UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL' INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN
INGEGNERIA INFORMATICA

Parrot Game, uno strumento interattivo per l'accertamento delle capacità discriminatorie del linguaggio nei bambini in età prescolare

Relatore: Chiar.mo
Prof. Antonio Rodà

Laureando: Loris De Marchi
Nr. Matr. 1011049

Anno Accademico 2013-2014

INDICE

0. Indice 2

1. Introduzione 3

2. Funzionalità di save/resume 6

3. Editor di playlist 13

4. Backup e ripristino dei setup delle impostazioni 17

5. Conclusioni 23

Bibliografia 24

1. Introduzione

La tesi si propone di descrivere le nuove funzionalità aggiunte all'applicazione per dispositivi mobili "Parrot Game Discrimination", nata dalla collaborazione tra il Dipartimento di Psicologia dello Sviluppo e della Socializzazione dell'Università di Padova e il Centro di Sonologia Computazionale (CSC) del Dei, Dipartimento di Ingegneria Informatica di Padova, per implementare un metodo di accertamento delle capacità discriminatorie dei suoni da parte di bambini affetti da particolari disturbi all'udito [3].

Un ulteriore obiettivo è quello di approfondire la ricerca scientifica riguardante il processo di produzione del linguaggio parlato da parte di individui con difficoltà uditive.

Il Parrot Game è stato progettato sotto forma di videogioco, reso stimolante ed interattivo dalla moderna tecnologia touch screen e dalla presenza di personaggi animati (un bambino e il suo pappagallino) che parlano ed interagiscono direttamente con l'utente.

L'applicazione fornisce l'opportunità di far ascoltare ai partecipanti coppie di suoni tra cui discriminare; i due suoni possono essere tra loro uguali, molto simili, non molto simili, differenti.

Le proprietà dei suoni esaminate sono il timbro, l'altezza, l'intensità.

Ogni partita è suddivisa in livelli con un grado crescente di difficoltà di riconoscimento dei suoni; è quindi possibile verificare il livello di udito del bambino (percezione globale del suono, percezione nella media, e via via più fine).

Alla fine di ogni livello il giocatore viene premiato con una medaglia, indipendentemente dalla quantità di risposte corrette che ha dato (questo perché il bambino non in grado di distinguere i suoni non comprenderebbe un'eventuale "sconfitta" alla fine del gioco).

Terminata la partita, è possibile controllare i risultati ottenuti per ogni coppia e valutare così la capacità discriminatoria del bambino.

Rispetto alla versione precedente (Figura 1.1) sono state apportate diverse migliorie (Figura 1.2), sia all'esperienza di gioco da parte del giocatore, che alla gestione delle impostazioni da parte dell'operatore che assiste il bambino.

E' infatti possibile interrompere la partita in corso per poi riprenderla in un secondo momento; ciò permette al bambino di fare delle pause tra una sessione di gioco e l'altra (utile in caso di partite più lunghe del normale o con suoni più complessi e impegnativi che richiedono più concentrazione da parte del giocatore) (capitolo 2).

Inoltre è stata aggiunta la possibilità di creare delle playlist direttamente all'interno dell'applicazione, utilizzando un apposito editor, senza dover caricare dei file di testo contenenti i nomi delle coppie da eseguire direttamente nella cartella dell'applicazione (capitolo 3).

Infine si possono ora creare dei set di impostazioni (tempo tra un suono e l'altro, numero di coppie per livello, ecc…) e passare rapidamente da un set all'altro (capitolo 4).

Per i dettagli implementativi, si faccia riferimento alla sitografia riportata [1].

Figura 1.1

Figura 1.2

2. Funzionalità di save/resume

2.1 Introduzione

La nuova versione dell'applicazione prevede la sospensione del gioco tra un livello e l'altro, in modo da poter riprendere la partita in un secondo momento, anche in seguito alla chiusura dell'applicazione stessa.

Si supponga di iniziare un nuovo gioco. Al termine di ogni livello viene notificato all'utente che può scegliere se andare avanti con i rimanenti livelli oppure fare una pausa (figura 2.1).

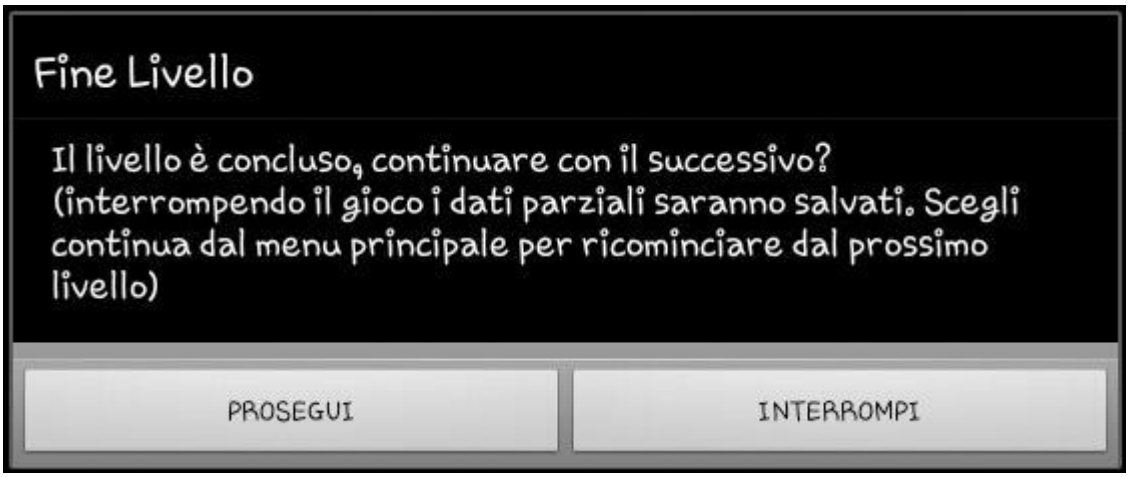

Figura 2.1

In caso si voglia continuare a giocare, il gioco prosegue normalmente fino al livello successivo, in cui verrà notificata ancora una volta la medesima possibilità di scelta.

In caso contrario, lo stato della partita viene salvato e si viene riportati alla schermata iniziale.

Lo stato rimane memorizzato anche in caso di chiusura dell'applicazione.

2.2 Descrizione dettagliata della funzione implementata

2.2.1 Nella versione precedente, alla fine di ogni livello veniva chiesto all'utente se preferisse continuare o interrompere il gioco. Nel secondo caso interveniva il codice seguente e semplicemente poi si tornava allo schermo del titolo:

```
setResult(RESULT_CANCELED);
finish();
```

La nuova versione invece, prima di tornare allo schermo del titolo, sospende i dati in corso con il seguente codice:

```
suspendGame();
setResult(RESULT_OK);
finish();
```

Dove `suspendGame()` è un metodo che salva nelle SharedPreferences le impostazioni utilizzate per la partita corrente e nel database le risposte parziali, i loro tempi e l'eventuale maschera utilizzata per simulare la casualità delle coppie:

```
private void suspendGame() {

   // save temp data

   // variables
   SharedPreferences sh =
   getSharedPreferences(Constants.RESUMING_SETTINGS, MODE_PRIVATE);
   SharedPreferences.Editor ed = sh.edit();

   ed.putInt(TEMP_CURRENT_LEVEL, currentLevel);
   ed.putInt(TEMP_CURRENT_COUPLE, currentCouple);
   ed.putInt(TEMP_LEVEL_FOR_REWARD, levelForReward);
   ed.putBoolean(TEMP_END_OF_GAME, endOfGame);
   ed.putInt(TEMP_CURRENT_BKGND, currentBackGroundIndex);
   ed.putInt(TEMP_SKY_BKGND, currentSkyBackGroundIndex);

   // arrays
   DatabaseManager mng = new DatabaseManager (getApplicationContext());
   mng.insertTempData(m, answers, answerTimes);

   // resuming mode ON
   ed.putBoolean(RESUMING, true).commit();
}
```

Notare come il flag *RESUMING* nelle SharedPreferences venga portato a **true**, in modo da notificare al sistema che sono presenti dati di gioco in sospeso.

2.2.2 Dallo schermo del titolo, selezionando "Inizia!" oppure "Opzioni", verrà notificato all'utente che proseguendo con la sua scelta i dati in sospeso saranno cancellati. Questa scelta è stata fatta poiché è ragionevole pensare che iniziando un nuovo gioco i dati precedenti non servano più; inoltre accedendo alle opzioni è possibile cambiare impostazioni che renderebbero inconsistente lo stato della partita salvata.

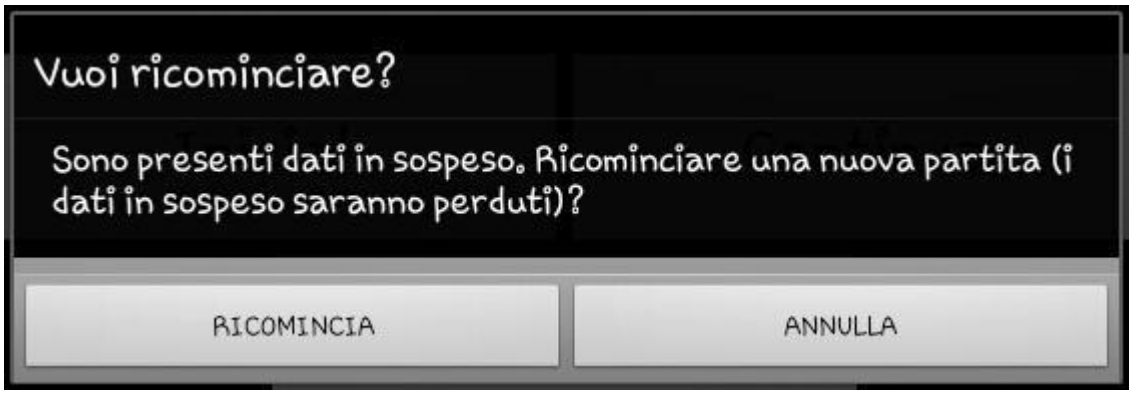

Figura 2.2

Nel caso del pulsante "Inizia!", nel caso ci sia una partita in sospeso, ciò viene notificato all'utente, che può scegliere nella finestra di dialogo (figura 2.2) se annullare la sua scelta oppure cominciare una nuova partita. Altrimenti semplicemente viene avviata una nuova partita (`startNew()`):

```
if (preferences.getBoolean(RESUMING, false)) restartDialog();
   else {
   startNew();
}
```

Dove `startNew()` é:

```
private void startNew() {
   SharedPreferences preferences =
         getSharedPreferences(Constants.RESUMING_SETTINGS,
         MODE_PRIVATE);
   preferences.edit().putBoolean(RESUMING, false).commit();
   Intent t = new Intent(AndroMusicActivity.this,
         StartFormActivity.class);
   AndroMusicActivity.this.startActivity(t);
}
```

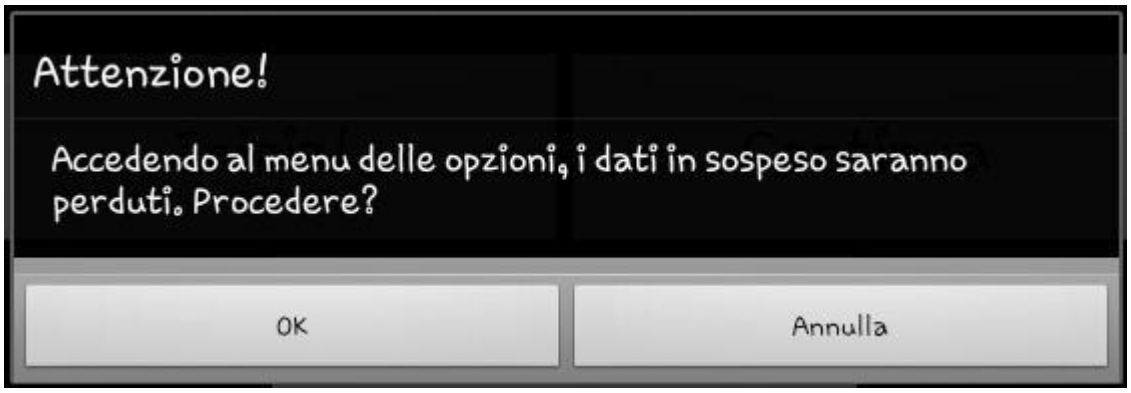

Figura 2.3

Nel caso del pulsante "Opzioni", invece, la finestra in figura 2.3 notifica che i dati in sospeso saranno cancellati.

```
private void startOptionsActivity() {
   SharedPreferences preferences =
         getSharedPreferences(Constants.RESUMING_SETTINGS,
         MODE_PRIVATE);
   preferences.edit().putBoolean(RESUMING, false).commit();
   Intent t = new Intent(AndroMusicActivity.this,
         ConfigActivity.class);
   AndroMusicActivity.this.startActivity(t);
}
```

In entrambi i casi, il flag `RESUMING` nelle SharedPreferences viene portato a `false` in modo tale da notificare al sistema che non c'è più alcun gioco in sospeso da ripristinare. Scegliendo il pulsante "Continua" nello schermo del titolo avviene il recupero della sessione di gioco (figura 2.4). Nel caso non esistano dati in sospeso, ciò viene notificato all'utente (figura 2.5).

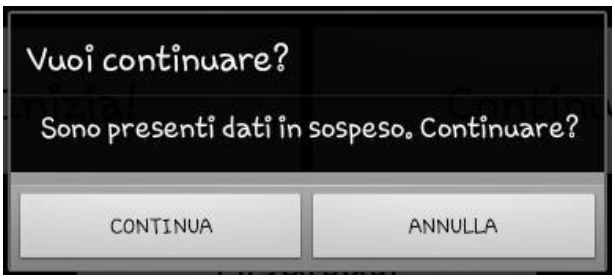

Figura 2.4

Figura 2.5

2.2.3 Altrimenti viene lanciata l'activity relativa al gioco e il seguente codice gestirà il recupero della sessione di gioco precedentemente salvata nel database:

```
private void resumeGame() {

   // resume temp data

   // variables
   SharedPreferences sh =
         getSharedPreferences(Constants.RESUMING_SETTINGS,
         MODE_PRIVATE);
    currentLevel = sh.getInt(TEMP_CURRENT_LEVEL, 1);
   currentCouple = sh.getInt(TEMP_CURRENT_COUPLE, 0);
   levelForReward = sh.getInt(TEMP_LEVEL_FOR_REWARD, 0);
   endOfGame = sh.getBoolean(TEMP_END_OF_GAME, false);
   currentBackGroundIndex = sh.getInt(TEMP_CURRENT_BKGND, 0);
   currentSkyBackGroundIndex = sh.getInt(TEMP_SKY_BKGND, 0);

   // arrays
   DatabaseManager mng = new DatabaseManager (getApplicationContext());

   m = new int[totalCouples];
   answers = new boolean[totalCouples];
   answerTimes = new long[totalCouples];

   mng.getTempData(m, answers, answerTimes);

   // resuming mode OFF
   sh.edit().putBoolean(RESUMING, false).commit();
}
```

Vengono quindi recuperati dalle SharedPreferences i valori salvati e dal database i risultati parziali mediante il metodo `mng.getTempData(m, answers, answerTimes)`). Inoltre, dato che i dati in sospeso non servono più poiché il gioco è stato ripreso, il flag *RESUMING* nelle SharedPreferences viene riportato a **false**.

2.3 Struttura dei dati

Utilizzo una tabella SQLite per memorizzare i risultati temporanei di una partita. Di seguito il codice che crea la tabella:

```
/**
 * @author Loris
 * creation of temp data table for suspending/resuming game
 */

String createTableTempData = "create table " +
   DatabaseConstants.TABLE_TEMP_DATA + "( " +
   DatabaseConstants.COLUMN_INDEX + " integer primary key, " +
   DatabaseConstants.COLUMN_MASK + " integer, " +
   DatabaseConstants.COLUMN_ANSWERS + " integer, " +
   DatabaseConstants.COLUMN_ANSWER_TIMES + " integer);";

db.execSQL(createTableTempData);
Log.d("CREATETABLE",createTableTempData);
```

2.4 Dettaglio delle funzioni del database manager utilizzate

```
/**
 * Insert temp data in database
 * @author Loris
 */

public boolean insertTempData(int[] m, boolean[] answers, long[]
   answerTimes) {

        //...

        ContentValues values;
        for (int i=0; i<m.length;i++) {
              values = new ContentValues();
              values.put(DatabaseConstants.COLUMN_INDEX, i);
              values.put(DatabaseConstants.COLUMN_MASK, m[i]);
              values.put(DatabaseConstants.COLUMN_ANSWERS, answers[i]?1:0);
              values.put(DatabaseConstants.COLUMN_ANSWER_TIMES,
                    answerTimes[i]);

            sqlite.insert(DatabaseConstants.TABLE_TEMP_DATA, null, values);
            Log.d("inserted "+i, ""+m[i]+answers[i]+answerTimes[i]);
        }

        //...
}
```

```
/**
 * Restore temp data from database
 * @author Loris
 */

public void getTempData(int[] m, boolean[] answers, long[] answerTimes){

        //...

        String[] columns = { DatabaseConstants.COLUMN_INDEX,
              DatabaseConstants.COLUMN_MASK,
              DatabaseConstants.COLUMN_ANSWERS,
              DatabaseConstants.COLUMN_ANSWER_TIMES };

        Cursor cursor = sqlite.query(DatabaseConstants.TABLE_TEMP_DATA,
              columns, null, null, null, null, null);

        if (cursor != null) {
              if (cursor.moveToFirst() ) {
                    int index;
                    for (int i=0; i<cursor.getCount(); i++) {
                          index = cursor.getInt(0);
                          m[index] = cursor.getInt(1);
                          answers[index] = cursor.getInt(2)>0;
                          answerTimes[index] = cursor.getLong(3);
                          cursor.moveToNext();
                    }

                    //...
              }
        }

        //...
}
```

3. Editor di playlist

3.1 Introduzione

L'activity si propone di mettere a disposizione un ambiente dove poter creare e modificare le playlist, cioè liste di coppie di file audio da riprodurre durante lo svolgimento del gioco. L'interfaccia utente si presenta come in figura 3.1; la parte di destra presenta due colonne contenenti ciascuna tutti i file audio presenti nella cartella di sistema "media". Selezionando un file per ogni colonna si genera una coppia. E' possibile aggiungere alle playlist nuove coppie di file audio, riordinare o eliminare le coppie già inserite, salvare le modifiche, o caricare playlist precedentemente create per modificarle. La separazione in livelli dipende dal valore `levelMeanNumber` impostato nelle opzioni, il quale determina quante coppie deve possedere ogni livello.

Figura 3.1

3.2 Descrizione dettagliata della funzione implementata

3.2.1 Il pulsante "Nuova playlist" genera una finestra di dialogo (figura 3.2) dove poter inserire il nome di una nuova playlist; una volta digitato il nome, crea la playlist corrispondente. La creazione viene effettuata creando un nuovo file nella cartella delle playlist come da codice seguente (si omettono per semplicità di lettura i controlli effettuati sul nome del file letto dalla casella di testo); viene inoltre settata come playlist corrente la nuova playlist (`switchToPlaylist = newPlaylist`):

```
//...
EditText newPlaylistName = (EditText)((AlertDialog
   dialog).findViewById(R.id.newName);
String newPlaylist = newPlaylistName.getText().toString()+".txt";
newFile(newPlaylist);
switchToPlaylist = newPlaylist;
//...
```

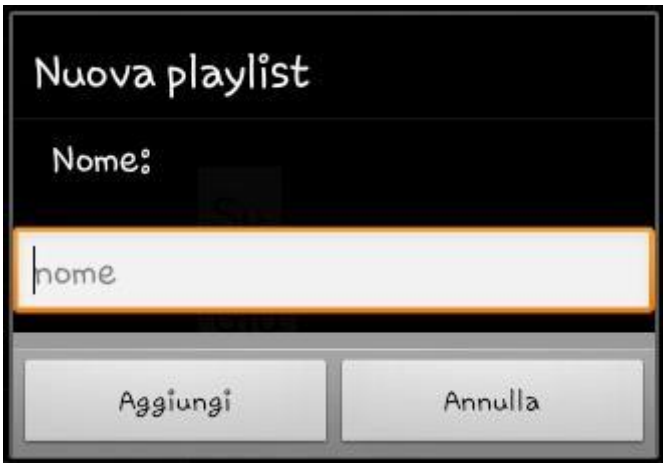

Figura 3.2

3.2.2 Il pulsante "Aggiungi coppia" legge i nomi selezionati nelle due colonne, crea la coppia corrispondente (dopo essersi accertato che sia una coppia di file audio validi), e la inserisce in coda nella playlist:

```
File f1 = new File(samplesPath+txt1.getText());
File f2 = new File(samplesPath+txt2.getText());
if (f1.isFile() && f2.isFile()) {
   AudioCouple a = new AudioCouple(f1,f2);
   lm.addCouple(a);
   //...
}
```

3.2.3 E' possibile cambiare l'ordine delle coppie inserite semplicemente agendo sui pulsanti "Su" e "Giù". Per eliminare una coppia basta selezionarla e utilizzare il pulsante "Elimina". Il codice è molto semplice, utilizza infatti metodi della classe ListManager (`switchElement()` e `removeCurrent()`), già presente nella versione precedente dell'applicazione:

```
// SU
lm.switchElement(ListManager.UP);
isPlaylistModified = true;

// GIU'
lm.switchElement(ListManager.DOWN);
isPlaylistModified = true;

// ELIMINA
lm.removeCurrent();
isPlaylistModified = true;
```

3.2.4 Il pulsante "Salva playlist" rende effettive le modifiche apportate alla playlist su cui si sta lavorando, copiando la lista di coppie scelte in un omonimo file di testo mediante il metodo `writeCouples()` (già presente nella versione precedente dell'applicazione, ma modificato per poter svolgere anche questa nuova funzione). Inoltre il flag di modifica della playlist `isPlaylistModified` viene settato a **`false`** poiché le modifiche sono state salvate.

```
//...
FileManager.writeCouples(lm.getGoodList());
isPlaylistModified = false;
//...
```

3.2.5 Lo spinner situato in alto permette di scegliere la playlist da modificare. Alla scelta di una playlist dalla lista, essa viene caricata dal file di testo corrispondente e visualizzata a schermo. In caso ci sia una playlist a cui si stavano apportando delle modifiche, una finestra di dialogo notificherà all'utente la possibilità di salvarla o di annullare le modifiche (figura 3.3):

```
//...
if (isPlaylistModified) {
   switchDialog.show();
}
else {
   switchPlaylist();
}
//...
```

3.2.6 Il pulsante "Indietro" riporta l'utente alla schermata delle opzioni, se le modifiche alla playlist corrente sono state salvate. In caso contrario, prima di tornare alla schermata precedente, una finestra di dialogo notificherà all'utente la possibilità di salvare o meno le modifiche apportate (figura 3.3).

Figura 3.3

4. Backup e ripristino dei setup delle impostazioni

4.1 Introduzione

La prima versione dell'applicazione prevedeva di poter impostare un unico set di preferenze nell'activity "Opzioni" (figura 4.1). Ciò obbligava l'operatore ad intervenire manualmente su ogni preferenza ad ogni esperimento. La nuova versione mette invece a disposizione più set personalizzabili di preferenze, in modo tale da poter passare da una configurazione all'altra in maniera semplice e veloce (figura 4.2).

Inoltre l'unica playlist veniva prima modificata in questa activity. Con la nuova versione questa operazione di editing delle playlist ha un'activity propria, descritta nel capitolo 3. Ciò ha permesso, in aggiunta ad una maggiore gestibilità delle playlist, di alleggerire la visualizzazione dei dati sullo schermo, agevolandone la comprensione.

Figura 4.1

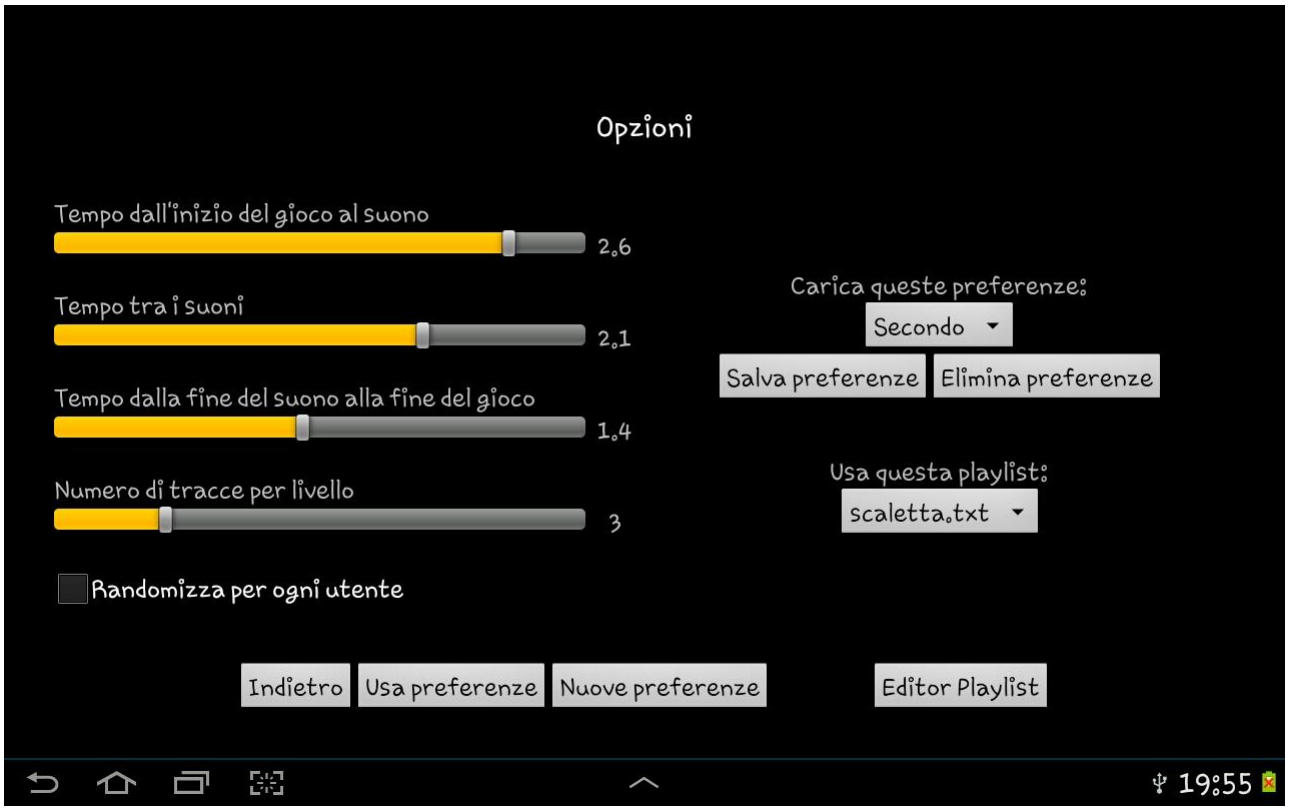

Figura 4.2

4.2 Descrizione dettagliata della funzione implementata

4.2.1 Il pulsante "Nuove preferenze" visualizza una finestra di dialogo in cui viene richiesto il nome del nuovo set da creare. La creazione avviene invocando il metodo newSettingsRecord() del DatabaseManager e passandogli il *nome* del set appena inserito e le *impostazioni* attuali lette dalle scrollbar e dalla checkbox con il metodo getUsingSettings():

```
mng.newSettingsRecord(newSetting, getUsingSettings());
```

4.2.2 Il pulsante "Salva preferenze" sovrascrive nel database il set corrente, con le impostazioni aggiornate delle scrollbar e della check box.

```
mng.saveSetting(settingsSp.getSelectedItem().toString(),
timeBetween, timeBegin, timeFinish, levelMeanNumber,
isRandomized?1:0);
```

4.2.3 Il pulsante "Elimina preferenze" elimina il record relativo al set corrente

```
mng.deleteSettingsRecord(settingsSp.getSelectedItem().toString());
```

4.2.4 Lo spinner "Carica queste preferenze" presenta la lista dei set fin ora memorizzati. Alla selezione di uno di questi vengono istantaneamente lette dal database le impostazioni associate al nome del set scelto:

```
float[] set = mng.getSettingsByName(name);
timeBetween = set[0];
timeBegin = set[1];
timeFinish = set[2];
levelMeanNumber = (int)set[3];
isRandomized = set[4]>0;
```

e successivamente caricate nelle relative scrollbar, checkbox e textview:

```
betweenTimeBar.setProgress((int)(timeBetween*BAR_RES));
beginTimeBar.setProgress((int)(timeBegin*BAR_RES));
finishTimeBar.setProgress((int)(timeFinish*BAR_RES));
levelBar.setProgress(levelMeanNumber-1); cb.setChecked(isRandomized);

tbi.setText(""+timeBetween);
tbegin.setText(""+ timeBegin);
tfinish.setText(""+ timeFinish);
tli.setText("" + levelMeanNumber);
```

4.2.5 Il pulsante "Usa preferenze" rende effettive le modifiche alle impostazioni, che saranno da questo momento applicate all'ambiente di gioco, salvandole nelle relative Shared Preferences.

4.3 Struttura dei dati

Per memorizzare i vari set ho scelto di utilizzare una tabella SQLite, che memorizza il nome del set e i valori delle scrollbar e della checkbox. Di seguito il codice che crea la tabella:

```
/**
 * @author Loris
 * creation of settings table
 */

String createTableSettings = "create table " +
       DatabaseConstants.TABLE_SETTINGS + "( " +
       DatabaseConstants.COLUMN_NAME + " text primary key, " +
       DatabaseConstants.COLUMN_BETWEEN + " real, " +
       DatabaseConstants.COLUMN_BEGIN + " real, " +
       DatabaseConstants.COLUMN_FINISH + " real, " +
       DatabaseConstants.COLUMN_MEAN + " integer, " +
       DatabaseConstants.COLUMN_RANDOM + " integer);";

db.execSQL(createTableSettings);
Log.d("CREATETABLE",createTableSettings);
```

4.4 Dettaglio delle funzioni del database manager utilizzate

```
/**
 * Create a new record with the specified name
 * @author Loris
 */

public void newSettingsRecord(String name, float[] settings) {

  //...
        ContentValues values;
        values = new ContentValues();

        values.put(DatabaseConstants.COLUMN_NAME, name);
        values.put(DatabaseConstants.COLUMN_BETWEEN, settings[0]);
        values.put(DatabaseConstants.COLUMN_BEGIN, settings[1]);
        values.put(DatabaseConstants.COLUMN_FINISH, settings[2]);
        values.put(DatabaseConstants.COLUMN_MEAN, (int)settings[3]);
        values.put(DatabaseConstants.COLUMN_RANDOM, (int)settings[4]);

        sqlite.insert(DatabaseConstants.TABLE_SETTINGS, null, values);
        sqlite.close();

  //...
}
```

```java
/**
 * Save given settings in the specified record, if exists (if not, create
 * it)
 * @author Loris
 */

public void saveSetting(String name,
  float between, float begin, float finish, int mean, int random) {

  deleteSettingsRecord(name);

  //...
        ContentValues values;
        values = new ContentValues();

        values.put(DatabaseConstants.COLUMN_NAME, name);
        values.put(DatabaseConstants.COLUMN_BETWEEN, between);
        values.put(DatabaseConstants.COLUMN_BEGIN, begin);
        values.put(DatabaseConstants.COLUMN_FINISH, finish);
        values.put(DatabaseConstants.COLUMN_MEAN, mean);
        values.put(DatabaseConstants.COLUMN_RANDOM, random);

        sqlite.insert(DatabaseConstants.TABLE_SETTINGS, null, values);

  //...
}



/**
 * Delete the record with the specified name, if exists
 * @author Loris
 */
public void deleteSettingsRecord(String name) {

  //...
        SQLiteDatabase sqlite = this.getReadableDatabase();
        sqlite.delete(DatabaseConstants.TABLE_SETTINGS,
                    DatabaseConstants.COLUMN_NAME+"='"+name+"'", null);

  //...
}
```

```java
/**
 * Return an array containing the set of settings corresponding
 * to the record of the specified name
 * @author Loris
 */

public float[] getSettingsByName(String name) {

  float[] settings = null;

  //...
          settings = new float[5];
          String[] columns = { DatabaseConstants.COLUMN_BETWEEN,
              DatabaseConstants.COLUMN_BEGIN,
              DatabaseConstants.COLUMN_FINISH,
              DatabaseConstants.COLUMN_MEAN,
              DatabaseConstants.COLUMN_RANDOM };
          Cursor cursor =
              sqlite.query(DatabaseConstants.TABLE_SETTINGS, columns,
              DatabaseConstants.COLUMN_NAME + "='" + name + "'", null, null,
              null, null);

          if (cursor != null) {
              if (cursor.moveToFirst() ) {
                      settings[0]=cursor.getFloat(0);
                      settings[1]=cursor.getFloat(1);
                      settings[2]=cursor.getFloat(2);
                      settings[3]=(float)cursor.getInt(3);
                      settings[4]=(float)cursor.getInt(4);
              }
          }
  //...
  return settings;
}
```

5. Conclusioni

In questo elaborato sono state presentate le modifiche funzionali apportate all'applicazione “Parrot Game Discrimination”.

In particolare le funzioni di backup delle impostazioni e di editing delle playlist hanno semplificato non poco il lavoro nella fase di preparazione del gioco da parte dell’operatore incaricato di assistere il bambino.

Inoltre la funzione di save/resume permette di effettuare test più lunghi o complessi con il medesimo bambino, dandogli la possibilità di riposarsi tra un livello e l’altro e facendo quindi in modo che anche gli ultimi livelli siano svolti con un alto livello di concentrazione.

In questo modo si possono evitare eventuali risposte “casuali” (che andrebbero ad intaccare i risultati ottenuti nei livelli precedenti e quindi la valutazione globale del test) date da un bambino sfinito o che non ha più voglia di continuare il gioco.

Il software realizzato è stato sperimentato con successo nel corso di una tesi di laurea specialistica in psicologia [2]. In particolare, il Parrot Game è stato utilizzato per accertare le capacità discriminatorie dell'udito in 27 bambini con sviluppo tipico e 14 bambini affetti da sindrome di Down, permettendo di evidenziare importanti tendenze e relazioni tra le capacità discriminatorie e l'evoluzione della produzione linguistica.

Bibliografia

Fonti utilizzate:

[1] GOOGLE. **Android Developer's Guide,** http://developer.android.com.

[2] CHIARA SQUADRANI. **Discriminazione fonologica e vocabolario recettivo nella Sindrome di Down. Il ruolo della discriminazione di suoni non linguistici,** introduzione, Master's thesis, University of Padova, 2013.

[3] SERENA ZANOLLA. **Interactive and Multimodal Environments for Learning,** capitolo 6, Doctoral thesis, University of Udine, 2013.

Ulteriori approfondimenti:

[4] Y. KELLER-BELL AND R. A. FOX. **A Preliminary Study of Speech Discrimination in Youth with Down Syndrome. Clinical Linguistics and Phonetics**, 305–317, Aprile 2007.

[5] L. K. BENNETTS AND M. C. FLYNN. **Improving the Classroom Listening Skills of Children with Down Syndrome by using Sound-Field Amplification. Down Syndrome Research and Practice**, 19–24, 2002.

[6] J. DENNIS. **Hearing Problems in People With Down's Syndrome. Notes for Parents and Carers**, 2001.

[7] B. MCCORMICK. **The Medical Practioner's Guide to Paediatric Audiology**. Cambridge University press, 1995.