

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

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 Java™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エミュレーター一起動

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
ファイル(F) 編集(E) 検索(S) 表示(V) 挿入(I) アウトライン(O) ツール(T) ウィンドウ(W) ヘルプ(H)
1 import javax.microedition.lcdui.*;
2 import javax.microedition.midlet.*;
3
4 public class RayTest extends MIDlet{
5
6     static RayTestCanvas c;
7
8     public RayTest(){
9         c = new RayTestCanvas();
10        Display.getDisplay(this).setCurrent(c);
11    }
12
13    public void startApp(){
14    }
15
16    public void pauseApp(){
17    }
18
19    public void destroyApp(boolean unconditional){
20    }
21
22 }
23
    
```

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

その後、「implements 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。

```

RayTestCanvas.000.java - WZ C PROGRAM EDITOR
ファイル(F) 編集(E) 検索(S) 表示(V) 挿入(I) アウトライン(O) ツール(T) ウィンドウ(W) ヘルプ(H)
1 import java.util.*;
2 import javax.microedition.lcdui.*;
3
4 public class RayTestCanvas extends Canvas implements Runnable{
5
6     final int SLEEP_COUNT = 30; // ms
7
8     long CYCLE_TIME = SLEEP_COUNT; // 1サイクルあたりの時間
9     long startTime; // 処理開始時刻
10    long passTime; // 処理にかかった時間
11
12    RayTestCanvas(){
13    }
14
15    public void paint(Graphics g){
16
17        // ここに全ての処理を書く
18    }
19
20    public void run(){
21
22        while(true){
23
24            // 処理開始前に時刻を記録
25            startTime = System.currentTimeMillis();
26
27            // 全処理と描画
28            repaint();
29
30            // 仮想画面を実画面へ転送
31            serviceRepaints();
32
33            try{
34                // 処理にかかった時間を計算
35                passTime = System.currentTimeMillis() - startTime;
36
37                // 【処理にかかった時間】を差し引いてスリープ
38                if((0 <= passTime) && (passTime < CYCLE_TIME)){
39                    try{
40                        // ウェイト
41                        Thread.sleep(CYCLE_TIME - passTime);
42                    } catch (Exception e){
43                        //
44                    }
45                }
46            } catch (Exception e){
47                System.out.println("Exception: run() " + e);
48                e.printStackTrace();
49            }
50        }
51    }
52
53 }
54
55
56
57
58
    
```


Androidアプリその1：永続ループ処理方法

```

RayTest.org.java.andr - WZ C PROGRAM EDITOR
ファイル(F) 編集(E) 検索(S) 表示(V) 挿入(I) アウトライン(O) ツール(T) ウィンドウ(W) ヘルプ(H)
1 package com.taito.android.raytest;↓
2 ↓
3 import android.app.Activity;↓
4 import android.os.Bundle;↓
5 import android.view.Window;↓
6 ↓
7 public class RayTest extends Activity↓
8 {↓
9 ↓
10 → @Override↓
11 → public void onCreate( Bundle icycle )↓
12 → {↓
13 ↓
14 →     super.onCreate( icycle );↓
15 ↓
16 →     // タイトルバー非表示↓
17 →     requestWindowFeature( Window.FEATURE_NO_TITLE );↓
18 ↓
19 →     // Viewをセット↓
20 →     setContentView( new RayTestView( this ) );↓
21 ↓
22 → }↓
23 ↓
24 ↓
25 ↓
16.1 (0009) 挿入 ヘルプ SJIS.LF
    
```

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
ファイル(F) 編集(E) 検索(S) 表示(V) 挿入(I) アウトライン(O) ツール(T) ウィンドウ(W) ヘルプ(H)
1 package com.taito.android.raytest;↓
2 ↓
3 import android.content.Context;↓
4 import android.graphics.*;↓
5 import android.view.View;↓
6 import android.view.KeyEvent;↓
7 import android.graphics.Canvas;↓
8 import android.graphics.drawable.Drawable;↓
9 ↓
10 import java.util.*;↓
11 ↓
12 public class RayTestView extends View↓
13 ↓
14 →     final int SLEEP_COUNT = 30; // ms↓
15 ↓
16 →     long CYCLE_TIME = SLEEP_COUNT; // 1サイクルあたりの時間↓
17 →     long startTime; // 処理開始時刻↓
18 →     long passTime; // 処理にかかった時間↓
19 ↓
20 →     public RayTestView( Context context )↓
21 →     {↓
22 →         super( context );↓
23 ↓
24 →         // このViewにフォーカスをセット (キーを取るのにも必要) ↓
25 →         setFocusable( true );↓
26 ↓
27 ↓
28 →     @Override↓
29 →     protected void onDraw( Canvas canvas )↓
30 →     {↓
31 →         // 処理開始前に時刻を記録↓
32 →         startTime = System.currentTimeMillis();↓
33 ↓
34 →         // ここに全ての処理を書く↓
35 ↓
36 →         // 描画↓
37 →         repaint();↓
38 ↓
39 →     private void repaint()↓
40 →     {↓
41 →         // 再描画 (引数無しなので全画面) ↓
42 →         invalidate();↓
43 ↓
44 →         try↓
45 →         {↓
46 →             // 処理にかかった時間を計算↓
47 →             passTime = System.currentTimeMillis() - startTime;↓
48 ↓
49 →             // [処理にかかった時間]を差し引いてスリープ↓
50 →             if( ( 0 <= passTime ) && ( passTime < CYCLE_TIME ) )↓
51 →             {↓
52 →                 try↓
53 →                 {↓
54 →                     Thread.sleep( CYCLE_TIME - passTime );↓
55 →                 }↓
56 →                 catch( Exception e )↓
57 →                 {↓
58 →                 }↓
59 →             }↓
60 →             catch( Exception e )↓
61 →             {↓
62 →             }↓
63 →         }↓
64 →     }↓
65 ↓
66 ↓
67 ↓
68 ↓
69 ↓
70 ↓
1.1 (0070) 挿入 ヘルプ SJIS.SR+LF
    
```



MIDPアプリその1：キー検出方法

```

% RayTestCanvas_001.java - WZ C PROGRAM EDITOR
1 import java.util.*;
2 import javax.microedition.lcdui.*;
3
4 public class RayTestCanvas extends Canvas imp
5
6     final int SLEEP_COUNT = 30; // ms
7
8     long CYCLE_TIME = SLEEP_COUNT; // 1サイクル
9     long startTime; // 処理開始時刻
10    long passTime; // 処理にかかった時間
11
12    int allkeycode_org;
13    int allkeycode;
14
15    RayTestCanvas() {
16    }
17
18    public void keyPressed(int keyCode) {
19        int action = keyCode;
20
21        if (action < 0) // ゲームキーはマイナ
22            action = getGameAction(keyCode);
23
24
25        if (action == UP) {
26            allkeycode_org |= 0x00001000;
27        }
28        if (action == LEFT) {
29            allkeycode_org |= 0x00002000;
30        }
31        if (action == FIRE) {
32            allkeycode_org |= 0x00010000;
33        }
34        if (action == RIGHT) {
35            allkeycode_org |= 0x00004000;
36        }
37        if (action == DOWN) {
38            allkeycode_org |= 0x00008000;
39        }
40        if (action == KEY_NUM1) {
41            allkeycode_org |= 0x00000002;
42        }
43        if (action == KEY_NUM2) {
44            allkeycode_org |= 0x00000004;
45        }
46        if (action == KEY_NUM3) {
47            allkeycode_org |= 0x00000008;
48        }
49        if (action == KEY_NUM4) {
50            allkeycode_org |= 0x00000010;
51        }
52        if (action == KEY_NUM5) {
53            allkeycode_org |= 0x00000020;
54        }
55        if (action == KEY_NUM6) {
56            allkeycode_org |= 0x00000040;
57        }
58        if (action == KEY_NUM7) {
59            allkeycode_org |= 0x00000080;
60        }
61        if (action == KEY_NUM8) {
62            allkeycode_org |= 0x00000100;
63        }
64        if (action == KEY_NUM9) {
65            allkeycode_org |= 0x00000200;
66
67
68     if (action == KEY_STAR) {
69         allkeycode_org |= 0x00000400;
70     }
71     if (action == KEY_NUM0) {
72         allkeycode_org |= 0x00000001;
73     }
74     if (action == KEY_POUND) {
75         allkeycode_org |= 0x00000800;
76     }
77
78     public void keyReleased(int keyCode) {
79         int action = keyCode;
80
81         if (action < 0) // ゲームキーはマイナ
82             action = getGameAction(keyCode);
83
84
85         if (action == UP) {
86             allkeycode_org &= 0xFFFFFFF;
87         }
88         if (action == LEFT) {
89             allkeycode_org &= 0xFFFFFFF;
90         }
91         if (action == FIRE) {
92             allkeycode_org &= 0xFFFFFFF;
93         }
94         if (action == RIGHT) {
95             allkeycode_org &= 0xFFFFFFF;
96         }
97         if (action == DOWN) {
98             allkeycode_org &= 0xFFFFFFF;
99         }
100        if (action == KEY_NUM1) {
101            allkeycode_org &= 0xFFFFFFF;
102        }
103        if (action == KEY_NUM2) {
104            allkeycode_org &= 0xFFFFFFF;
105        }
106        if (action == KEY_NUM3) {
107            allkeycode_org &= 0xFFFFFFF;
108        }
109        if (action == KEY_NUM4) {
110            allkeycode_org &= 0xFFFFFFF;
111        }
112        if (action == KEY_NUM5) {
113            allkeycode_org &= 0xFFFFFFF;
114        }
115        if (action == KEY_NUM6) {
116            allkeycode_org &= 0xFFFFFFF;
117        }
118        if (action == KEY_NUM7) {
119            allkeycode_org &= 0xFFFFFFF;
120        }
121        if (action == KEY_NUM8) {
122            allkeycode_org &= 0xFFFFFFF;
123        }
124        if (action == KEY_NUM9) {
125            allkeycode_org &= 0xFFFFFFF;
126        }
127        if (action == KEY_STAR) {
128            allkeycode_org &= 0xFFFFFFF;
129        }
130        if (action == KEY_NUM0) {
131            allkeycode_org &= 0xFFFFFFF;
132
133     if (action == KEY_POUND) {
134         allkeycode_org &= 0xFFFFFFF;
135     }
136     }
137
138     public void keyHandler() {
139         allkeycode = allkeycode_org;
140
141         if ((allkeycode & 0x00000001) == 0x00000001) // 0キー
142             ;
143         if ((allkeycode & 0x00000002) == 0x00000002) // 1キー
144             ;
145         if ((allkeycode & 0x00000004) == 0x00000004) // 2キー
146             ;
147         if ((allkeycode & 0x00000008) == 0x00000008) // 3キー
148             ;
149         if ((allkeycode & 0x00000010) == 0x00000010) // 4キー
150             ;
151         if ((allkeycode & 0x00000020) == 0x00000020) // 5キー
152             ;
153         if ((allkeycode & 0x00000040) == 0x00000040) // 6キー
154             ;
155         if ((allkeycode & 0x00000080) == 0x00000080) // 7キー
156             ;
157         if ((allkeycode & 0x00000100) == 0x00000100) // 8キー
158             ;
159         if ((allkeycode & 0x00000200) == 0x00000200) // 9キー
160             ;
161         if ((allkeycode & 0x00000400) == 0x00000400) // STARキー
162             ;
163         if ((allkeycode & 0x00000800) == 0x00000800) // POUNDキー
164             ;
165
166         if ((allkeycode & 0x00002000) == 0x00002000) // 左キー
167             ;
168         else if ((allkeycode & 0x00001000) == 0x00001000) // 上キ
169             ;
170         else if ((allkeycode & 0x00004000) == 0x00004000) // 右キ
171             ;
172         else if ((allkeycode & 0x00008000) == 0x00008000) // 下キ
173             ;
174         else if ((allkeycode & 0x00003000) == 0x00003000) // 左上
175             ;
176         else if ((allkeycode & 0x00006000) == 0x00006000) // 右上
177             ;
178         else if ((allkeycode & 0x00009000) == 0x00009000) // 左下
179             ;
180         else if ((allkeycode & 0x0000C000) == 0x0000C000) // 右下
181             ;
182         else // キーがなにも押されていない
183             ;
184
185         if ((allkeycode & 0x00010000) == 0x00010000) // 決定キー
186             ;
187     }
188
189     public void paint(Graphics g) {
190         // ここに全ての処理を書く
191         keyHandler();
192     }
193
194     public void run() {
195
196     }
197
198     public void run() {
199
200     }
    
```

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

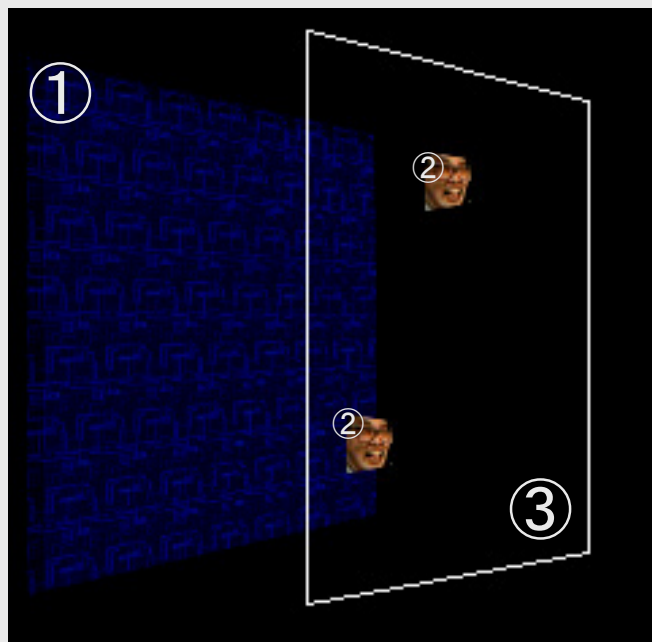
永続ループ内で直接キーフラグを見
て処理を行ってもかまいません

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. KE
Y_STATE)」やDoJaの
「getKeypadState()」との互換性を
取りやすいです



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;↓
    
```

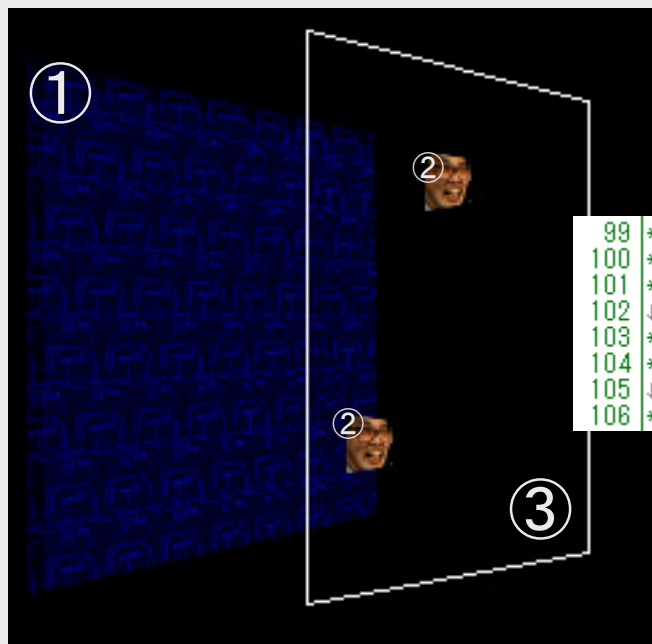
```

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);↓
347 ↓
    
```

Androidアプリその1 : 画像描画方法全般



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

```

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 );↓
300 ↓
    
```

```

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エミュレーター一起動
Androidエミュレーター一起動

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
- ・ e t c、 、 、

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いました

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

```

20 private void record(boolean writeflag){
21     try{
22         * byteArr = new ByteArrayOutputStream();
23         * dout = new DataOutputStream(byteArr);
24     }
25     * // レコードストアを開く
26     * RecordStore recordStore = RecordStore.openRecordStore("ArkanoidQWGA", true);
27     * // 初めての起動なので初期データを書き込む
28     * if(recordStore.getNumRecords() == 0){
29     *     * dout.writeInt(highScore);
30     *     * dout.writeInt(arrivalRound);
31     *     * dout.writeInt(enemySwitch);
32     * }
33     * recordStore.addRecord(byteArr.toByteArray(), 0, byteArr.toByteArray().length);
34     * }
35     * }
36     * else{
37     *     * if(writeflag == true){// 現在の状態を書き込み
38     *         * dout.writeInt(highScore);
39     *         * dout.writeInt(arrivalRound);
40     *         * dout.writeInt(enemySwitch);
41     *     * }
42     *     * recordStore.setRecord(1, byteArr.toByteArray(), 0, byteArr.toByteArray().length);
43     *     * }
44     *     * else if(writeflag == false){// 前回データがあるのでそれを読む
45     *         * byte[] tmp;
46     *         * tmp = new byte[recordStore.getRecordSize(1)];
47     *         * din = new DataInputStream(new ByteArrayInputStream(tmp));
48     *         * recordStore.getRecord(1, tmp, 0);
49     *         * highScore = din.readInt();
50     *         * arrivalRound = din.readInt();
51     *         * enemySwitch = din.readInt();
52     *     * }
53     *     * tmp = null;
54     *     * din.close();
55     *     * }
56     *     * }
57     * }
58     * }
59     * byteArr.close();
60     * dout.close();
61     * recordStore.closeRecordStore();
62     * }
63     * catch(Exception e){
64     * }
65     * }
    
```

```

1 private void record( boolean writeflag ){
2     {
3     * if(writeflag == true){// 現在の状態を書き込み
4     *     * SharedPreferences settings = this.getContext().getSharedPreferences(PREFS_NAME, 0);
5     *     * SharedPreferences.Editor editor = settings.edit();
6     *     * editor.putInt("highScore", highScore);
7     *     * editor.putInt("arrivalRound", arrivalRound);
8     *     * editor.putInt("enemySwitch", enemySwitch);
9     *     * editor.commit();
10    * }
11    * else if(writeflag == false){// 前回データがあるのでそれを読む
12    *     * SharedPreferences settings = this.getContext().getSharedPreferences(PREFS_NAME, 0);
13    *     * highScore = settings.getInt("highScore", 0);
14    *     * arrivalRound = settings.getInt("arrivalRound", 0);
15    *     * enemySwitch = settings.getInt("enemySwitch", 0);
16    * }
17    * }
    
```

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

e t c, , ,

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とでしょう

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)
徳富 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