2023年度

「実世界情報実験2 モバイル端末プログラミング」資料

Rev. 1.0 (20230923)

立命館大学 情報理工学部 実世界情報コース

担当：柴田史久，藤井 康之

学生証番号　　氏名

目 次

1　モバイルプログラミングの基本 (1)

概要

統合開発環境 Android Studio を利用したプログラミングについて基本的な事項を学ぶ．Android Studio の起動方法や設定，アプリケーション作成手順について学習する．Android Studio のインストール方法については別の資料を参照すること．

keywords

Android Studio，プロジェクトの作成，アプリケーションのビルド，レイアウトエディタ，エミュレータでの実行，実機での実行，画面レイアウト

準備

Android Studio のエンコーディングを設定･確認しておく．Android Studio を起動し，表示される Welcome 画面において，左のメニューの Customize を選択し，その中の All settings... をクリックして Settings 画面を表示する図 1.1 参照）．

左側のメニューにおける Editor → File Encodings の設定を確認し，図 1.2 中の矢印で示した Global Encoding と Project Encoding がともに UTF-8 になっていることを確認する．なっていなければ，UTF-8 に変更する．

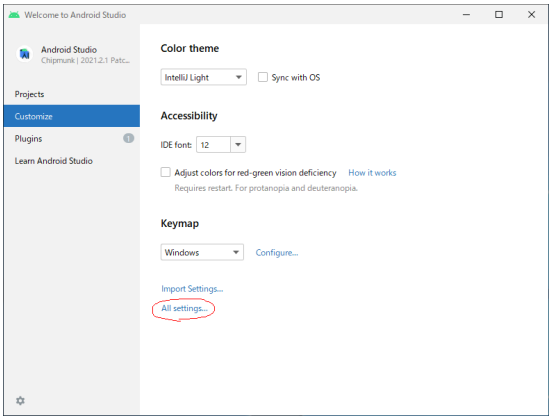

図 1.1: 2021.2.1 の Welcome 画面．
バージョンによって表示が一部異なる場合がある．

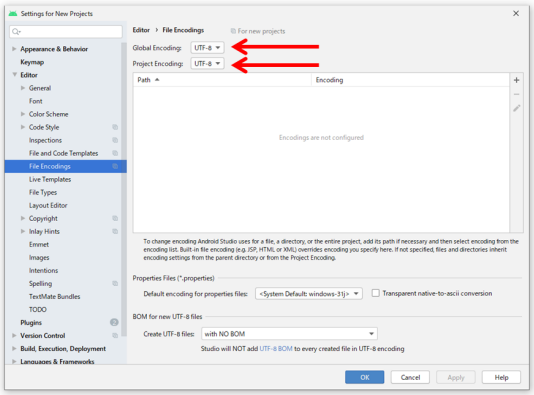

図 1.2: エンコーディングの設定．
UTF-8 に設定し，”with NO BOM”にする．

注意事項

Android Studio は頻繁にバージョンアップが行われるため，画面の細部が異なっている場合がある．また，Mac 版については，画面構成などが一部異なっている場合がある．

プロジェクトの作成にあたっては，ファイル名やフォルダ名に全角文字を使わず半角英数字を用いること．

1.1 課題 1-1

初めてのアプリケーションとして，以下の手順にそって HelloAndroid を作成する．

1.1.1 プロジェクトの新規作成

Welcome 画面（図 1.1 参照）において，左のメニューから Project を選び，”New Project” ボタンを押して新規プロジェクトの作成を開始する．

最初に New Project の画面（図 1.3 参照）では，作成するプロジェクトの種類を選ぶ．ここでは Phone and Tablet の選択肢の中から，「Empty Views Activity」[1]を選択する．Empty Views Activity（旧 Empty Activity．以下は旧名称は記載しないので適宜読み替えること）は，初期画面として無地の背景が表示されるシンプルなアクティビティである．今後のプロジェクト作成において，他の種類の背景を初期画面にする場合は，該当するものを選択するとよい．

Next ボタンを押すと Empty Views Activity の各種設定を行う画面（図 1.4 参照）に進むので，表 1.1 の項目を設定せよ．なお，”isXXXXXX” の部分については，自分の RAINBOW ID に読み替えること．通常，パッケージの名称はドメイン名を逆順にしたものをベースに作成する．ここでは，立命館大学情報理工学部の実世界情報コースに割り当てられたドメイン名をベースにしている．

Minimum SDK の項目では，”API 23: Android 6.0 (Marshmallow)” を選択している．SDK のバージョンを低く指定すると，より多くのデバイスで動作するが，利用できる機能が少なくなる．本実験向けに準備している Zenfone 3 Max は Android 6.0.1 を搭載しているため，ここでは API 23 を選んでいるが，自分が所有している端末向けにアプリケーションを作成する場合には，所有している端末の Android バージョンにあわせて設定を変更するとよい．

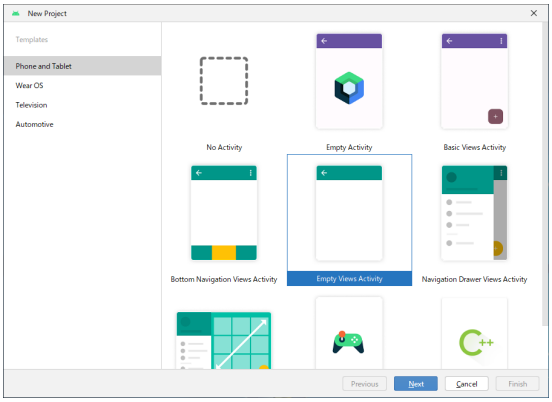

図 1.3: New Project の画面

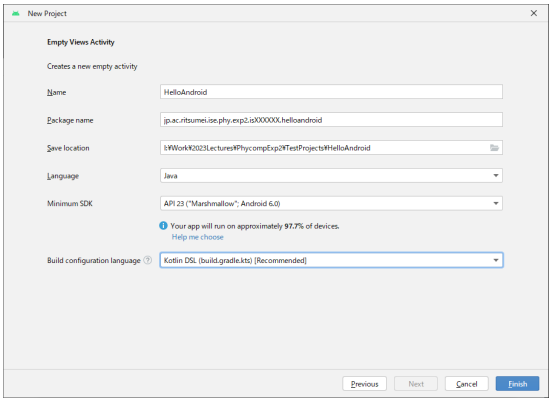

図 1.4: Empty Views Activity に関する設定画面

画面の右下にある Finish を押すとプロジェクトの作成が始まり，必要に応じてファイルの追加ダウンロードなどを行いながら，最終的には Android Studio の画面が現れる（図 1.5）[2]．

[1] 2023 年 6 月以降のバージョン（Android Studio Flamingo 2022.2.1 以降）から名称が変更になった．以前のバージョンでは「Empty Activity」を選択すること．

[2] 最初にプロジェクトを作成した場合は大量のファイルをダウンロードする場合がある．高速なネットワークに接続した状態で作業することを推奨する．バックグラウンドでの処理の進み具合は，Android Studio 画面の下部に表示されているので，何らかの処理が進んでいる場合は他の作業をせずに様子をみること．

表 1.1: New Project の設定内容

項目	設定内容	解説
Name	HelloAndroid	プロジェクト名
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.helloandroid	パッケージ名[a]
Save location	D:¥workspace¥HelloAndroid	プロジェクトの保存場所[b]
Language	Java	使用する言語を選択
Minimum SDK	API 23: Android 6.0 (Marshmallow)	API の SDK バージョン
Build configuration language[c]	Kotlin DSL (build.gradle.kts) [Recommended]	ビルドスクリプトの言語[d]

[a]自身の RAINBOW アカウントにあわせて適宜修正すること．他のアプリケーションとの競合を避ける必要がある点に注意せよ．
[b]適宜修正すること．これは，D ドライブ（USB メモリ）の workspace フォルダの下に HelloAndroid フォルダを作成する例．
[c]旧バージョンではこの項目は存在しない．以降では，この項目は省略する．
[d]以前の言語は Groovy(build.gradle)

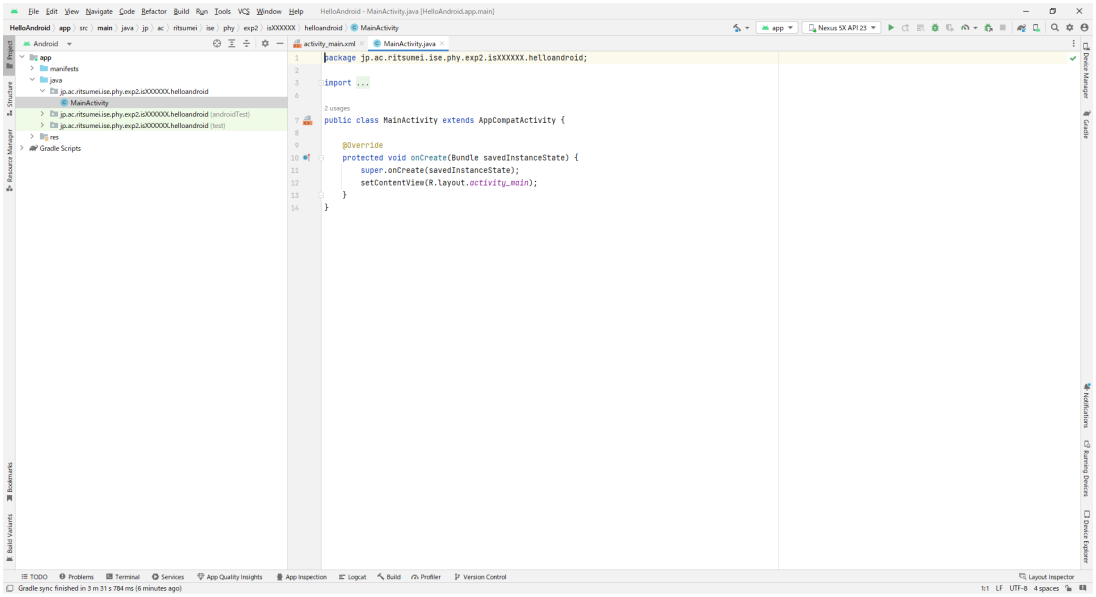

図 1.5: Android Studio の画面

1.1.2 Android Studio の画面

画面中央に Tips of the Day が表示されている場合は，不要なので閉じてよい．

Android Studio の基本画面は図 1.6 のような構成になっている．画面の左にあるのがプロジェクトツールウィンドウで，その右にあるのがエディタウィンドウである．また，下側や右側に別のウィンドウが表示される場合もある．Android Studio には多くのツールウィンドウがある．画面の左右や下部に表示／非表示やツールの種類を切り替えるボタン（例えば，左側であれば「Project」や「Structure」など）があるので，必要に応じて利用するとよい．

図 1.7 にプロジェクトツールウィンドウで重要な項目を列挙するので，新規作成されたプロジェクトにどのようなものがあるのかを確認せよ．

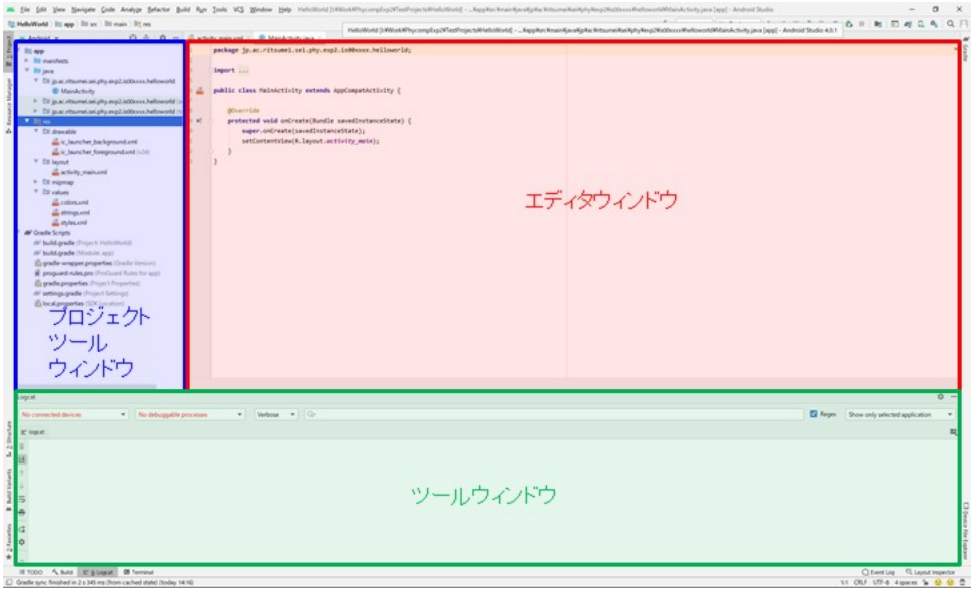

図 1.6: Android Studio の画面構成．左側にプロジェクトツールウィンドウ，右側にエディタウィンドウ，下側にツールウィンドウが表示されている場合．

1.1.3 レイアウトエディタ

エディタウィンドウにおいて，”activity_main.xml” と書かれたタブをダブルクリックするとレイアウトエディタが表示される．[3]

レイアウトエディタでは，ビュー（ボタンなどの部品）を視覚的なデザインエディタで配置することができる．図 1.8 にレイアウトエディタの画面配置を示す．レイアウトエディタの領域は以下の通りである．

Palette
: Design Editor で利用するビュー（ウィジェット，レイアウト等）のリストを表示．

Component Tree
: レイアウトで配置されているビューの階層構造を表示．

Design Editor
: デザイン（左側）とブループリント（右側）を表示．デザイン側でビューを配置し，ブループリント側でビューの制約を確認．

Attributes
: 現在選択されているビューの属性（アトリビュート，プロパティということも）を表示．

レイアウトエディタの詳細について知りたいときは以下の URL を確認するとよい．

Layout Editor による UI の作成
: `https://developer.android.com/studio/write/layout-editor`

[3]初めて表示する際には，ビルドなどの実行のために時間がかかる場合がある．またいつまでも表示されずにエラーが発生する場合は，トラブルシューティング B の章を確認せよ．

図 1.7: プロジェクトツールウィンドウ

1.1.4 文字列の変更

Android Studio では Empty Views Activity [4]を使った新規プロジェクトを作成すると，何もしなくても ”Hello World!” という文字列が画面の真ん中に表示されるアプリケーションが完成している．そこで，この文字列を ”Hello Android World!” に変更する．ここでは，直接文字列を入力して変更することとする．以下の手順にしたがって文字列を修正せよ．なお，本格的なアプリケーションを作成する場合には，文字列リソースを作成してそれを利用するとよい．

1. Component Tree もしくは Design Editor から，”Hello World!”を表示している TextView を選択する．
2. Attributes 領域の TextView に関する Common Attributes から，text（文字列）を見つけ，”Hello Android World!” に修正する．
3. （オプション）textSize など，他の属性を変更してもよい．textSize は，Common Attributes の中の textAppearance のサブメニューにある．

1.1.5 エミュレータでの実行

ここまでの作業が終了したら，エミュレータを使って作成したアプリケーションを実行せよ．

エミュレータでアプリケーションを実行するには，Android Studio のツールバー（右上，図 1.9）の三角形のボタンの左側（図中で”No Device”と表示されている部分）でアプリケーションを実行するエミュレータ端末を選んだ上で，三角形ボタンを押せばよい．

エミュレータ端末の管理については，付録の A.1 節を参照せよ[5]．

選択するとエミュレータの画面が表示され（図 1.10），アプリケーションのビルドを行った後，しばらくすると作成したアプリケーションの画面が現れる（図 1.11）．

なお，エミュレータの操作について詳しく知りたい場合は，以下の URL を参考にすること．

[4]アクティビティとは，アプリケーションの画面を提供するコンポーネントである．詳細については，Android 開発者サイトのアクティビティのページ（`https://developer.android.com/guide/components/activities?hl=ja`）を確認すること．

[5]1 つのエミュレータ端末毎に数 GB のハードディスク容量を消費するため，空き容量に注意すること．

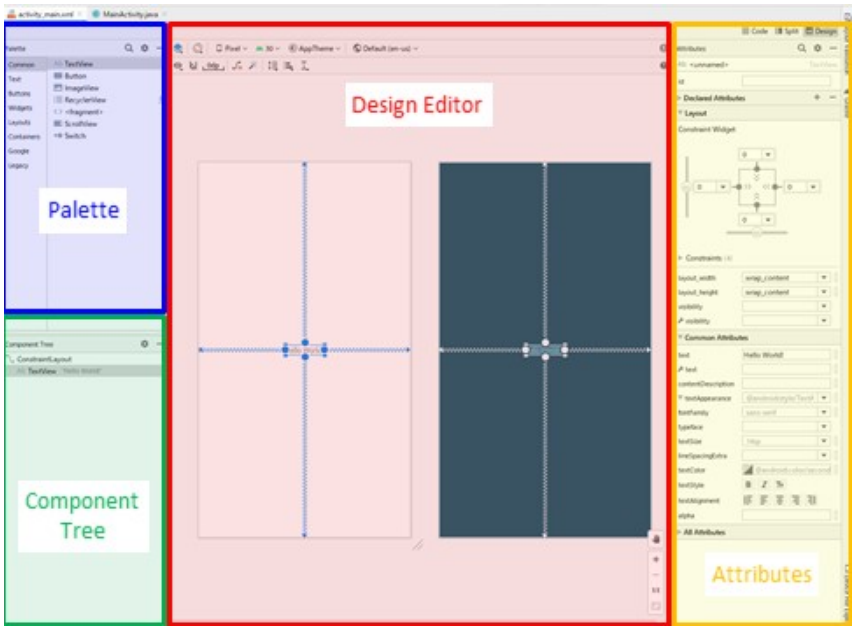

図 1.8: レイアウトエディタ

Android Emulator 上でアプリを実行する

`https://developer.android.com/studio/run/emulator?hl=ja`

図 1.9: ツールバー

1.1.6 実機による実行（オプション）

実機を使って作成したアプリケーションを実行することもできる[6].

まずは Android 端末を USB コネクタに接続する．この際,「USB デバッグを許可しますか？」というダイアログが表示された場合には,「はい」と入力する．また,「USB をファイル転送に使用しますか？」というダイアログに対しても「はい」を入力する．

実機でアプリケーションを実行するには，実機を USB コネクタに接続した状態で，Android Studio のツールバー（右上，図 1.9）の三角形のボタンの左側（図中で”No Device”と表示されている部分）でアプリケーションを実行する端末を選んだ上で，三角形ボタンを押せばよい．

実機にアプリケーションが転送され，しばらくすると作成したアプリケーションの画面が現れる．

1.2 課題 1-2

ボタンを押すと決められた範囲の整数をランダムで表示するサイコロアプリを作成せよ．プロジェクトの設定は表 1.2 の通りとする．

[6] 自身が所有する Android 端末で実行する場合,「開発者オプション」の「USB デバッグモード」をオンにする必要がある．インターネットなどを検索し,「開発者オプション」を表示する方法や「USB デバッグモード」をオンにする方法を調べるとよい．また，1.1.1 節に記載したように，プロジェクトを作成する際に自身の端末の OS バージョンにあわせてプロジェクトを作成する点に留意すること．

図 1.10: エミュレータ端末の起動画面例

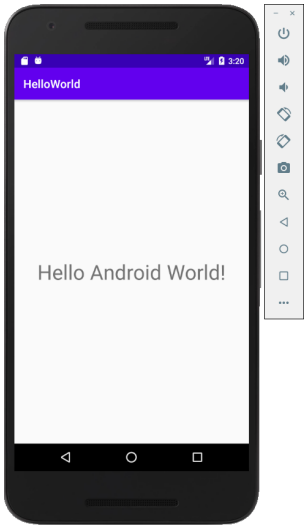

図 1.11: アプリケーションの実行例

表 1.2: 課題 1-2 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	SimpleDice
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.simpledice[a]
Save location	D:¥workspace¥SimpleDice[b]
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

[a]最初のプロジェクトと同様に isXXXXXX は自分の RAINBOW ID に読み替えること．以降，同様．
[b]最初のプロジェクトとは異なるフォルダ名にすること．以降，同様．

1.2.1 サイコロアプリ作成の手順

以下の手順にそって，サイコロアプリの作成せよ．ポイントとなる事項については，次節以降に説明しているのでそちらを参照すること．

1. 新規プロジェクト（SimpleDice）を作成
2. レイアウトエディタに切り替え，”Hello World!” の TextView を削除
3. パレットの Text の中から TextView を選択し，レイアウトに TextView を追加
   (a) 画面上部に配置
   (b) Common Attributes の text を 0 に変更
   (c) Common Attributes の textSize を 128sp に変更（”128sp”という文字列を入力）
   (d) Common Attributes の textAlignment を ”Align Center” に設定[7]
   (e) Attributes の id を dice に変更
4. パレットの Buttons の中から Button を選択し，レイアウトに Button を追加[8]

[7]Android Studio のバージョンによっては，textAppearance の中に存在する場合もある．また，”Align Center” ではなく ”Center” になっているバージョンも存在する
[8]Android Studio のバージョンによって画面の配色や，ボタンの色・形状などが異なる場合があるが気にしなくてよい．

(a) 画面下部に配置

(b) Layout の layout_width を 150dp に変更（”150dp”という文字列を入力）

(c) Layout の layout_height を 75dp に変更（”75dp”という文字列を入力）

(d) Common Attributes の text を ”d6” に変更

(e) Common Attributes の textSize を 32sp に変更（”32sp”という文字列を入力）

(f) Common Attributes の textAlignment を ”Align Center” に設定

(g) Attributes の id を button に変更

5. レイアウトエディタで TextView と Button の位置関係を調整（図 1.12 参照）
6. ボタンを押したときの処理を MainActivity.java に追加（詳細は後述）
7. Button に，上で作成したボタンを押したときに呼び出される処理を追加（詳細は後述）

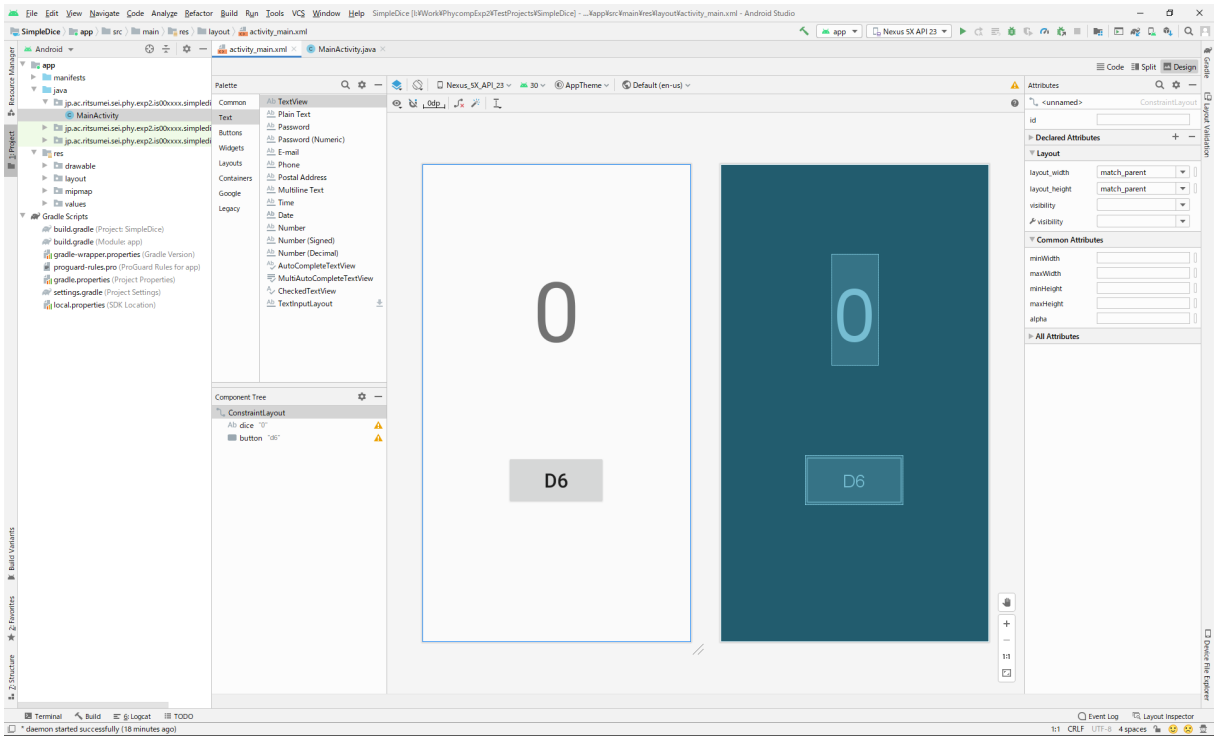

図 1.12: サイコロアプリの画面レイアウト

1.2.2 ボタンを押したときの処理

Android Studio の機能を使って，ボタンを押したときの処理を追加する．

まず，ボタンを押したときの処理内容をプログラムとして準備する．エディタウィンドウで MainActivity.java を表示し，以下に示した onClickButton という名前のメソッドを追加する．このメソッドは，ボタンを押した際に呼び出されるもので，dice という id を持つビュー（ここでは TextView）を探し出し，そのテキストの内容を 1〜6 の範囲の整数に置き換える，という処理になっている．乱数の発生には，java.util.Random クラスを利用している．

プログラムをそのまま入力すると，View，TextView，Random クラスの部分でエラーが発生し，赤字で表示される．エラーが発生しているクラスの部分にカーソルを持っていった状態で Alt キーと Enter

キーを同時に押すと，修正候補が表示される．例えば，Random クラスの部分のエラーを解消するには java.util.Random クラスを import すればいいため，”Import class”という修正候補を選択すればよい．他の View，TextView クラスについても同様に修正すること．

```
public void onClickButton(View view) {
    TextView dice = (TextView)findViewById(R.id.dice) ;
    Random random = new Random() ;
    dice.setText(String.format("%d", random.nextInt(6) + 1)) ;
}
```

次に，レイアウトエディタに切り替えて作成したボタンを選択すると，ボタンの属性が表示される．Common Attributes の中の onClick という属性の右端のプルダウンを選ぶと，MainActivity の中に上で作成した onClickButton というメソッドが表示されるので，これを登録する（図 1.13）．

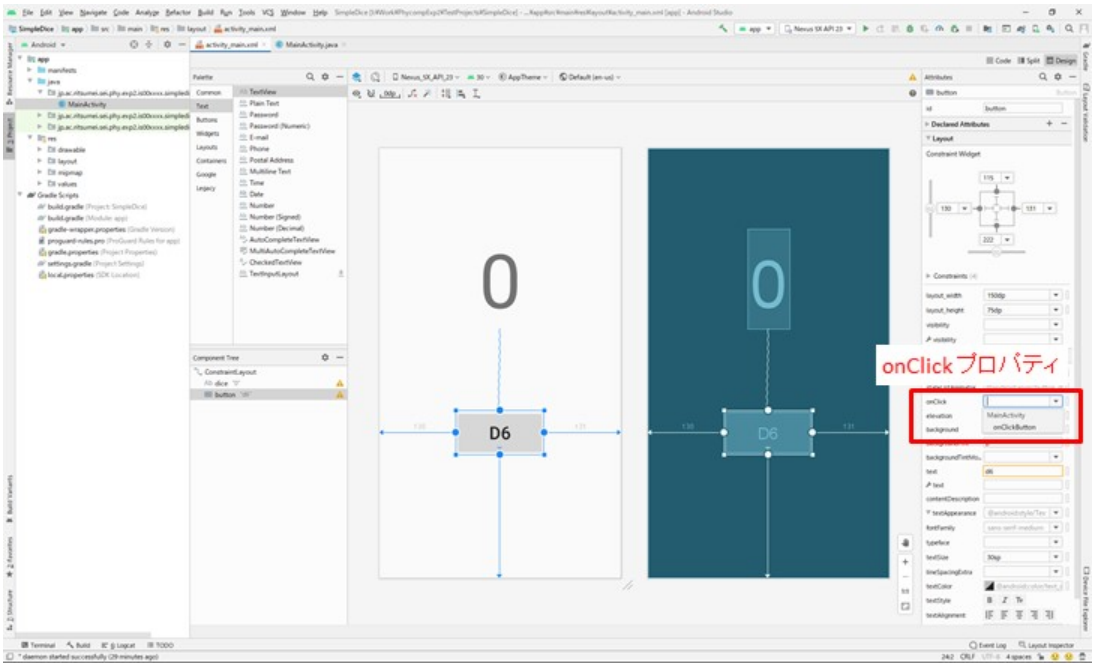

図 1.13: onClick プロパティ

ここまでの作業でサイコロアプリが完成したはずなので，エミュレータおよび実機で動作を確認せよ．エミュレータにおけるアプリの起動直後の画面を図 1.14 に，サイコロを振った後の画面を図 1.15 に示す．

1.3 課題 1-3

図 1.16 に示した画面配置を参考に，複数種類のサイコロ（ここでは d4, d6, d8, d12, d20 としているが種類・数については自由）を選んで，決められた範囲の整数（例えば，d4 であれば 1〜4 の整数）をランダムで表示するサイコロアプリ 2 を作成せよ．ボタンなどの配置には Layouts の LinearLayout を利用するとよい．プロジェクトの設定は表 1.3 の通りとする．

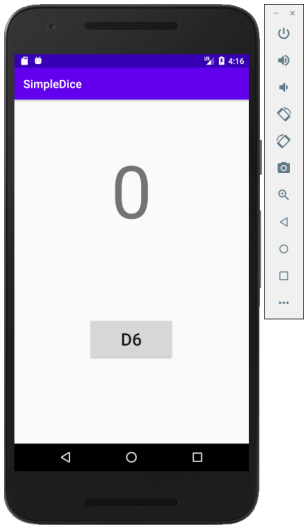

図 1.14: 起動直後の画面

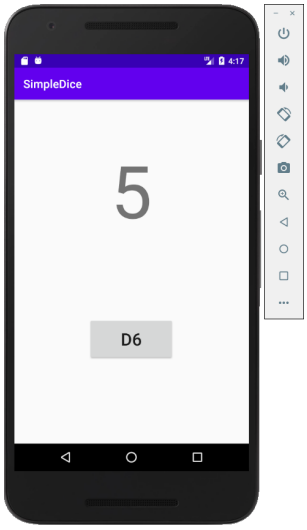

図 1.15: サイコロを振った後の画面

表 1.3: 課題 1-3 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	SimpleDice2
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.simpledice2
Save location	D:¥workspace¥SimpleDice2
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

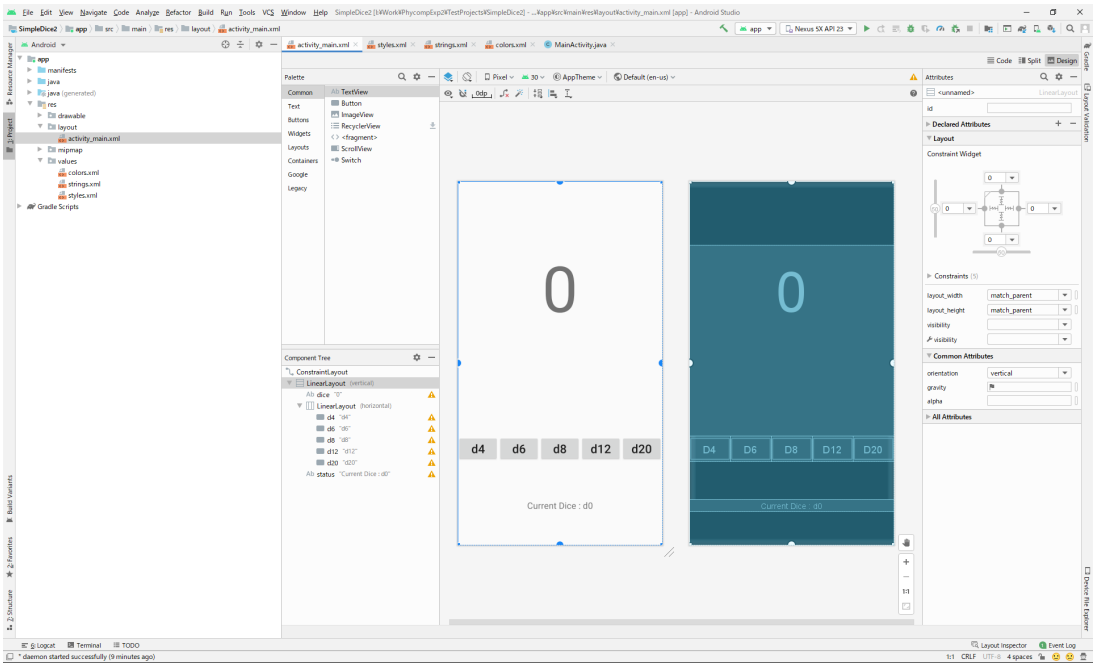

図 1.16: 課題 1-3 の画面配置

2　モバイルプログラミングの基本 (2)

概要

Android プログラミングにおける基本的な GUI や画面遷移，インテントについて学習する．

keywords

基本的な GUI，画像リソース，画面遷移，インテント

2.1　課題 2-1

コンピュータ（Android 端末）を相手にじゃんけんをするアプリケーション（以下，じゃんけんアプリ）を作成せよ．

じゃんけんアプリの概要

じゃんけんアプリは 2 つの画面（スタート画面（図 2.1）とゲーム画面（図 2.2，図 2.3）参照）を持ち，スタート画面において Start ボタンを押すと，ゲーム画面へと遷移してじゃんけんが始まる．プレイヤが，じゃんけんボタン（グー・チョキ・パーのいずれかのボタン）を選択すると，コンピュータがランダムで生成した手と比較して，勝敗が決まる．再度，じゃんけんボタンを押すともう一度勝負できる．また，画面下部にある Exit ボタンを押すとスタート画面に戻る．

プロジェクトの設定は表 2.1 の通りとする．

表 2.1: 課題 2-1 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	Jankenpon
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.jankenpon
Save location	D:¥workspace¥Jankenpon
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

2.1.1　じゃんけんアプリ作成の手順

以下の手順にそって，じゃんけんアプリを作成せよ．ポイントとなる事項については，次節以降に説明しているのでそちらを参照すること．

1. 新規プロジェクト（Jankenpon）を作成
2. レイアウトエディタに切り替え，アクティビティから ”Hello World!” の TextView を削除
3. manaba+R からじゃんけんの画像（kadai2image.zip）をダウンロード[9]
4. ダウンロードした画像を drawable フォルダに配置
   (a) エクスプローラーでダウンロードした画像をコピー（画像ファイルをマウスで選択して Ctrl+C でコピー）（図 2.4）

[9]イラスト素材については，素材ライブラリー.com（https://www.sozai-library.com/）のサイトから許諾を得て利用しています．実験以外での利用については，別途，素材ライブラリー.com の利用規約を確認してください．

図 2.1: スタート画面

図 2.2: じゃんけん前の画面

図 2.3: じゃんけん後の画面

(b) プロジェクトの drawable フォルダにペースト（マウスの右クリックメニューから選択するか，Ctrl+V でもペースト可能）（図 2.5 参照）

(c) コピー先は drawable-v24 ではなく drawable を選択（図 2.6）

(d) ファイル名を適宜設定[10]（図 2.7）

5. 図 2.8 を参考にレイアウトエディタでスタート画面を作成

(a) ImageView を 3 つ配置し，グー，チョキ，パーの画像を設定[11]

(b) TextView を配置し，タイトルを設定

(c) スタートボタンを配置

6. 図 2.9 を参考にレイアウトエディタでゲーム画面を作成

(a) File メニューから New → Activity → Empty Views Activity を選択

(b) Configure Activity の画面で，Activity Name を ”GameActivity” に変更し，ゲーム画面を作成（図 2.10 参照）

(c) レイアウトエディタを利用して要素を配置

7. じゃんけんの機能を実装

(a) Start ボタンにスタート画面からゲーム画面への遷移機能を追加（詳細は後述）

(b) ゲーム画面のじゃんけんボタンにじゃんけん機能を追加

(c) ゲーム画面の Exit ボタンにスタート画面に戻る機能を追加（詳細は後述）

2.1.2 画面の遷移

ここでは，スタート画面からゲーム画面に遷移する方法について説明する．新しい画面（アクティビティ）を開くにはインテント（Intent）という仕組みを使う[12]．インテントによる画面遷移の基本的な流れは以下

[10] ファイルによるリソースの名前は英小文字，数字，アンダースコアのいずれかである必要がある点に注意せよ．

[11] Common Attributes の scaleType を fitCenter にすると画像が ImageView の大きさに調整される

[12] インテントの詳細については，インターネットなどで調べるとよい．例えば，Android 開発者向けのページに一般的なインテント（URL `https://developer.android.com/guide/components/intents-common?hl=ja`）が掲載されている．

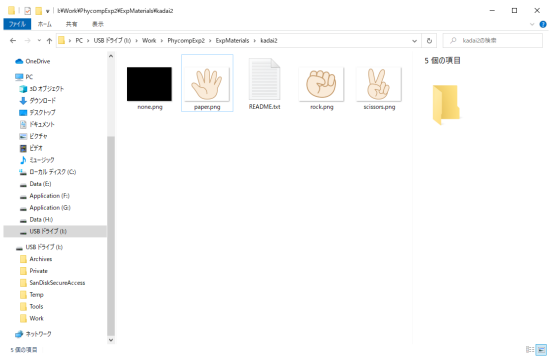

図 2.4: エクスプローラーの画面

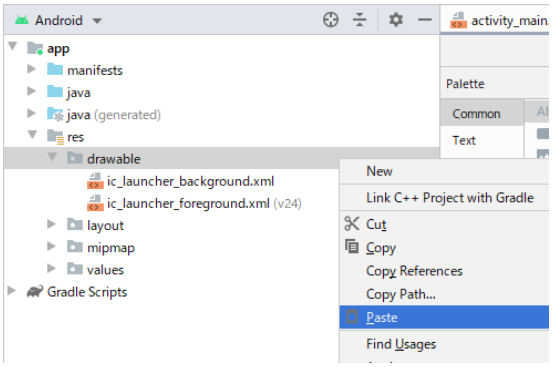

図 2.5: プロジェクトの drawable フォルダに Paste する様子

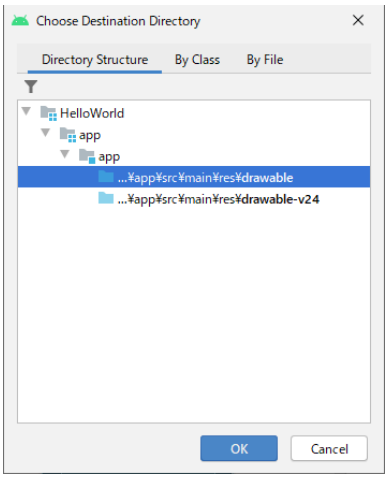

図 2.6: コピー先の選択

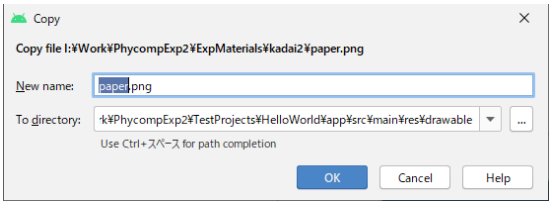

図 2.7: ファイル名の設定

のようになる．

1. 元のアクティビティで Intent クラスのインスタンスを設定
2. putExtra メソッドを利用して遷移先に送りたいデータを Intent のインスタンスにセット
3. startActivity メソッドで画面を遷移
4. 遷移先のアクティビティで getXXExtra メソッドを使って送ったデータを取り出す
5. 必要に応じて finish メソッドを使って元のアクティビティに遷移

まず，元のアクティビティ（MainActivity.java）に上記の 1〜3 の処理を記述する．ここでは遷移先に送りたいデータはないので，以下のようなプログラムを追加する．メソッド名はどのようなものでもよいが，「public void メソッド名 (View view)」というシグニチャにする必要がある．

Intent クラスのインスタンスを生成する際に，第 1 引数としては呼び出し元のアクティビティ（ここでは MainActivity クラス自身にコードを書くので自身のインスタンスを表す this を指定）を，第 2 引数として呼び出し先のクラス（ここでは，GameActivity.class）を指定する．その後，Intent クラスのインスタンスを引数として startActivity メソッドを呼び出し，画面を遷移させる．作成したメソッドは，課題 1-2 と同様に Start ボタンの onClick 属性で指定する．

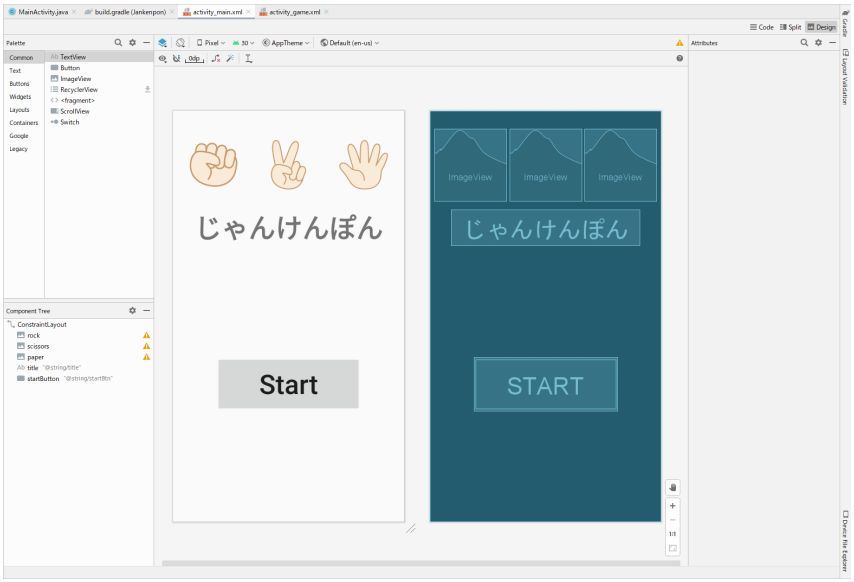

図 2.8: スタート画面のレイアウト

```
public void onStartButtonTapped(View view) {
    Intent intent = new Intent(this, GameActivity.class) ;
    startActivity(intent) ;
}
```

2.1.3 スタート画面への遷移

遷移先の Activity の finish メソッドを実行すれば，現在のアクティビティを終了し，元のアクティビティへ戻ることができる．GameActivity.java に以下のメソッドを追加し，EXIT ボタンのクリックでこのメソッドが呼び出されるようにすればよい．

```
public void onExitButtonTapped(View view) {
    finish() ;
}
```

2.2 課題 2-2

課題 2-2 については選択課題とする．以下のいずれかの課題を選択せよ．なお，余力のあるものが両方の課題に取り組むことは妨げない．

2.2.1 じゃんけんぽんアプリの改良

課題 2-1 で作成したじゃんけんぽんアプリを改良し，じゃんけんの思考ルーチンを強化せよ．また，これまでの勝敗を記録するなど，自分のアイデアでアプリケーションの完成度を高めよ．ただし，思考ルーチン

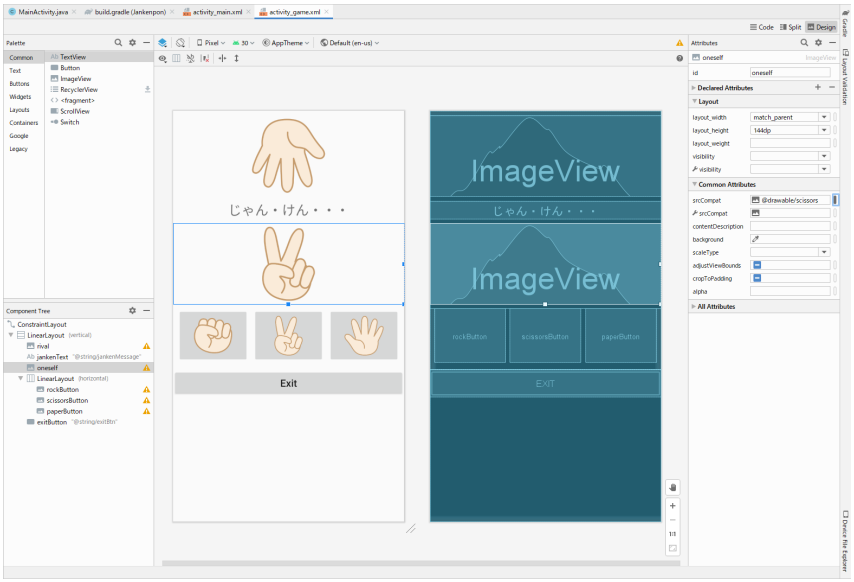

図 2.9: ゲーム画面のレイアウト

については，後出し（すなわち，プレイヤの手を知ってからコンピュータがその情報を利用して手を決める）はなしとする．

2.2.2 電卓アプリの作成

電卓アプリを作成せよ．画面のレイアウト，ボタンの配置などは自分で考えること．四則演算に加えて，平方根の計算などの機能を加えてもよい．自分だけのオリジナル電卓アプリを完成させよ．

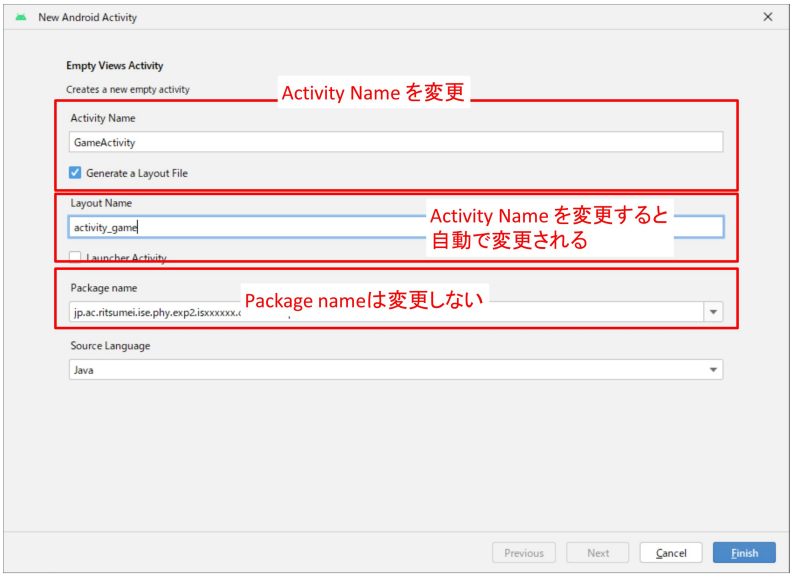

図 2.10: ゲーム画面の追加

3　モバイルプログラミングの基本 (3)

概要

図形の描画や画像表示，アニメーションなどについて学習する．SurfaceView を利用した図形・画像の描画方法やタッチイベントの扱い，スレッドを用いてアニメーションを実現する方法を学ぶ．

keywords

イメージの表示，図形の描画，アニメーション，SurfaceView，コールバック，タッチイベント

3.1　課題 3-1

以降に説明する手順にそって，SurfaceView クラスを継承した MySurfaceView クラスを作成し，図形（円）の描画を行うプログラム（CanvasSample01）を作成せよ．プロジェクトの設定は表 3.1 の通りとする．

表 3.1: 課題 3-1 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	CanvasSample01
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.canvassample01
Save location	D:¥workspace¥CanvasSample01
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

3.1.1　CanvasSample01 の作成手順

ここでは，黄色の背景色の画面中央に，半径 50 ピクセルの赤い円を描くプログラムを作成する．CanvasSample01 の作成手順は以下のとおりである．ポイントとなる事項については，次節以降に説明しているのでそちらを参照すること．

1. 新規プロジェクト（CanvasSample01）を作成
2. 作成される Empty Views Activity から ”Hello World!” の TextView を削除
3. SurfaceView クラスを継承した MySurfaceView クラスを新規作成（詳細は後述）[13]．
4. MySurfaceView クラスに SurfaceHolder.Callback インタフェースを実装（詳細は後述）
5. MySurfaceView に必要な処理を追加
   (a) コンストラクタにおいて自身を SurfaceHolder からのコールバックを受け取るように設定（詳細は後述）
   (b) surfaceCreated メソッドの中に，画面を描画する処理を追加（詳細は後述）
6. MySurfaceView を MainActivity のレイアウトエディタで配置[14]

[13] 新規クラスの作成手順は Android Studio のバージョンによって多少異なる．
[14] レイアウトエディタを使わずに，MainActivity クラスの onCreate メソッドのおいて `setContentView(new MySurfaceView())` のように直接配置することもできる．

(a) レイアウトエディタのパレットで Project を選択[15]

(b) 作成した MySurfaceView が表示されるので選択してレイアウトの中央に配置

(c) MySurfaceView のデザインエディタにおいて MySurfaceView の上下左右の Constraints が，画面の上下左右とつながるように設定（図 3.3）

3.1.2 SurfaceView クラスの継承

File メニューから New → Java Class を選び，MySurfaceView という名前の新規クラスを作成する．新規に作成した MySurfaceView.java は，以下のようなコードになる．

```
package jp.ac.ritsumei.ise.phy.exp2.is00xxxx.canvassample01;

public class MySurfaceView {
}
```

MySurfaceView は android.view.SurfaceView クラスの派生クラスとするので，以下のように SurfaceView クラスを extends するように修正し，android.view.SurfaceView クラスを import する．

```
package jp.ac.ritsumei.ise.phy.exp2.is00xxxx.canvassample01;

import android.view.SurfaceView;

public class MySurfaceView extends SurfaceView {
}
```

この時点で，MySurfaceView クラスのコンストラクタが親クラス（SurfaceView）のコンストラクタと整合していないため，クラスの先頭にエラーが表示されているはずである．エラー箇所の上に表示される ”電球” マークをクリックすると ”Create constructor matching super” という項目が表示されるので，これを選択する．すると図 3.1 が表示されるので，実装する 4 つのコンストラクタをすべて選択し，OK ボタンを押す．複数を選択する際には，コントロールキーを押しながらマウスでクリックするとよい．

3.1.3 SurfaceHolder.Callback インタフェースの実装

SurfaceView は，例えば画面のサイズが途中で変化するなど，自身の Surface の状態変化を監視する必要がある．状態の変化を監視するのは，SurfaceView に準備された SurfaceHolder インタフェース（android.view.SurfaceHolder）である．SurfaceHolder インタフェースには SurfaceHolder.Callback インタフェースが含まれており，画面変化が発生したときに発生するイベントを SurfaceHolder.Callback インタフェースに含まれる 3 つのメソッド（表 3.2）によって処理する．

以下のように SurfaceHolder インタフェースを利用するための import 文を追加し，MySurfaceView クラスに SurfaceHolder.Callback インタフェースを実装（implements）することを明示する．

[15] MySurfaceView を作成後に Project が表示されていないときは，一度，Android Studio のプロジェクトを保存・終了して，再度読み込みなおしてみること．

表 3.2: 実装が必要なメソッド

メソッド名	説明
surfaceCreated	サーフェスが作られたときに呼ばれるメソッド
surfaceChanged	サーフェスに変化があったときに呼ばれるメソッド
surfaceDestroyed	サーフェスが破棄されたときに呼ばれるメソッド

```
package jp.ac.ritsumei.ise.phy.exp2.is00xxxx.canvassample01;

import android.content.Context;          /* コンストラクタ作成の際に自動で追加される */
import android.util.AttributeSet;        /* コンストラクタ作成の際に自動で追加される */
import android.view.SurfaceView;
import android.view.SurfaceHolder;

public class MySurfaceView extends SurfaceView implements SurfaceHolder.Callback {

/* 以下省略 */
```

この時点で，表 3.2 に示したメソッドを MySurfaceView クラスで実装していないために，クラスの先頭にエラーが表示されているはずである．エラー箇所の上の ”電球” マークをクリックすると ”Implement methods” という項目が表示されるので，これを選択する．これを選択する．すると，図 3.2 が表示されるので，実装する 3 つのメソッドをすべて選択し，OK ボタンを押す．この際，追加されるメソッドの引数に @NonNull アノテーションを追加するために，androidx.annotation.NonNull が import される．@NonNull アノテーションはフィールドやメソッドの引数, メソッドの Return 値に Null を許容しないことを表明するアノテーションである[16]．

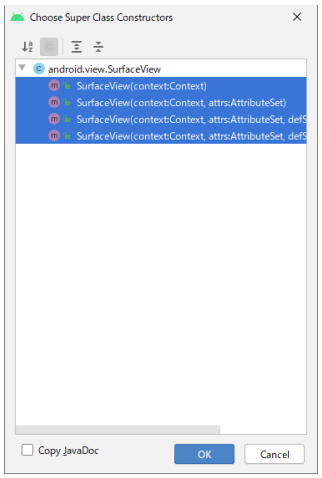

図 3.1: 実装するコンストラクタの選択

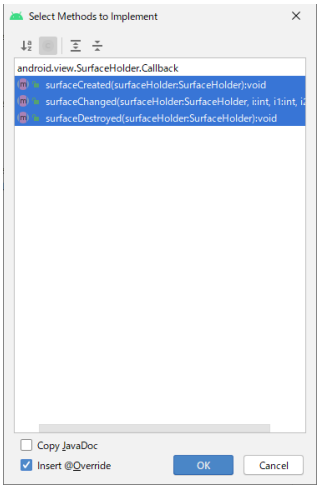

図 3.2: 実装するメソッドの選択

[16]アノテーションの詳細については，インターネットなどで調べるとよい．例えば，Android 開発者向けのページにアノテーションによるコード検査の改善に関する説明（URL `https://developer.android.com/studio/write/annotations?hl=ja`）が掲載されている．

3.1.4 コールバックの登録

SurfaceView には，SurfaceHolder を取得するための getHolder メソッドが準備されているのでこれを利用する．ここでは画面描画の際の利便性を考えて，SurfaceHolder を格納するフィールド変数 sHolder を準備しておく．コンストラクタが 4 つあるので，すべてのコンストラクタでコールバックの登録ができるように initialize メソッドを準備し，この中でコールバックの登録を行う．以下がプログラムコードである．

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

    private SurfaceHolder sHolder ;  // SurfaceHolder を格納

    public MySurfaceView(Context context) {
        super(context);
        initialize() ;
    }

    public MySurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initialize() ;
    }

    public MySurfaceView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        initialize() ;
    }

    public MySurfaceView(Context context, AttributeSet attrs, int defStyleAttr,
int defStyleRes) {
        super(context, attrs, defStyleAttr, defStyleRes);
        initialize() ;
    }

    private void initialize() {
        sHolder = getHolder() ;       // SurfaceHolder を取得
        sHolder.addCallback(this) ;   // 自身をコールバックとして登録
    }

    /* 以下，SurfaceHolder.Callback インタフェースのメソッドは省略 */
}
```

3.1.5 画面の描画

surfaceCreated メソッドは，Surface が作成された際に呼ばれるメソッドである．ここでは，surfaceCreated メソッドの中で，別途作成する drawCanvas メソッドを呼び出すこととし，drawCanvas メソッドの中で画面の描画を行う．

```
@Override
public void surfaceCreated(@NonNull SurfaceHolder surfaceHolder) {
    drawCanvas();  // 描画処理は drawCanvas メソッドに記述
}
```

画面に図形などを描画するには，SurfaceHolder から Canvas クラスのインスタンスを取得し，Canvas クラス（android.graphics.Canvas）と Paint クラス（android.graphics.Paint）を使って描画する必要がある．具体的には，SurfaceHolder の lockCanvas メソッドを使って，他のスレッドから Canvas オブジェクトを操作されないように Canvas をロックした上で，Canvas オブジェクトを取得し，描画する．

Canvas クラスには描画に必要なメソッドが準備されており，例えば円を描画したければ drawCircle メソッドを利用する．主な描画用メソッドを表 3.3 に示す．Canvas クラスには多数のメソッドが存在するため，詳細は以下の URL を確認するとよい．

Canvas

`https://developer.android.com/reference/android/graphics/Canvas`

表 3.3: Canvas クラスの主な描画用メソッド

メソッド名	説明	メソッド名	説明
drawArc	円弧を描画	drawBitmap	画像を描画
drawCircle	円を描画	drawColor	背景を塗りつぶす
drawLine	線の描画	drawPoint	点を描画
drawRect	矩形を描画	drawText	文字を描画

ここでは，黄色の背景に赤い円を描画するので，drawColor と drawCircle を利用する．Paint クラスのインスタンスを作り，描画色を赤色に設定した上で，drawCircle メソッドを呼び出せば赤い円が描画される．色の指定には，Color クラス（android.graphics.Color）を利用する．

円は画面の中央に描画する必要があるため，Canvas クラスの getWidth メソッドと getHeight メソッドを利用して Surface の幅と高さを取得し，円の描画に利用する．円の半径については予め RADIUS として定義しておく．

描画が終わったら SurfaceHolder の unlockCanvasAndPost メソッドを呼び出して，Canvas のロックを解除し，画面表示を更新する．

```
    private final static float RADIUS = 50.0f ;  // 円の半径

    private void drawCanvas() {
        Canvas c = sHolder.lockCanvas() ;  //
        c.drawColor(Color.YELLOW) ;
        Paint p = new Paint() ;
        p.setColor(Color.RED) ;
        int width = c.getWidth() ;
        int height = c.getHeight() ;
        c.drawCircle(width / 2, height / 2, RADIUS, p) ;
        sHolder.unlockCanvasAndPost(c) ;
    }
```

ここまで作成したら，MySurfaceView の MainActivity の上にレイアウトエディタで配置せよ（図 3.3）．一度，Project を閉じて，再度開きなおすと，レイアウトエディタの Palette に Project という項目が追加され，その中に先ほど作成した MySurfaceView がある．配置してレイアウトを整えれば完成なので，ContraintLayOut において，追加した MySurface の 4 つの辺を，それぞれ親画面の 4 つの辺と結びつける[17]．完成したら動作確認をせよ．エミュレータでの実行例を図 3.4 に示す．

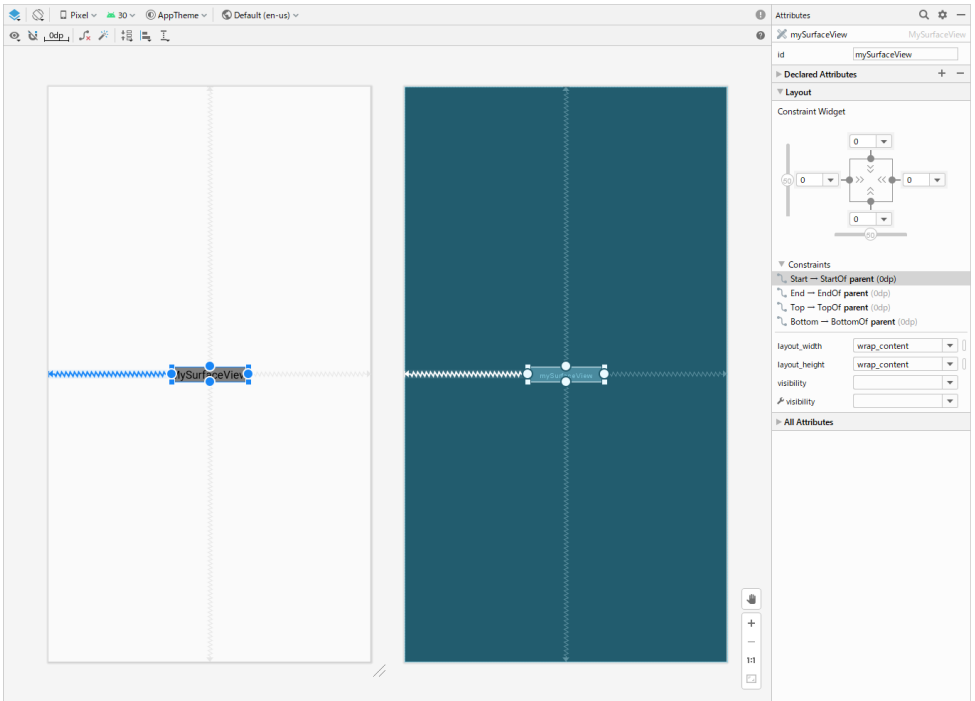

図 3.3: レイアウトエディタでの MySurfaceView の配置

[17] 4 辺を結びつけると，画面右の Layout 属性の部分において，Constraints の設定が,「Start → StarOf parent」「End → EndOf parent」「Top → TopOf parent」「Bottom → BottomOf parent」となる．

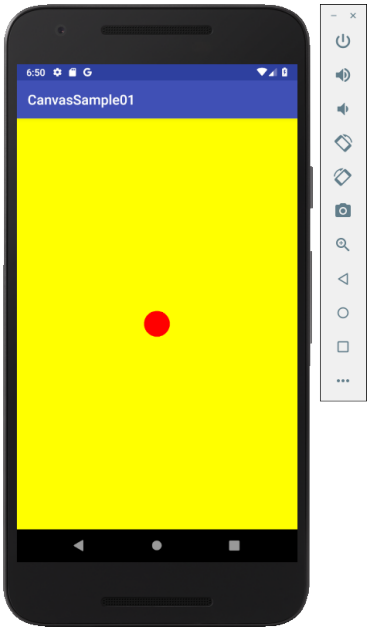

図 3.4: 課題 3-1 の実行例

3.2 課題 3-2

課題 2-1 で用いたじゃんけんの画像を使って，SurfaceView の Canvas に画像が表示されるプログラム CanvasSample02 を作成せよ．このプログラムは以下の仕様を満たすこととする．画像の描画や回転方法などについては次節以降に説明しているのでそちらを参照すること．課題 3-2 の設定内容を表 3.4 に示す．タッチの検出や画像の描画までのプログラム作成の流れは課題 3-1 と同様である．

- 表示される画像は drawable フォルダに配置したグー・チョキ・パーのいずれかの画像とする
- 画面をタッチすると画像がランダムに切り替わる
- 画像は画面の中央に表示される
- 画像の向きは 0〜359 度の範囲でランダムに回転する

表 3.4: 課題 3-2 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	CanvasSample02
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.canvassample02
Save location	D:¥workspace¥CanvasSample02
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

3.2.1 タッチの検出

Android においてもっとも一般的な入力インタフェースはタッチスクリーンである．ここでは，ユーザによるタッチスクリーンの操作を受け付ける方法について説明する．

Android でユーザによるタッチスクリーンの操作（タッチイベント）を受け取るには，onTouchEvent メソッドを利用する．タッチイベントの呼び出しの優先順位は，上位の View から下位の View へと順番になっており，いずれの View でも処理されなかった場合は，Activity のタッチイベントが呼び出される（表 3.5 参照）．

onTouchEvent メソッドの返り値の型は boolean となっている．標準は false だが，返り値に true を設定するとタッチイベントを消化したことになり，他の View や Activity への通知が抑制される．

表 3.5: タッチベントの呼び出し

クラスとメソッド名	説明
View.onTouchEvent	上位レイヤのメソッドから順に呼び出される
Activity.onTouchEvent	View で処理されなかった場合に呼び出される

ある View においてタッチイベントを処理したい場合，その View に以下の形式にそった onTouchEvent メソッドを追加し，必要な処理を実装すればよい．

```
    @Override
    public boolean onTouchEvent(MotionEvent event) {

        /* タッチイベントに対する処理 */

        return super.onTouchEvent(event) ;  // 下位の View, Activity のタッチイベント
の結果を返す
    }
```

touchEvent メソッドの引数である MotionEvent クラスのインスタンスには発生したタッチイベントの情報が格納されており，それらの情報を引き出すメソッドが準備されている（表 3.6 参照）．これらの情報をもとに必要な処理を実装する．アクションの種類には表 3.7 に示すようなものがある．

この課題では，画面がタッチされた際に表示する画像を切り替えればいいため，MotionEvent クラスのインスタンスから getAction メソッドでアクションの種類を取り出した上で，タッチ（ACTION_DOWN）であれば，画像を表示するような処理を追加すればよい．

表 3.6: MotionEvent クラスの主なメソッド

メソッド名	説明
getX()	タッチされた画面の X 座標
getY()	タッチされた画面の Y 座標
getAction()	タッチイベントのアクション
getDownTime()	押されていた時間（ms）
getEdgeFlags()	スクリーンの端の判定

表 3.7: MotionEvent クラスの主な定数

定数名	説明
ACTION_DOWN	0x00 タッチ（押下）
ACTION_UP	0x01 タッチ終了
ACTION_MOVE	0x02 指を上げずにスライド
ACTION_CANCEL	0x03 キャンセル
ACTION_OUTSIDE	0x04 UI の範囲外の押下

3.2.2 画像の描画

ここでは drawable フォルダに配置した画像リソースを画面に描画することを考える．画像リソースを読み込むには BitmapFactory クラス（android.graphics.BitmapFactory）の decodeResource メソッドを利

用する．読み込んだ画像は Bitmap クラス（android.grapihics.Bitmap）のインスタンスとして得られるので，drawBitmap メソッド（表 3.3 参照）を使用して Canvas に描画する．なお，画像の読み込みは負荷の高い処理のため，できるだけ読込回数を削減するような工夫をすることが推奨される．

例えば，rock という名前の画像ファイル（rock.png）を座標 (0, 0) に描画するには以下のようなコードを書けばよい．なお，canvas および paint はそれぞれ Canvas クラスおよび Paint クラスのインスタンスである．

```
Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.rock);
canvas.drawBitmap(bmp, 0, 0, paint) ;
```

3.2.3 Canvas の平行移動・回転・スケーリング

Canvas を使った描画では，描画する図形・画像を平行移動・回転・スケーリングすることができる．これらの操作をするためのメソッドを表 3.8 に示す．

drawBitmap メソッドは，画像の左上を基準に描画するため，じゃんけんの画像を平行移動，回転，スケーリングして描画位置を調整するとよい．

表 3.8: 平行移動・回転・スケーリング

メソッド名	説明
void translate(float x, float y)	座標を x, y だけ移動
void rotate(float degrees)	座標を degrees だけ回転
void scale(float x, float y)	座標を軸方向に x, y 倍にスケーリング

実行画面を図 3.5 に示す．

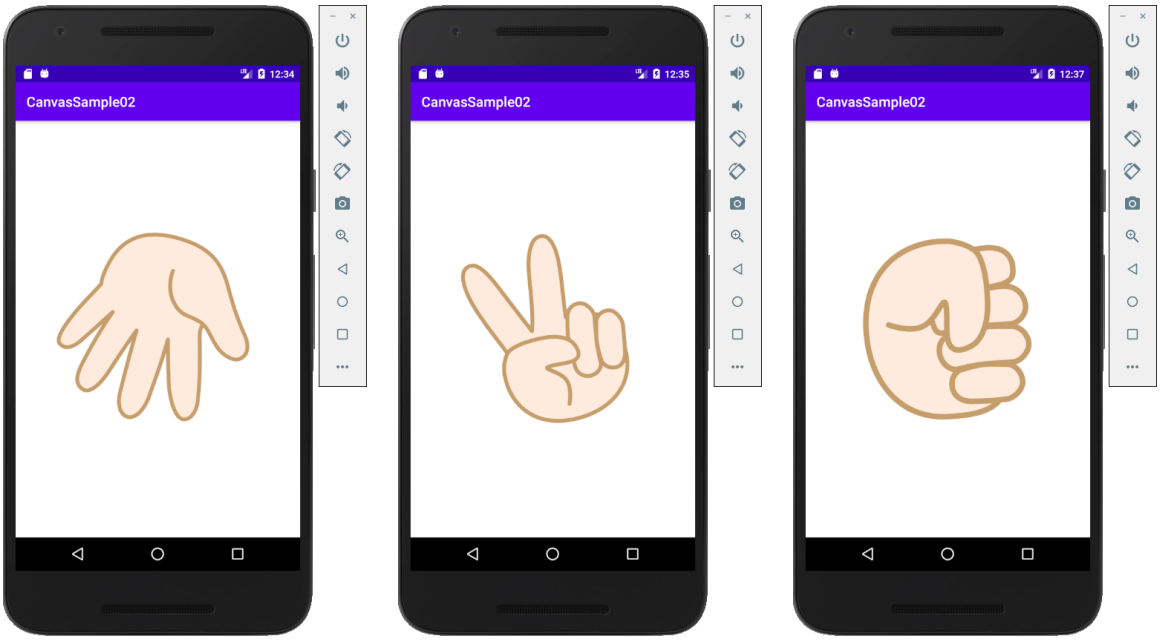

図 3.5: CanvasSample02 の実行画面

3.3 課題 3-3

課題 3-1 で画面に表示した赤い円をアニメーションさせるプログラム（CanvasSample03）を作成せよ．このプログラムは以下の仕様を満たすこととする．課題 3-3 の設定内容を表 3.9 に示す．キャラクタを移動させる方法については次節以降に説明しているのでそちらを参照すること．

- 初期状態では赤い円は画面中央にある
- スタートすると画面右方向に動き始め，画面の端に接触すると移動方向が反転する
- 画面をタッチすると左右方向の移動を停止し，下方向に移動を開始し，画面の端に接触すると移動方向が反転する
- 再度画面をタッチすると画面左右どちらかの方向（従前の方向）に動き始める
- さらに再度画面をタッチすると画面上下どちらかの方向（従前の方向）に動き始める
- 以降，上記を繰り返す

表 3.9: 課題 3-3 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	CanvasSample03
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.canvassample03
Save location	D:¥workspace¥CanvasSample03
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

3.3.1 スレッドによるアニメーションの実装

移動するキャラクタを表示するためには，一定の時間間隔で画面を書き換える必要がある．これまでの課題では，画面が作られた際に呼び出されるコールバックメソッド（surfaceCreated）やタッチイベントが発生した際に呼び出されるコールバックメソッド（onTouchEvent）は扱ってきたが，これらの呼び出しは該当するイベントが発生した際のみである．そのため，一定の時間間隔で画面を書き換えるためには別の仕組みを利用する必要がある．

ここではこれをスレッドを使って実現する方法について説明する．おおよその処理の流れは以下のようになる．

1. Surface が作成された際にスレッドを作成・起動する
2. スレッドの処理（run メソッド）の中で一定の時間間隔で画面を描画する処理を実行する
3. Surface が破棄された際にスレッドを停止する

3.3.2 Runnable インタフェースによるスレッドの利用

Java でスレッドを利用するには，Thread クラスを継承して利用する方法と Runnable インタフェースを実装して利用する方法があるが，ここでは後者について説明する．

これまでの課題と同様の流れで，SurfaceView クラスを継承した MySurfaceView クラスを実装する．スレッドを使うために，インタフェースとして，これまでの SurfaceHolder.Callback に加えて，Runnable インタフェースも実装することになる．

これまでの課題では surfaceCreate メソッドの中で直接，画面に描画を行っていたが，ここではかわりにスレッドの起動のみを行う．また，Surface が破棄された際にスレッドを停止する必要があるので，その処理を surfaceDestroyed メソッドに追加する．具体的には下記のようなコードになる．

最初に implements によって Runnable インタフェースを実装することを宣言し，それによって実装を要求される run メソッドを追加する．

```
public class MySurfaceView extends SurfaceView
    implements Runnable, SurfaceHolder.Callback {
    /*            ↑ Runnable インタフェースを実装することを宣言  */

    private Thread thread ;  /* Thread クラスのインスタンスを準備  */

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        thread = new Thread(this) ;  /* 新しくスレッドを作成 */
        thread.start() ;             /* スレッドをスタートさせる */
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        thread = null ;              /* スレッドを停止させる */
    }

    /* 省略  */

    @Override                        /* Runnable インタフェースに必要な run() メソッド
*/
    public void run() {

        /* run メソッド実装の詳細は後述  */

    }

}
```

3.3.3 run メソッドの実装

Runnable インタフェースに必要な run メソッドの実装のポイントは，一定の時間間隔で画面を描画する処理である．システムの現在時刻をミリ秒単位で取得するメソッドを利用して描画タイミングの管理を行う．

FPS には予め 1 秒間に描画するフレーム数を与える．FTIME には，FPS から計算したミリ秒単位の 1 フレームあたりの時間が計算される．FPS にあまり大きな値を設定してしまうと，画面の描画処理がおいつかなくなる．逆に小さな値に設定すると画面の描画にカクツキがあらわれる．

また，先の課題ではタッチイベントが発生した際に画面を描画するようにしていたが，この課題では画面を描画するのではなく，赤い円の動きを制御するようなコードが必要となる点に注意せよ．

```
    static final long FPS = 30 ;
    static final long FTIME = 1000 / FPS ;

    @Override
    public void run() {
        long loopC = 0 ;  // ループ数のカウンタ
        long wTime = 0 ;   // 次の描画までの待ち時間（ミリ秒）
        /*  必要に応じて描画に関する初期化をここに書く  */
        long sTime = System.currentTimeMillis() ;  // 開始時の現在時刻
        while (thread != null) {
            try {
                loopC++ ;
                drawCanvas() ;      /*  drawCanvas メソッドで画面を描画  */
                wTime = (loopC * FTIME) - (System.currentTimeMillis() - sTime) ;
                if (wTime > 0) {
                    Thread.sleep(wTime) ;
                }
            } catch (InterruptedException e) {
                /* 必要に応じて割り込み発生時の例外処理を追加 */
            }
        }
    }
```

以下に参考として，赤い円が画面の中央付近を時計回りに回転運動するような drawCanvas メソッドのプログラムコードを示す．

```
    private int degree = 0 ;

    private void drawCanvas() {
        Canvas c = sHolder.lockCanvas() ;
        c.drawColor(Color.YELLOW) ;
        Paint p = new Paint() ;
        p.setColor(Color.RED) ;
        float x = (float)Math.cos(Math.toRadians(degree));
        float y = (float)Math.sin(Math.toRadians(degree));
        c.drawCircle(c.getWidth() / 2 + x * 200.0f, c.getHeight() / 2 + y *
200.0f, 50.0f, p);
        degree += 6 ;
        degree = degree % 360;
        sHolder.unlockCanvasAndPost(c) ;
    }
```

4　モバイルプログラミングの基本 (4)

概要

Android 端末に搭載された各種センサの利用方法について学習する．センサを利用する際の処理の流れや加速度センサ等を使った具体的なプログラム作成に取り組む

keywords

センサマネージャー（SensorManager），加速度センサ，地磁気センサ，イベントリスナー

4.1　課題 4-1

Android 端末のセンサを利用したプログラムについて理解するために，加速度センサの値を取得して TextView に表示するプログラム（SensorSample01）を作成せよ．プロジェクトの設定は表 4.1 の通りとする．

表 4.1: 課題 4-1 の設定内容

項目	設定内容
Template	Empty Views Activity
Name	SensorSample01
Package	jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.sensorsample01
Save location	D:¥workspace¥SensorSample01
Language	Java
Minimum SDK	API 23: Android 6.0 (Marshmallow)

4.1.1　SensorSample01 の作成手順

ここでは，加速度センサから得られた値を文字列として TextView を使って表示するプログラムを作成する．SensorSample01 の作成手順は以下の通りである．ポイントとなる事項については，次節以降に説明しているのでそちらを参照すること．

1. 新規プロジェクト（SensorSample01）を作成
2. 作成される Empty Views Activity から ”Hello World!” の TextView を削除
3. MainActivity にセンサマネージャーとセンサを取得する処理を追加（詳細は後述）
4. MainActivity にセンサの値を監視するイベントリスナを追加（詳細は後述）
5. MainActivity にセンサの監視を開始する処理と終了する処理を追加（詳細は後述）
6. TextView をレイアウトに追加し，取得したセンサ値を表示する処理を作成（詳細は後述）

4.1.2　センサを利用する処理の流れ

Android 端末においてセンサを利用する場合の一般的な処理の流れは以下のようになる．

1. getSystemService メソッドを使って SensorManager クラスのインスタンスを取得

2. SensorManager クラスの getDefaultSensor メソッドを使って必要なセンサの Sensor オブジェクトを取得
3. センサを利用するクラスにセンサの値を監視するための SensorEventListener インタフェースを実装
   (a) センサ値が更新された際の処理を SensorEventListener インタフェースで規定された onSensorChanged メソッドとして作成
   (b) センサの精度が変更された際の処理を SensorEventListener インタフェースで規定された onAccuracyChanged メソッドとして作成
4. SensorManager クラスの registerListener メソッドでセンサの監視を開始
5. SensorManager クラスの unregisterListener メソッドでセンサの監視を終了

4.1.3 センサマネージャとセンサの取得

まずはセンサマネージャとセンサを取得する処理を追加する．

Android 端末に用意された各種サービスを取得するためのメソッドとして getSystemService メソッドが準備されている．これまで作成してきた MainActivity クラスは，AppCompatActivity クラスを継承しており，さらにその上の継承関係は以下のようになっている．getSystemService メソッドはこれらの親クラスの中で Context クラスの抽象メソッドとして定義され，ContextWrapper クラスで実装されているメソッドである．

```
java.lang.Object
   +-- android.content.Context
          +-- android.content.ContextWrapper
                 +-- android.view.ContextThemeWrapper
                        +-- android.app.Activity
                               +-- androidx.fragment.app.FragmentActivity
                                      +-- androidx.appcompat.app.AppCompatActivity
```

getSystemService メソッドの引数には，Context クラスで定義された定数を指定する．センサを管理する SensorManager クラスのインスタンスを取得するには，Context.SENSOR_SERVICE という定数を指定する．

さらに，得られた SensorManager クラスの getDefaultSensor メソッドを使って，実際に利用するセンサ（Sensor クラスのインスタンス）を取得する．getDefaultSensor に引数には必要とするセンサの種類を表 4.2 に示す定数で指定する．詳細については以下の URL を確認するとよい．

Sensor

`https://developer.android.com/reference/android/hardware/Sensor`

センサマネージャとセンサの取得はプログラムが起動した際に行えばよいので，MainActivity クラスの onCreate メソッドの中に記述すればよい．取得したセンサマネージャとセンサは他のメソッドからも利用するのでフィールド変数に格納しておく．プログラムコードとしては以下のようになる．

表 4.2: センサの種類（抜粋）

定数	センサの種類
Sensor.TYPE_ACCELEROMETER	加速度センサ
Sensor.TYPE_GRAVITY	重力センサ
Sensor.TYPE_GYROSCOPE	ジャイロセンサ
Sensor.TYPE_LIGHT	照度センサ
Sensor.TYPE_MAGNETIC_FIELD	地磁気センサ
Sensor.TYPE_PRESSURE	圧力センサ
Sensor.TYPE_PROXIMITY	近接センサ
Sensor.TYPE_TEMPERATURE	温度センサ

```
private SensorManager sManager ; // センサマネージャ
private Sensor accSensor ;  // 加速度センサ

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    sManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE) ;
    accSensor = sManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) ;
}
```

4.1.4 イベントリスナの登録

上記で準備したセンサから実際に値を取得するには，センサの値を監視するイベントリスナが必要となる．センサ値を監視するのは，SensorEventListener インタフェースを実装したクラスである．

SensorEventListener インタフェースを実装するクラスは，MainActivity クラスの場合と，その他のクラスを準備する場合があるが，ここでは MainActivity クラスに実装することとする．MainActivity クラスで実装することを宣言するために，クラス定義の冒頭を以下のように修正する．

```
public class MainActivity extends AppCompatActivity implements SensorEventListener
{
    // 以下省略
```

SensorEventListener インタフェースでは表 4.3 に示す 2 つのメソッドをオーバーライドする必要がある．この時点で，表 4.3 に示したメソッドを MainActivity クラスで実装していないために，クラスの先頭にエラーが表示されているはずである．エラー箇所の上の ”電球” マークをクリックすると ”Implement methods” という項目が表示されるので，これを選択する．表に示した 2 つのメソッドを選択し，OK ボタンを押すとプログラムコードに 2 つのメソッドが追加される．

ここでは，センサの精度の変更については無視をし，センサ値が変更された場合の処理だけ追加することとする．

表 4.3: 実装が必要なメソッド

メソッド名	説明
onAccuracyChanged	センサの精度が変更された場合に呼ばれるメソッド
onSensorChanged	センサ値が更新された場合に呼ばれるメソッド

onSensorChanged メソッドは，センサ値が変更された場合に，センサイベント（SensorEvent）クラスのインスタンスを引数として渡される．このインスタンスには表 4.4 に示すプロパティが格納されている．詳細については以下の URL を確認するとよい．

SensorEvent

`https://developer.android.com/reference/android/hardware/SensorEvent`

表 4.4: センサイベントのプロパティ

プロパティ	説明
int accuracy	センサの精度
Sensor sensor	イベントを起こした Sensor オブジェクト
long timestamp	イベントが起こった時刻
final float[] values	センサ値を格納した配列

sensor に格納されているのはイベントを起こした Sensor オブジェクトである．values に格納されるセンサ値は，センサの種類によってその内容が異なる．加速度センサを利用する場合，values[0]，values[1]，values[2] がそれぞれ X 軸，Y 軸，Z 軸方向の加速度となっている．ここでは，センサの種類が加速度センサ（Sensor.TYPE_ACCELEROMETER）であることを確認した上で，各軸のセンサ値をフィールド変数に格納することとする．Android 端末と各軸の関係は図 4.1 を参照せよ．

プログラムコードは以下のようになる．

```
private float x, y, z ;

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        x = event.values[0] ;
        y = event.values[1] ;
        z = event.values[2] ;
    }
}
```

4.1.5 センサ監視処理の開始と終了

Android 端末のセンサを利用するには，センサの監視を開始するタイミングと終了するタイミングを決める必要がある．ここではアクティビティが表示された際にセンサの監視を開始し，アクティビティが非表

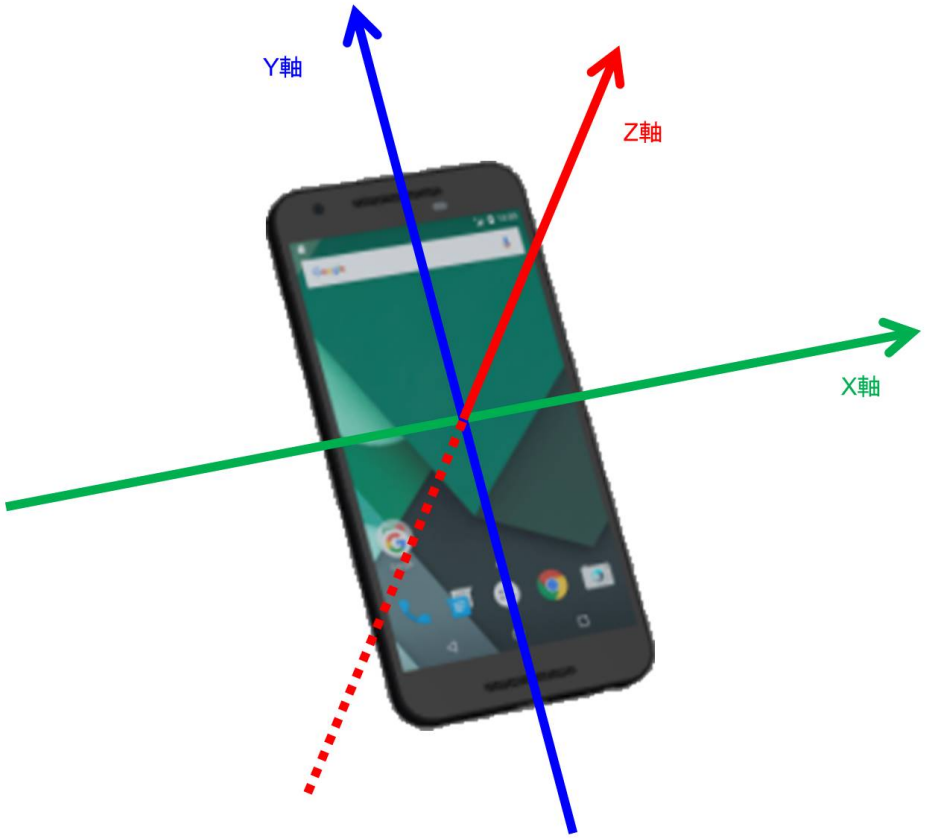

図 4.1: Android 端末と各軸の関係

示になったらセンサの監視を終了することとする．前者については，onResume メソッドを利用し，後者については onPause メソッドを利用する．これらのメソッドがどのようなタイミングで呼び出されるかについては，以下の URL にあるアクティビティについての説明を読むとよい．

アクティビティ

`https://developer.android.com/guide/components/activities`

センサの監視は onResume メソッドの中で行う．センサの監視を開始するには，SensorManager クラスの registerListener メソッドを使って，呼び出しを受けるリスナーを指定する．registerListener メソッドの引数は表 4.5 を参照せよ．

表 4.5: registerListener メソッド

引数	説明
SensorEventListener listener	SensorEventListener インタフェースを実装したクラスのインスタンス
Sensor sensor	監視対象のセンサオブジェクト
int samplingPeriodUs	センサ値の更新頻度

ここでは，センサイベントリスナーを MainActivity クラス自身に実装したので，登録するリスナは自分自身になる．監視対象は，onCreate メソッドで取得した加速度センサである．センサ値の更新頻度については，表 4.6 に示す 4 つの定数が指定でき，用途に応じて使い分ける．リアルタイム性が重視されるゲーム

などでは更新頻度を高くする必要があるが，センサ値が更新されるたびに onSensorChanged メソッドが呼び出されるため，onSensorChanged メソッドでの処理内容が重い（＝処理に時間がかかる）場合は，イベントが累積して遅れが生じる恐れがあるため，更新頻度の設定と onSensorChanged メソッド内部での処理内容についてはよく検討する必要がある．

表 4.6: センサ値の更新頻度

引数	説明
SENSOR_DELAY_FASTEST	できるだけ速くセンサ値を取得
SENSOR_DELAY_GAME	ゲームに適した更新頻度
SENSOR_DELAY_NORMAL	スクリーン方向の変更に適した更新頻度
SENSOR_DELAY_UI	ユーザインタフェースに適した更新頻度

一方，センサの監視を終了する処理は onPause メソッドの中で行う．SensorManager クラスの unregisterListener メソッドを使って，センサを監視するリスナを登録解除すれば監視が終了する．リスナは MainActivity 自身のため，onPuase メソッドの中で，これを引数にして unregisterListener メソッドを呼び出せばよい．ここまでのプログラムコードは以下のようになる．

```
@Override
protected void onResume() {
    super.onResume();
    sManager.registerListener(this, accSensor, SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
    super.onPause();
    sManager.unregisterListener(this);
}
```

4.1.6 センサ値の表示

レイアウトエディタで TextView を追加し，これに対して setText メソッドを使って取得したセンサ値を文字列としてセットすればよい．センサ値のセットは onSensorChanged メソッド内部で行うとよい．なお，表示される文字列が小さい場合は，Common Attributes の textSize 属性を変更するとよい．

具体的には，以下のような手順である．

1. レイアウトエディタのパレットから TextView を選択し画面に配置
2. 配置した TextView の id を ”sensorStatusView” に設定（一意な文字列ならなんでもかまわない）
3. MainActivity に TextView クラス（android.widget.TextView）のインスタンスを保持する変数 sensorStatus を追加
4. findViewById メソッドを利用して id が ”sensorStatusView” である TextView を探し出し，sensorStatus に格納
5. TextView クラスの setText メソッドを使ってセンサから取得したセンサ値を文字列としてセット

なお，加速度を利用する場合，Android 端末の向きがかわって画面の方向が変更されると煩雑になる．そこで，画面を縦方向に固定する setRequestedOrientation メソッドを利用するとよい．onCreate メソッドの内部で setRequestedOrientation メソッドを以下のように呼び出すと画面が縦方向に固定される．

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

画面を横方向に固定する場合には，ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE を，自動回転にする場合には，ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED を設定すればよい．

ここまでをまとめると，onCreate メソッドは以下のようなプログラムコードになる．

```
private SensorManager sManager;
private Sensor accSensor;

private TextView sensorStatus;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    sensorStatus = (TextView)findViewById(R.id.sensorStatusView);

    sManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
    accSensor = sManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}
```

4.1.7 エミュレータでのセンサ値の変更

エミュレータで動作を確認する場合，エミュレータ端末の加速度値を変更する必要がある．エミュレータの拡張コントロールパネルを用いると様々なハードウェアの値をエミュレーションすることができる．図 4.2 に示すように，エミュレータ画面の横にあるボタンの内，一番下にある「···」ボタンを押すと，”Extended controls” という別のウィンドウが表示される（図の右側）．このウィンドウの ”Virtual sensors” というボタンを押すとウィンドウ内に端末が表示される．この端末をマウスでドラッグすると加速度センサの値を変更することができる．図 4.3 および図 4.4 に課題 4-1 の実行例を示す．

4.2 課題 4-2（オプション課題）

Android 端末のセンサを利用したアプリとして，加速度センサを用いた玉転がしアプリを作成せよ．課題 4-1 を参考に，得られた加速度の値を使って画面上に描画した玉が転がる（移動する）ようにせよ．このアプリは以下の仕様を満たすようにすること．画面の描画には 3 章で学んだ SurfaceView を利用するとよい．

- 玉は半径 50 の円とする
- 玉は加速度センサの値にしたがって画面内を移動する
- 玉は画面の端に到達すると跳ね返る

図 4.2: エミュレータの拡張コントロールの表示手順

なお，移動量の計算については以下を参考にするとよい．
加速度 α と速度 v の関係は，初速度を v_0 とし，経過時間を t 秒とすると

$$v = v_0 + \alpha t$$

で与えられる．また，初速度 v_0，加速度 α の物体が t 秒の間に移動する距離 d は

$$d = v_0 t + \frac{1}{2}\alpha t^2$$

である．

ここで t は前回センサ値を取得した時刻と今回センサ値を取得した時刻の間の経過時間と考えればよい．現在時刻の取得には，System.currentTimeMillis メソッドを利用するとよい[18]．また，加速度 α は加速度センサから得られる値を利用することとする．初速度 v_0 については，前回センサ値を取得した際に速度がいくらになったかを計算すればよい．

これらは X 軸と Y 軸で独立に計算することができるため，X 軸，Y 軸それぞれ毎に変数を準備して計算すればよい．移動距離はメートル単位，時間は秒単位である点に注意すること．移動距離は Android 端末の画面上で適切な移動量にするために係数をかけるとよい．

4.3 課題 4-3（オプション課題）

課題 4-3 については選択課題とする．以下のいずれかの課題を選択せよ．なお，余力のあるものが両方の課題に取り組むことは妨げない．

4.3.1 玉転がしゲームの作成

課題 4-2 を発展させてゲームを作成せよ．以下のようなアイデアを組み込んでゲームとして成立させること．なお，すべてのアイデアを採用する必要はなく，オリジナルなアイデアを入れることも妨げない．

- ゴール地点を設定し，初期位置からゴールまでの到達時間を競う
- 動き回る敵キャラクタを作成し，玉が敵に接触したペナルティを与える
- 障害物を設置する

[18] System.currentTimeMillis メソッドが返すのはミリ秒である点に注意せよ．

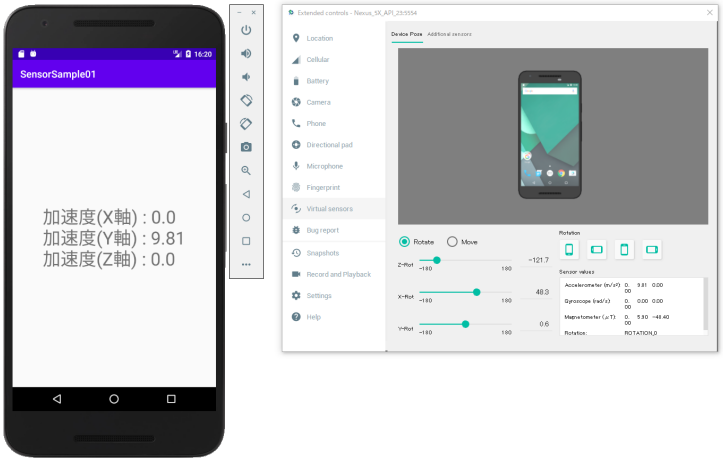

図 4.3: エミュレータ端末が垂直の状態

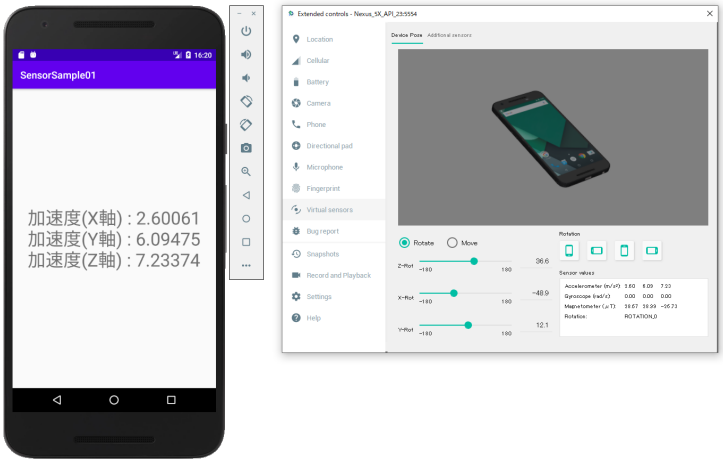

図 4.4: エミュレータを斜めに傾けた状態

- ステージの概念を導入し，ステージが進むにつれて難易度が上がるようにする
- 壁を設置し，迷路のようにする
- ワープポイントを作る
- スコアの要素を導入する

4.3.2 コンパスアプリの作成

加速度センサと地磁気センサを組み合わせてコンパスアプリを作成せよ．図 4.5 にエミュレータでの実行例を示す．この例では，方位角を示す数字と磁針しか描画されていないが，デザインはこの通りである必要はない．また，この他のセンサ情報なども組み合わせてさらに様々なセンサ情報が表示されるアプリに拡張してもよい．

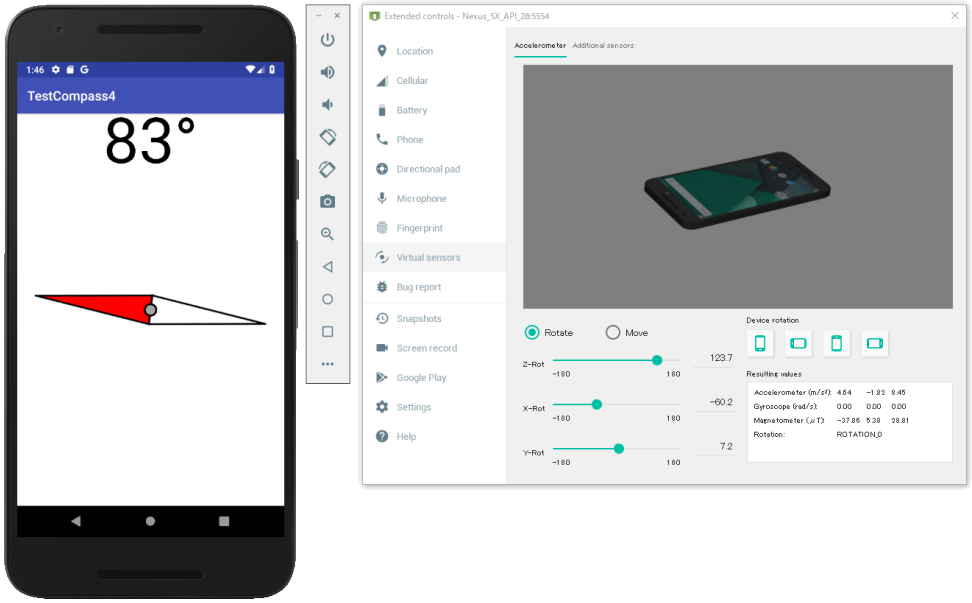

図 4.5: コンパスアプリの実行例

A　付録: 補足事項

ここでは，Android のプログラムを開発する上で参考になる情報について説明する．

A.1　仮想デバイスの作成と管理

Android の開発では，実機ではなくエミュレータを使った動作確認をすることができる．エミュレータでシミュレートしたい端末の管理には Virtual Device Manager [19] を利用する．

Virtual Device Manager は，Tools メニューから Device Manager をたどることで起動できる．エミュレータが未定義の状態では，画面の右側に図 A.1 のような画面が表示される．

左上の Create Device ボタンを押すと，ハードウェアの選択画面（Select Hardware）（図 A.2）が表示される．左側の Category ではハードウェアの種類を選択できる．中央の表では，Google が提供しているそのカテゴリーの端末が表示されており，解像度などのスペックが確認できる．新規のエミュレータとしては，この中から選ぶこともできるし，新たなスペックの端末を定義することも可能である．ここでは Nexus 5X を選択することとする．

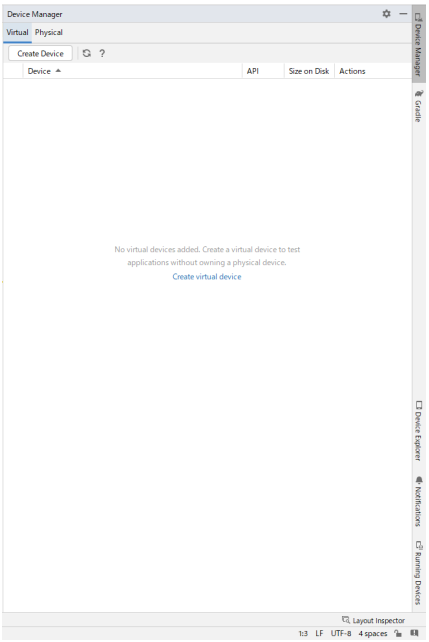

図 A.1: Device Manager の初期画面

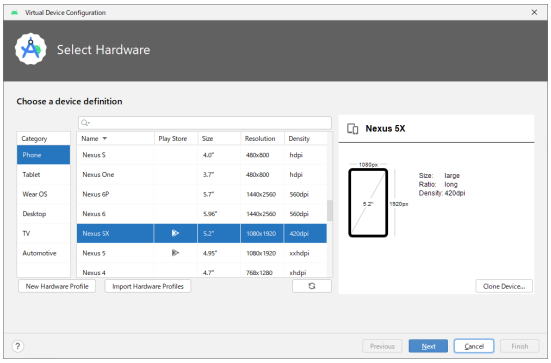

図 A.2: ハードウェアの選択画面

Nexus 5X を選択すると次にシステムイメージの選択画面（System Image）が表示される（図 A.3）．それぞれのシステムイメージは，API レベルや Target が異なっているため，エミュレータのスペックとして必要なシステムイメージを選択すればよい．”Recommended” タブには Google が推奨するシステムイメージが，”x86 Images” タブには x86 系の CPU 向けのシステムイメージが，”Other Images” タブにはその他の CPU 向けのシステムイメージがリストアップされている．ここでは，実験向けに準備した Android 端末（Zenfone 3 Max ZC553KL）にあわせて API レベル 23（x86_64）を選択することとする．自分が所有する Android 端末がある場合，それにあわせてもよい．

なお，Release Name 部分にダウンロードのアイコンが記載されているものは，先にダウンロードアイコンをクリックしてシステムイメージをダウンロードする（図 A.4）．License Agreement が表示された場合

[19] 以前は，AVD (Android Virtual Device) Manager の名称が使われていた

は Accept を選択してダウンロードを続行する．その後，作成した AVD の名前などを設定すれば新規エミュレータの設定は完了する（図 A.5 参照）．

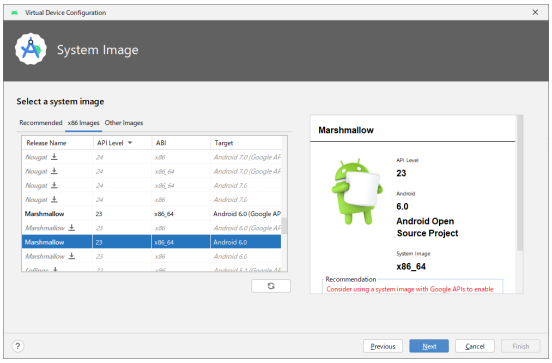

図 A.3: システムイメージの選択画面

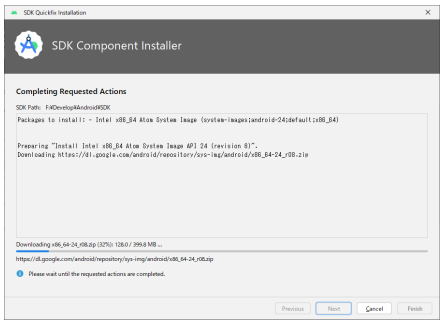

図 A.4: システムイメージのダウンロード

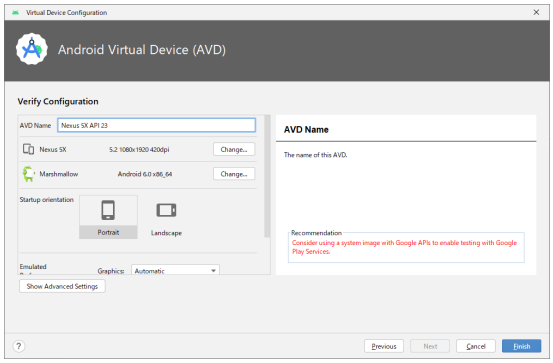

図 A.5: AVD の設定確認

Zenfone 3 Max ZC553KL のエミュレータを定義するのであれば，ハードウェアの選択画面で New Hardware Profile を押し，新たなプロファイルとして以下のように設定するとよい．

表 A.1: Zenfone 3 Max ZC553KL の設定

項目	設定内容
Device Name	Zenfone 3 MAX ZC553KL
Device Type	Phone/Tablet
Screen size	5.5
Resolution	1080 x 1920

詳細については以下の URL を確認すること．なお，1 つのエミュレータ定義でハードディスクの容量を 2〜3GB 分（エミュレートする端末のメモリ容量に比例し，場合によってはさらに大きくなることもある）消費するため，実験で作成する定義は 1 つにすること．自宅などで Android Studio を利用する場合も，ハードディスクの容量には注意するとよい．

仮想端末の作成と管理

`https://developer.android.com/studio/run/managing-avds?hl=ja`

A.2　オートコンプリートについて

Android Studio のコーディングでは，図 A.6 に示すように初めて利用するクラスを入力した際に赤い文字で表示されることがある．赤文字で示されているのはエラーが出ている状態であり，このままではビルドができない．この図の例では，Paint というクラス名から Android Studio が推測して，android.graphics.Paint クラスではないかとのメッセージを出力している．このような場合，メッセージが正しいのであれば，Alt + Enter キーを押すことによって自動で補完が為され，android.graphics.Paint クラスがインポートされる．

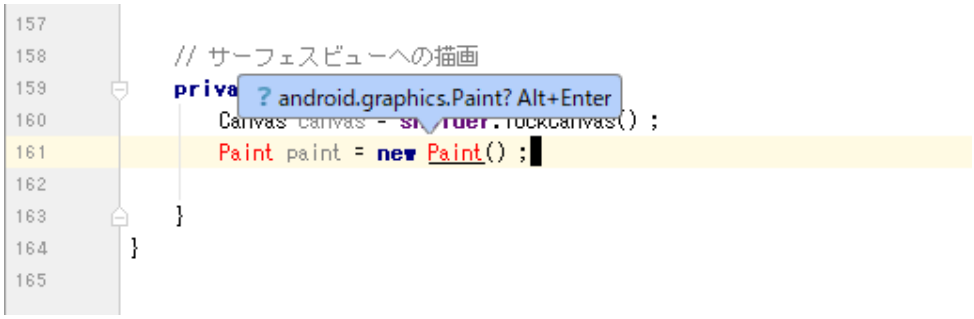

図 A.6: オートコンプリートの例

A.3　Windows におけるパフォーマンスの改善に関して

一部のウイルス対策ソフトウェアは Android Studio のビルドプロセスの妨げとなり，ビルドの実行速度を大幅に低下させることがあります．詳細や対策については下記の URL を確認してください．なお，設定を変更する際には自己責任で行ってください．

Windows で Android Studio のパフォーマンスを最適化する

`https://developer.android.com/studio/intro/studio-config?utm_source=android-studio#optimize-studio-windows`

A.4　プロジェクトの複製

既に作成したプロジェクトを元に，改良版のプロジェクトを作成したい場合がある．このような場合，プロジェクトのフォルダをそのままコピー＆リネームするだけでは，正しく動作しない．以下では，プロジェクトを複製して名前（パッケージ名なども含む）を変更する手順を説明する．

1. Android Studio をいったん終了する．
2. Windows のファイルエクスプローラーなどでプロジェクトのルートフォルダをコピーし，名前を変更する．
3. コピーしたルートフォルダ内の .idea フォルダと拡張子が .iml のファイル（存在する場合）を削除する．
4. ルートフォルダ内の settings.gradle ファイルをエディタで開き，rootProject.name を新しい名前に変更する．
5. Android Studio を起動し，先ほどコピーした新しいプロジェクトをオープンする．

6. Android Studio の Refactor 機能でパッケージ名を書き換える．プロジェクトツールウィンドウ（プロジェクトツリー）のパッケージ名（jp.ac.ritsumei.ise.phy.exp2.isXXXXXX.helloworld など）の上で右クリックをし，Refactor から Rename を選択する．Warning が出るが気にせずに「Rename package」ボタンを押し，Rename したい名前を入力した上で,「Refactor」ボタンを押す．画面左下のウィンドウに変換対象のファイル一覧が表示されるので「Do Refactor」ボタンを押して実行する．

7. プロジェクトツールウィンドウの build.gradle(Module: app) を開き，defaultConfig の中の applicationId を変更後の名前に修正する．

8. 右上に表示される「Sync Now」を押して gradle を再ビルドする．

9. アプリの表示名を変更する場合には，res/values/strings.xml の中の app_name を変更する．

B　付録: トラブルシューティング

ここでは，よく発生するエラー／トラブルについての対処法を説明する．必要に応じて確認すること．

B.1　原因不明のエラー

プロジェクト内のファイルについて何らかの不整合が生じてエラーが発生する場合がある．画面最下部のステータス表示部において ”Gradle Build Running” というメッセージが表示されている場合は，何らかの処理が進んでいる状態なのでそのまま何もせずにしばらく様子を見ることを推奨する．処理途中で別の作業を行うとファイルの不整合が生じる可能性がある．

また，エラーの原因が不明の場合，”Build” メニューから ”Clean Project” を実行し，まずはプロジェクトの初期化を行う．その上で，同じメニューの ”Rebuild Project” を実行して，プロジェクトの再構築を行ってみるとよい．ファイルの不整合が原因の場合，一連の作業によってエラーが解消される場合がある．

なお，ビルドにあたっては大量ファイルを読み書きするため，プロジェクトはできるだけ高速なディスク上に保存しておくことを推奨する．

B.2　レイアウトエディタが表示されない場合

Android Studio が利用するツールのバージョンの不整合などが理由で，レイアウトエディタが正常に表示されず図 B.1 のようなエラーが発生する場合がある[20]．画面下部のツールウィンドウにエラーが表示されており，バージョンが 26.1.0 と 27.1.1 で不整合を起こしていることがわかる．

図 B.1: エラー表示

```
Error:Execution failed for task ':app:preDebugAndroidTestBuild'.
> Conflict with dependency 'com.android.support:support-annotations' in project
':app'. Resolved versions for app (26.1.0) and test app (27.1.1) differ. See
https://d.android.com/r/tools/test-apk-dependency-conflicts.html for details.
```

このエラーはビルド用のスクリプトの中での不整合によって発生しているため，プロジェクトツールウィンドウから，Gradle Scripts → build.gradle (Module: app) を選択し，エディタウィンドウに表示する．その上で図 B.2 に示した 3 箇所について修正する．修正後，画面上部にある「Sync now」を押すと，修正結果が反映される．

B.3　複数の Android Studio での開発

Android Studio が保存するプロジェクトには，Android SDK（Android のソフトウェア開発環境）のフォルダパスの情報などが記録されている．そのため，環境の異なる Android Studio でプロジェクトを開く際

[20] 詳細は確認していないが，このエラーは本実験用にセットアップした Android Studio Version 3.0.1 特有のエラーだと思われる．

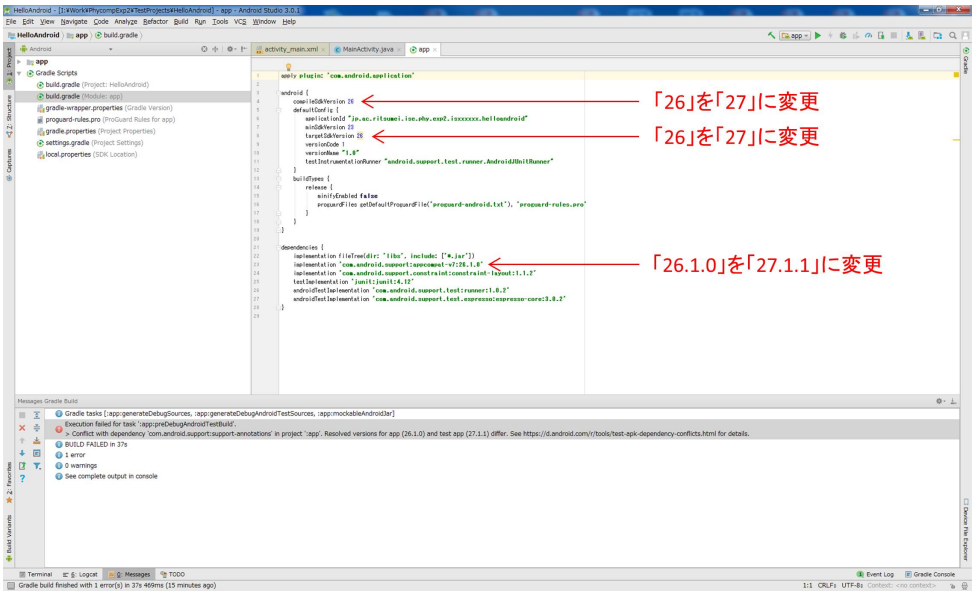

図 B.2: build.gradle (Module: app) の内容と修正部分

には，図 B.3 のようなエラーが表示される場合がある．メッセージにあるように修正して処理を進めてくれるため，気にせずに「OK」ボタンを押せばよい．

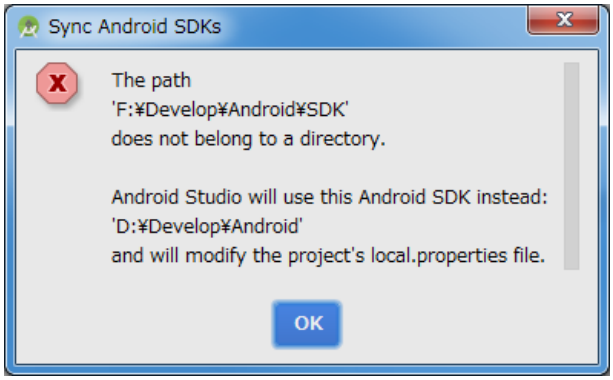

図 B.3: Android SDK のフォルダパスのエラー例

B.4　モジュール等の追加インストール

Android Studio では，ビルドなどの際に，必要なモジュール等が不足している場合には，ダウンロードとインストールを要求する場合がある．多くの場合，表示されるメッセージにしたがって追加のモジュール等をインストールすれば，そのまま正常にビルドを続けることができる．

B.5　ADB の不具合に起因するエミュレータ起動失敗

ADB（Android Debug Bridge）に不具合が発生したために，エミュレータの起動が失敗する場合があります．エラーとしては，例えば以下のようなものが表示されます．

```
Adb connection Error:Connection reset by peer
Cannot reach ADB server, attempting to reconnect.
```

この現象は以下のような設定変更を行うと解消されることがあるので試してみてください.

1. File メニューの Settings を開く
2. Build, Execution, Deployment の項目中の Debugger の Android Debug Bridge (adb) の設定において，”Enable adb mDNS for wireless debugging” の項目のチェックをはずす

C　参考文献

本実験資料は，以下の文献を参考にしている．

- はじめての Android プログラミング（改訂版），金田浩明，ソフトバンククリエイティブ，ISBN: 4-7973-9166-4
- Android Studio ではじめる Android プログラミング入門，掌田津耶乃，秀和システム，ISBN: 4-7980-4698-3
- Java からはじめよう Android プログラミング，大津真，インプレス，ISBN: 4-8443-3877-2