POLITECNICO
DI MILANO

LEGO DIGITAL SONORO 2

Documentazione Tecnica

Giacomo Bresciani

Indice

Capitolo 1

Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è quello di mostrare le scelte implementative fatte durante lo sviluppo dell'applicazione *Lego Digital Sonoro 2*. Esso si divide in 4 sezioni principali: *Database*, *Logica di gioco*, e *Interfaccia Utente*.

1.2 Descrizione dell'applicazione

Lego Digital Sonoro 2 è un videogioco per piattaforma Android progettato esclusivamente per dispositivi tablet. Lo scopo del gioco è quello di comporre parole a partire da tessere colorate rappresentanti diverse sillabe. Ogni tessera ha un colore associato direttamente alla sillaba che rappresenta e non appena viene toccata il dispositivo riproduce (tramite il sintetizzatore vocale) il suono della sillaba. Toccando due tessere una dopo l'altra è possibile comporre una parola che, se corretta, verrà aggiunta all'elenco delle parole trovate sotto forma di immagine composta dalle due tessere. La schermata consente inoltre di riascoltare le parole trovate, sia in italiano sia in inglese. Una partita può essere composta da più schermata; ogni schermata termina quando sono state trovate tutte le parole componibili con le sillabe mostrate (2 o 4).
Durante il corso della partita l'applicazione registra i tempi in cui vengono identificate le parole, e le invia ad una mail configurabile nelle impostazioni. E' presente inoltre una modalità cooperativa a due giocatori a turni. Ogni turno consente ad un giocatore un tentativo per indovinare una parola.

Capitolo 2

Database

2.1 Struttura

Il database SQLite è stato implementato tramite SugarORM[1], un ORM (Object-Relational Mapping) sviluppato appositamente per Android. In questo modo le entità del database sono direttamente mappate su classi Java, utilizzando il pattern DAO (Data Access Object).
Il database contiene 5 entità all'interno del package `it.gbresciani.legodigitalsonoro.model` :

`Word` Contiene la parola, sia in italiano sia in inglese, e le sillabe che lo compongono.

`Syllable` Contiene la sillabla e l'esadecimale del colore associato.

`WordStat` Permette di tracciare il momento in cui una parola viene trovata.

`GameStat` Contiene le statistiche di una particolare partita

In figura 2.1 sono riportati dettagli delle entità.

2.2 Inizializzazione

Il database viene inizializato la prima volta che l'applicazione viene aperta con i dati in formato *JSON* presenti nei files `words.json` e `syllable.json`

[1] **#TODO** Link

Figura 2.1:

contenuti nella cartella degli `assets`.
Il listing 2.1 contiene un estratto del file `words.json`

Il mapping degli elementi JSON con la classe Java associata avviene tramite la libreria Open Source GSON[2], sviluppata da Goolge, che fa uso di annotazioni per mappare gli attributi delle classi con i campi JSON.
L'operazione di conversione dei dati da JSON a classi Java e il successivo inse-

[2] **#TODO** Link

Listing 2.1: Porzione del file `words.json`

```
[
  {
    "lemma":"arco",
    "eng":"arc",
    "syllable1":"ar",
    "syllable2":"co"
  },
  ...
```

rimento all'interno del database, in quanto operazioni relativamente lunghe, non vengono eseguite all'interno del *Thread* principale ma in uno differente grazie ad un `IntentService` contenuto in `it.gbresciani.legodigitalsonoro.services.InitDBService`.

Capitolo 3

Logica di gioco

La logica di gioco è affidata all'Activity `PlayActivity`; essa gestisce inoltre i `Fragments` (vedi capitolo 4) che visualizzano l'Intefaccia Utente.

3.1 Glossary

In questa sezione vengono descritti acuni termini utili alla comprensione delle successive sezioni.

Page Una partita è composta da una o più pagine. Una pagina può contenere 2 o 4 sillable e da 1 a 4 parole da trovare

User role Durante una partita multi player i giocatori si distinguono in "master", colui che "orchestra" la partita, e "slave", colui che vi partecipa in modo passivo

Turno Durante una partita multi player rappresenta la possibilità per un giocatore di provare a indovinare una parola. Il giocatore che non è in possesso del turno attende la mossa dell'altro

3.2 Event Bus

Per far comunicare i vari componenti dell'applicazione (`Activities`, `Fragments`, `Services`) è stato utilizzato Otto[1], un *Event Bus*, sviluppaato appositamen-

[1] **#TODO** Link

te per Android, che permette ai vari componenti di postare eventi su un unico Bus e per riceverli in modo asincrono.
La classe `BusProvider` permette, tramite un metodo statico, di accedere allo stesso Bus in ogni punto dell'applicazione.
All'interno del package `package it.gbresciani.legodigitalsonoro.events;` son presenti numerose classi (ad esempio `WordClickedEvent` o `PageCompletedEvent`) che rappresentano i diversi eventi generati e "ascoltati" dai componenti dell'app.
La classe `NineBus` è una piccola estensione del Bus predefinito di Otto che permette tutti gli eventi sul `Main Thread`, anche se generati in un altro `Thread`.

3.3 GameState

`GameState` è la classe che rappresenta lo stato di una partita, sia essa single o multi player. Viene inizializzato da `PlayActivity`, e aggiornato durante il corso della partita in seguito ai diversi eventi prodotti dagli altri componenti dell'app.
I suoi attributi sono:

pageNumber Il numero di pagina corrente

pages Il numero di pagine totali della partita in corso (stabilito nelle impostazioni)

pageWordsToFindNum Il numero di parole ancora da trovare nella pagina corrente

wordsAvailable Un `ArrayList` contentente le parole ancora da trovare

wordsFound Un `ArrayList` contenente le parole gia trovate

syllables Un `ArrayList` contenente le syllabe presenti nella pagina corrente

currentPlayer Se la partita è multi player contiene una stringa che rappresenta il ruolo del giocatore che detieni in quel momento il turno

currentPlayerDeviceId Se la partita è multi player contiene una stringa che rappresenta il id del device che detieni in quel momento il turno

La classe `GameState` contiene attributi annotati con `@SerializedName` in modo da poter essere convertita in un oggetto JSON (tramite GSON) ed essere inviata al secondo giocatore nel caso di partita multiplayer (maggiori dettagli nella sezione **??**).

3.4 Statistiche di gioco

Le satistiche della partita, come accennato nel capitolo sul database, vengono salvate tramite i DAO `WordStat` e `GameStat` che contengono, ad esempio, i tempi di ritrovamento delle parole, le parole stesse, la pagine su cui sono state trovate o eventualmente l'id del giocatore che le ha trovate.

3.5 Game flow

Tutta le fasi di una partita sono gestite dall'`Activity` `PlayActivity`. All'avvio, cioè durante la chiamata al metodo `onCreate()` essa esegue le seguenti operazioni:

1. Carica i settaggi della partita
2. Carica i suoni di gioco
3. Istanziaril sintetizzatore vocale `TextToSpeech`
4. In base al tipo di partita (single o multi player) inizializza o meno il Bluetooth e setta la variabile `multi` a "MASTER" o "SLAVE"

I principali metodi di `PlayActivity` sono:

`startGame()` Si occupa di inizializzare gli oggetti per le statistiche e chiama `constructPage()` and `startPage()`

`constructPage()` Inizializza lo stato della partita (`GameState`) con i dati relativi alla pagina corrente. Chiama i metodi accessori `Helper.chooseSyllables()` (sceglie in modo casuale le sillabe per la pagina) e `Helper.permuteSyllablesInWords()` (calcola le permutazione delle sillabe scelte per trovare le parole componibili)

Listing 3.1: Struttura delle satistiche inviate

```
Dispositivo N: device id
Tempo totale: mm:ss
N - word: mm:ss (# of page on which the word was found)
...
```

`startPage()` Inizializza i `Fragments` che mostreranno l'interfaccia utente

`updateState()` Si occupa di gestire la logica di gioco: determina se tutte le parole sono state trovate o se interrompere il gioco quando non è il proprio turno e posta sul Bus l'evento `StateUpdatedEvent` in modo che i `Fragments` possano reagire correttamente

L'`Activity` presenta inoltre diversi metodi annotati con `@Subscribe` che permettono di reagire agli eventi postati da altri componenti.

3.6 Invio statistiche

L'invio delle statistiche avviene tramite l'invio di una mail all'indirizzo settato nelle impostazioni dell'applicazione.
L'invio è gestito dall'`IntentService` `GenericIntentService` al quale viene passato l'id corrispondente alla partita appena conclusa. Interrogando il database esso carica i dati e li inserisce nel corpo dell'email formattandoli come mostrato nel listing 3.1. Il listing 3.2 ne mostra invece un esempio.

3.7 Multiplayer

Come già detto la modalità multi player segue le stesse regole di quella single player se non per il fatto che i due giocatori provano a turno ad trovare una parola.
La comunicazione tra i due devices avviene tramite Bluetooth. La classe che si occupa di stabilire la connessione è `BluetoothService`

Listing 3.2: Esempio di statistiche

```
- One player -

Dispositivo 1: BC:20:A4:73:6B:49
Tempo totale: 00:13
2 - fumo: 00:02 (1)
1 - muro: 00:13 (2)

- Two Player -

Dispositivo 1: BC:20:A4:73:6B:49
Dispositivo 2: D4:0B:1A:15:FD:AD
Tempo totale: 00:14
2 - ceno: 00:02 (1)
1 - noce: 00:09 (1)
2 - buco: 00:14 (2)
```

Capitolo 4

User Interface

4.1 Master-Details layout

As decided in the Design Document, the main screen of the application, containing the list of alarms, is presented in different ways depending on the screen size of the device: in particular, on large screens the list and the details of alarms are shown one next to each other. To get this kind of layout were exploited `Fragmen`, UI components which have the particularity of being able to be reused in multiple `Activity` maintaining their application logic. The classes used to achieve this behavior are:

- MainActivity
- AlarmDetailActiviy
- AlarmListFragment
- AlarmDetailFragment

It's has been also created a file `refs.xml` in the folder res/values-sw600dp:

```
<resources>
    <item type="layout"
        name="activity_alarm_list">@layout/activity_alarm_twopane</item>
</resources>
```

along with files `activity_alarm_list.xml` and `activity_alarm_twopane.xml` in the folder res/layout. Starting `MainActivity` and executing:

```
inflater.inflate(R.layout.activity_alarm_list, container, false);
```

Android will know which of the two layouts to choose, thanks to the alias contained in `refs.xm`, depending on the size of the display. In fact the system will choose the resources contained in folders with suffix "-sw600dp" when the width of the display will be greater than 600 dp. In this way it is possible to recognize whether the code is run on a tablet or a smartphone then choosing to put side by side `AlarmListFragment` and `AlarmDetailFragment` in the first case or only `AlarmListFragment` in the latter.

Figura 4.1: Master-Details layout - Tablet

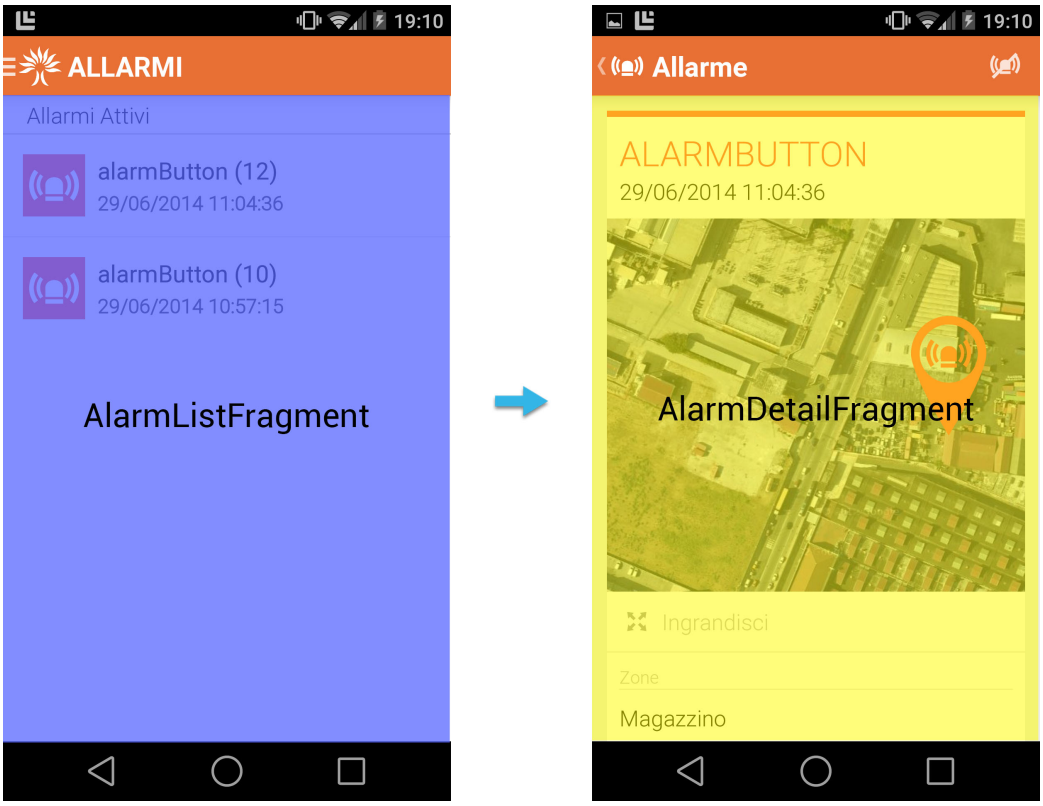

Figura 4.2: Master-Details layout - Phone

4.2 AlarmDetailActivity

This `Activity` is launched thanks to an `Intent` which contains an extra parameter that indicates the id of the alarm that will displayed:

```
Intent detailIntent = new Intent(mActivity,
    AlarmDetailActivity.class);
detailIntent.putExtra(AlarmDetailFragment.ARG_ITEM_ID, id);
```

4.3 NotificationsManager

he management of notifications has been enclosed in the class `NotificationsManager`. The class has two `Notifications`:

NOTIF_ONGOING_SERVICE Notification permanently associated with the background service as described in section **??**; the text of the notification is updated with the appropriate methods. Tapping it launches the app.

NOTIF_ONGOING_ALARMS Notification displayed permanently until there are new alarms not yet readed. Tapping it launches the alarm list.

Capitolo 5

External libraries

They are listed here external libraries used for application development.

Sprinkles
: Libreria Open Source ORM

Gson
: Java library that allows you to convert between Java objects and their representation Json.

Crouton
: Open Source library that allows to display alerts and confirmation messages.

Joda-Time
: Java library for managing dates.

Crashlytics
: Bug-tracking solution with a web inteface.

SmoothProgressBar
: Open Source library which improves the smoothness of the default Android progressbar.