Lab3
物件控制與監聽事件

本節目的：

- 透過程式碼使用 Xml 畫面元件。
- 設定監聽事件回應使用者操作。

3.1 元件與監聽事件

3.1.1 取得畫面元件

前面章節我們完成了 Layout（Xml）的設計，而實際要去使用這些元件，我們需要把畫面中的元件與主程式（Kotlin）連結。一開始，我們會在 Activity 來編寫我們的程式碼。

在建立好的專案中，會配置一個程式檔：MainActivity 以及一個布局配置：activity_main.xml，如圖 3-1 所示。

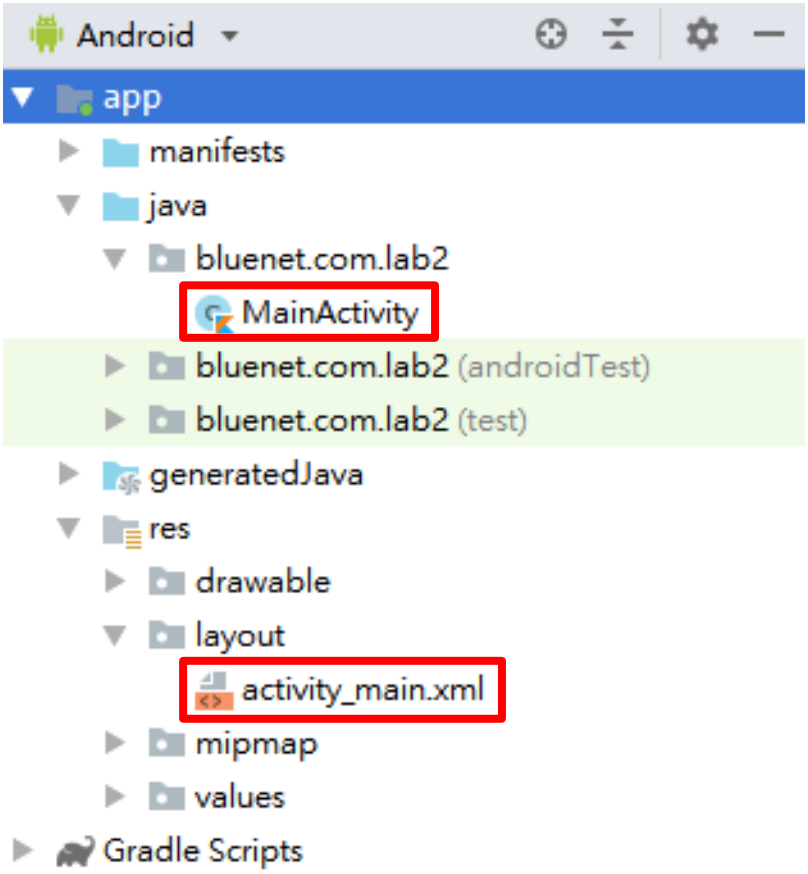

圖 3-1 Android 專案架構

首先 activity_main.xml 延續使用第二章節的畫面設計，配置結果如圖 3-2 所示下。

圖 3-2 猜拳遊戲預覽畫面（左）與布局元件樹（右）

圖 3-2 中的元件如果希望能在程式中使用，則在 xml 上需要設定一組對應的 id，id 的用途是為該元件貼上一個識別標籤，在 Android 中必須透過 id 才能找到對應的元件。

如下方的程式碼片段中，EditText 的 id 為 gamer、TextView 的 id 為 status、RadioGroup 的 id 為 radioGroup…以此類推。

```
<EditText
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:id="@+id/gamer" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="請輸入名字以開始遊戲"
    android:id="@+id/status"
    android:textColor="#000000"
    android:textSize="22sp" />

<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:orientation="horizontal"
    android:id="@+id/radioGroup">
```

完成了畫面設計，接著我們便可以在 MainActivity 中去取得 activity_main.xml 的元件。

打開 MainActivity 我們可以看到以下程式碼：

```
package bluenet.com.lab2

import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

由於 Android 主程式與 Xml 畫面本身是不同的程式碼，如果要在主程式中要顯示畫面，就必須要明確指定要使用哪頁 Xml 畫面。setContentView()方法會指定這個 Activity 所要使用的 Xml 畫面，這邊系統已經事先指定了（R.layout.activity_main），也就是透過 R 類別指定了 activity_main.xml。

在 Kotlin 中，當我們要取得 Xml 內的元件時有兩種方法，第一種是使用 findViewById()將元件綁定，如以下程式碼所示：

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val ed_name = findViewById<EditText>(R.id.ed_name)
        val tv_text = findViewById<TextView>(R.id.tv_text)
        val radioGroup =
                    findViewById<RadioGroup>(R.id.radioGroup)
        val btn_scissor =
                findViewById<RadioButton>(R.id.btn_scissor)
    }
}
```

findViewById()方法會從 Xml 畫面中找到對應 id 的元件（R.id.xxx），並回傳元件到程式碼中，如上面 xml 程式碼的 ed_name、tv_text、radioGroup、btn_scissor 等。

```
<EditText
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:id="@+id/ed_name" />
```

```
val ed_name = findViewById<EditText>(R.id.ed_name)
```

findViewById()回傳的結果為 View 型態，View 型態是畫面元件的原型類別，所有畫面元件都屬於 View 型態，如 EditText、TextView、Button，如圖 3-3 所示。

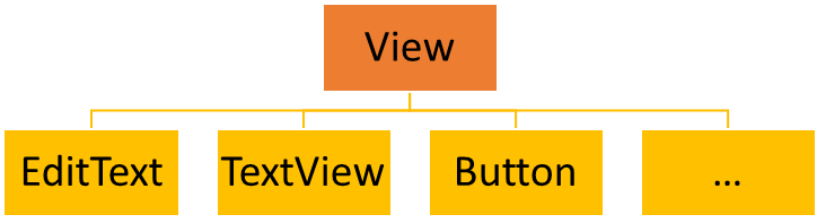

圖 3-3 View 成員

回傳 View 型態，也就表示 findViewById()只能知道他取到了是一個畫面元件，但是不知道是什麼元件，因此只能回傳說找到了一個「畫面元件」。所以在程式碼當中，我們需要明確的告知這是 EditText 元件，主要就是將 findViewById()回傳的 View 型態，並藉由<EditText>的型態語法，轉型成 EditText 元件。

編寫完 findViewById()後，應該會發現 EditText、TextView、Button 都是圖 3-4 的 EditText 會顯示 **Unresolved...**，因為要使用這些元件時，程式碼中還必須匯入對應的 android 的 widget 類別，我們必須在程式碼上加入 Import。

```
val ed_name = findViewById<EditText>(R.id.ed_name)
val tv_text = findViewById<                    v_text)
val radioGroup = findViewById<RadioGroup>(R.id.radioGroup)
val btn_scissor = findViewById<RadioButton>(R.id.btn_scissor)
```

Unresolved reference: EditText

圖 3-4 找不到元件的 class

Import 可以手動加入，也可以讓 Android Studio 自動產生，用鼠標點擊有問題的元件，如圖 3-5 所示。然後按下 Alt+Enter 後就可以自動匯入，如圖 3-6 所示，或是會彈出一個小列表，這時選擇需要的 class 自動匯入。

? android.widget.EditText? Alt+Enter

```
val ed_name = findViewById<EditText>(R.id.ed_name)
val tv_text = findViewById<TextView>(R.id.tv_text)
val radioGroup = findViewById<RadioGroup>(R.id.radioGroup)
val btn_scissor = findViewById<RadioButton>(R.id.btn_scissor)
```

圖 3-5 匯入需要的 class

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.EditText

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val ed_name :EditText! = findViewById<EditText>(R.id.ed_name)
        val tv_text = findViewById<TextView>(R.id.tv_text)
        val radioGroup = findViewById<RadioGroup>(R.id.radioGroup)
        val btn_scissor = findViewById<RadioButton>(R.id.btn_scissor)
    }
}
```

圖 3-6 Import EditText 元件後就不再顯示錯誤

第二種方式是透過添加對 Xml 的依賴，不需要額外的綁定（findView）就可以直接使用 Xml 的元件。

```
1   package bluenet.com.lab2
2
3   import android.support.v7.app.AppCompatActivity
4   import android.os.Bundle
5
6   class MainActivity : AppCompatActivity() {
7       override fun onCreate(savedInstanceState: Bundle?) {
8           super.onCreate(savedInstanceState)
9           setContentView(R.layout.activity_main)
    kotlinx.android.synthetic.main.activity_main.ed_name? Alt+Enter
11          ed_name.setText("王小明")
12      }
13  }
```

圖 3-7 匯入畫面布局的 xml

3.1.2 事件處理

使用者在操作 APP 的過程中，會對於畫面物件產生事件，例如點擊、輸入、觸碰等，而程式中透過對物件設置監聽器去等待事件被觸發。如點擊某個元件，程式碼就會被啟動去做特定的工作，這種回饋動作就是透過監聽器來實現。

Android 有提供內建的監聽器給元件做使用。不同類別的元件，能使用的監聽器也不同。而監聽器的命名規則通常為 On_XXX_Listener，如 OnClickListener，每個監聽器內部會預定幾種方法，當觸發時會執行對應的方法。

而監聽器種類繁多，以下則會就幾個常見的監聽器做說明：

- OnClickListener：

```
btn_mora.setOnClickListener {
}
```

OnClickListener 是最常使用的監聽器，他可以在使用者對元件做出**點擊（按下後立刻放開）做出回應**。該元件使用 setOnClickListener()的方法將該監聽器做連結，之後就可以等待該元件觸發按下的動作。

- OnLongClickListener：

```
btn_mora.setOnLongClickListener {
    false
}
```

OnLongClickListener 可以在使用者對元件做出**長按(持續按住不放開超過 1 秒）做出回應**，當事件成立時會觸發，元件使用 setOnLongClickListener ()的方法將該監聽器做連結。

- OnTouchListener：

```
btn_mora.setOnTouchListener { v, event ->
    false
}
```

OnTouchListener 提供使用者對元件做出的**觸摸事件(按下、移動手勢、離開...等）做出回應**，當事件成立時會觸發 setOnTouchListener，參數 event 為觸發的事件。

- OnCheckedChangeListener：(RadioButton 必須是 RadioGroup 的子層級)

```
radioGroup.setOnCheckedChangeListener { group, chechedId ->
}
```

OnCheckedChangeListener 是 RadioGroup 專用的監聽事件，它的用途在於監聽 RadioGroup 子層級的 RadioButton 被按下時，會觸發 onCheckedChanged()。

onCheckedChanged()的第二個參數，會回傳剛才按下的元件 Id，或是我們也可以用 radioGroup.getCheckedRadioButtonId()的方法取得元件 Id。

說明

這邊補充說明，在選擇監聽器時，Android Studio 會在我們打字的同時用表單列出可用的語句，要設定監聽器時可以打上 setOn…就可以篩選出下圖 3-8 中能用的監聽事件。

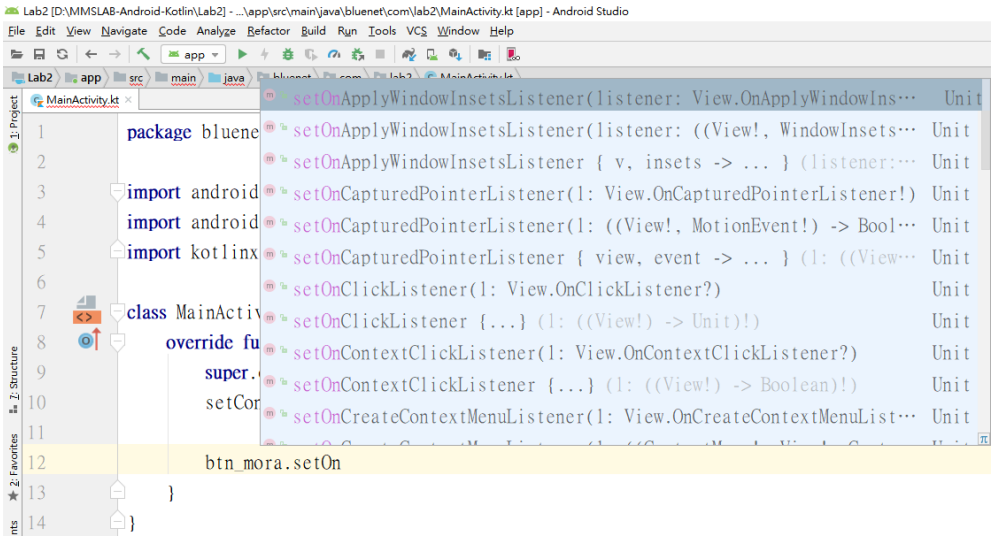

圖 3-8 加入按鈕監聽事件

3.2 猜拳遊戲程式設計

- 延續前一個 Lab 的結果，實做一個完整的猜拳功能，如圖 3-9 所示。

圖 3-9 猜拳遊戲實機畫面

- 選擇剪刀石頭布三個選擇之一，並按下開始遊戲，系統會以亂數回應勝負，下方會顯示出輸入的名稱、勝利者與雙方所出的拳，如圖 3-10 所示。

圖 3-10 玩家勝利（左）、平手（中）、電腦勝利（右）

3.2.1 加入監聽與判斷式

開啟 MainActivity，並對 Button 設定 OnClickListener 的監聽事件，在這裡我們加入姓名的判斷與剪刀石頭布的判斷式。

```
package bluenet.com.lab2

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btn_mora.setOnClickListener {
            //判斷字串是否是空白來要求輸入姓名
            if(ed_name.length()<1)
                tv_text.text = "請輸入玩家姓名"
            else{
                //顯示玩家姓名與我方出拳
                tv_name.text = "名字\n${ed_name.text}"
                tv_mmora.text = "我方出拳\n${if(btn_scissor.isChecked) "剪刀"
                    else if(btn_stone.isChecked) "石頭" else "布"}"
                //Random()產生介於 0~1 之間不含 1 的亂數，乘 3 產生 0~2 當作電腦的出拳
                val computer = (Math.random()*3).toInt()
                tv_cmora.text = "電腦出拳\n${if(computer==0) "剪刀"
                    else if(computer==1) "石頭" else "布"}"
                //用三個判斷式判斷並顯示猜拳結果
                when{
                    btn_scissor.isChecked && computer==2 ||
                    btn_stone.isChecked && computer==0 ||
                            btn_paper.isChecked && computer==1 ->{
                        tv_winner.text = "勝利者\n${ed_name.text}"
                        tv_text.text = "恭喜你獲勝了！！！"
                    }
                    btn_scissor.isChecked && computer==1 ||
                    btn_stone.isChecked && computer==2 ||
                            btn_paper.isChecked && computer==0 ->{
                        tv_winner.text = "勝利者\n 電腦"
                        tv_text.text = "可惜，電腦獲勝了！"
                    }
                    else ->{
                        tv_winner.text = "勝利者\n 平手"
                        tv_text.text = "平局，請再試一次！"
                    }
                }
            }
        }
    }
}
```

說明

這邊補充 TextView 可透過程式碼直接修改 text 屬性，而 EditText 必須使用 setText()才能設定字串內容。為了確保資料類型一定為字串，從 TextView 與 EditText 取得 text 屬性時，建議要再額外加上 toString()的轉型取得字串型態的資料。