Estudo e experimentação do Android

Acesso páginas Web

Acesso a Web Services (JSON)

Acesso a Web

- Para aceder a uma página Web
  - Incluir a permissão de acesso à internet no ficheiro de manifesto

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

  - Usar um objecto cliente para o protocolo http
    - `HttpURLConnection` ou `HttpsURLConnection`
      - Para pedidos em *clear text*, a partir da API 28 é necessário adicionar à estrutura *application* do ficheiro de manifesto o atributo `android:usesCleartextTraffic="true"`
  - O pedido deve ser realizado no contexto de uma *thread* secundária

getData

```
const val LIMIT = 4096
...
private fun getData(strUrl: String?): String? {
    val resp = StringBuilder()
    var conn: HttpURLConnection? = null
    try {
        val url = URL(strUrl)
        conn = url.openConnection() as HttpURLConnection
        conn.readTimeout = 10000
        conn.connectTimeout = 15000
        conn.requestMethod = "GET"
        conn.doInput = true
        conn.connect()
        val code: Int = conn.responseCode
        if (code == HttpURLConnection.HTTP_OK /*200*/) {
            var count = 0
            conn.inputStream.bufferedReader().forEachLine {
                count++
                if (count >= LIMIT)
                    return@forEachLine
                resp.append(it + "\n")
            }
        } else {
            resp.append("Error: $code")
        }
    } catch (_: Exception) {
        return null
    } finally {
        conn?.inputStream?.close()
        conn?.disconnect()
    }
    return resp.toString()
}
```

getDataAsync

- Chamar a função getData no contexto de uma *thread* secundária

```
fun getDataAsync(strURL: String, result: MutableLiveData<String?>) {
    thread {
        val strContent = getData(strURL)
        result.postValue(strContent)
    }
}
```

- Exemplo de utilização

```
val webContent : MutableLiveData<String?> = MutableLiveData()
webContent.observe(this) {
   ...
}
getDataAsync("https://www.isec.pt",webContent)
```

Verificar estado da rede

```
fun verifyNetworkState(context: Context): Boolean {
    val connMgr = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager?
    val networkInfo = connMgr?.activeNetworkInfo
    if (networkInfo != null && networkInfo.isConnected)
        return true
    return false
}

fun verifyNetworkStateV2(context: Context):Boolean {
    val connMgr = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    connMgr.allNetworks.forEach { network ->
        connMgr.getNetworkCapabilities(network).apply {
            if (this?.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) == true ||
                this?.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) == true)
                    return true
        }
    }
    return false
}
...
    if (!NetUtils.verifyNetworkStateV2(this)) {
        Toast.makeText(this,"No network available",Toast.LENGTH_LONG).show()
        finish()
        return
    }
...
```

Exercício com WebServices

- Desenvolver uma aplicação para apresentar informação meteorológica atual para Coimbra
  - Usar o serviço disponibilizado pela WeatherAPI
    - https://www.weatherapi.com
    - É necessário registar para ter uma API Key
  - Exemplo básico de utilização com retorno da informação a JSON

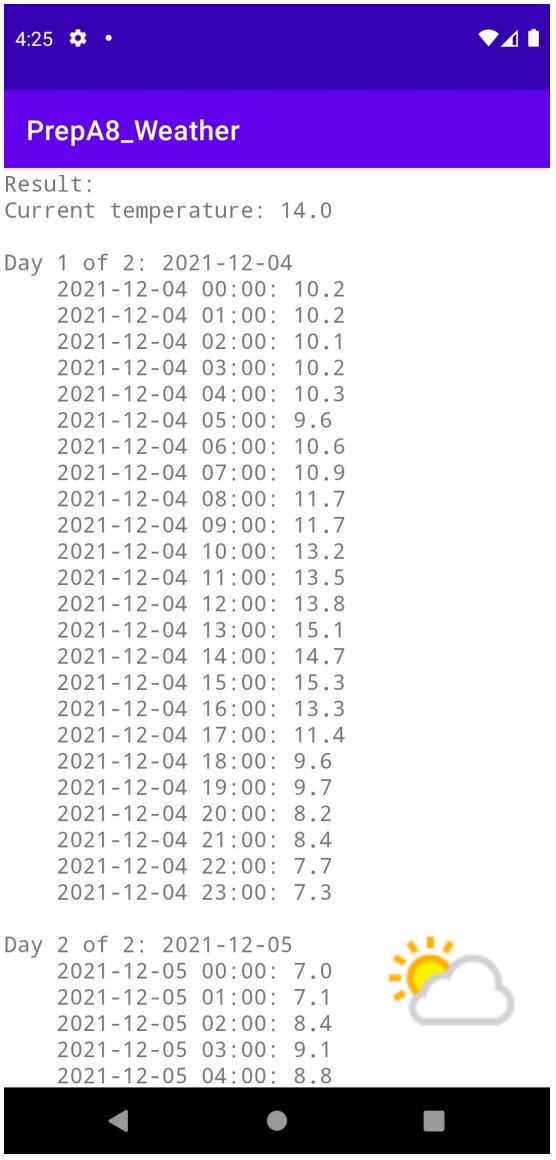
```
https://api.weatherapi.com/v1/forecast.json?
                key=<APIKey>&q=Coimbra&days=2&aqi=no&alerts=no
```

Exercício

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ScrollView
        android:id="@+id/scrollView2"
        android:layout_width="0dp" android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
        <TextView
            android:id="@+id/tvContent"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="monospace" android:textSize="16sp"
            android:text="(none)" />
    </ScrollView>

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="64dp" android:layout_height="64dp"
        android:layout_marginEnd="16dp" android:layout_marginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:srcCompat="@drawable/ic_launcher_foreground" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



JSON

- Formato simples, alternativo ao *XML* mas com objectivos semelhantes, para representação de estruturas de dados
  - Exemplo:

```
{
    “name”: “Zeferino Pancracio”,
    “age”: 57,
    “emails” : [
          “zefe@world.com”,
          “zepa@gmail.com”
    ]
}
```

- Dois tipos de objectos
  - `JSONObject ( { … } )`
    - Dados armazenados segundo o modelo `<atributo> : <valor>`
  - `JSONArray ( [ … ] )`
    - Dados armazenados como uma sequência de valores separados por ‘,’
    - Os valores podem ser objectos `JSONObject` ou `JSONArray`

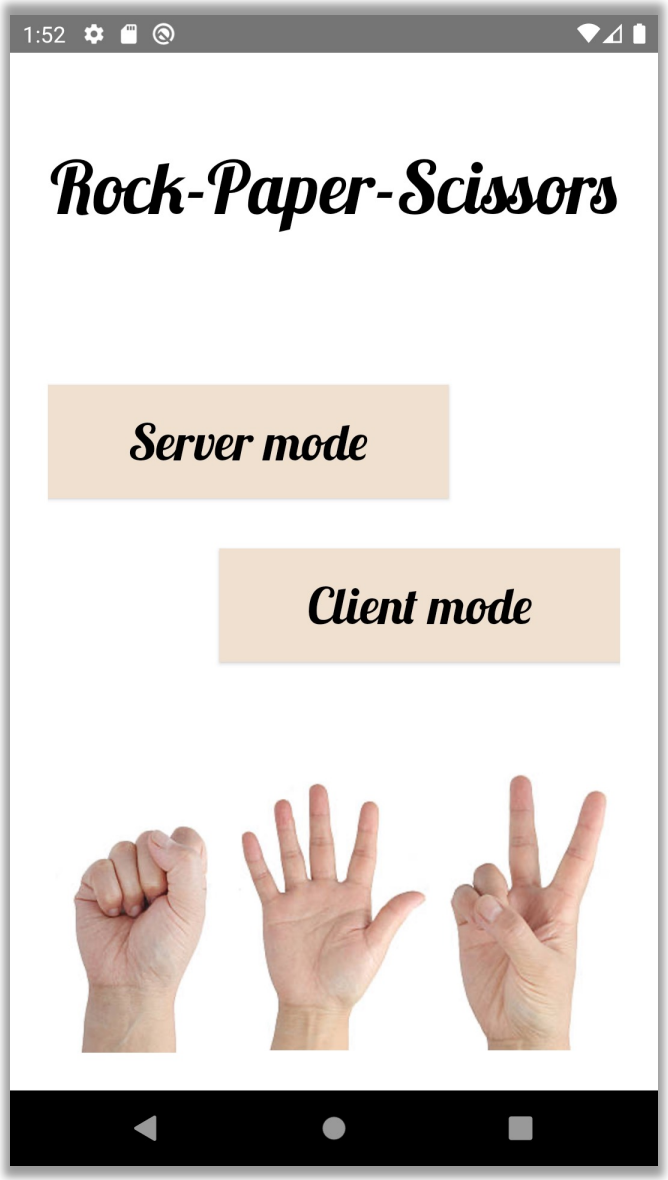
Estudo e experimentação do Android

Jogo multijogador

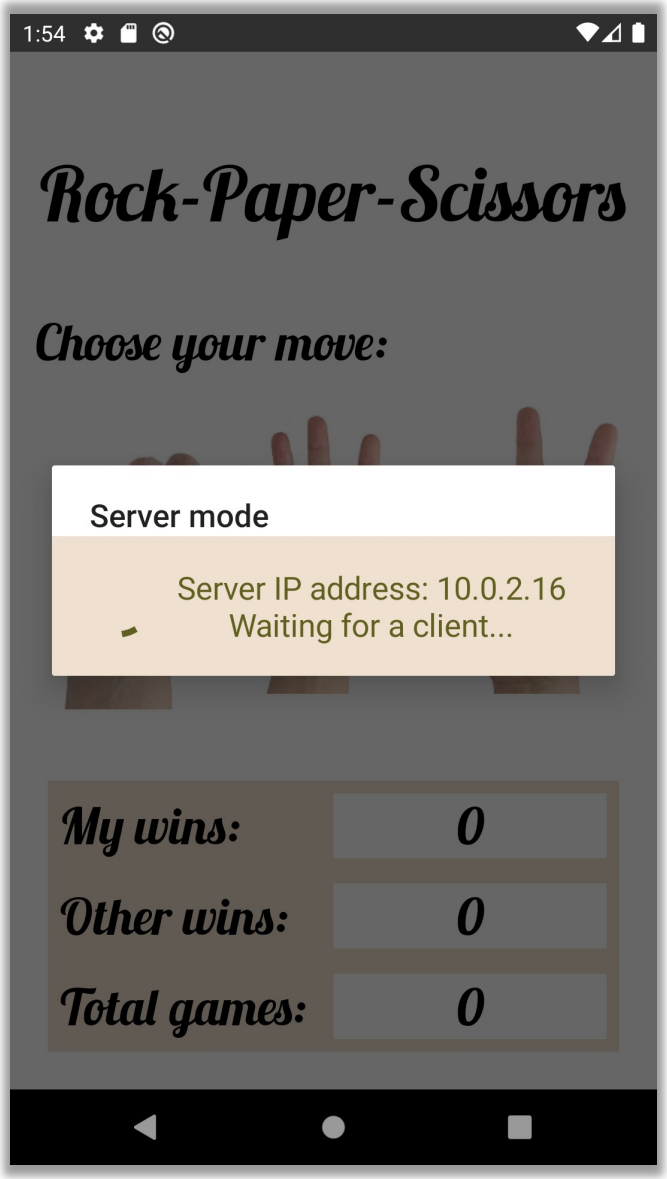
Objetivo

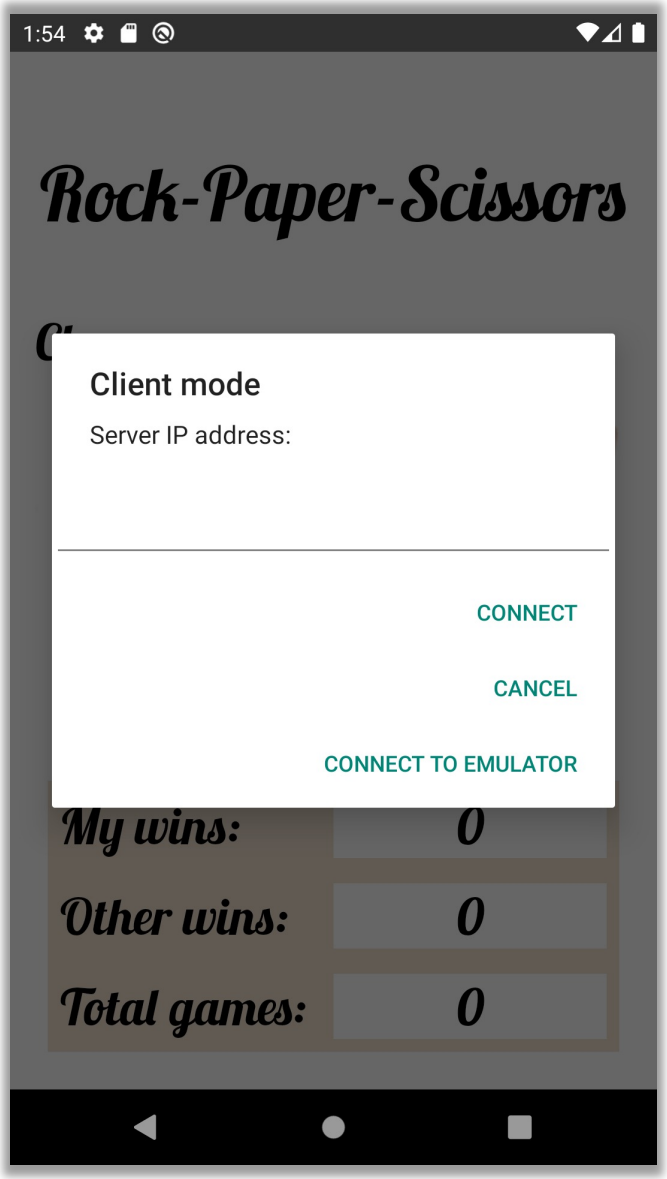
- Desenvolvimento de jogo Pedra-Papel-Tesoura
- Multijogador
  - Comunicação através de *sockets*
    - `java.net, java.io`
  - Um dispositivo será designado de servidor
    - Fica à espera que outro dispositivo se conecte
  - O outro dispositivo será designado de cliente
    - Deverá ser definido o IP do servidor
      - Como opção podemos pesquisar outros dispositivos na rede
        - *Broadcast*
        - *Wifi peer-to-peer (P2P)*
        - *Network Service Directory (NSD)*

Ecrã inicial

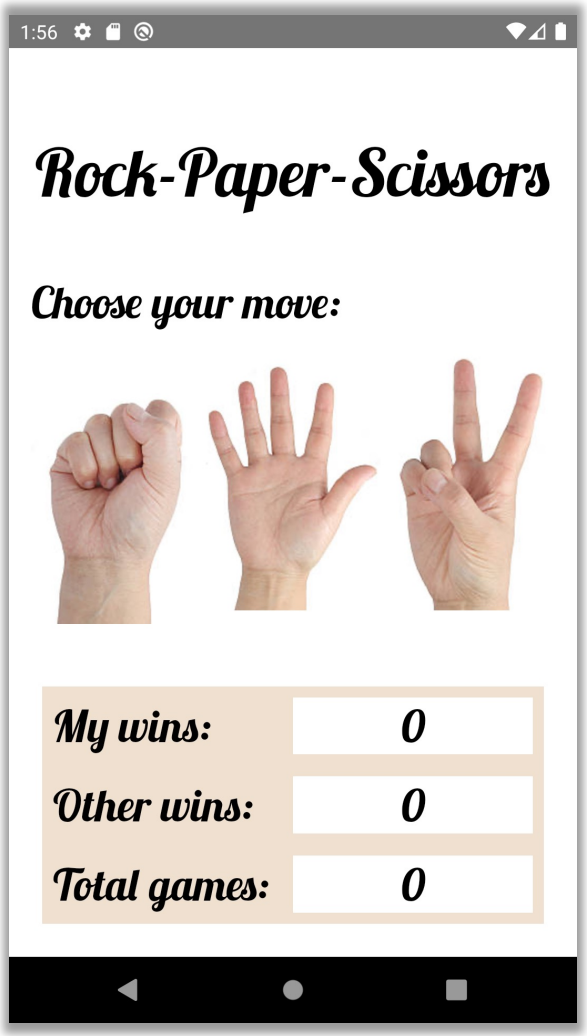


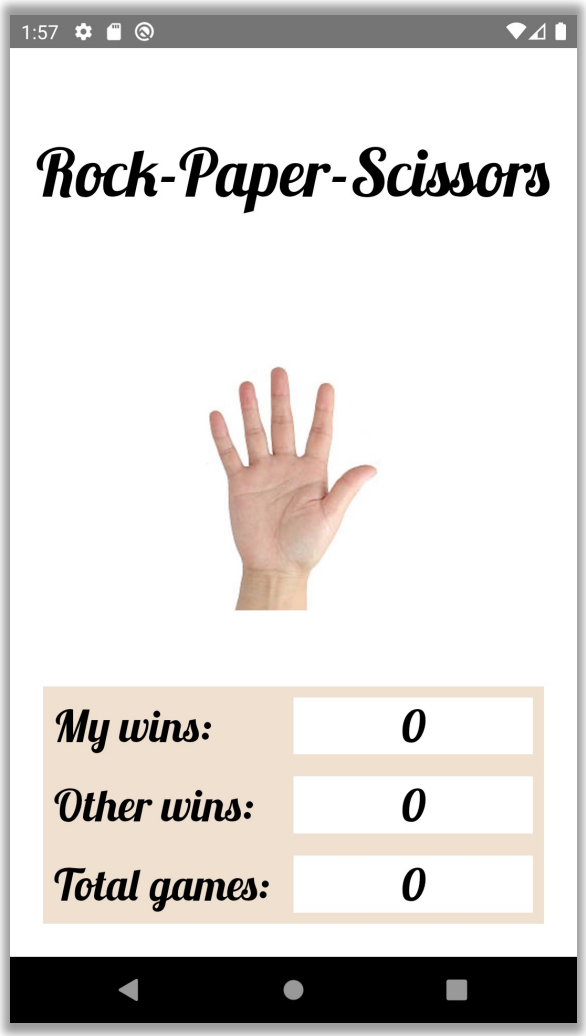
Configuração de modos

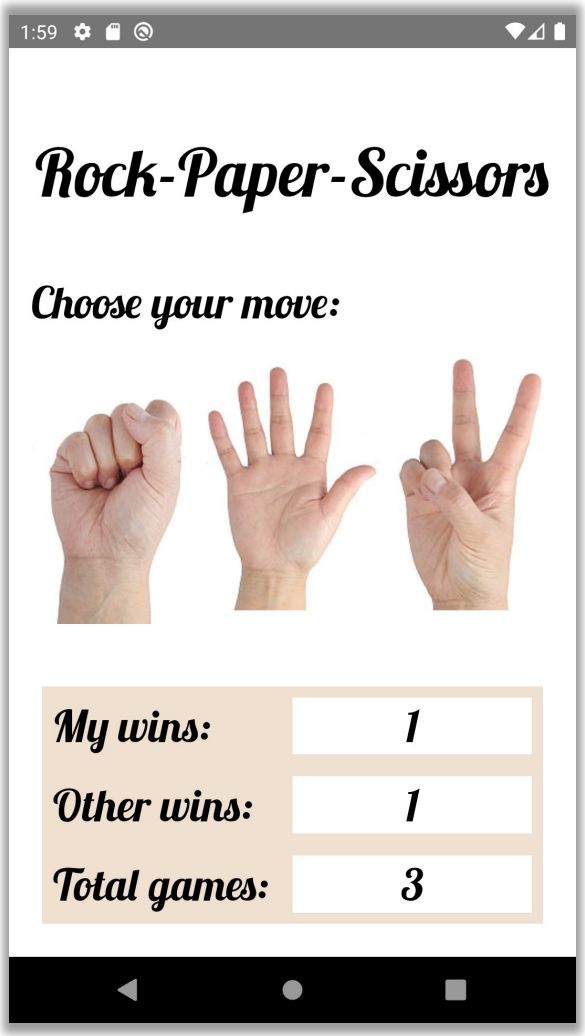




Jogo


1:56
Rock-Paper-Scissors
Choose your move:
My wins: 0
Other wins: 0
Total games: 0


1:57
Rock-Paper-Scissors
My wins: 0
Other wins: 0
Total games: 0


1:59
Rock-Paper-Scissors
Choose your move:
My wins: 1
Other wins: 1
Total games: 3

Teste da aplicação

- *Device (Server/Client) ⇔ Device (Server/Client)*
  - Sem problemas
- *Device (Server) ⇔ Emulator (Client)*
  - O *device* físico tem que atuar como servidor uma vez que não existe forma de conectar ao servidor executado num emulador
  - O emulador pode ligar ao exterior/servidor
- *Emulator (Server) ⇔ Emulator (Client)*
  - É necessário configuração de portos diferentes e reencaminhamento entre portos (ver slide seguinte)
  - Alguns emuladores possuem problemas internos de implementação da *stack* IP, ex.: API<=24 ou API >= 29

Teste com emuladores

- Para permitir o teste com emuladores tem que ser indicado um porto diferente para o servidor "ficar à escuta" relativamente ao que é usado para o cliente se ligar
- É necessário ativar um redireccionamento de tráfego TCP/IP entre os portos definidos para o servidor e cliente
  - Configuração de redireccionamento de portas apenas no emulador que age como servidor
    - `telnet localhost 5554`
      - macOS: `nc localhost 5554`
    - `auth <authentication_token>`
    - `redir add tcp:9998:9999`
    - `exit`
- No emulador cliente a ligação deve ser feita para o IP 10.0.2.2

Exercícios

- Introduzir as seguintes funcionalidades:
  - Ver a jogada do adversário na verificação do jogo
  - Depois do adversário jogar tem 5 segundos para realizar a jogada
    - Mostrar tempo com uma barra de progresso
  - Quem atingir primeiro as 10 vitórias ganha o jogo
    - Mostrar mensagem através de uma `AlertDialog`
  - Mostrar um ecrã *splash* durante 3 segundos no início da execução