６　フリックによってロケットを動かす

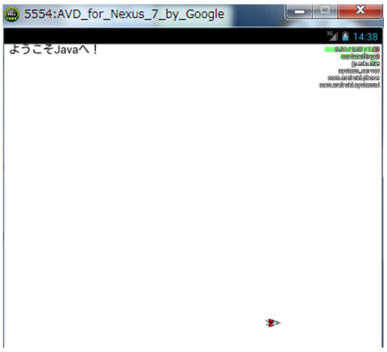
6.1　ソースコード

(1) テンプレート　Step の①の箇所に　Step060View　を入力してください。

(2) 次のアプリケーションを新規作成してください。

　　Step050View をコピー&ペーストして、ファイル名を「Step060View」に変更してください。

プロジェクト名：StepPro????(年組席)　　アプリケーション名：Step060View




```
/*    年    組    席   名前
 * Step060View
 *     フリックによってロケットを動かす
 */
・・・
import android.view.GestureDetector;
import android.graphics.Matrix;

public class Step060View extends SurfaceView
    implements
        SurfaceHolder.Callback,
        GestureDetector.OnGestureListener,
        GestureDetector.OnDoubleTapListener
{
    ・・・
    private Bitmap imageInit;
    private float downX;//ダウン時のＸ座標
    private float downY;//ダウン時のＹ座標
    private float flingX;//フリック時のＸ座標
    private float flingY;//フリック時のＹ座標
    private GestureDetector gestureDetector;//ジェスチャーディテクター
    private int direction;//メージの進行方向　1:上　　2:下　　3:右　　4:左
    static int moveStepSize=5;//image の１ステップの移動ピクセル数
    private int sleepTime=100;//スリープ設定時間（ミリ秒）
    private float rotateX;//回転の中心のＸ
    private float rotateY;//回転の中心のｙ

    //コンストラクタ
    public Step060View(Context context)
    {
        super(context);//context:アプリケーション環境の情報を保持する
        //画像の読み込み
        Resources r=context.getResources();//リソースオブジェクトの取得
        image=BitmapFactory.decodeResource(r, R.drawable.rocket);//読み込み
        imageInit=image;
        //サーフェイスホルダの作成
        holder=getHolder();//サーフェイスフォルダの取得
        holder.addCallback(this);//サーフェイスフォルダの通知先の指定
        holder.setFixedSize(getWidth(), getHeight());//サイズの指定
```

```
        this.rotateX = this.image.getWidth()/2.0f;
        this.rotateY = this.image.getHeight()/2.0f;
        //ジェスチャーディテクターの生成
        gestureDetector = new GestureDetector(context,this);
        setFocusable(true);//フォーカス指定
    }
    //サーフェイスの生成
    public void surfaceCreated(SurfaceHolder holder)
    {
        ・・・
        executor.scheduleAtFixedRate(
            new Runnable()
            {
                public void run()
                {
                    draw(canvas);
                    switch(direction)//image の進行方向
                    {
                        case 1://上
                            py-=moveStepSize;
                            break;
                        case 2://下
                            py+=moveStepSize;
                            break;
                        case 3://右
                            px+=moveStepSize;
                            break;
                        case 4://左
                            px-=moveStepSize;
                            break;
                        default://通常
                            py-=moveStepSize;
                    }
                    //端に着いたら反対側から出てくる
                    if(px<-image.getWidth()) px=getWidth();
                    if(px>getWidth()) px=-image.getWidth();
                    if(py<-image.getHeight()) py=getHeight();
                    if(py>getHeight()+image.getHeight()) py=-image.getHeight();
                }
            }, 0, sleepTime, TimeUnit.MILLISECONDS);
    }
    public void draw(Canvas canvas)
    {
        //Canvas オブジェクトをロックして取得
        canvas=holder.lockCanvas();
        canvas.save();
        canvas.drawColor(Color.WHITE);//背景を塗りつぶす
        canvas.drawText(TEXT1, 10, 20, paint);//文字列の表示
        canvas.drawBitmap(image, px, py, null);//画像の表示
        canvas.restore();
        holder.unlockCanvasAndPost(canvas);//実画面に反映
    }
    public boolean onTouchEvent(MotionEvent event)
    {
        gestureDetector.onTouchEvent(event);//ジェスチャーディテクターの設置
```

```
        return true;
    }
    //サーフェイスの変更
    public void surfaceChanged
                        (SurfaceHolder holder, int format, int w, int h){}
    //サーフェイスの破棄
    public void surfaceDestroyed(SurfaceHolder holder){executor.shutdown();}

    public boolean onDoubleTap(MotionEvent arg0) {return false;}
    public boolean onDoubleTapEvent(MotionEvent e) {return false;}
    public boolean onSingleTapConfirmed(MotionEvent e) {return false;}
    public boolean onDown(MotionEvent e) {return false;}
    //フリック時に呼ばれる　　画面に指を少しだけスライドさせる操作
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
                                                              float velocityY)
    {
        downX=e1.getX();//ダウン時の X 座標の取得
        downY=e1.getY();//ダウン時の Y 座標の取得
        flingX=e2.getX();//フリック時の X 座標の取得
        flingY=e2.getY();//フリック時の Y 座標の取得
        Matrix matrix= new Matrix();
        //image を初期状態に戻す
        image=imageInit;
        image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                    image.getHeight(), matrix, true);
        double kaku=Math.atan2(Math.abs(downY-flingY),
              Math.abs(downX-flingX))*180/Math.PI;//移動角度　アークタンジェント
        //Log.v("Tag",Double.toString(kaku));//デバック用 Logcat 出力
        if (kaku<45.1)//移動角度 45.1
        {
            if(flingX>downX)
            {
                direction=3;//右
                matrix.postRotate(90.0f,rotateX,rotateY);
                image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                            image.getHeight(), matrix, true);
            }else{
                direction=4;//左
                matrix.postRotate(-90.0f,rotateX,rotateY);
                image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                            image.getHeight(), matrix, true);
            }
        }else{
            if(flingY>downY)
            {
                direction=2;//下
                matrix.postRotate(180.0f,rotateX,rotateY);
                image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                           image.getHeight(), matrix, true);
            }else{
                direction=1;//上
                matrix.postRotate(0.0f,rotateX,rotateY);
                image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                            image.getHeight(), matrix, true);
            }
```

```
        }
        return true;
    }
    public void onLongPress(MotionEvent e) {}
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
                                            float distanceY) {return false;}
    public void onShowPress(MotionEvent e) {}
    public boolean onSingleTapUp(MotionEvent e) {return false;}
}
```

6.2　ジェスチャーディテクターの生成

フリックとは、画面に指を少しだけスライドさせる操作のことです。エミュレータでは、マウスで少しドラッグすることです。ジェスチャーイベントの一種で、これ以外に８種類あります。日本語で「弾く」「払い落とす」という意味があります。

ジェスチャーイベントを利用するには、まずジェスチャーディテクターを生成します。ディテクターとは日本語で「探知装置」という意味があります。ジェスチャーディテクターを生成するには、GestureDetector クラスを使います。

```
gestureDetector = new GestureDetector(context,this);
```

このプログラムでは、Step060View クラス自身を通知先とするリスナーとして指定しています。

■ジェスチャーイベント

日本語表記	英語表記	操作方法
タップ	Tap Single tap	画面を指先(ペン先)で１回叩く
ダブルタップ 二回タップ	Double tap	画面を指先(ペン先)で２回叩く(突く)
ロングタップ ロングプレス 長押し	Long press Long tap	画面を指先(ペン先)で長く押す
フリック	Flick	画面に触れた指先(ペン先)を少しスライドさせ、素早く払う(弾く)ようにタッチ
右フリック	Flick Right Fling right Right flick	左から右にフリック→
左フリック	Flick Left Fling left Left flick	右から左にフリック←
上フリック	Flick up Fling up Up flick	下から上にフリック↑
下フリック	Flick down Fling down Down flick	上から下にフリック↓
マルチタッチ マルチタップ	Multi-tap Multitech	同時に２本以上の指で操作すること
ピンチ操作 ピンチズーム	Pinch Pinch to Zoom Pinch-Zoom	同時に２本の指で操作すること

	Pinch and Zoom	
ピンチイン	Pinch-in Pinch-close	2 本の指の間隔を縮めて画面をつまむようにする動作
ピンチアウト	Pinch-out Pinch-open	2 本の指の間隔を広げる動作

6.3　ジェスチャーディテクターの設置

```
public boolean onTouchEvent(MotionEvent event)
{
    gestureDetector.onTouchEvent(event);//ジェスチャーディテクターの設置
    return true;
}
```

ジェスチャーディテクターは、タッチイベントの通知先となる onTouchEvent()メソッド内に設置します。GestureDetector クラスの onTouchEvent()に引数としてタッチイベントを渡します。

6.4　ジェスチャーイベント発生時の処理を行うインタフェースの実装

```
public class Step060View extends SurfaceView
    implements
        SurfaceHolder.Callback,
        GestureDetector.OnGestureListener,
        GestureDetector.OnDoubleTapListener
{
```

ジェスチャーイベント発生時の処理を行う GestureDetector.OnGestureListener インタフェース GestureDetector.OnDoubleTapListener インタフェースを実装します。

このプログラムでは、Step060View 自身がリスナーになるので、implements キーワードを使って定義した後、実際に処理を行うメソッドを記述します。

```
//フリック時に呼ばれる　　画面に指を少しだけスライドさせる操作
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
float velocityY)
{
```

文法

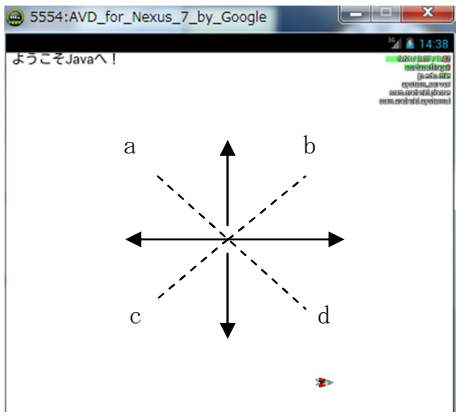
```
onFLing(MotionEvent event1, MotionEvent event2, float velocity,
                                                    float velocity)
```

event1　　・・・　　ダウン時のタッチイベント
event2　　・・・　　フリック時のタッチイベント
velocityX　・・・　　X 軸の速度（ピクセル／秒）
velocityY　・・・　　Y 軸の速度（ピクセル／秒）

```
downX=e1.getX();//ダウン時の X 座標の取得
downY=e1.getY();//ダウン時の Y 座標の取得
flingX=e2.getX();//フリック時の X 座標の取得
flingY=e2.getY();//フリック時の Y 座標の取得
Matrix matrix= new Matrix();
//image を初期状態に戻す
image=imageInit;
image=Bitmap.createBitmap(image, 0, 0, image.getWidth(),
                                    image.getHeight(), matrix, true);
//移動角度　アークタンジェント
```

```
double kaku=Math.atan2(Math.abs(downY-flingY),
                                        Math.abs(downX-flingX))*180/Math.PI;
//Log.v("Tag",Double.toString(kaku));//デバック用 Logcat 出力
if (kaku<45.1)//移動角度 45.1
{
```

移動角度が 45.1 度を基準にして上下左右の判断をしています。



(a)と(b)の破線の間は、上へ　　(c)と(d)の破線の間は、下へ
(a)と(c)の破線の間は、右へ　　(b)と(d)の破線の間は、左へ

（１）double kaku=Math.atan2(Math.abs(downY-flingY),
Math.abs(downX-flingX))*180/Math.PI;

①Math.atan2(Math.abs(downY-flingY), Math.abs(downX-flingX))

角度をラジアンで求める。

②*180/Math.PI

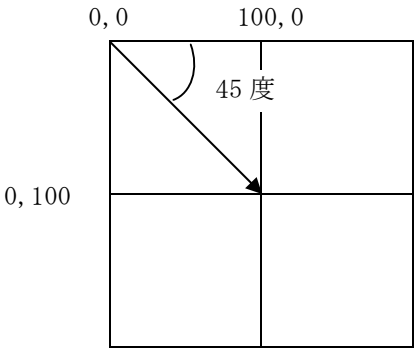
ラジアンから角度の変換は、ラジアン ＊ 180 / 円周率です。
角度の範囲は-180～+180 になります。

③Math.PI　… 円周率

（２）public static double atan2(double y, double x)

atan2 関数は座標の逆正接(アークタンジェント)を計算して返します。
(単位はラジアン)

例えば、Math.atan2(1, 1)とした場合、角度は 45 度になりタンジェントは tan(45 度) = 1 / 1 = 1 となります。

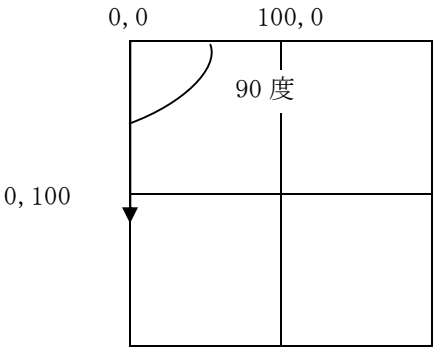


atan2 関数はタンジェントの結果が 1 となるような角度を取得するために、タンジェントの結果ではなく座標を指定します。

つまりこの関数は座標を指定することで、原点からその座標に引いた直線の X

軸からの角度を取得することができます。

なお X 座標が 0 だった場合、Y 座標も 0 ならば角度は 0、Y 座標が正の値ならば 90 度、Y 座標が負の値ならば-90 度となります。(tan0=0 になります)



tan や sin、cos は、角度に対する関数です。例えば、tan60°＝√3　のように角度を入力すると値が出てきます。逆に、**アークタンジェント**などは、数値に対する関数です。

arctan√3=60°

などのように、数値を入力すると角度が出てきます。

そして、タンジェントとアークタンジェントの関係は逆関数という関係です。逆関数というのは、原因と結果が逆になるような関数です。

例えば、

45° → tan →1

1　→ arctan →45°

のように、「1」と「45°」が逆の位置にあるような関係を**逆関数**といいます。

6.5 度数と弧度(ラジアン)との変換

通常、度(度数)とラジアン(弧度)の変換には以下のような式を使います。

ラジアン = 度数 *（π / 180）

度数 = ラジアン *（180 / π）

(1)そもそもπとは何か

πとは「円周の長さを直径で割った率」のことです。図 1 だと、「円周率(π) = 円周の長さ(C) ÷ 直径(d)」となります。そのため、直径 1 の円の円周の長さは約 3.14 となります。また同時に、「円周の長さ(C) = 円周率(π) × 直径(d)」となります。

円周率を求めるためには「円周の長さ」が必要となり、円周の長さを求めるためには「円周率」が必要となります。それでは円周率の 3.14...とはどのように求められるのでしょうか？実は、別に様々な円周率の計算方式がありそれらを使って円周率の 3.14...を近似で求めているのです。

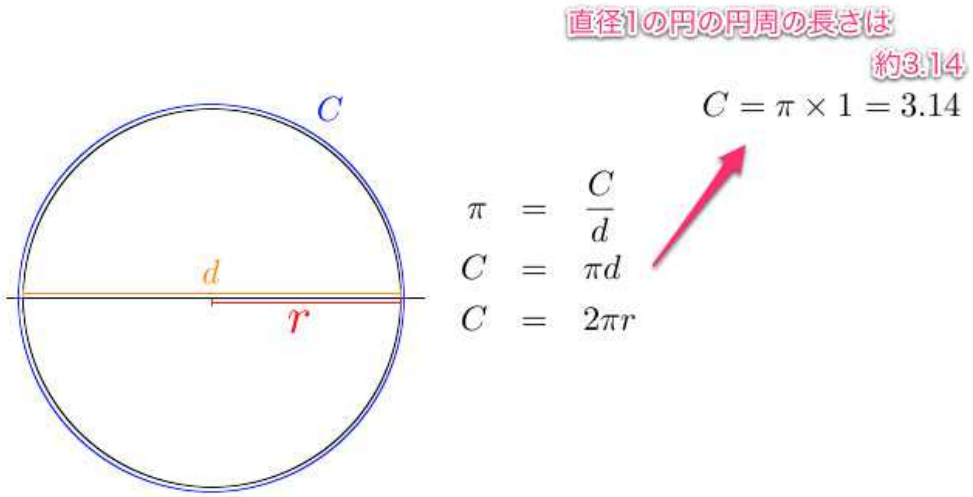


図 1: 円周率とは

(2) ラジアンとは何か

1 ラジアン(rad)とは、「半径と円周の弧の長さと等しくなる角度」とされています。図 2 では、「半径(r) = 弧(l)となる場合の角度(θ)が 1rad」となります。

弧の長さ(l)は、角度(θ)に比例して大きくなるので「弧の長さ(l) = 半径(r) × 角度(θ)」となります。相互変換できるのでどちらでも良いのですが、厳密には、この角度というのは「度(°)」ではなく、「ラジアン(rad)」です。

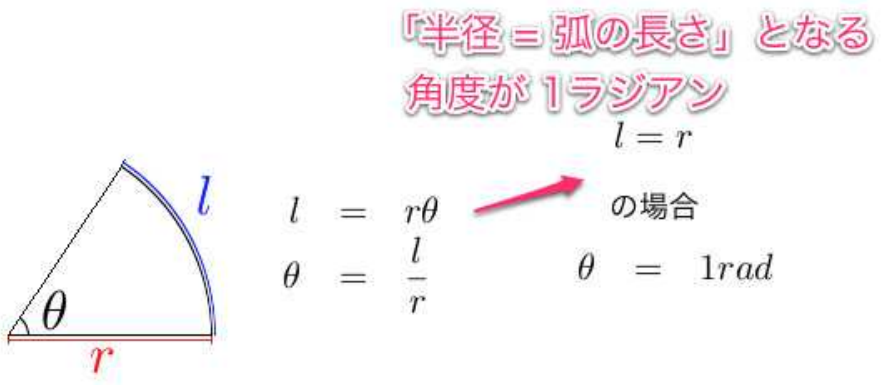


図 2: ラジアンとは

回転させるという行為は円を書くということですが、その角度を求めるのになぜ sin 関数や cos 関数ではラジアンを使うのでしょうか？それは単に、円との相性が良い為です。「半径(r)が 1」の単位円で考えてみると、「弧(l) = 角度(θ)」となり「弧(l)は角度(θ)によってのみ決まる」ということになります。つまり、どれだけの距離を移動すれば良いかがラジアンから求められるということです。

(3) 度とラジアンの求め方

さて、いよいよタイトルの通り「度数と弧度との変換」ですが、図 3 の通りの計算式で求められます。上 2 つよりも長くなってはいますが簡単です。例えば、「角度(θ)が 180° の場合」を考えます。

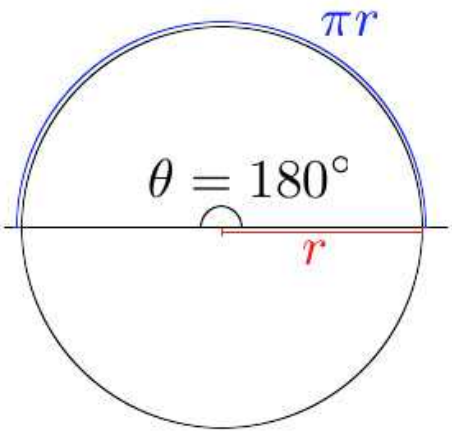
その場合、図 1 で求めた式を利用して「円周の長さの半分(C/2) = 2πr/2 = πr」となります。そして、図 2 で求めた式を利用し、「角度(θ) = 弧(l)/半径(r)」に代入すると「角度(θ) = πr/r = π」となります。つまり、「180° = π (rad)」です。

「180° の場合、πrad」なのですから「1° は(π/180)rad」「1rad は(180/π)° 」となります。つまり、最終的には、最初に書いた式の

　　ラジアン = 度数 *（π / 180）

　　度数 = ラジアン *（180 / π）

となります。



$$\theta = \frac{\pi r}{r}$$
$$\theta = \pi$$
$$180^\circ = \pi$$
$$(360^\circ = 2\pi)$$

よって

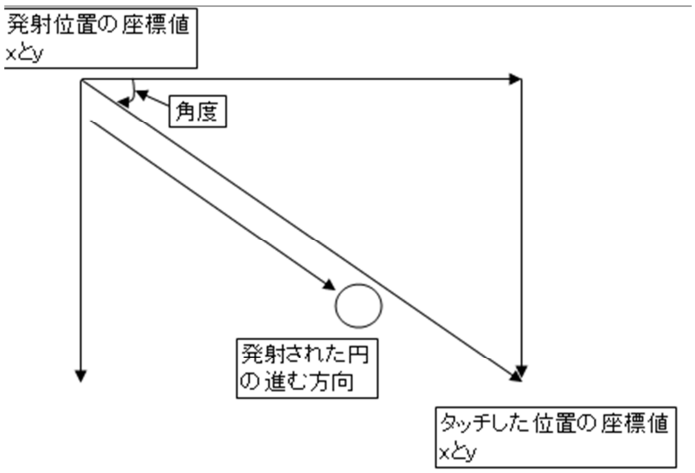
$$1^\circ = \frac{\pi}{180},\ 1rad = \frac{180}{\pi}$$

$$\text{ラジアン} = \frac{\pi}{180}\theta^\circ$$
$$\text{度数} = \frac{180}{\pi}\theta rad$$

図 3: 度数とラジアンの求め方

（参考）タッチした方向に描画した円を発射する

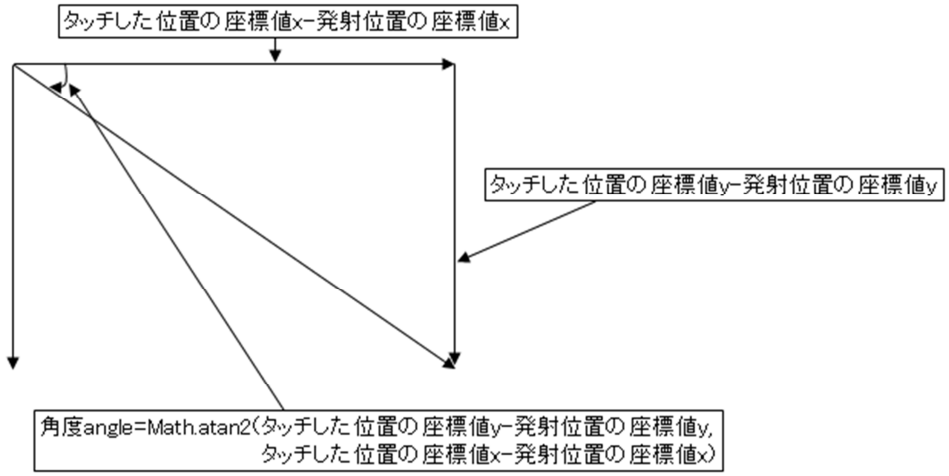
Android のタッチした方向に描画した円を発射するには、発射されるスタートの座標値からタッチした座標値を求める。



①スタートの座標値から、タッチした座標値の角度を考える

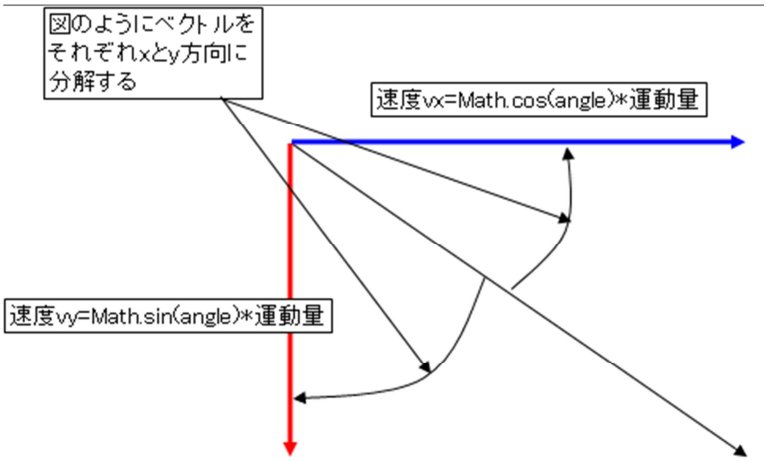
この場合の角度を求めるには、アークタンジェントを使う。

このアークタンジェントを求めるには、Math クラスの atan2(double y,double x)を使用する。これに当てはめると double angle=Math.atan2(発射位置の y-タッチした位置の y,発射位置の x-タッチした位置の x)となる。この、atan2()メソッドは極座標に変換するするものです。



②今度は速度を求める。

角度を求めたので、今度は求めた角度を使って速度を求める。
この速度を求めるには、タッチした座標値と発射する座標値を x と y それぞれ分けて、それに運動量を掛けると求められる。
この考え方はベクトルの分解と言う考え方を使う。
そうすると
　double 速度 x=Math.cos(angle)*運動量
　double 速度 y=Math.sin(angle)*運動量
から、速度を求めることが可能となる。



あとは、タッチイベントの処理メソッド内で発射する座標値とタッチする座標値を Point クラスを使用してそれぞれ求めることで発射することが出来る。

（参考）現在位置からランダムな方向(角度)に任意の距離移動するプログラム

シミュレーション等で，現在位置からランダムに決定した角度に対して，任意の距離移動させたい場合などが結構ある。以下方法を記述する。

◇ランダムに角度 θ(ラジアン)を決定する。

角度$\theta = 2\pi Z$
Z は 0 以上 1 未満の範囲をとる一様確率変数.

・Java での記述方法

```
Random r = new Random();
double radian = 2.0 * Math.PI * r.random();
```

◇角度 θ の方向に，現在の座標(x, y)から距離 d 分だけ移動した座標値を求める。つまり移動先を決める。

移動後の座標 x = 現在の座標 x + 任意の距離 d × cosθ
移動後の座標 y = 現在の座標 y + 任意の距離 d × sinθ

・Java での記述方法

```
double next_x = x + d * Math.cos(radian);
double next_y = y + d * Math.sin(radian);
```