TAITO

株式会社タイトー

MIDPアプリ移植入門

株式会社　タイトー
ON!AIR事業本部ゲーム企画部開発課
藤本　豊

株式会社タイトー

SPACE INVADERS 30th ANNIVERSARY

自己紹介（携帯アプリ歴）

1999年 2月　ｉモード、EZweb、J-スカイ
　　6月　他社、待ち受け画像コンテンツスタート
　　11月　会員50万人突破の報道
2000年 3月　会員100万人突破の報道
2001年 1月　ｉアプリ
　　6月　JavaTMアプリ（現S!アプリ）
　　7月　EZアプリ（Java）
　　|
　　BREWとか
　　|
　　以降ずっと携帯アプリ開発
　　現在に至る、、、

注意

※　Canvasだけを使用したアプリケーションに特化しています

※　Androidアプリ開発の細かい部分の説明は割愛しています

※　説明にあるコードは説明用に用意したものです

なぜMIDPから？

現在の携帯アプリプログラマーであれば、ほぼMIDP、
DoJaプログラマと言えるから

会社にとりあえず説得しやすいから

アプローチ

Androidアプリケーション開発の勉強をしようと思いました

既存の携帯Javaアプリの移植を目標とすることで、短期間、集中的に差異を把握することにしました

移植対象となる携帯Javaアプリは「CLDC + MIDP」の仕様範囲で製作されていることが望ましいと考えました

拡張ライブラリを使っているアプリでは面倒なことが予想されるので

MIDPアプリのコードを解析しながら、
Androidアプリのコードに置き換えていくことにしました

MIDPアプリその１

MIDPエミュレーター起動

MIDPアプリその１を移植するにあたって、考慮するコード

- 永続ループ処理方法
- キー検出方法
- 画像描画方法全般

MIDPアプリその１：永続ループ処理方法

RayTest.000.java - WZ C PROGRAM EDITOR

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class RayTest extends MIDlet{

    static RayTestCanvas c;

    public RayTest(){
        c = new RayTestCanvas();
        Display.getDisplay(this).setCurrent(c);
    }

    public void startApp(){
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean unconditional){
    }

}
```

RayTestCanvas.000.java - WZ C PROGRAM EDITOR

```
import java.util.*;
import javax.microedition.lcdui.*;

public class RayTestCanvas extends Canvas implements Runnable{

    final int SLEEP_COUNT = 30;// ms

    long CYCLE_TIME = SLEEP_COUNT;// １サイクルあたりの時間
    long startTime;// 処理開始時刻
    long passTime;// 処理にかかった時間

    RayTestCanvas(){
    }

    public void paint(Graphics g){

        // ここに全ての処理を書く

    }

    public void run(){

        while(true){

            //処理開始前に時刻を記録
            startTime = System.currentTimeMillis();

            // 全処理と描画
            repaint();

            // 仮想画面を実画面へ転送
            serviceRepaints();

            try{
                // 処理にかかった時間を計算
                passTime = System.currentTimeMillis() - startTime;

                // [処理にかかった時間]を差し引いてスリープ
                if(( 0 <= passTime )&&(passTime < CYCLE_TIME)){
                    try{
                        // ウェイト
                        Thread.sleep(CYCLE_TIME - passTime);
                    }
                    catch(Exception e){
                    }
                }
            }
            catch(Exception e){
                System.out.println("Exception: run() " + e);
                e.printStackTrace();
            }

        }

    }
}
```

Canvasではコンストラクタ（RayTestCanvas()）、paint()、と実行されます。

その後、「inplements Runnable」しているので run() が実行されます。

run() の中では「while(true)」としているので、永久ループします。

永久ループの中 repaint() を呼ぶことで、paint()が呼ばれるので、永続ループとなります。

このサンプルでは1回のループ処理が30msecより早く終わっても、差分時間sleep処理を行うことで、安定した間隔でのループ処理を実現しています。

Androidアプリその１：永続ループ処理方法

RayTest.org.java.andr - WZ C PROGRAM EDITOR

```
package com.taito.android.raytest;
import android.app.Activity;
import android.os.Bundle;
import android.view.Window;
public class RayTest extends Activity
{
    @Override
    public void onCreate( Bundle icicle )
    {
        super.onCreate( icicle );
        // タイトルバー非表示
        requestWindowFeature( Window.FEATURE_NO_TITLE );
        // Viewをセット
        setContentView( new RayTestView( this ) );
    }
}
```

onCreate() で setContentView() すると、View のコンストラクタ（RayTestView()）、onDraw() の順番で実行される。

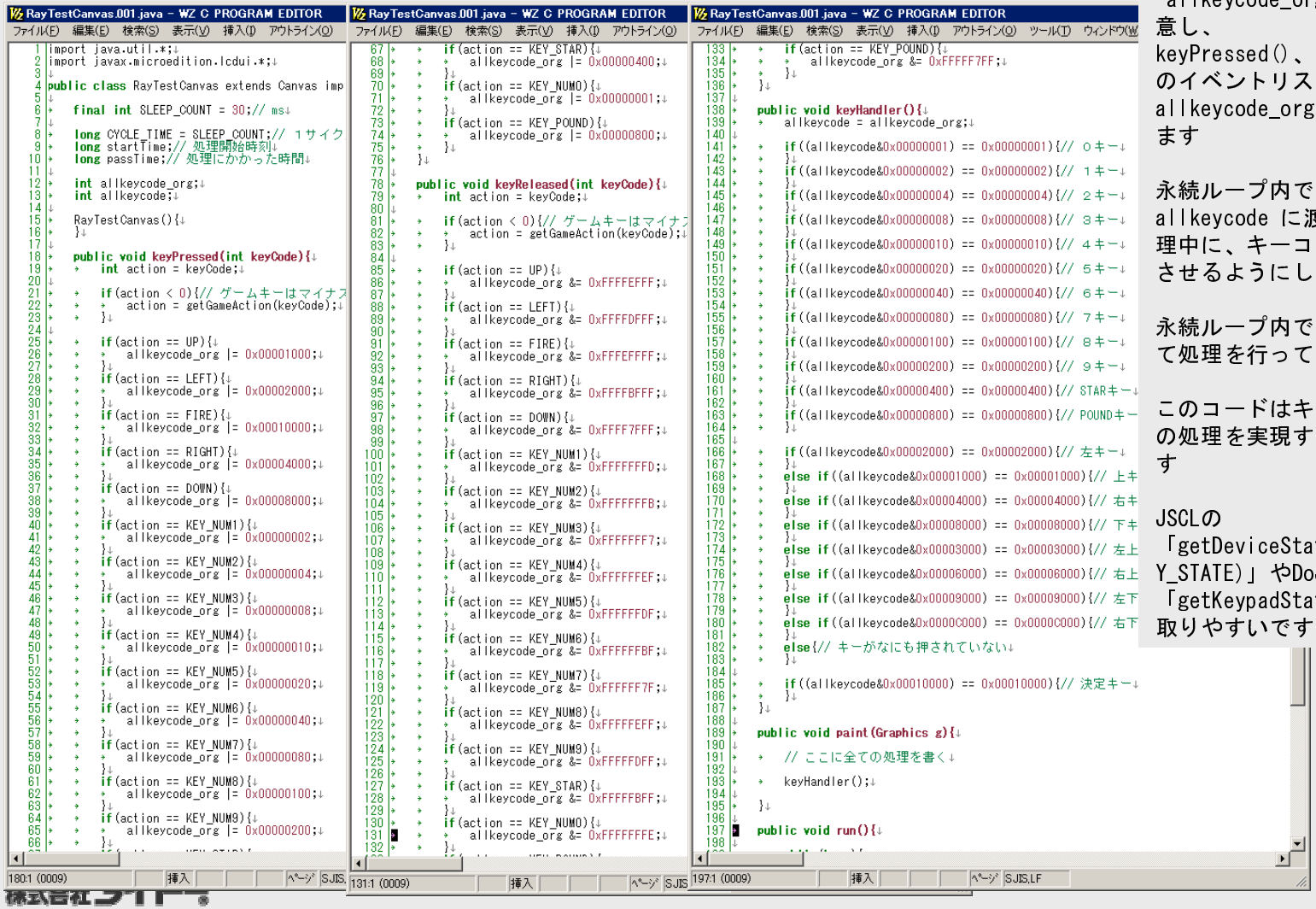
onDraw() 内で invalidate() を実行することで onDraw() が再度実行され、永続ループとなる

MIDPアプリ同様にウェイト処理がされています

RayTestView.org.java.andr - WZ C PROGRAM EDITOR

```
package com.taito.android.raytest;

import android.content.Context;
import android.graphics.*;
import android.view.View;
import android.view.KeyEvent;
import android.graphics.Canvas;
import android.graphics.drawable.Drawable;

import java.util.*;

public class RayTestView extends View{

    final int SLEEP_COUNT = 30;// ms

    long CYCLE_TIME = SLEEP_COUNT;//１サイクルあたりの時間
    long startTime;//処理開始時刻
    long passTime;//処理にかかった時間

    public RayTestView( Context context ){
        super( context );

        // このViewにフォーカスをセット（キーを取るのにも必要）
        setFocusable( true );
    }

    @Override
    protected void onDraw( Canvas canvas ){
        // 処理開始前に時刻を記録
        startTime = System.currentTimeMillis();

        // ここに全ての処理を書く

        // 描画
        repaint();
    }

    private void repaint()
    {
        // 再描画（引数無しなので全画面）
        invalidate();

        try{
            // 処理にかかった時間を計算
            passTime = System.currentTimeMillis() - startTime;

            // [処理にかかった時間]を差し引いてスリープ
            if(( 0 <= passTime )&&(passTime < CYCLE_TIME)){
                try{
                    Thread.sleep(CYCLE_TIME - passTime);
                }
                catch(Exception e){
                }
            }
        }
        catch(Exception e){
        }
    }

}
```

MIDPアプリその１：キー検出方法



キー検出用フラグ変数 allkeycode_org、allkeycode を用意し、keyPressed()、keyReleased() 二つのイベントリスナー内で allkeycode_org のフラグ制御を行います

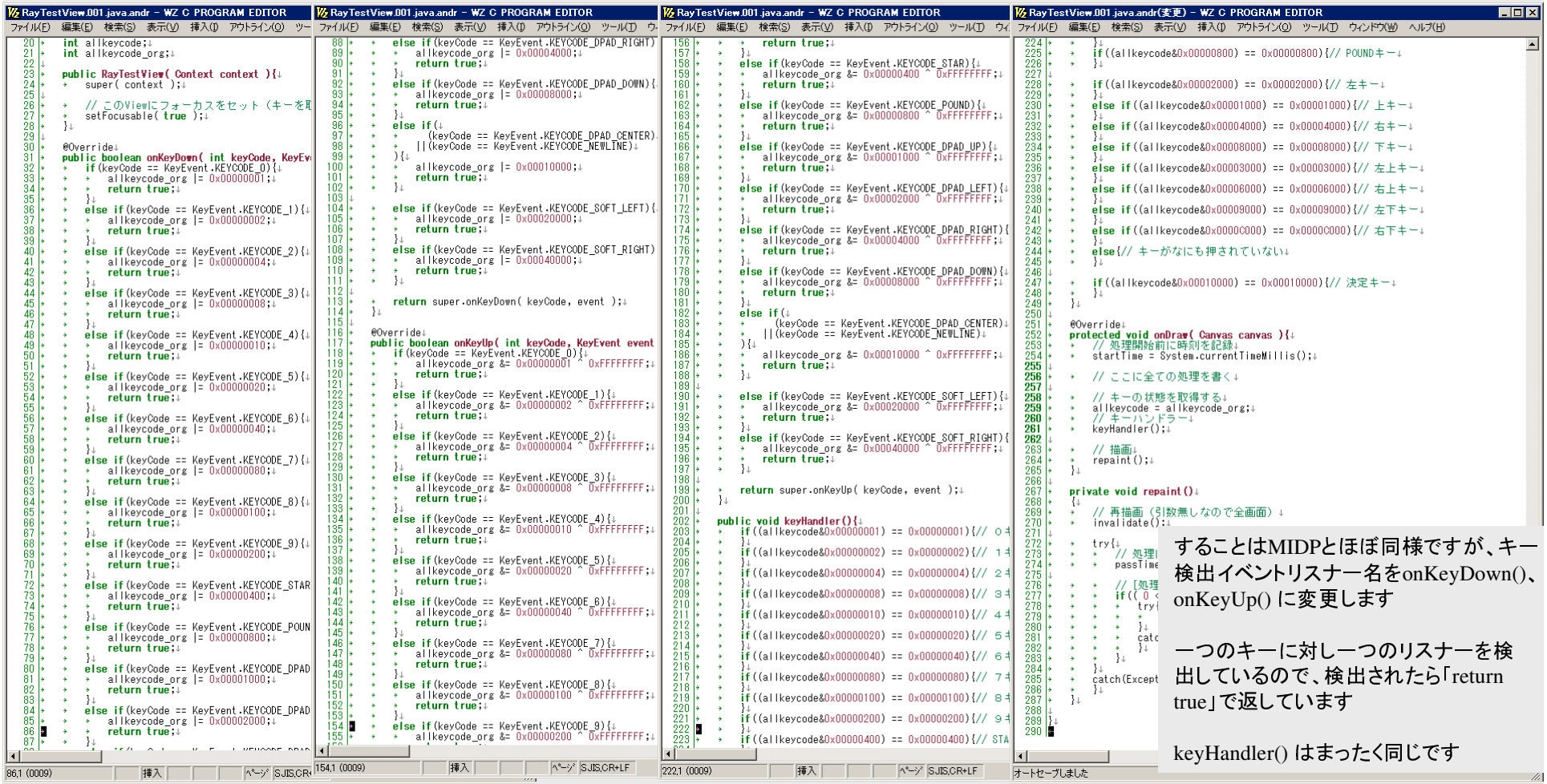
永続ループ内で allkeycode_org を allkeycode に渡し、1回のループ処理中に、キーコードを一度だけ反映させるようにします

永続ループ内で直接キーフラグを見て処理を行ってもかまいません

このコードはキーを押し続けた場合の処理を実現するにとどまっています

JSCLの「getDeviceState(DeviceControl.KEY_STATE)」やDoJaの「getKeypadState()」との互換性を取りやすいです

Androidアプリその１：キー検出方法

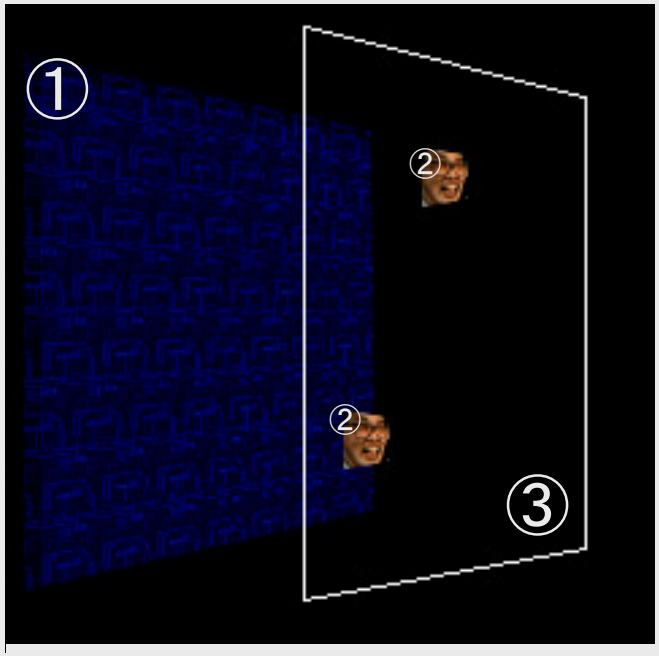


することはMIDPとほぼ同様ですが、キー検出イベントリスナー名をonKeyDown()、onKeyUp() に変更します

一つのキーに対し一つのリスナーを検出しているので、検出されたら「return true」で返しています

keyHandler() はまったく同じです

MIDPアプリその１：画像描画方法全般



①は背景画像です
Image.createImage() により、可変タイプのImageとして生成し、あまり描き変えの必要が無いキャラクターを起動時に描いておきます
永続ループ内ではグラフィックスコンテキストにそのまま描画します

②のように常に動くキャラクターはそのままグラフィックスコンテキストに描画します

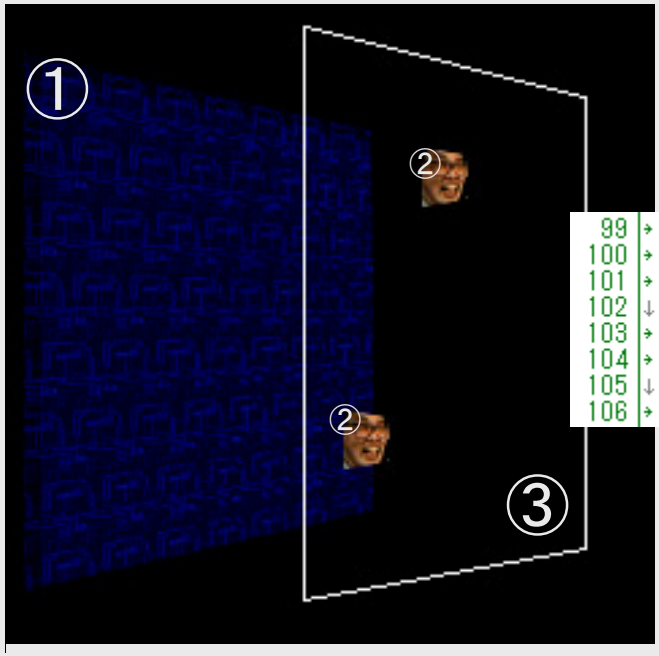
③はグラフィックスコンテキストです

```
43  ////////// バックグラウンド・グラフィックス //////////
44  Graphics bgg;
45  ////////// バックグラウンド・フレームバッファ //////////
46  Image bgfb;// 可変タイプとして生成する
47
48  Image[] charimg = new Image[2];
49  Image bg;
```

```
53  /**
54   * コンストラクタ
55   *
56   **/
57  RayTestCanvas(){
58      try{
59          charimg[0] = Image.createImage("/icon.png");
60          charimg[1] = Image.createImage("/icon.png");
61          bg = Image.createImage("/bg.png");
62      }
63      catch(Exception e){
64          System.out.println("Exception: scene == 0" + e);
65          e.printStackTrace();
66      }
67
68      bgfb = Image.createImage(SCREEN_WIDTH, SCREEN_HEIGHT);
69      bgg = bgfb.getGraphics();
70
71      bgg.drawImage(bg, 0, 0, bgg.TOP|bgg.LEFT);
72  }
```

```
340 public void paint(Graphics g){
341
342     // キーハンドラー
343     keyHandler();
344
345     // スクリーンバッファ描画
346     g.drawImage(bgfb, 0, 0, g.TOP|g.LEFT);
```

Androidアプリその１：画像描画方法全般




```
 32 ⇢   private final Paint paint = new Paint();↓
 33 ↓
 34 ⇢   private Canvas bgg;↓
 35 ⇢   private Bitmap bgfb;↓
 36 ↓
 37 ⇢   private Drawable[] charaimg = new Drawable [2];↓
 38 ⇢   private Drawable bg;↓
```

```
 99 ⇢   ⇢   charaimg[0] = this.getResources().getDrawable( R.drawable.fujimoto );↓
100 ⇢   ⇢   charaimg[1] = this.getResources().getDrawable( R.drawable.fujimoto );↓
101 ⇢   ⇢   bg = this.getResources().getDrawable( R.drawable.bg );↓
102 ↓
103 ⇢   ⇢   bgfb = Bitmap.createBitmap( SCREEN_WIDTH, SCREEN_HEIGHT, true );↓
104 ⇢   ⇢   bgg = new Canvas( bgfb );↓
105 ↓
106 ⇢   ⇢   drawImage( bg, bgg, 0, 0 );↓
```

```
281 ⇢   @Override↓
282 ⇢   protected void onDraw( Canvas g )↓
283 ⇢   {↓
298 ⇢   ⇢   // メインスクリーンをキャンバスに描画↓
299 ⇢   ⇢   g.drawBitmap( bgfb, 0, 0, paint );↓
```

```
 2 ⇢   protected void drawImage( Drawable d, Canvas c, int x, int y )↓
 3 ⇢   {↓
 4 ⇢   ⇢   try{↓
 5 ⇢   ⇢   ⇢   int w = d.getIntrinsicWidth();↓
 6 ⇢   ⇢   ⇢   int h = d.getIntrinsicHeight();↓
 7 ↓
 8 ⇢   ⇢   ⇢   d.setBounds( x, y, x+w, y+h );↓
 9 ⇢   ⇢   ⇢   d.draw( c );↓
10 ⇢   ⇢   }↓
11 ⇢   ⇢   catch(Exception e){↓
12 ⇢   ⇢   }↓
13 ⇢   }↓
14 ↓
15 ⇢   private void drawImage( Drawable d, Canvas c, int x, int y, int w, int h )↓
16 ⇢   {↓
17 ⇢   ⇢   try{↓
18 ⇢   ⇢   ⇢   d.setBounds( x, y, x+w, y+h );↓
19 ⇢   ⇢   ⇢   d.draw( c );↓
20 ⇢   ⇢   }↓
21 ⇢   ⇢   catch(Exception e){↓
22 ⇢   ⇢   }↓
23 ⇢   }↓
```

MIDP での Image を Drawable で、Graphics を Canvas で、可変Image を Bitmap に置き換えてみました

MIDP の Graphics.drawImage() と名前を合わせた、Drawable を Canvas に描画する drawImage() を用意しました

MIDPアプリその１に関しては、ここまでの変更だけで
とりあえず動くものとなりました。

Androidアプリその１

Androidエミュレーター起動

ある程度わかったので、商用に配信しているMIDPアプリを移植してみることにしました

MIDPアプリその２

MIDPエミュレーター起動
Androidエミュレーター起動

ANDROID で ARKANOID

MIDPアプリその２を移植するにあたって、考慮するコード

- ソフトキー処理
- レコードストア
- リソースファイルの読み込み
- ｅｔｃ、、、

ソフトキー

MIDPでは「implements CommandListener」し、
setCommandListener(this)することで、commandAction()でソフトキー押下イベントが取得できます

本来であれば「android.view.Menu」を使ってCommandListenerを再現するのが望ましいと思います

ですが、今回は目先のソフトキー押下取得に気を取られて、先ほどのキー取得の中にソフトキー処理を含めてしましました

タッチ操作なども考えると、ここは修正する余地があると思います

MIDP、DoJaで互換性の薄い箇所でもあります

レコードストア

MIDPではRecordStoreを使い更新データの保持などを行います

Androidで同じことをするには「android.content.SharedPreferences」を使うのが良いでしょう

```
private void record(boolean writeflag){
    try{
        byteArr = new ByteArrayOutputStream();
        dout = new DataOutputStream(byteArr);

        // レコードストアを開く
        RecordStore recordStore = RecordStore.openRecordStore("ArkanoidQVGA", true);

        // 初めての起動なので初期データを書き込む
        if(recordStore.getNumRecords() == 0){
            dout.writeInt(highScore);
            dout.writeInt(arrivalRound);
            dout.writeInt(enemySwitch);

            recordStore.addRecord(byteArr.toByteArray(), 0, byteArr.toByteArray().length);
        }
        else{
            if(writeflag == true){// 現在の状態を書き込み
                dout.writeInt(highScore);
                dout.writeInt(arrivalRound);
                dout.writeInt(enemySwitch);

                recordStore.setRecord(1, byteArr.toByteArray(), 0, byteArr.toByteArray().length);
            }
            else if(writeflag == false){// 前回データがあるのでそれを読む
                byte[] tmp;
                tmp = new byte[recordStore.getRecordSize(1)];
                din = new DataInputStream(new ByteArrayInputStream(tmp));
                recordStore.getRecord(1, tmp, 0);

                highScore = din.readInt();
                arrivalRound = din.readInt();
                enemySwitch = din.readInt();

                tmp = null;
                din.close();
            }
        }

        byteArr.close();
        dout.close();
        recordStore.closeRecordStore();
    }
    catch(Exception e){
    }
}
```

```
private void record( boolean writeflag )
{
    if(writeflag == true){// 現在の状態を書き込み
        SharedPreferences settings = this.getContext().getSharedPreferences(PREFS_NAME, 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putInt("highScore", highScore);
        editor.putInt("arrivalRound", arrivalRound);
        editor.putInt("enemySwitch", enemySwitch);
        editor.commit();
    }
    else if(writeflag == false){// 前回データがあるのでそれを読む
        SharedPreferences settings = this.getContext().getSharedPreferences(PREFS_NAME, 0);
        highScore = settings.getInt("highScore", 0);
        arrivalRound = settings.getInt("arrivalRound", 0);
        enemySwitch = settings.getInt("enemySwitch", 0);
    }
}
```

リソースファイル読み込み

```
2 MIDP:↓
3 InputStream is = Connector.openInputStream("resource:///wall.bin");↓
4 ↓
5 Android:↓
6 InputStream is = this.getResources().openRawResource(R.raw.wall);↓
```

「¥res¥raw」ディレクトリを作り、そこに必要なデータファイルを置くだけで良いです

後は通常のJava同様の処理が可能です

MIDP版より変更した箇所はまだありますが、大きな変更はこの程度です。

入門ということなのでこの辺でご勘弁を

残っている作業

サウンド
イメージリソースの持ち方
サスペンド／レジューム時の処理
通信
ｅｔｃ，，，

でも、多分そんなに大変ではないと思います
既存のMIDPアプリは要点さえつかめば簡単にAndroidアプリにすることができるでしょう

携帯アプリプログラマーの皆さん、良かったですね

でも、本当はMIDPアプリの移植なんてしている場合ではない

実機での動作
デモンストレーション

実機デモにご協力いただいた皆様

東京工科大学
コンピュータサイエンス学部
大学院 コンピュータサイエンス専攻
田胡研究室
　　田胡　和哉　教授（タゴ　カズヤ）
　　徳冨　武志　様（トクトミ　タケシ）

株式会社　タイトー
サーバー事業部サーバー係
　　森田　誉志（モリタ　タカシ）
　　三塚　淳史（ミツカ　アツシ）

最後に

この発表を日々厳しいスケジュールで

携帯アプリを開発している

全ての携帯アプリプログラマーに
捧げます

ご静聴ありがとうございました