ANDROIDマーケットでの配信を経験してみて

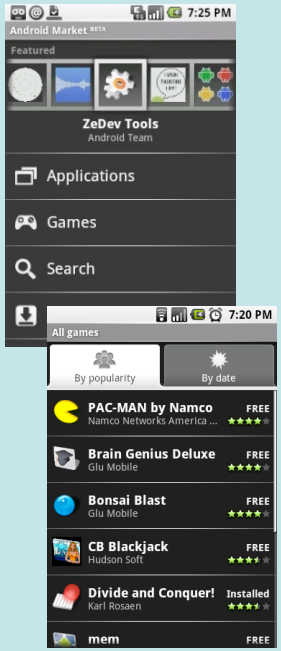
株式会社ハドソン　新規事業本部　西岡光治

ANDROIDマーケットのスキーム

ユーザー

課金

ANDROID
マーケット



売上

Google
&
各国のキャリア

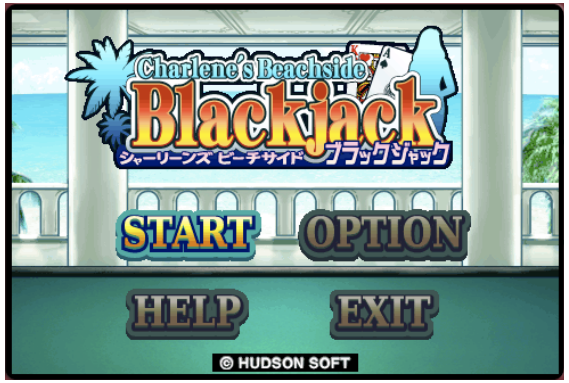
30%

70%
（各国の税務処理負担含む）

HUDSON

無料アプリの配信

シャーリーンズ ビーチサイド シリーズ



ブラックジャック　　クロンダイク　　リバーシ

有料アプリの配信

ボンバーマン道場

ネクタリス

App Store（iPhone）との違い

価格設定(表示)

売上レポート

PCプラットフォームがない

無料と有料が同一カテゴリー

ANDROIDマーケット以外への横断的な展開

複数プラットフォームへの展開

Androidを利用した簡単なゲーム開発

株式会社ハドソン　新規事業制作本部　遠藤大介

アジェンダ

現在、スマートフォン市場が非常に活発的になっています。

その中で、無料で誰でも開発する事ができるプラットフォーム

「Android」を利用した簡単なゲーム開発方法をご紹介いたします。

1. メインループの作成方法

メインループとは

リアルタイムな処理が必要となるゲームで

任意で処理を停止しない限り、無限にループし続ける処理の事を指します。

必要となるクラス

java.lang.Thread

サンプルコード1

```
private class MainLoop extends Thread {
    private boolean mFinish = false;

    /**
     * コンストラクタ
     */
    public MainLoop() {
    }

    /**
     * ゲームループの起動
     */
    @Override public void start() {
        mFinish = false; // メインループ管理フラグの初期化
        super.start();   // Threadを開始させる
    }
    ...
```

サンプルコード2

```
    ...
    /**
     * ゲームループ
     */
    @Override public void run() {
        // メインループ管理フラグによって終了させる
        while( !mFinish ) {
            // ゲーム本編の動作を記述する
        }
    }

    /**
     * ゲームループの終了
     */
    public void finish() {
        try {
            mFinish = true;   // メインループ管理フラグを終了へ
            super.join();     // スレッドの終了を待つ
        }
        catch( Throwable x ) {}
    }
}
```

2. SurfaceViewを利用した簡単な2D描画

SurfaceViewとは

画面に画像やなどを描画する際、必要な描画領域を提供してくれる

クラスです。

（それ以外にも色々な機能が存在しています）

必要となるクラス

android.view.SurfaceHolder

android.view.SurfaceView

android.view.SurfaceHolder.Callback

android.graphics.BitmapFactory

サンプルコード1

```
public class MainLoopView extends SurfaceView implements SurfaceHolder.Callback {
    private MainLoop       mMainLoop   = null;
    private SurfaceHolder mHolder      = null;
    private boolean        mHasSurface = false;
    private Bitmap         mBitmap     = null;
    ....

    /**
     * コンストラクタ
     * @param context
     */
    public MainLoopView(Context context ) {
        super(context);
        mHolder = this.getHolder();
        // Surfaceの領域を指定
        mHolder.setFixedSize( MainLoopView.CANVAS_W, MainLoopView.CANVAS_H );
        // Surfaceのピクセルフォーマットを指定
        mHolder.setFormat( PixelFormat.RGB_565 );
        // Surfaceの生成にGPUを使用するように指定
        mHolder.setType( SurfaceHolder.SURFACE_TYPE_GPU );
        // コールバックに自身を登録
        mHolder.addCallback( this );
        mHasSurface = false;
        ....
        // 画像データの生成（メモリ不足でOutOfMemoryExceptionが発生する可能性有り）
        mBitmap = BitmapFactory.decodeResource( context.getResources(), R.drawable.icon );
    }
```

サンプルコード2

```
/**
 * Surface が生成された際にコールされる
 * @param holder
 */
public void surfaceCreated( SurfaceHolder holder ) {
    mHasSurface = true;
    // メインループを開始させる
    if( mHasSurface ) { mMainLoop.start(); }
}

/**
 * Surface の属性が変更された際にコールされる
 * @param holder
 * @param format
 * @param width
 * @param height
 */
public void surfaceChanged( SurfaceHolder holder, int format, int width, int height ) {

}

/**
 * Surface が破棄された際にコールされる
 * @param holder
 */
public void surfaceDestroyed( SurfaceHolder holder ) {
    mHasSurface = false;
}
```

サンプルコード3

```
    private class MainLoop extends Thread {
        ....
        /**
         * ゲームループ
         */
        @Override public void run() {
            mStartTime  = System.currentTimeMillis();
            Paint paint = new Paint();
            while( !mFinish ) {
                // キャンバスをロックして、転送可能な状態にする
                Canvas canvas = MainLoopView.this.mHolder.lockCanvas();
                if( canvas == null ) { continue; }

                paint.setColor( Color.WHITE );
                canvas.drawBitmap( mBitmap, 0, 0, paint );
                // キャンバスをアンロックして、画面に描画する
                MainLoopView.this.mHolder.unlockCanvasAndPost( canvas );
            }
        }
        ....
    }
}
```

ピックアップポイント

SurfaceHolder.addCallback(SurfaceHolder.Callback)

このメソッドをコールしてSurfaceHolder.Callbackをimplementsしたクラスを登録しなければ、surfaceCreated surfaceChanged surfaceDestroyedのイベントを取得する事ができなくなります。

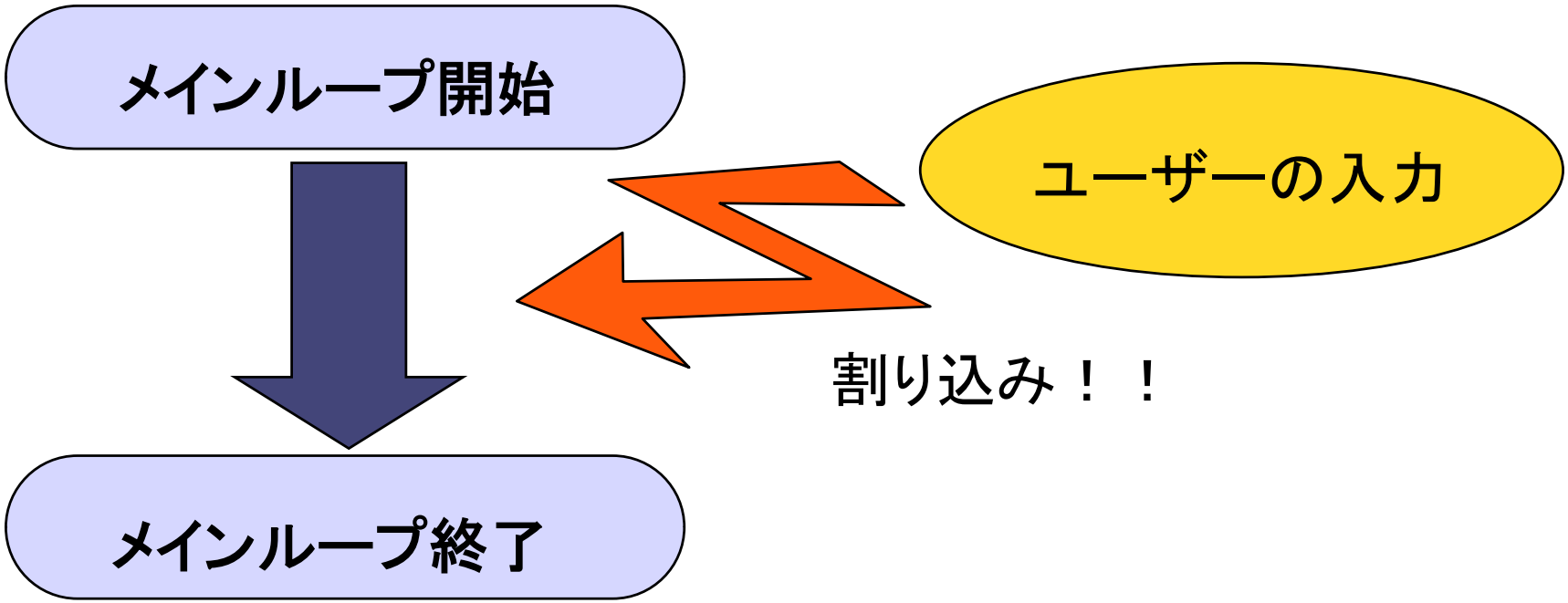
特にsurfaceCreatedは描画する事ができるsurfaceが生成された時にコールされますので、非常に有効なイベントとなります。

3. 様々な入力について

androidの入力デバイスについて

Androidではトラックボール、タッチスクリーン、キーボードの3つの入力がサポートされています。

各種入力情報を取得するメソッドの実行形式はイベントドリブン型となっており、メインループとは違うスレッドで行われる事になります。



3−1. トラックボールイベントの取得方法

トラックボールとは

ポインティングデバイスの一種で、マウスを動かす代わりに

ボールのような箇所を回す事により操作を行う事ができるデバイスです。

必要なクラス

android.view.Viewまたはandroid.view.Viewを継承しているクラス
※SurfaceViewやGLSurfaceViewなど

android.view.MotionEvent

サンプルコード

```
public class MainLoopView extends SurfaceView implements SurfaceHolder.Callback {
    ....

    @Override public boolean onTrackballEvent( MotionEvent event ) {
        // トラックボールの動作状態を判断する
        switch( event.getAction() ) {
        case MotionEvent.ACTION_MOVE:
            // トラックボールを動かしたとき
            mCursorX += event.getX();
            mCursorY += event.getY();
            break;
        case MotionEvent.ACTION_DOWN:
            // トラックボールを押したとき
            break;
        case MotionEvent.ACTION_UP:
            // トラックボールを押して離したとき
            break;
        }
        return true;
    }
}
```

ピックアップポイント

public boolean onTrackballEvent(MotionEvent event)

MotionEvent.getX() MotionEvent.getY()でトラックボールを転がした時
の強さを取得できます。

※実機とエミュレータでは戻り値に違いがあるので検証が必要となります。
値は移動後の座標などではないので利用する場合は加工をすると
使用しやすくなります。

3−2. タッチスクリーンイベントの取得方法

タッチスクリーンとは

画面を指や専用機器で触れる事により端末を操作する事ができる、

入力装置の事を指します。

銀行などのATMが一番身近で代表的な物です。

必要なクラス

android.view.Viewまたはandroid.view.Viewを継承しているクラス
※SurfaceViewやGLSurfaceViewなど

android.view.MotionEvent

サンプルコード

```
public class MainLoopView extends SurfaceView implements SurfaceHolder.Callback {
    ....

    @Override public boolean onTouchEvent( MotionEvent event ) {
        // タッチスクリーンの動作状態を判断する
        switch( event.getAction() ) {
        case MotionEvent.ACTION_DOWN:
            // タッチスクリーンに触れたとき
            mTouchX = event.getX();
            mTouchY = event.getY();
            break;
        case MotionEvent.ACTION_UP:
            // タッチスクリーンを離したとき
            mReleaseX = event.getX();
            mReleaseY = event.getY();
            break;
        case MotionEvent.ACTION_MOVE:
            // タッチスクリーンを触れながら動かしたとき
            mMovingX  = event.getX();
            mMovingY  = event.getY();
            break;
        }
        return true;
    }
}
```

ピックアップポイント

public boolean onTouchEvent(MotionEvent event)

ユーザーが画面をタッチし続けるとイベントが連続的にコールされ、
システムに負荷がかかってしまいます。

Thread.sleepにてコールされるタイミングを制御する事によって、
システムへの負荷を軽減できます。

3−3．キーイベントの取得方法

キー入力インターフェース

現在、Androidでは大きく分けてソフトウェアキーボードと

ハードウェアキーボードの2種類のキー入力がサポートされています。

必要なクラス

android.view.Viewまたはandroid.view.Viewを継承しているクラス
※SurfaceViewやGLSurfaceViewなど

android.view.KeyEvent

サンプルコード

```
public class MainLoopView extends SurfaceView implements Callback {
    ....

    @Override public boolean onKeyDown( int keyCode, KeyEvent event ) {
        /*
          キーが押されたときにコールされる
          →keyCodeにはandroid.view.KeyEventで定義されている数値が入っている
            int型は32ビットと仮定し処理を行う
        */
        int param = keyCode / 32;
        int code  = keyCode % 32;
        mKeyParams[param] |= ( 1 << code );
        return super.onKeyDown( keyCode, event );
    }

    @Override public boolean onKeyUp( int keyCode, KeyEvent event ) {
        /*
          キーが離されたときにコールされる
          →keyCodeにはandroid.view.KeyEventで定義されている数値が入っている
            int型は32ビットと仮定し処理を行う
        */
        int param = keyCode / 32;
        int code  = keyCode % 32;
        mKeyParams[param] ^= ( 1 << code ) & mKeyParams[ param ];
        return super.onKeyUp( keyCode, event );
    }
}
```

4. サウンド再生について

サウンドファイルを扱う前に
サウンドデータをプロジェクト上で利用する場合は、

resフォルダにrawフォルダを作成して、そのフォルダにデータを

設置するようにしてください。

また、このフォルダはサウンドデータ以外にも利用できますので、

利用する事をオススメします。

使用する事ができるデータはWAVE MP3 OGGなどがあげられます。

android.media.MediaPlayer

良いと思う点

・データの様々な状態を取得する事ができる（再生中／再生終了など）

・サウンドだけではなくムービーの再生にも利用する事ができる

・設定を細かに行う事ができる

気になった箇所

・再生にかかる処理負荷が高い

・メモリ上に展開されているサウンドファイルを利用できない

※サウンドデータをかえる場合はインスタンスの生成からする

MediaPlayer(android.media.MediaPlayer)

```
public class MediaProcess implements MediaPlayer.OnCompletionListener  {
    public Context     mContext   = null;
    public MediaPlayer mBgmPlayer = null;

    /**
     * コンストラクタ
     * @param context - コンテキスト
     */
    public MediaProcess( Context context ) {
        /*
          再生するにはコンテキストが必要になる
          Activityクラスの getApplicationContext()などで取得可能
        */
        mContext = context;
    }

    /**
     * サウンド再生
     * @param resId - サウンドファイルのリソース番号
     */
    public void play( int resId ) {
        mBgmPlayer = MediaPlayer.create( mContext, resId );
        // 再生完了時のコールバックへ自身のクラスを登録
        mBgmPlayer.setOnCompletionListener( this );
        // 再生を開始する
        mBgmPlayer.start();
    }
    ....
```

```
    /**
     * サウンド停止
     */
    public void stop() {
        /*
          再生されていないMediaPlayerを停止しようとすると
          エラーが発生してしまうので再生チェックを行う
        */
        if( mBgmPlayer.isPlaying() ) { mBgmPlayer.stop(); }
    }

    /**
     * サウンドデータを破棄する
     */
    public void dispose() {
        // 再生中のデータを破棄しないように一度停止させる
        this.stop();
        // MediaPlayerが停止している状態で破棄するようにする
        mBgmPlayer.release(); mBgmPlayer = null;
        System.gc();
    }

    /**
     * 再生が完了した際にコールされる
     * @param mp - 再生が完了したMediaPlayer
     */
    public void onCompletion( MediaPlayer mp ) {
        // サウンド再生後の処理を記載
    }
}
```

android.media.SoundPool

良いと思う点

・データを展開し、登録することで再生をスムーズに行う事ができる

・再生にかかる処理負荷がMediaPlayerより少ない

気になった箇所

・MediaPlayerの様な状態取得が難しい（再生終了など）

・高レートのサウンドデータ再生にはあまり向いていない

SoundPool(android.media.SoundPool)

```
public class MediaProcess {
    private Context                 mContext   = null;
    private SoundPool               mSePlayer  = null;
    private HashMap<Integer,Integer> mSePoolMap = null;

    /**
     * コンストラクタ
     * @param context - コンテキスト
     * @param capacity - サウンドプールの容量
     */
    public MediaProcess( Context context, int capacity ) {
        mContext   = context;
        mSePlayer  = new SoundPool( capacity, AudioManager.STREAM_MUSIC, 100 );
        mSePoolMap = new HashMap<Integer,Integer>();
    }
    ....
```

```
....
/**
 * サウンドファイルをロードする
 * @param resId – サウンドファイルのリソース番号
 * @param idx   – 登録インデックス番号
 */
public void loadFromResrouce( int resId, int idx ) {
    /*
      サウンドデータのロードを行うと登録番地が返却されるので
      そのデータをHashMapへ登録し保持しておくようにする
     */
    int soundId = mSePlayer.load( mContext, resId, 1 );
    mSePoolMap.put( idx, soundId );
}

/**
 * サウンドを破棄する
 * @param idx – 登録インデックス番号
 */
public void dispose( int idx ) {
    /*
      SoundPoolに登録してあるデータを破棄する
     */
    this.stop();
    int id = mSePoolMap.get( idx );
    mSePlayer.unload( id );
    mSePoolMap.put( idx, -1 );
    System.gc();
}
....
```

```
    ....
    /**
     * サウンドを再生させる
     * @param idx
     */
    public void play( int idx ) {
        /*
          HashMapに登録されているデータ番号を取得し
          SoundPoolからデータ番号目のデータを再生させます。
        */
        int id = mSePoolMap.get( idx );
        mSeStreamId = mSePlayer.play( id, 1.0f, 1.0f, 1, 0, 1.0f );
    }

    /**
     * サウンドを停止させる
     */
    public void stop() {
        /*
          SoundPool.playで保持したPool番号を利用して
          SoundPool.stopを行います。
        */
        if( mSeStreamId == -1 ) { return; }
        mSePlayer.stop( mSeStreamId );
    }
}
```

ピックアップポイント

MediaPlayer.stop()

MediaPlayerを停止させるときにMediaPlayer.stop()をコールしますが、
再生されていないMediaPlayerに対してコールするとエラーが発生する
可能性があります。
停止させる場合は、MediaPlayer.isPlaying()でチェックしてから
停止させると安全です。

5. ファイルの保存

android.content.SharedPreferences

JavaのHashtableの様な利用方法です。

保存する値に対して名前を設定する必要があり、わかりやすいのですが

その反面、名前を設定するのが面倒になります。

キーとデータが関連づいているので命名規則に注意すれば、

データの並びには特に制限はなく追加や削除が楽です。

android.content.SharedPreferences

データの書き込み

```
// ContextからSharedPreferencesの取得及び生成を行います。
SharedPreferences        prefs = this.getSharedPreferences( "etc.xml", Context.MODE_PRIVATE );
// 取得したSharedPreferencesから書き込み用のEditorを生成します。
SharedPreferences.Editor ed    = prefs.edit();
// キーと値を設定します
ed.putString( "name", "AndroidでSharedPreferences!!" );
// comitを行わないとデータの保存が適用されないので注意が必要です。
ed.commit();
```

データの読み込み

```
// ContextからSharedPreferencesの取得及び生成を行います。
SharedPreferences prefs = this.getSharedPreferences( "etc.xml", Context.MODE_PRIVATE );

/*
  キーを指定して、そのキーに関連づけられたデータを取得してきます。
  取得できなかった場合は、第2引数の値が戻ってきます。
*/
String name = prefs.getString( "name", "名無しさん@お腹いっぱい" );

Log.d( "", name );
```

java.io.FileInputStream

java.io.FileOutputStream

バイナリレベルでデータを保存する手法です。

名前などは指定する必要がなく、バイト単位の管理になります。

データ保存等は楽になりますが、その反面管理が複雑になりやすいです。

領域の途中にデータなどを追加すると領域を再設定する必要があります。

java.io.FileOutputStream

```
FileOutputStream fo  = null;
DataOutputStream dos = null;
try {
    // ContextからFileOutputStreamを取得する
    fo  = this.getApplicationContext().openFileOutput( "etc.sp", Context.MODE_PRIVATE );
    dos = new DataOutputStream( fo );
    // 書き込みを行う
    dos.writeInt( 1000 );
    // データの書き込みを反映させる
    dos.flush();
}
catch( Throwable err ) {
    Log.e( "OutputError", err.toString() );
}
finally {
    /*
      安全のために、必ず最後には取得したストリームを閉じるように設定すること
      また、close自体もブロックしてあげないといけない
    */
    try {
        if( fo  != null ) { fo.close();  }
        if( dos != null ) { dos.close(); }
    }
    catch( Throwable err ) {

    }
}
```

java.io.FileInputStream

```
FileInputStream fi  = null;
DataInputStream dis = null;
try {
    // ContextからFileOutputStreamを取得する
    fi  = this.getApplicationContext().openFileInput( "etc.sp" );
    dis = new DataInputStream( fi );
    // データの読み込みを行う
    int val = dis.readInt();

    Log.d( "", "" + val );
}
catch( Throwable err ) {
    Log.e( "InputError", err.toString() );
}
finally {
    /*
      安全のために、必ず最後には取得したストリームを閉じるように設定すること
      また、close自体もブロックしてあげないといけない
    */
    try {
        if( fi  != null ) { fi.close();  }
        if( dis != null ) { dis.close(); }
    }
    catch( Throwable err ) {

    }
}
```

6. おまけ

System.gc()のコールを乱発しない

ガベージコレクタをコールすると軽いメモリ回収でも20-30msは
かかってしまいます。
20FPS保持するにも50ms必要となりますので、
FPSの保持がきつくなってしまいます。

メモリ確保を乱発しない

ローカルでも new int[10] などをコールし続けるとガベージコレクタが
コールされてしまいます。
システムへの負荷が高くなってしまいますのでメモリ確保するタイミングは
注意した方が良いです。

Javaのライブラリを乱用しない

とても便利なメソッドが存在しているJavaライブラリのメソッドですが、
場合によってはメモリ確保等を多数行っているライブラリもありますので、
計画的に利用した方が良いです。

端末のマナー状態を取得する

携帯機用のアプリケーションを開発する場合には、

マナーモードによるプレイも意識した方がより良いゲームが作れます。

```
AudioManager mng  = (AudioManager)Context.getSystemService(Context.AUDIO_SERVICE);
.....
// マナーモードのチェック
int silentMode = mng.getRingerMode();
if( silentMode != AudioManager.RINGER_MODE_NORMAL ) {
    // マナーモードだった場合の処理を入れてみてください
}
```

割り込み系処理を意識する

携帯電話端末では、ゲームプレイ中に電話がかかってきたりなどの
様々な割り込みが発生する事はよくあると思います。
それらにできるだけ対処する事によってより良いアプリケーションを
作る事ができます。
Androidではライフサイクルというシステムに従った復帰処理を
作成していく事になります。

HTC-03Aの場合

1．通話キーを押した場合

onPause → onResume

2．ホームボタンを押した場合

onPause → onResume

3．終話キーを押した場合

onSaveInstanceState → onDestroy → onCreate

終話キー長押し時の携帯電話オプション画面

終話キーを長押しすると突然ウィンドウが出てきます。

ゲーム中にこの状態になると画面の優先順位はこの画面になります。

この場合、リアルタイムゲームなどはゲームを一時停止してあげると

ユーザーライクなアプリになります。

```
public void onWindowFocusChanged( boolean hasWindowFocus ) {
    if( hasWindowFocus ) {
        // 復帰処理
    }
    else {
        // 停止処理
    }
}
```

今回の講演させていただいた内容

1. メインループの作成方法
2. SurfaceViewを使った簡単な2D描画
3. 様々な入力について
4. サウンドの再生方法
5. ファイルの保存
6. おまけ

ご清聴ありがとうございました