玉転がし

タブレット端末の特徴の一つとして，センサを使った動作や，指による画面操作がある．それらを活用して，図形を動かすアプリの例を示す．

1. プロジェクトを作る

Tama アプリケーションを作る，Tama プロジェクトを作る．

図 1　プロジェクト作成

プロジェクトの構成を設定する．

図 2　プロジェクトの構成

ランチャー・アイコンを設定する．

図 3　ランチャー・アイコンを設定する

BlankActivity を作る.

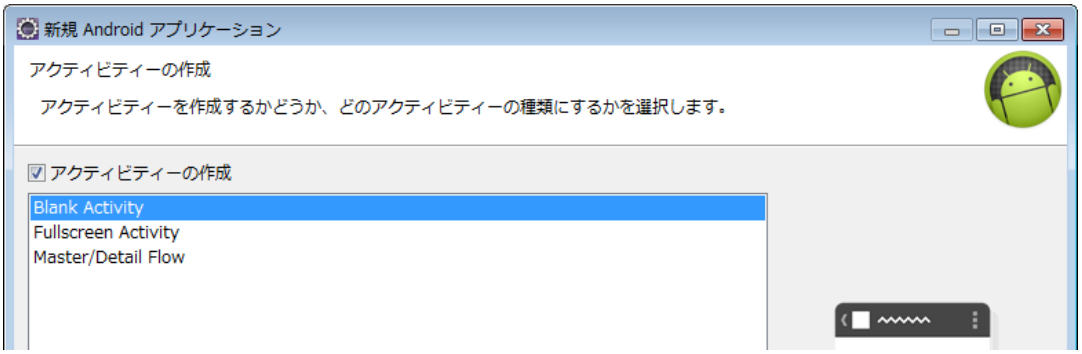

図 4　Blank Activity を作る

BlankActivity の名前を MainActivity とする.

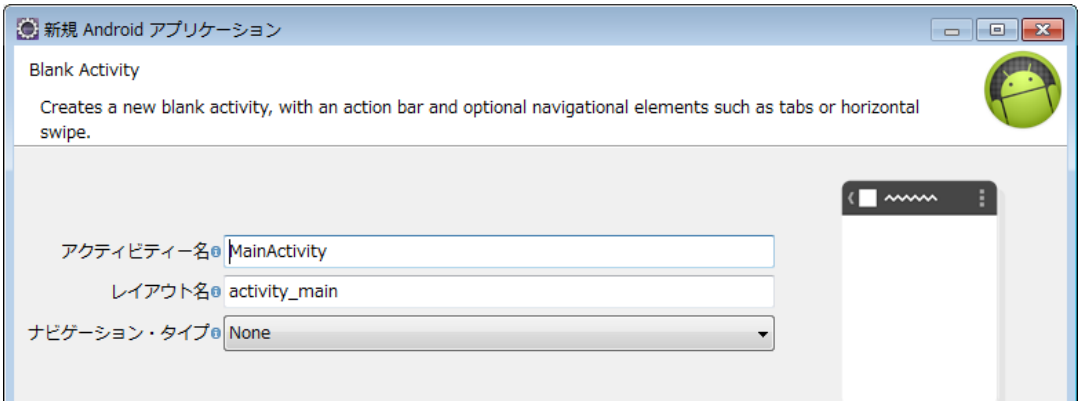

図 5　アクティビティの名前をつける

2. レイアウトを作成する

RelativeLayout をレイアウトファイルで定義し，表示する View をプログラムで作成して設置する．そのため，標準で作成されるレイアウトを XML 編集画面から削除し，グラフィカル・レイアウト画面にて，RelativeLayout を追加する．

activity_main.xml を開き，XML 編集タブを開く．現在書かれている内容を削除する．

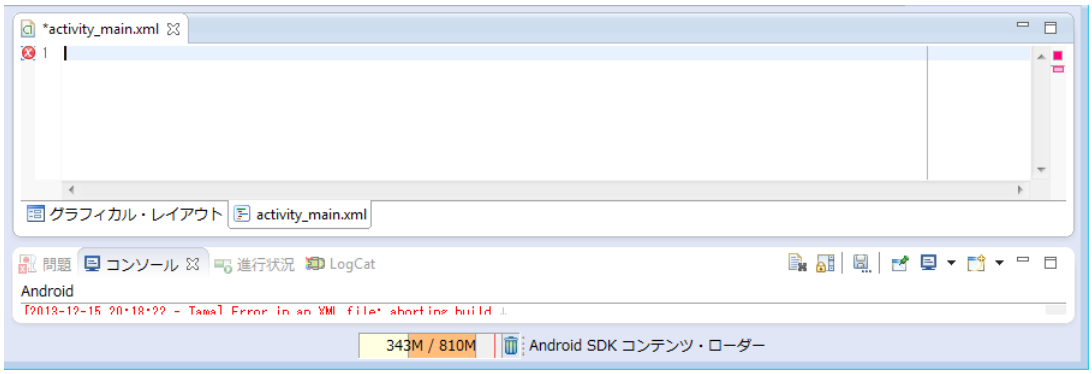

図 6　標準のレイアウトを削除する

グラフィカル・レイアウトタブに切り替えて，パレットの「レイアウト」から RelativeLayout をアクティブワークスペースにドラッグアンドドロップする．

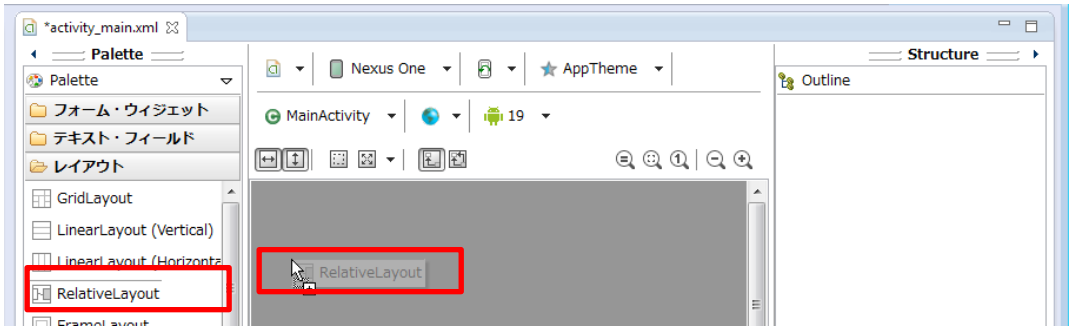

図 7　RelativeLayout を追加

ここで，RelativeLayout と View にそれぞれ ID をつける．ここでは，「wholeLayout」とする．さらに，センサの値を確認するための，TextView を追加する．その TextView の ID を「sensorView」とする．

結果として，リスト 1 のような XML ソースになる．

リスト 1 XML ファイルの状態

```
 1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 2      android:id="@+id/wholeLayout"
 3      android:layout_width="fill_parent"
 4      android:layout_height="fill_parent" >
 5
 6      <TextView
 7          android:id="@+id/sensorView"
 8          android:layout_width="fill_parent"
 9          android:layout_height="wrap_content"
10          android:text=""
11          android:textAppearance="?android:attr/textAppearanceMedium" />
12
13  </RelativeLayout>
```

3. マニフェストファイル

縦長の状態で固定させたいので，マニフェストファイルで指定する．AndroidManifest.xml ファイルを開き，固定したい Activity を選択する．Activity の属性（Attributes）の中で「Screen orientation」として「portrait」を指定する．

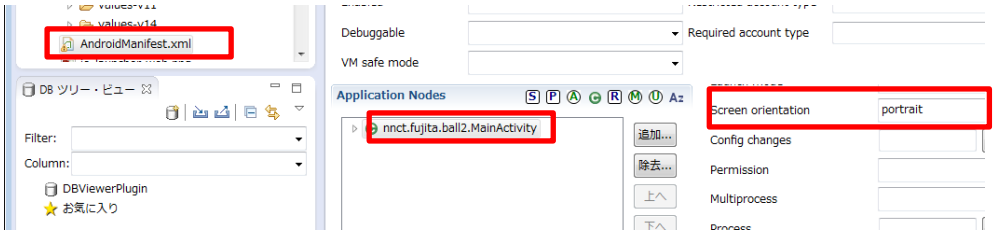

図 8　画面の表示方向を指定する

4. Tamaクラス

玉を描画する図形を表示するための Tama クラスを作成する．

プロジェクトフォルダを右クリックし，「新規」->「クラス」を選択する．

クラス名を「Tama」にするので，「名前」を「Tama」とする．

図 9　クラス名をつける

Tama クラスは View クラスを継承するので，スーパークラスの項目の「追加」をクリックする．

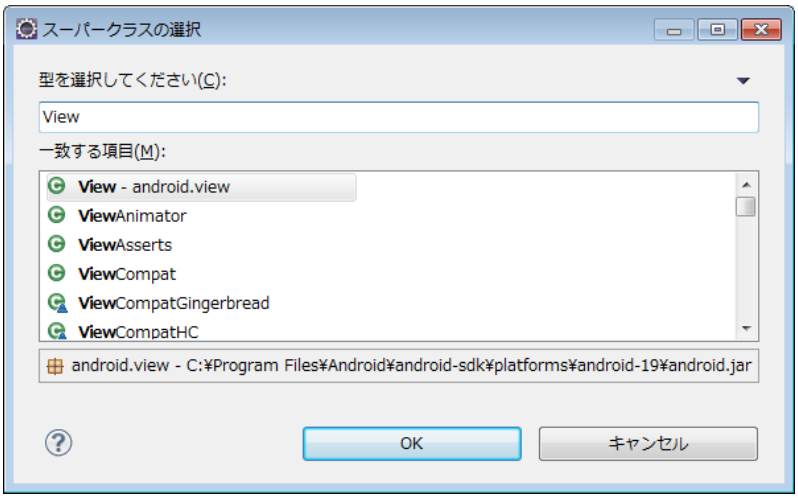

図 10　継承するクラスの追加

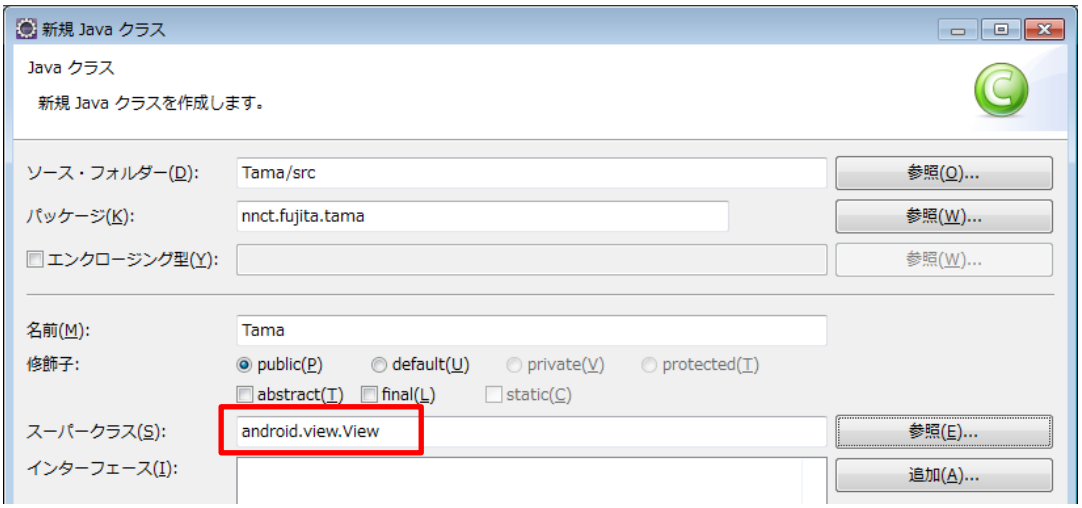

図 11　継承するクラスを追加した様子

Tama.java を開くと，Tama クラス名にエラーが表示されているので，カーソルを合わせると，引数が指定されたコンストラクタを定義する必要があることを助言してくれるので，Tama(Context)を選ぶ．

図 12 コンストラクタを定義する

描画するための内容を書き入れる.

リスト 2 Tama クラス

```
1  public class Tama extends View
2  {
3
4      float radius = 100.f;
5      Paint paint = new Paint();
6      public Tama(Context context) {
7          super(context);
8      }
9
10     // 描画処理を記述
11     @Override
12     protected void onDraw(Canvas canvas)
13     {
14         //描画する色を定義
15         paint.setColor(Color.argb(255, 0, 0, 255));
16         //塗りつぶしの円にする
17         paint.setStyle(Paint.Style.FILL_AND_STROKE);
18         paint.setStrokeWidth(0);
19         //円を描画
20         canvas.drawCircle((int)(radius * 1),(int)(radius * 1), radius, paint);
21     }
22     protected void onMeasure(int width,int height)
23     {
24         super.onMeasure(width, height);
25
26         // View の描画サイズを指定する
27         setMeasuredDimension((int)(radius * 2),(int)(radius * 2));
28     }
29 }
```

5. MainActivityクラス

(1) フィールド

MainActivity 内のフィールドをリスト 3 のように設定する.

リスト 3　MainActivity クラスのフィールド

```
1  private SensorManager mSensorManager;//センサーマネージャーのオブジェクト
2  private SensorEventListener sensorListener;//センサーリスナーのオブジェクト
3
```

```
 4 public Tama tama;//玉のオブジェクト
 5 public Point displaySize;//玉を表示できる領域を記憶する
 6 private RelativeLayout wholeLayout;//レイアウトファイルから取得する画面のレイアウト
 7
 8 public int displayLeft; //図形の左座標
 9 public int displayTop; //図形の上座標
10 public int distanceX; //タッチした x 座標と図形の左座標の差
11 public int distanceY; //タッチした y 座標と図形の上座標の差
```

(2) onCreateメソッド

MainActivity を表示するときに，玉を作り，画面上に表示するように，リスト 4 のように，onCreate メソッドに設定する．なお，玉には，玉をタッチにより移動するためのリスナを設定する．

リスト 4　MainActivity クラスの onCreate メソッド

```
 1 public void onCreate(Bundle savedInstanceState)
 2 {
 3    super.onCreate(savedInstanceState);
 4    setContentView(R.layout.activity_main);
 5
 6    //画面サイズを取得する
 7    WindowManager wm = (WindowManager) getSystemService(WINDOW_SERVICE);
 8    Display disp = wm.getDefaultDisplay();
 9    displaySize = new Point();
10    disp.getSize(displaySize);
11
12    //Tama オブジェクトをインスタンス化する
13    tama = new Tama(this);
14    //Tama オブジェクトにタッチしたときのリスナーを設定する
15    tama.setOnTouchListener(new View.OnTouchListener()
16    {
17       public boolean onTouch(View view, MotionEvent event)
18       {
19          switch (event.getAction())
20          {
21          //動かしたときの動作
22          case MotionEvent.ACTION_MOVE:
23             displayLeft = (int) event.getRawX() - distanceX;
24             displayTop = (int) event.getRawY() - distanceY;
25
26             //玉を移動させる
27             view.layout(displayLeft,
28                    displayTop,
29                    displayLeft + view.getWidth(),
30                    displayTop + view.getHeight());
31             break;
32
33          //タッチした時の動作
34          case MotionEvent.ACTION_DOWN:
45             //図形の左上座標とタッチ座標の差分をとる
36             distanceX = (int) (event.getRawX() - view.getLeft());
37             distanceY = (int) (event.getRawY() - view.getTop());
38             break;
39
40          //指を離したときの動作
41          case MotionEvent.ACTION_UP:
42             break;
43          }
```

```
44              return true;
45          }
46      });
47
48      //起動時に図形を置く場所を指定する
49      displayLeft = displaySize.x / 2;
50      displayTop = displaySize.y / 2;
51
52      //図形を移動する
53      tama.layout(displayLeft,
54              displayTop,
55              displayLeft + tama.getWidth(),
56              displayTop + tama.getHeight());
57
58      //画面に Draw オブジェクトを表示する
59      wholeLayout = (RelativeLayout) findViewById(R.id.wholeLayout);
60      wholeLayout.addView(tama);
61  }
```

(3) onCreateOptionsMenuメソッド

メニューを表示するメソッドを定義する．ここでは，デフォルトのままにする．

リスト 5　MainActivity クラスの onCreateOptionsMenu メソッド

```
1  @Override
2  public boolean onCreateOptionsMenu(Menu menu)
3  {
4      // Inflate the menu; this adds items to the action bar if it is present.
5      getMenuInflater().inflate(R.menu.main, menu);
6      return true;
7  }
```

(4) onStopメソッド

アクティビティが止まった時にセンサの変化に対する監視を解除するため，onStop メソッドにてリスナを解除する．

リスト 6　MainActivity クラスの onStop メソッド

```
1  protected void onStop()
2  {
3      // TODO Auto-generated method stub
4      super.onStop();
5      // Listener の登録解除
6      mSensorManager.unregisterListener(sensorListener);
7  }
```

(5) onResumeメソッド

MainActivity を起動する際に実行される onResume にて，センサから値を取得する処理と，センサを監視して，センサの値が変わったときに処理する内容を定義する．

センサに対して，sensorListner オブジェクトをリスナーとして設定する．

リスト 7　MainActivity クラスの onResume メソッド

```
1  @Override
2  protected void onResume()
3  {
4      super.onResume();
5
```

```
 6      //センサーの情報を取得するためのセンサーマネージャーを立ち上げる
 7      mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
 8
 9      //センサーが変化したときの処理内容が含まれたクラスをインスタンス化する
10      sensorListener = new RotationSet(this);
11
12      // 加速度センサの取得
13      List<Sensor> list = mSensorManager
14              .getSensorList(Sensor.TYPE_ACCELEROMETER);
15      if (list.size() > 0)
16      {
17          Sensor _accelerometer = list.get(0);
18          // SensorEventListenerを登録
19          mSensorManager.registerListener(sensorListener,
20                                      _accelerometer,
21                                      SensorManager.SENSOR_DELAY_UI);
22      }
23
24      // 地磁気センサの取得
25      List<Sensor> list2 = mSensorManager
26              .getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
27      if (list2.size() > 0)
28      {
29          Sensor _magneticField = list2.get(0);
30          // SensorEventListenerを登録
31          mSensorManager.registerListener(sensorListener,
32                                      _magneticField,
33                                      SensorManager.SENSOR_DELAY_UI);
34      }
35  }
```

(6) onWindowFocusChangedメソッド

ナビゲーションバーを除いた表示領域を取得したいので，表示する MainActivity にフォーカスがきまり，View の表示が定まった時点で，表示画面のサイズを取得する．

リスト 8　MainActivity クラスの onWindowFocusChanged メソッド

```
 1  public void onWindowFocusChanged(boolean hasFocus)
 2  {
 3      super.onWindowFocusChanged(hasFocus);
 4
 5      wholeLayout = (RelativeLayout) findViewById(R.id.wholeLayout);
 6      displaySize.x = wholeLayout.getWidth();
 7      displaySize.y = wholeLayout.getHeight();
 8  }
```

6. RotationSetクラス

以下のようにセンサに設定する，リスナにするためのクラスを RotationSet クラスとして定義する．

```
mSensorManager.registerListener(sensorListener,
                                _accelerometer,
                                SensorManager.SENSOR_DELAY_UI);
```

リスト 9　RotationSet クラス

```
 1  public class RotationSet implements SensorEventListener
```

```
{
    private boolean MagneticSensorReady;
    private boolean AccerelateSensorReady;
    float magneticValues[] = new float[3];
    float accelerometerValues[] = new float[3];

    //センサによる X-Y の移動量
    public int diffX, diffY;

    MainActivity activity;

    RotationSet(MainActivity activity)
    {
        this.activity = activity;
    }

    @Override
    public void onSensorChanged(SensorEvent event)
    {
        /* 回転行列 */
        float rotation_matrix[] = new float[16];
        /* 傾斜行列 */
        float inclination_matrix[] = new float[16];
        /* 向き */
        float orientation_matrix[] = new float[3];

        //センサごとに値を取得する
        switch (event.sensor.getType())
        {
            //地磁気センサのとき
            case Sensor.TYPE_MAGNETIC_FIELD:
                magneticValues = event.values.clone();
                MagneticSensorReady = true;
                break;
            //加速度センサのとき
            case Sensor.TYPE_ACCELEROMETER:
                accelerometerValues = event.values.clone();
                AccerelateSensorReady = true;
                break;
        }

        //地磁気センサと加速度センサの値が両方取得できたら
        if (AccerelateSensorReady && MagneticSensorReady)
        {
            AccerelateSensorReady = false;
            MagneticSensorReady = false;

            // 加速度センサと地磁気センサから，回転行列と傾斜行列を得る
            SensorManager.getRotationMatrix(rotation_matrix,
                                            inclination_matrix,
                                            accelerometerValues,
                                            magneticValues);
            //画面の回転状態に応じて，座標を変換する
            getOrientationAsRotation(rotation_matrix, orientation_matrix);


            TextView tv = (TextView)activity.findViewById(R.id.sensorView);
            tv.setText( "Z:" + Math.toDegrees(orientation_matrix[0]) + "¥n"+
                      "Y:" + Math.toDegrees(orientation_matrix[1]) + "¥n"+
                      "X:" + Math.toDegrees(orientation_matrix[2]) );

            //回転角の値をトリガに移動量を決める
            diffX = (int) Math.toDegrees(orientation_matrix[2]) * (-5);
```

```
65            diffY = (int) Math.toDegrees(orientation_matrix[1]) * (5);
66
67
68            //画面からはみ出ないように調整する
69            if (0 <= activity.displayLeft - diffX
70               && activity.displayLeft - diffX
71  < (activity.displaySize.x - activity.tama.getWidth()))
72            {
73               activity.displayLeft -= diffX;
74            }
75            else if (activity.displayLeft - diffX < 0)
76            {
77               activity.displayLeft = 0;
78            }
79            else if ((activity.displaySize.x - activity.tama.getWidth())
80  < activity.displayLeft - diffX)
81            {
82               activity.displayLeft        =        activity.displaySize.x        -
    activity.tama.getWidth();
83            }
84
85            if (0 <= activity.displayTop - diffY
86                  && activity.displayTop - diffY
87  < (activity.displaySize.y - activity.tama.getWidth()))
88            {
89               activity.displayTop -= diffY;
90            }
91            else if (activity.displayTop - diffY < 0)
92            {
93               activity.displayTop = 0;
94            }
95            else if (activity.displaySize.y - activity.tama.getWidth()
96  <= activity.displayTop - diffY)
97            {
98               activity.displayTop =
                        (activity.displaySize.y - activity.tama.getWidth());
99            }
100
101           //玉を移動させる
102           activity.tama.layout(activity.displayLeft,
103                                activity.displayTop,
104                                activity.displayLeft + activity.tama.getWidth(),
105                                activity.displayTop + activity.tama.getHeight());
106       }
107    }
108
109    public void getOrientationAsRotation(float[] rotate, float[] out)
110    {
111       // ディスプレイを取得する
112       Display disp = activity.getWindowManager().getDefaultDisplay();
113       //標準状態からのディスプレイの角度を求める
114       int dispDir = disp.getRotation();
115
116       // 画面回転してない場合はそのまま
117       if (dispDir == Surface.ROTATION_0)
118       {
119          SensorManager.getOrientation(rotate, out);
120       }
121       else
122       {
123          float[] outR = new float[16];
124
125          switch(dispDir)
126          {
```

```
127                 case Surface.ROTATION_90:
128
129                     SensorManager.remapCoordinateSystem(rotate,
130                                                         SensorManager.AXIS_Y,
131                                                         SensorManager.AXIS_MINUS_X,
132                                                         outR);
133                 break;
134                 case Surface.ROTATION_180:
135
136                     float[] outR2 = new float[16];
137                     SensorManager.remapCoordinateSystem(rotate,
138                                                         SensorManager.AXIS_Y,
139                                                         SensorManager.AXIS_MINUS_X,
140                                                         outR2);
141                     SensorManager.remapCoordinateSystem(outR2,
142                                                         SensorManager.AXIS_Y,
143                                                         SensorManager.AXIS_MINUS_X,
144                                                         outR);
145                 break;
146                 case Surface.ROTATION_270:
147                     SensorManager.remapCoordinateSystem(rotate,
148                                                     SensorManager.AXIS_MINUS_Y,
149                                                     SensorManager.AXIS_X,
150                                                     outR);
151                 break;
152             }
153             SensorManager.getOrientation(outR, out);
154         }
155     }
156
157     @Override
158     public void onAccuracyChanged(Sensor sensor, int accuracy)
159     {
160         // TODO 自動生成されたメソッド・スタブ
161     }
162 }
```

7. 実行する