UNIVERSITÀ DI PADOVA **FACOLTÀ DI INGEGNERIA**

TESI DI LAUREA

SOUNDRISE: SVILUPPO E VALIDAZIONE DI UN'APPLICAZIONE MULTIMODALE INTERATTIVA PER LA DIDATTICA BASATA SULL'ANALISI DI FEATURE VOCALI

Laureando: Marco Randon

Relatore: prof. Federico Avanzini
Correlatore: dott.ssa Serena Zanolla

Corso di Laurea Magistrale in Ingegneria Informatica

17 luglio 2012

Anno Accademico 2011/2012

Sommario

In questo lavoro si presentano i passi e gli strumenti utilizzati per lo sviluppo di *SoundRise* a partire da un prototipo *Pure Data* e i primi test effettuati per valutare e migliorare il risultato ottenuto.

SoundRise è un'applicazione multimodale e interattiva a fini didattici, finalizzata a proporre ai bambini della scuola primaria una modalità alternativa di apprendimento delle caratteristiche del suono. Il bambino può esplorare tali caratteristiche attraverso una rappresentazione grafica in tempo reale delle proprie produzioni vocali.

Protagonista di questo gioco è il sole e mediante la sua posizione sull'orizzonte di un paesaggio stilizzato, le sue dimensioni e il suo colore si cerca di dare una raffigurazione coerente delle caratteristiche sonore.

L'altezza del sole sull'orizzonte corrisponde all'altezza del tono prodotto dall'utente, la sua dimensione all'ampiezza della produzione, mentre la durata è rappresentata dal viso sorridente del sole che apre o chiude gli occhi in presenza o assenza di produzione vocale. Non potendo ottenere un'immagine ragionevole del timbro partendo dall'analisi delle caratteristiche della voce, quest'ultimo è sostituito con una visualizzazione delle cinque vocali della lingua italiana associando a ciascuna un colore col quale disegnare il sole.

Le caratteristiche possono essere ispezionate singolarmente oppure tutte assieme, lasciando piena libertà di scelta al bambino.

Si è cercato di proporre un'interfaccia semplice e amichevole, che fosse intuitiva e non causasse confusione o distraesse l'utente dall'obiettivo dell'applicazione, basandosi su alcune delle caratteristiche indicate come importanti per una applicazione di questo tipo nel lavoro di Anne-Marie Öster [1] dove si valuta l'applicazione clinica di un'applicazione di *speech training* nel trattamento di bambini affetti da deficit uditivo.

Tra gli obiettivi di questo lavoro vi è lo studio delle potenzialità offerte dall'uso di *Pure Data* e *libpd*, una libreria *C++* che consente di integrare un'istanza di *Pure Data* all'interno di una ap-

plicazione qualsiasi. Questa libreria permette di utilizzare *Pure Data* per creare in modo semplice e veloce un prototipo di ciò che si vuole realizzare e di convertirlo successivamente in un'applicazione multi-piattaforma facilmente distribuibile, in quanto non dipendente dalla presenza di *Pure Data* nel *computer* dell'utente. Molto interessante è la possibilità di creare applicazioni per i principali sistemi operativi mobili a partire dallo stesso codice.

Si è cercato inoltre di strutturare il progetto in modo da ottenere una piattaforma versatile e adattabile alla sperimentazione delle tecnologie di analisi audio nel trattamento di patologie che interessano la produzione vocale di base.

Questo progetto è stato testato su sistemi *Apple Mac OS X*, *Apple iOS*, *Microsoft WindowsXP* e *Microsoft Windows7*. Il test e l'eventuale adattamento su piattaforme *GNU/Linux* e *Google Android* saranno oggetto di lavori futuri.

Nel primo capitolo si propone una selezione di lavori riguardanti applicazioni a supporto di logopedisti e terapeuti nel trattamento di patologie che riguardano la produzione vocale, sia dovute a deficit fisici che neurologici. Infine si propone una panoramica sul campo molto recente delle applicazioni per dispositivi mobili a supporto di persone affette da disturbo autistico, dei loro familiari o dei terapeuti che si occupano di queste sindromi.

Nel secondo capitolo si presentano gli obiettivi di questo lavoro, un'analisi degli strumenti utilizzati, le caratteristiche dei progetti per lo sviluppo delle tre versioni dell'applicazione e le linee guida da rispettare per realizzare una *patch Pure Data* adatta ad essere utilizzata con *libpd*. Infine si descrivono le parti più interessanti del codice sorgente prodotto.

Nel terzo capitolo si illustrano le caratteristiche offerte dell'interfaccia grafica dell'applicazione e le modalità di utilizzo.

Infine nel quarto capitolo vengono esposti i risultati di un test sull'usabilità dell'applicazione sottoposto a un pubblico eterogeneo di utenti.

Ringraziamenti

Desidero ringraziare i proff. Federico Avanzini, Antonio Rodà e Sergio Canazza per la possibilità di realizzare questo lavoro e per il sostegno e l'aiuto forniti.

Un ringraziamento particolare va alla dott.ssa Serena Zanolla per l'idea iniziale, il piacevole confronto riguardo le scelte di progetto, il continuo sostegno e il fondamentale contributo nelle fasi finali di test. Si ringrazia i bambini e il personale della Scuola Primaria Elisa Frinta di Gorizia per la collaborazione durante i test dell'applicazione.

Non potrò mai ringraziare a sufficienza mamma e papà per il loro amore e la loro guida nell'oceano di esperienze che è la vita; i doni più preziosi che potessero offrirmi. Grazie a Gianni e Pamela per essere dei buoni fratelli e a tutta la grande famiglia in cui ho avuto il piacere di percorrere questi anni della mia vita.

Un ricordo particolare al Capitaño per l'importante e duraturo esempio di vita umile e sincera. E grazie a mia zia suor Maria Margherita che mi ricorda sempre nelle sue preghiere.

Come non ricordare la mia seconda famiglia, quelle persone che mi hanno fatto dono della loro amicizia e che condividono con me le parti più importanti della loro vita.

Grazie agli illustri professori del Conservatorio di Vallorcola coi quali ho speso momenti importanti di crescita personale e musicale, all'Appartamento dove ho passato quattro anni meravigliosi che mai potrò dimenticare.

Grazie agli amici conosciuti durante questi anni di studio, alla "crew" della Da, e naturalmente a Stefano che ha condiviso le fatiche e le soddisfazioni di questo lavoro di tesi.

Grazie "alle botti" ritrovo abituale per lo spritz del mercoledì, l'obiettivo motivazionale che consente di girare la boa del metà settimana...

Infine un ringraziamento speciale ad "Ale." per la pazienza e il coraggio di leggere e correggere questa tesi e per i preziosi consigli di stile.

Indice

Sommario **i**

Ringraziamenti **iii**

1 Applicazioni a supporto dei logopedisti e app di aiuto a persone affette da disturbo autistico **1**

2 Sviluppo applicazione *stand-alone* **7**
2.1 Obiettivi 7
2.2 Ricerca degli strumenti adatti allo sviluppo 10
2.3 Ambienti di sviluppo 11
2.4 *libpd* 12
2.5 *openFrameworks* e *ofxPd* 14
2.6 Trasformazione di una *patch Pure Data* per l'utilizzo con *libpd* 16
2.7 Descrizione dei progetti software 24
2.8 Descrizione del codice 27
2.8.1 I *file main* e *testApp* 27
2.8.2 *AppCore*: il cuore dell'applicazione 32
2.8.3 Le classi di supporto 35

3 L'interfaccia utente di SoundRise **37**
3.1 La selezione delle caratteristiche da analizzare 38
3.2 La pagina di *setup* 39

4 Validazione dell'applicazione **43**
4.1 Test sull'usabilità dell'interfaccia e sul funzionamento dell'applicazione 43

5 Conclusioni **47**

Appendici **51**

A Codice **51**
A.1 Licenza 51
A.2 La dichiarazione della classe AppCore 52
A.3 L'implementazione della classe AppCore 53
A.4 La classe di supporto srSole 64
A.5 La classe di supporto srButton 70
A.6 Le definizioni delle costanti 72

Elenco delle figure

1.1 Proloquo2Go, l'app di comunicazione alternativa per Apple iPad creata da AssistiveWare. 5

2.1 Console di comando del prototipo SoundRise 8
2.2 Esempi del feedback visivo del prototipo per le quattro catteristiche del suono e per i quattro paesaggi proposti 9
2.3 La patch SoundRiseCore usata come DSP nell'app SoundRise 18
2.4 Segnali di configurazione della patch SoundRiseCore.pd 19
2.5 Blocchi di (a) ingresso audio, (b) ricezione codice sezione e (c) riconoscimento delle vocali 19
2.6 Il blocco di analisi del pitch 21
2.7 Blocchi di analisi dell'ampiezza e della durata 22
2.8 Esempio di espressione calcolata con l'oggetto `[expr]` o con gli oggetti matematici di Pure Data 23
2.9 L'albero delle directory dei progetti (a) iOS, (b) Mac OS X e (c) Windows 25

3.1 L'applicazione SoundRise all'apertura 38
3.2 Esempi del feedback visivo dell'applicazione SoundRise per l'analisi di intensità, altezza, durata, vocali e di tutte le caratteristiche assieme 40
3.3 La finestra di setup di SoundRise 41

Elenco delle tabelle

4.1 Risultati dei test sull'usabilità e il funzionamento dell'applicazione 45

Elenco del codice

2.1 main.cpp 28
2.2 main.cpp versione iOS 28
2.3 testApp.h 29
2.4 testApp.h versione iOS 30
2.5 L'inizializzazione dell'applicazione in testApp.cpp 31
2.6 L'inizializzazione dell'applicazione in testApp.mm versione iOS 32
A.1 AppCore.h 52
A.2 AppCore.cpp 53
A.3 srSole.h 64
A.4 srSole.cpp 65
A.5 srButton.h 70
A.6 srButton.cpp 70
A.7 srConstants.h 72

Capitolo 1

Applicazioni a supporto dei logopedisti e app di aiuto a persone affette da disturbo autistico

Dagli anni '90 la disponibilità di strumenti come *personal computer* sufficientemente performanti per gestire informazioni audiovisive ed effettuare analisi e *processing* audio ha consentito di studiare e progettare sistemi di assistenza all'apprendimento del parlato e alla correzione di eventuali difetti nella produzione vocale. L'interesse di terapeuti e insegnanti [2] riguardo sistemi di questo tipo ha ulteriormente supportato queste ricerche e l'evoluzione dei *software* di *speech training* utilizzabili anche con persone affette da gravi deficit uditivi [1], [3].

L'apprendimento del parlato avviene nel periodo della prima infanzia, proprio negli anni in cui la velocità di sviluppo e la capacità di apprendimento si presentano al massimo delle potenzialità [4], e, in presenza di deficit uditivi e altri deficit neurologici come le sindromi di Down o Asperger, possono insorgere problemi nella produzione vocale che impediscono un pieno inserimento sociale dell'individuo.

Le terapie tradizionali richiedono la presenza di personale qualificato come negli incontri terapeuta-paziente o della supervisione dei genitori nelle sessioni di esercitazione in ambiente domestico, ma non prevedono sistemi efficaci per consentire al paziente di esercitarsi in autonomia. Le tecnologie di riconoscimento del parlato non solo possono venire in aiuto al terapeuta fornendo strumenti avanzati e flessibili in grado di agevolarne il lavoro, ma soprattutto sono in grado di offrire mezzi semplici e funzionali all'esercizio autonomo del paziente nella produzione vocale. In questo senso grande importanza assume la presentazione grafica delle caratteristiche sonore e il *design* dell'interfaccia di utilizzo che non dovrebbe guardare soltanto alle esigenze del personale qualificato, ma soprattutto alla capacità di veicolare le necessarie informazioni al-

l'utente al quale si rivolgono i programmi e le terapie, ovvero il bambino. Come dice Ling [5] "l'aspetto cruciale non è la quantità di informazioni che possono essere presentate al bambino, ma quanto può percepire e in che modo può imparare a processarle".

Rannan [3], un *software* di assistenza nella riabilitazione uditiva di bambini con impianto cocleare, funge da gestore di sessione di terapia in cui il terapista, i genitori ed i pazienti stessi possono progettare e personalizzare le sessioni selezionando i materiali rieducativi e regolandone i parametri principali. Il sistema copre le fasi principali di riabilitazione ed utilizza migliaia di materiali terapeutici come toni puri, rumori ambientali, parole mono e multi sillabiche, contrasti fonetici, parole familiari, frasi familiari, melodie e immagini. Il funzionamento consiste nel proporre dei materiali sonori al paziente che deve scegliere una immagine da associarvi tra quelle presentate in una interfaccia grafica adatta a un bambino, semplice e in stile cartone animato.

SPECO [6], [7], un progetto europeo per lo sviluppo di un sistema di *speech training* multilingua, ha prodotto un *software* audiovisivo di supporto alla correzione della pronuncia e all'apprendimento del parlato indirizzato ai bambini tra i cinque e i dieci anni di età. Le correzioni della pronuncia avvengono presentando in tempo reale una rappresentazione visuale dei parametri del parlato in modo comprensibile e interessante per i bambini pur rimanendo corretto dal punto di vista acustico-fonetico. Al paziente viene proposto un materiale sonoro da replicare finché la rappresentazione visiva della propria produzione vocale non si avvicina a quella del materiale presentato, utilizzando cioè un feedback visivo e uditivo durante l'esercitazione vocale. Questa modalità di funzionamento consente al paziente un utilizzo in autonomia.

STAR [2] si propone come un sistema *software* capace di distinguere tra una "buona" e una "insufficiente" pronuncia di una parola prodotta dal bambino in risposta a uno stimolo testuale, visivo o verbale da un vocabolario di 1000 parole. Jones [8] definisce "insufficiente" una produzione difficilmente comprensibile dalla maggior parte delle persone, caratterizzata da borbottii o mancanza di definizione delle espressioni. Al contrario una "buona" produzione vocale permette al bambino di partecipare con sicurezza nelle comunicazioni pubbliche quotidiane e consente una lettura e una pronuncia accurata. Questo progetto si avvale dei modelli nascosti di Markov [9], una tecnologia di grande successo nella modellazione di sequenze vocali molto utilizzata nei sistemi automatici di riconoscimento del parlato. Questa tecnica è stata ampiamente studiata nel riconoscimento della voce negli adulti e si basa su una stima statistica dei parametri vocali estratti da un numero consistente di produzioni fonetiche. E' facile comprendere, dunque, come il riconoscimento del parlato dei bambini sia problematico da parte di questi sistemi, non essendo questi addestrati con parametri estratti da produzioni di bambini. Questa ricerca si è concentrata

nel raccogliere un consistente numero di parametri vocali estratti da circa 16000 parole pronunciate da bambini in lingua inglese e nel valutare le prestazioni di un sistema HMM addestrato con questi campioni. Sebbene gli autori non abbiano curato un'interfaccia specifica per l'utilizzo da parte di bambini, ai quali viene soltanto richiesto verbalmente, graficamente, testualmente o con una combinazione di questi stimoli di pronunciare la parola desiderata, i risultati sono stati promettenti. Inoltre si è visto come i bambini non si sentano intimiditi dal parlare a un computer, considerato come un interlocutore che non giudica e sentendosi liberi di approcciarsi alla pronuncia di parole sconosciute.

Speech Illumina Mentor (SIM) [10] si propone come un *software* di *speech training* indirizzato ai bambini in età scolare con deficit uditivo e si avvale di due componenti: un modulo di addestramento che presenta al bambino esempi di produzioni e un modulo di pratica che presenta in forma di gioco un sistema di riconoscimento vocale che analizza le produzioni del bambino. Il modulo di addestramento presenta una animazione del movimento articolatorio esterno e interno, nel secondo caso mostra il movimento dei componenti del tratto orofaringeo normalmente nascosti, come lingua, palato duro, palato molle e faringe. La visualizzazione di queste animazioni avvengono in modo simile all'utilizzo di un riproduttore video dove tutti i parametri della riproduzione sono accessibili (*play*, *stop*, *forward*, *rewind*, *zoom* e *volume*). Durante una sessione di pratica il sistema analizza i tentativi del bambino di produrre la sillaba richiesta e propone una risposta visuale per indicare il grado di successo o fallimento. Si utilizza uno stile giocoso e personaggi animati per stabilire una buona comunicazione e motivare l'utente. SIM può essere visto come una serie di moduli didattici specifici per l'apprendimento dei diversi fonemi con diversi gradi di difficoltà, tutti utilizzanti la struttura vista precedentemente (un componente di addestramento e uno di pratica).

ARTUR [11] è invece un progetto pluriennale del *KTH Royal Institute of Technology* per lo studio e lo sviluppo di un sistema computerizzato di insegnamento del parlato con correzione articolatoria mediante la visualizzazione di un modello tridimensionale del tratto orofaringeo. Si tratta di un progetto complesso ancora in corso che al 2005 presentava solamente il modello tridimensionale del tratto vocale come componente completata e utilizzabile, oltre a una interfaccia grafica di base per i test. Le pubblicazioni più recenti presentano ricerche riguardanti la mappatura delle caratteristiche sonore con il movimento del modello tridimensionale o delle valutazioni sull'utilità di sistemi di questo tipo nell'insegnamento del parlato. Questi esperimenti utilizzano una configurazione chiamata "Mago di Oz", dove l'analisi delle produzioni vocali è demandata a un terapeuta in quanto le componenti di analisi audio non sono ancora complete. È

il terapeuta che indica al sistema come rispondere alle produzioni vocali dell'utente.

Un sistema che molto si avvicina all'idea del lavoro presentato in questa tesi è visiBabble [12], un *software* che si prefigge l'obiettivo di aumentare la vocalizzazione nei primi mesi di vita. L'interfaccia, estremamente semplice data l'età degli utenti, presenta alcuni personaggi che si animano in tempo reale in risposta alle produzioni sillabiche, le attività sono iniziate dall'infante, sono focalizzate sull'infante e non includono eventi di addestramento di alcun tipo.

Un altro campo significativo in cui le tecnologie informatiche stanno avendo risultati molto positivi riguarda il trattamento di persone e bambini affetti da disturbi dello spettro autistico, una sindrome ancora poco compresa, che include una varietà di sintomatologie e per la quale sono state proposte teorie e terapie molto diverse nel secolo scorso [13]. Si rileva un'evoluzione della definizione clinica: inizialmente questi disturbi erano considerati come una forma di schizofrenia, successivamente si ipotizzò che potesse trattarsi di una forma della sindrome di Down, almeno fino agli anni '40 del Novecento quando si iniziò a considerarla una specifica sindrome patologica grazie a Leo Kanner, il quale parlò di "autismo infantile precoce".

Gli individui affetti da autismo mostrano un ampio spettro di difficoltà e disabilità, che possono influenzare notevolmente le capacità intellettive generali, tuttavia si possono identificare una serie di sintomi comuni. La *National Autistic Society* (*NAS* 2003) individua nei seguenti tre i sintomi più frequenti:

- ridotta interazione sociale (difficoltà nelle relazioni sociali, incapacità di relazionarsi agli altri in modo significativo, ridotta capacità di comprendere i sentimenti altrui);
- ridotta comunicazione sociale (difficoltà di comunicazione verbale e non verbale, ad esempio, difficoltà nella comprensione del significato dei gesti, delle espressioni facciali o del tono della voce);
- ridotta immaginazione (difficoltà nello sviluppo del gioco e della fantasia, ad esempio, avere una gamma limitata di attività creative, in molti casi copiate e perseguite in modo rigido e ripetitivo).

Inoltre spesso si osserva una forte resistenza al cambiamento nella routine, la presenza di comportamenti ripetitivi associati ad un repertorio significativamente ridotto di attività e di interessi e la necessità a volte ossessiva di ambienti stabili e conosciuti. A differenza di un handicap fisico, che impedisce alle persone di interagire fisicamente con l'ambiente, le persone affette da autismo hanno difficoltà a dare un senso al mondo, in particolare il mondo sociale. L'autismo può inoltre essere accompagnata da difficoltà di apprendimento.

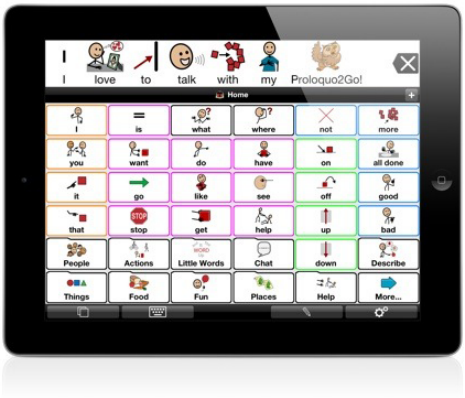


Figura 1.1: *Proloquo2Go, l'app di comunicazione alternativa per Apple iPad creata da AssistiveWare.*

Recentemente, la disponibilità e l'inserimento nelle terapie di strumenti informatici mobili come i *tablet* o agenti robotici autonomi [14] sta facendo emergere nuove possibilità e ipotesi.

È interessante in questo senso il caso di un ventisettenne affetto da mancanza di comunicazione verbale che ora, grazie all'uso dell'applicazione *Proloquo2Go* [15] per *Apple iPad*, riesce a comunicare con sicurezza nella vita quotidiana e persino a rispondere alle domande di una intervistatrice di un programma televisivo [16].

Proloquo2Go (Figura 1.1) è una soluzione per dispositivi *iOS* che fornisce un tipo di comunicazione alternativa alle persone con difficoltà espressiva o prive di comunicazione verbale. Un sintetizzatore vocale genera parole o frasi in risposta al tocco sui simboli presentati a video che rappresentano oggetti di uso comune o azioni, o sui tasti della tastiera virtuale con predizione delle parole digitate.

Sul sito *TouchAutism* [17] si può trovare un buon numero di applicazioni per sistemi mobili a supporto delle terapie per bambini autistici. Per alcune si tratta di esempi di come convertire gli strumenti utilizzati nelle terapie tradizionali in giochi multimediali che grazie alla combinazione di risposte audio e video aiutano a conquistare l'interesse del bambino. Altre invece sono applicazioni a supporto del terapeuta, ad esempio "*Preference and reinforcer assessment*" consente al terapeuta di individuare le preferenze del paziente tra un insieme di oggetti da utilizzare come rinforzo nelle terapie. Altre ancora sono esempi di *social story* multimediali, con una voce narrante, immagini e testo, che hanno l'obiettivo di insegnare al bambino le motivazioni di determinati comportamenti sociali, ad esempio l'importanza di giocare "a turno" con i propri

compagni, dell'igiene, di mantenere il contatto visivo durante una conversazione o di prendersi un momento e fare respiri profondi per calmarsi e non essere aggressivi o arrabbiati.

La maggior parte di queste applicazioni non prende in considerazione le produzioni vocali degli utenti, ma utilizza soltanto il dispositivo di input tattile dello schermo dei terminali mobili e reagisce con risposte audiovisive.

Una applicazione come *SoundRise* potrebbe diventare una piattaforma di sperimentazione e adattamento delle tecnologie di elaborazione audio e riconoscimento delle vocalizzazioni e del parlato alle terapie rivolte a queste patologie.

Capitolo 2

Sviluppo applicazione *stand-alone*

2.1 Obiettivi

Il lavoro qui presentato, ideato e sviluppato in collaborazione con la dott.ssa Serena Zanolla dell'Università degli Studi di Udine, e dei proff. Federico Avanzini, Antonio Rodà e Sergio Canazza del Dipartimento di Ingegneria dell'informazione dell'Università degli Studi di Padova, doveva riguardare lo sviluppo di una *app* per dispositivi *iOS* a partire dal prototipo realizzato come *patch Pure Data* [18] e valutare le potenzialità del progetto *libpd* (Paragrafo 2.4), una libreria scritta in linguaggio *C* che permette di integrare il nucleo di *Pure Data* in un qualsiasi programma.

Il prototipo permette di visualizzare con segni grafici ispirati ai disegni dei bambini le quattro caratteristiche del suono:

- altezza (*pitch*): ovvero un attributo di sensazione acustica secondo cui i suoni possono essere ordinati su una scala musicale dal basso verso l'alto [19];
- intensità: ovvero una descrizione psicoacustica dell'entità di una sensazione acustica secondo cui i suoni possono essere ordinati su una scala dal piano al forte;
- durata: ovvero l'arco di tempo in cui un suono viene percepito o emesso da un corpo in vibrazione;
- timbro: è la caratteristica che ci consente di distinguere il suono di uno strumento da quello di un altro.

In questo lavoro e nel prototipo il timbro è stato sostituito dalle cinque vocali della lingua italiana, in quanto la caratteristica principale dell'applicazione è dare all'utente la possibilità di

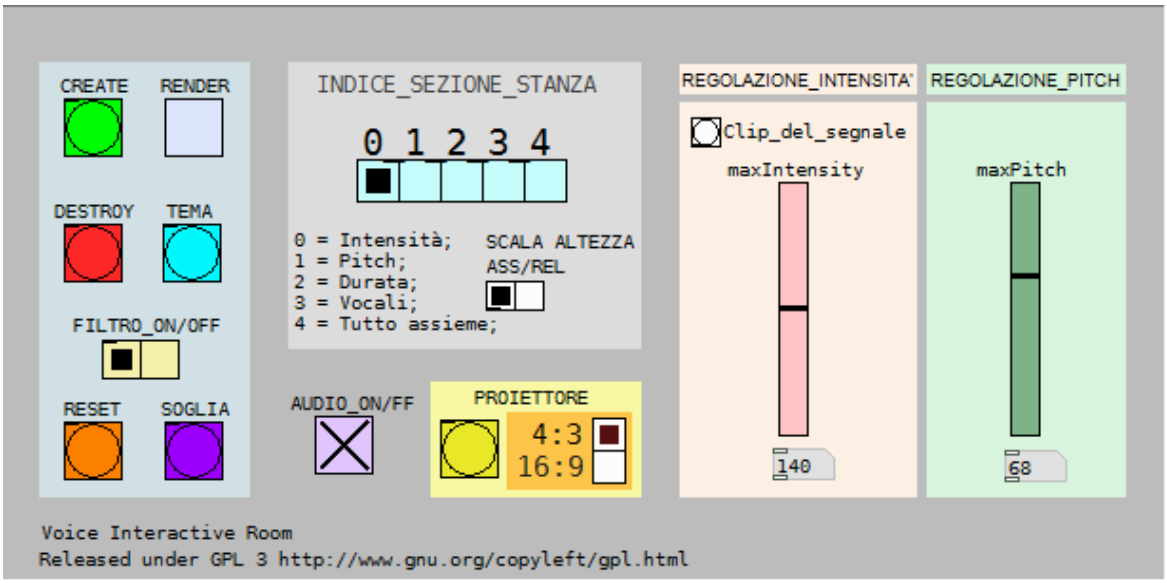


Figura 2.1: *Console di comando del prototipo SoundRise*

sperimentare queste caratteristiche con la propria voce, di conseguenza l'analisi del timbro non darebbe informazioni coerenti.

Il prototipo mette a disposizione una *console* di comando (Figura 2.1) molto minimale realizzata con gli oggetti dell'interfaccia grafica di *Pure Data* dove è possibile regolare alcuni parametri delle scale di altezza e intensità, attivare o disattivare il filtro equalizzatore in ingresso, scegliere la caratteristica audio da analizzare e aprire la finestra dove viene disegnato il *feedback* visivo, con la possibilità di posizionarla su una lavagna interattiva multimediale collegata al *computer* come monitor esterno.

Per quanto riguarda il *feedback* visivo si è scelto come protagonista un sole disegnato su un paesaggio a scelta dell'utente tra quattro proposti (Figura 2.2).

Il sole rappresenta le caratteristiche della voce secondo la seguente mappa:

- altezza ⇒ altezza del sole sull'orizzonte;
- intensità ⇒ dimensione del sole;
- durata ⇒ quando c'è suono il sole apre gli occhi e la bocca sorridente;
- vocali ⇒ colore del sole.

I colori da abbinare alle vocali sono stati scelti in base a uno studio sulle associazioni non casuali tra grafemi e colori in popolazioni sinestetiche e non [20].

(a) analisi altezza (b) analisi intensità

(c) analisi durata (d) analisi vocali

Figura 2.2: *Esempi del feedback visivo del prototipo per le quattro catteristiche del suono e per i quattro paesaggi proposti*

Gli accoppiamenti scelti sono i seguenti:

- [a] ⇒ rosso;
- [ɔ] ⇒ arancione;
- [ɛ] ⇒ verde;
- [i] ⇒ blu;
- [u] ⇒ grigio.

La presenza sempre più importante di dispositivi tecnologici anche portatili, come *smartphone* e *tablet*, nella vita quotidiana delle persone di tutte le età nei paesi tecnologicamente più avanzati giustifica l'idea di realizzare applicazioni di questo tipo che possono portare nelle case o nelle tasche degli utenti strumenti utili all'apprendimento [21].

La *app* avrebbe dovuto garantire le stesse caratteristiche e funzionalità della versione *Pure Data* integrando i controlli della *console* direttamente nella finestra di *feedback*, fornendo quindi un'interfaccia grafica più semplice, adatta agli schermi sensibili al tocco propri dei dispositivi mobili.

Durante la ricerca degli strumenti adatti alla realizzazione, grazie alle funzionalità del *framework* di sviluppo scelto, si è deciso di sviluppare una applicazione multi-piattaforma che potesse essere di più facile distribuzione data la possibilità di non dipendere da *Pure Data*.

Di seguito si elencano le caratteristiche definitive:

- indipendente dalla piattaforma;
- non richiedere una versione di *Pure Data* installata sulla macchina;
- garantire le stesse caratteristiche e funzionalità della versione *Pure Data*;
- interfaccia grafica semplice e adatta agli schermi sensibili al tocco.

2.2 Ricerca degli strumenti adatti allo sviluppo

Sebbene *SoundRise* non utilizzi una rappresentazione grafica complessa o difficilmente riproducibile con tecniche grafiche a due dimensioni, si è scelto di utilizzare le librerie per grafica tridimensionale *OpenGL* per costruire un sistema in grado di reagire agli stimoli in tempo reale.

Si ricorda che in questa fase la ricerca degli strumenti da utilizzare era focalizzata sullo sviluppo di una *app* per sistemi *iOS*.

Si sono quindi presi in considerazione alcuni tra i più conosciuti *framework opensource* per lo sviluppo di videogiochi sui sistemi mobili della *Apple* che potessero semplificare lo sviluppo dell'interfaccia grafica, tra i quali:

- *The Sparrow framework* [22], una libreria in linguaggio *Objective C*, il linguaggio nativo dei sistemi operativi di casa *Apple* dalla decima generazione, che semplifica la creazione di videogiochi a due dimensioni nascondendo l'uso delle librerie *OpenGL* con una sintassi simile al linguaggio di *scripting Adobe Flash*;
- *iSGL3D* [23], una libreria in linguaggio *Objective C* che permette di creare velocemente videogiochi con grafica tridimensionale;

- *cocos2d* [24], una libreria per videogiochi con grafica bidimensionale che si appoggia a *OpenGL* inizialmente scritta in linguaggio *Phyton* e successivamente convertita in *Objective C*;

- *Moai SDK* [25], una libreria per videogiochi con grafica bidimensionale che si appoggia a *Lua*, un linguaggio di *scripting* compilabile per dispositivi *iOS*, *Android* e *HTML5*;

- *emo-framework* [26], una libreria per videogiochi che utilizza *Squirrel* come linguaggio di *scripting*, un linguaggio orientato agli oggetti che nasconde le chiamate alle funzioni *OpenGL* e permette la compatibilità della *app* tra dispositivi *iOS* e *Android*.

Questi *framework* presentano caratteristiche piuttosto simili, ma nessuno permette di utilizzare la libreria *libpd* nello stesso progetto, presumibilmente per un conflitto tra il ciclo di *rendering* grafico di *OpenGL* e il ciclo di *processing* audio di *libpd*.

Diversamente, *openFrameworks* [27], un *toolkit* di sviluppo di applicazioni multi-piattaforma, multimediali e creative scritto nel linguaggio *C++*, non è affetto da questo problema. Questo *framework* è progettato come un contenitore di altre librerie e la sua struttura modulare consente di integrarne facilmente di nuove per aumentare le funzionalità messe a disposizione. Tra questi moduli non ufficiali, chiamati *addon*, si trova *ofxPd* che va ad aggiungere le funzionalità di *libpd* a *openFrameworks*.

La seconda caratteristica di questo *framework*, forse la più interessante ai fini di questo lavoro, è la capacità di creare con poco sforzo applicazioni indipendenti dalla piattaforma. È a questo punto che gli obiettivi del lavoro si sono allargati a comprendere una versione della applicazione funzionante per ognuno dei sistemi operativi maggiormente utilizzati, così da poter offrire un prodotto per *personal computer* che non dipenda dalla presenza di *Pure Data*.

2.3 Ambienti di sviluppo

Per non complicare eccessivamente il progetto si è scelto di sviluppare una versione di *SoundRise* per sistemi operativi *Microsoft Windows*, *Apple Mac* e *Apple iOS*, fermo restando che *openFrameworks* garantisce la compatibilità con sistemi operativi *Google Android* e *GNU/Linux*, ma si è deciso di non includere il test di queste versioni.

L'ambiente di lavoro consigliato per i sistemi *Apple* comprende l'*IDE*[1] *Xcode* [28] e gli *SDK* per i sistemi operativi desiderati, senza dimenticare il comodo *Simulatore iOS* che permette di controllare il funzionamento del proprio codice per sistemi mobili senza doverlo far girare su un dispositivo fisico. Naturalmente questo non sostituisce il test effettuato sul terminale reale, ma permette di procedere con la stesura del codice con la sicurezza data dai test e più velocemente di come sarebbe dovendo lanciare l'*app* sul dispositivo.

Per quanto riguarda invece i sistemi *Microsoft Windows*, il progetto *openFrameworks* consiglia l'*IDE opensource Code::Blocks* [29] e le librerie di sviluppo *MinGW* [30] un minimale ma completo *tool* di sviluppo *opensource* per creare applicazioni native *Microsoft Windows*.

2.4 *libpd*

Lo sviluppo di *libpd* iniziò nell'estate 2010 con un *porting* di *Pure Data* nel sistema operativo *Google Android*, prendendo come punto di partenza il tentativo di Naim Falandino, Scott Fitzgerald, Peter Kirn, and Hans-Christoph Steiner. All'epoca era già disponibile il *kit* di sviluppo nativo per *Android*, ma le applicazioni dovevano ancora essere scritte principalmente in linguaggio *Java* e non esisteva la possibilità di interfacciarsi con le *API* audio attraverso codice scritto in linguaggio *C*. Questa limitazione si rivelò un fatto positivo, poiché richiese lo sviluppo di un *wrapper Java* per *Pure Data* che fornisse una semplice *API* per il *processing* di segnali e lo scambio di messaggi. Dopo varie sistemazioni questo *wrapper* venne scomposto in due parti: un traduttore da linguaggio *Java* a linguaggio *C* chiamato *JNI* e una libreria in linguaggio *C* che divenne la prima versione di *libpd*.

Il cuore e l'anima di una libreria *DSP* [2] è una funzione *callback* di rendering che preleva campioni di ingresso e calcola campioni di uscita. Questa è la descrizione più semplice del concetto di elaborazione di un segnale, e lo scopo principale di *libpd* è quello di estrarre questa funzionalità di *Pure Data* e renderla disponibile come *callback* di elaborazione audio.

Il secondo obiettivo di *libpd* è quello di consentire lo scambio diretto di segnali di controllo e messaggi *MIDI* tra *Pure Data* e il codice cliente. *libpd* supporta lo scambio dei tipi di dato standard di *Pure Data*, come *bang*, *float*, simboli e messaggi, così come le liste e i messaggi tipizzati, fatta eccezione per i messaggi puntatore che sono l'unico tipo di messaggio non supportato.

[1] *Integrated Development Environment* (ambiente di sviluppo integrato).
[2] *Digital Signal Processing*.

In sostanza, *libpd* è costituito da variazioni della funzione *callback* di elaborazione per i diversi tipi di campioni (*short*, *float*, *double*), da un insieme di funzioni per l'invio di messaggi a *Pure Data*, e una serie di puntatori a funzione per la ricezione di messaggi da *Pure Data*. Al fine di ricevere i messaggi, il codice cliente deve implementare le funzioni di ricezione adeguate e assegnarle ai puntatori a funzione appropriati di *libpd*.

Le funzioni di *libpd* si concentrano quindi sull'elaborazione del segnale e sullo scambio di messaggi; gli aspetti di *Pure Data* che riguardano l'*editing*, le interazioni con l'interfaccia grafica e le interazioni con un sistema *desktop* sono stati deliberatamente rimossi nel processo di creazione di *libpd*. In particolare, *libpd* non include alcun tipo di *driver* audio, nessun *driver MIDI*, non comprende alcun genere di interfaccia utente, non garantisce la sincronizzazione dei *thread* e non mantiene un riferimento intrinseco del tempo. Questo approccio è essenziale allo scopo di *libpd*: poter essere incorporato nel maggior numero di applicazioni.

In ambienti come *openFrameworks*, *Python*, e *Processing*, non includere *driver* audio significa poter utilizzare l'audio nativo e i metodi di ingresso e uscita *MIDI* per ciascuno di questi ambienti, semplificando lo sviluppo e la distribuzione.

A partire dalla versione 0.42 di *Pure Data*, il flusso del tempo può essere pilotato tramite un *plugin scheduler*, che consente di effettuare a intervalli regolari le chiamate alla funzione che gestisce i tempi di elaborazione dei messaggi e del *DSP* [31]. Questo consente di rimuovere tutte le funzionalità di temporizzazione da *libpd*, e di fornire invece una funzione di *callback* `process(...)` per attivare l'elaborazione dei messaggi e dei *tick* del *DSP* a intervalli prestabiliti.

libpd tiene traccia del tempo solo in termini del numero di campioni richiesti finora (equivalentemente, in relazione al numero di chiamate alla funzione *callback* di elaborazione). Per esempio, durante l'esecuzione con la dimensione predefinita del *buffer* di 64 *frame*, ad una frequenza di campionamento di 44100 Hz, ogni invocazione della funzione di *callback* di elaborazione audio avanzerà il tempo interno di *libpd* di 64/44100 secondi. Poiché le invocazioni della funzione *callback* di elaborazione audio sono di norma generate dall'interfaccia audio, questo significa che *libpd* e l'interfaccia audio saranno sempre sincronizzati, senza richiedere ulteriori sforzi al programmatore.

Per quanto riguarda invece la gestione dei *thread* di elaborazione audio, si è deciso di lasciare il controllo a un livello più alto di astrazione dove risulta più semplice ed efficace da coordinare. L'unico problema di questa decisione riguarda l'uso di *external multi-threaded* che potrebbero non funzionare correttamente; un piccolo prezzo da pagare per la semplicità e flessibilità offerta da questo tipo di approccio.

Per quanto riguarda l'uso di *external*, sulle piattaforme che supportano le librerie caricate dinamicamente (per esempio, *Android*), è sufficiente compilarli e posizionarli sul percorso di ricerca, *libpd* li caricherà in base alle esigenze. Sulle piattaforme senza caricamento dinamico (ad esempio, *iOS*), gli *external* dovranno essere collegati staticamente e il codice cliente dovrà occuparsi di richiamare esplicitamente il metodo di configurazione di ogni *external*, con uno sforzo supplementare trascurabile.

libpd comprende dei *wrapper* per i linguaggi di programmazione *Java*, *Objective-C* e *Python* che rendono le funzionalità di *libpd* disponibili per il relativo linguaggio, effettuano la conversione tra i tipi di dati personalizzati di *Pure Data* e i tipi di dati standard del linguaggio di destinazione, forniscono un'interfaccia orientata agli oggetti per le funzioni di scambio messaggi e puntatori a funzione, e sono infine *thread-safe*.

Al fine di ricevere messaggi da *Pure Data*, il codice cliente deve implementare i metodi necessari dell'interfaccia `PdReceiver` e registrare un oggetto ricevitore tramite i metodi della classe `PdBase`. Oltre a queste funzionalità di base, i *wrapper* includono anche un insieme di classi di utilità più generale.

2.5 *openFrameworks* e *ofxPd*

openFrameworks è un *toolkit opensource* scritto in *C++* progettato per aiutare il processo creativo fornendo un quadro semplice e intuitivo per la sperimentazione. Il *toolkit* è stato progettato per inglobare diverse librerie di uso comune, tra cui:

- *OpenGL*, *GLEW*, *GLUT*, *libtess2* e *cairo* per la grafica;
- *rtAudio*, *PortAudio* o *FMOD* e *Kiss FFT* per l'*input*, l'*output* e l'analisi audio;
- *FreeType* per la gestione dei font *TrueType*;
- *FreeImage* per la gestione delle immagini;
- *Quicktime* e *videoInput* per la gestione dei flussi video;
- *Poco* per diverse altre utilità.

Il codice è scritto per garantire la compatibilità con diversi sistemi operativi: sono supportati i cinque più importanti (*Microsoft Windows*, *Apple Mac OS* e *iOS*, *GNU/Linux* e *Google Android*),

oltre a due ambienti di sviluppo integrato proprietari (*Apple Xcode* e *Microsoft Visual Studio*) e due *opensource* (*Code::Blocks* ed *Eclipse*).

È progettato per permettere al programmatore di dedicarsi alla fase creativa piuttosto che ai problemi legati alle varie piattaforme, anche se un minimo sforzo è richiesto per assecondare alcuni comportamenti differenti che queste possono presentare.

openFrameworks presenta una struttura all'interno della quale lo sviluppatore può posizionare il progetto della propria applicazione e usufruire delle funzionalità messe a disposizione dal nucleo del *toolkit* e dai vari *addon*. Le cartelle principali sono le seguenti:

- *libs*: contiene le librerie fondamentali e il nucleo di *openFrameworks*;
- *addons*: include le librerie esterne al nucleo che forniscono delle funzionalità di base molto richieste come la gestione del protocollo *XML*, del protocollo *OSC*, della grafica vettoriale oltre alla gestione delle funzionalità dei dispositivi mobili e delle particolari funzioni messe a disposizione dai sistemi operativi di questi dispositivi; inoltre contiene gli eventuali *addon* aggiunti dal programmatore per aumentare le funzionalità messe a disposizione dal *toolkit*;
- *apps*: contiene i progetti delle applicazioni sviluppate con *openFrameworks* e un insieme di applicazioni di esempio che mostrano come poter usufruire delle funzionalità del *framework*

Gli *addon* ufficiali messi a disposizione da *openFrameworks* che offrono le funzionalità viste in precedenza sono i seguenti:

- *ofx3DModelLoader*: per gestire alcuni tipi di modelli tridimensionali;
- *ofxAccelerometer*: per utilizzare gli accelerometri presenti nei dispositivi mobili;
- *ofxAndroid*: consente di interfacciarsi e usare le funzioni di base del sistema operativo *Google Android*;
- *ofxAssimpModelLoader*: per gestire dei modelli tridimensionali più avanzati;
- *ofxDirList*: fornisce dei metodi per la navigazione delle cartelle indipendenti dal sistema operativo;
- *ofxiPhone*: consente di interfacciarsi e usare le funzioni di base del sistema operativo *Apple iOS*;

- *ofxMultiTouch*: per rispondere agli eventi generati dagli schermi *multitouch*;
- *ofxNetwork*: fornisce dei metodi per utilizzare il protocollo di rete *TCP/IP* indipendenti dal sistema operativo;
- *ofxOpenCv*: permette di utilizzare le funzionalità della libreria di elaborazione video *real time OpenCV*;
- *ofxOsc*: fornisce metodi per utilizzare il protocollo *OSC* (*Open Sound Control*) per lo scambio di messaggi tra dispositivi di elaborazione audio;
- *ofxSynth*: fornisce metodi per varie tipologie di sintesi ed effetti audio;
- *ofxThreadedImageLoader*: per gestire il caricamento di un numero consistente di immagini in modo asincrono;
- *ofxVectorGraphics*: per gestire la grafica vettoriale e la creazione di documenti *EPS*;
- *ofxVectorMath*: fornisce metodi per il calcolo matriciale;
- *ofxXmlSettings*: fornisce dei metodi per semplificare il salvataggio di informazioni su file *XML*;

In questa cartella si possono collocare gli *addon* esterni, come *ofxPd* che permette di utilizzare *libpd* in applicazioni scritte utilizzando questo *framework*.

ofxPd, sviluppato da Dan Wilcox, è un *wrapper* per *libpd* specifico per *openFrameworks* e permette di utilizzare un'istanza dell'ambiente audio *Pure Data* all'interno di un'applicazione *openFrameworks*. Si possono scambiare segnali audio, messaggi e segnali *MIDI* tra le *patch Pure Data* e il codice dell'applicazione, inoltre fornisce la possibilità di creare applicazioni *multithread*.

2.6 Trasformazione di una *patch Pure Data* per l'utilizzo con *libpd*

Pure Data permette a utenti senza nessuna conoscenza di un qualsiasi linguaggio di programmazione di costruire sistemi di elaborazione audio anche molto complessi.

Le funzionalità di *libpd* consentono di utilizzare *Pure Data* come uno strumento per la creazione di prototipi da convertire poi in applicazioni che non richiedano la presenza di *Pure Data* nel sistema in cui verranno eseguite. Inoltre permette la collaborazione tra utenti che abbiano conoscenze esclusivamente nel campo dell'elaborazione audio e sviluppatori che non ne abbiano, ma che si occupino dello sviluppo dell'interfaccia e della logica della applicazione.

Le cose a cui porre attenzione nella creazione di una *patch* da utilizzare con *libpd* sono il numero di canali audio di ingresso e uscita che si vogliono utilizzare e l'insieme dei punti di invio e ricezione per lo scambio dei messaggi tra la *patch* e il codice dell'applicazione.

L'importante è costruire una *patch* che mantenga una suddivisione netta tra la parte destinata all'elaborazione audio e la parte destinata alla gestione dell'interfaccia, creando i necessari oggetti `[send]` e `[receive]` per lo scambio dei messaggi tra le varie parti. Il codice cliente utilizzerà questi stessi oggetti per comunicare con la *patch*, inviando messaggi derivanti dagli eventi dell'interfaccia grafica e da eventuali sensori o aggiornando l'interfaccia grafica in risposta a messaggi ricevuti dalla *patch*. Lo sviluppatore potrà quindi disinteressarsi del funzionamento interno della *patch* e preoccuparsi soltanto di caricarla e usarla come una *black box*. Per quanto riguarda lo scambio dei segnali audio tra la *patch* e il codice cliente, *libpd* utilizza gli oggetti standard di ingresso e uscita eventualmente presenti, gli stessi che si usano in qualsiasi *patch Pure Data* per il collegamento all'interfaccia audio.

Nel caso del lavoro qui presentato, il prototipo *Pure Data*, oltre a fornire una semplice interfaccia grafica, utilizza la libreria *GEM* (*Graphics Environment for Multimedia*) per gestire il *feedback* visivo in risposta all'elaborazione dei parametri vocali. Nella fase di sviluppo dell'applicazione *stand-alone* si è quindi provveduto a separare definitivamente la parte di elaborazione audio, esponendola nella *patch* `SoundRiseCore.pd` (Figura 2.3), dalla parte che si occupa di gestire l'interfaccia grafica e il *feedback* visivo.

Particolare attenzione si è rivolta ai segnali di configurazione della *patch* che vengono inviati dalla *patch* stessa in fase di caricamento. Se riguardano soltanto la parte di elaborazione audio e non devono essere controllati dall'utente utilizzatore dell'applicazione, questi devono essere presenti nella *patch* di elaborazione audio per una corretta inizializzazione (Figura 2.4). In questo caso si inserisce un oggetto `[receive]` per far partire questi segnali di configurazione che verrà utilizzato sia dal codice cliente in fase di inizializzazione dell'applicazione, sia dalla *patch* destinata alla gestione dell'interfaccia grafica per quanto riguarda il prototipo *Pure Data*. In questo modo lo sviluppo del prototipo e dell'applicazione *stand-alone* possono proseguire in contemporanea semplificando l'aggiornamento di entrambi in caso di modifiche. Inoltre la suddivisione

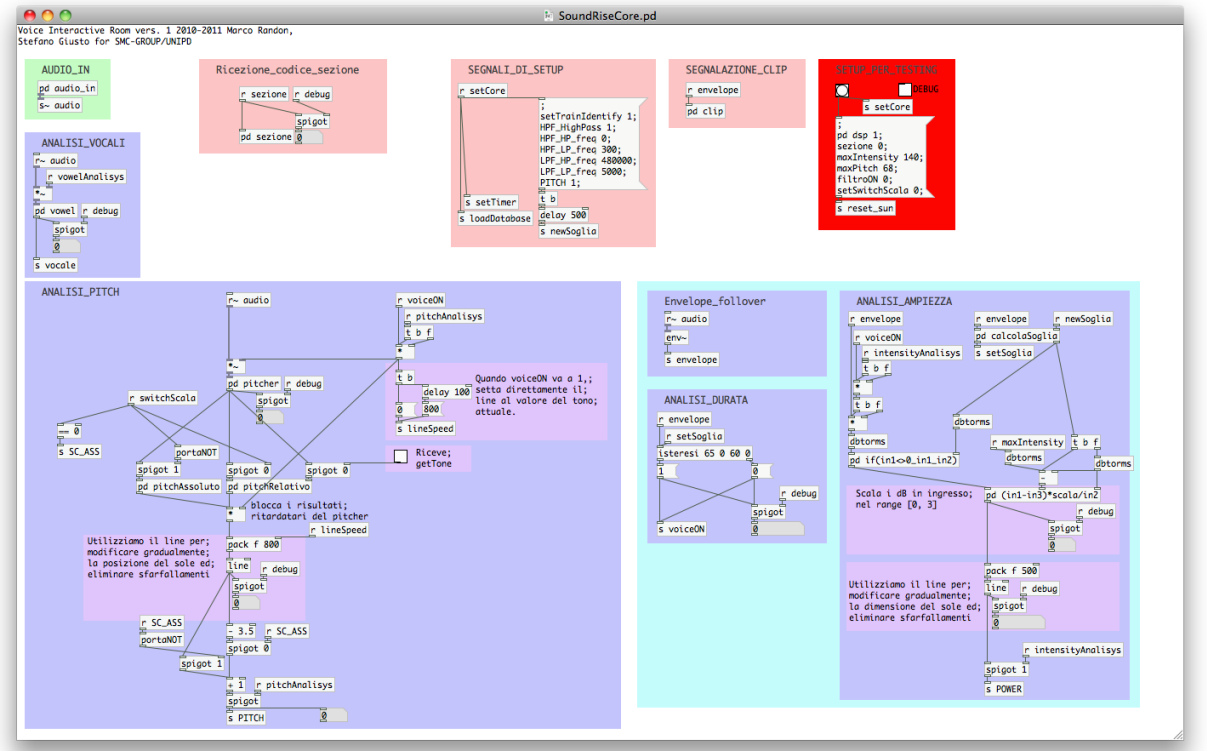


Figura 2.3: *La patch SoundRiseCore usata come DSP nell'app SoundRise*

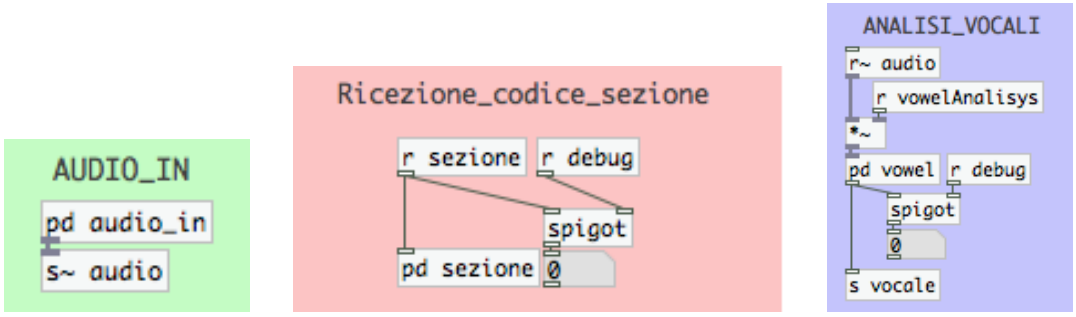
della *patch* in blocchi logici semplifica la ricerca e la correzione di eventuali errori o la valutazione di soluzioni alternative in un blocco di analisi dei parametri del suono senza preoccuparsi che le modifiche interessino gli altri blocchi.

In particolare i segnali qui utilizzati servono per inizializzare i filtri del segnale audio in ingresso, il valore del *pitch* che definisce la posizione iniziale del sole e la modalità di funzionamento del classificatore delle vocali. Nel blocco evidenziato in rosso chiamato "Setup per testing" sono presenti i segnali che dovranno essere inviati dal codice nella fase di inizializzazione dell'applicazione e che qui vengono usati per poter testare la *patch* `SoundRiseCore.pd` senza bisogno di caricare anche la *patch* dedicata all'interfaccia grafica.

In Figura 2.5 sono riportati i blocchi che gestiscono l'ingresso audio, la ricezione del codice di sezione e l'analisi delle vocali.

Il blocco di ingresso audio (Figura 2.5(a)), ovvero il riferimento al quale *Pure Data* da una parte e il codice cliente dall'altra inviano lo *stream* proveniente dall'interfaccia audio, attenua le frequenze del segnale che non portano informazioni utili lasciando inalterate le frequenze tipiche

Figura 2.4: *Segnali di configurazione della patch SoundRiseCore.pd*



(a) (b) (c)

Figura 2.5: *Blocchi di (a) ingresso audio, (b) ricezione codice sezione e (c) riconoscimento delle vocali*

della voce. Si utilizza un filtro passa banda composto da un banco di 4 filtri di Butterworth del terzo ordine che garantiscono una risposta piatta nella banda passante [32], due passa-basso con frequenza di taglio di 300 Hz e due passa-alto con frequenza di taglio di 5000 Hz. Questa prima elaborazione permette al sistema di esibire un comportamento adeguato anche in ambienti rumorosi.

Il blocco di ricezione del codice di sezione (Figura 2.5(b)) indica quale caratteristica del suono la *patch* debba analizzare e riceve questa informazione dalla *patch* di gestione dell'interfaccia grafica, dal codice cliente oppure da dispositivi esterni tramite messaggi *OSC*, questo per poter funzionare in collaborazione con la Stanza Logo-Motoria [33].

Il blocco di analisi delle vocali (Figura 2.5(c)) definisce una *subpatch* dove viene usata la libreria *timbreID* [18, 34], una collezione di *external Pure Data* in grado di estrarre caratteristiche audio quali *cepstrum*, *MFCC* (*Mel-Frequency Cepstral Coefficients*) e *BFCC* (*Bark-Frequency*

Cepstral Coefficients) tra i più importanti [35], e un classificatore che accetta liste arbitrarie di caratteristiche sonore e cerca di trovare la corrispondenza migliore tra le caratteristiche sonore in ingresso e quelle memorizzate in una fase iniziale di allenamento. In questo lavoro si usa l'*external* `[bfcc~]` che garantisce buoni risultati nel riconoscimento delle vocali e si è allenato il classificatore con 60 vettori di caratteristiche sonore per ogni vocale e altrettanti vettori che descrivono l'*input* del microfono in assenza di produzione vocale. Le caratteristiche sonore sono state ottenute da una selezione di sei registrazioni audio di bambini di 10 e 11 anni di età ottenendo 10 campioni di caratteristiche per ogni vocale. Come si vede nella tabella (4.1) e in [18] il riconoscimento delle vocali presenta dei problemi di affidabilità che potrebbero dipendere dalle registrazioni utilizzate nella fase di addestramento del classificatore e richiede un ulteriore lavoro di studio.

Il blocco di analisi del pitch (Figura 2.6) utilizza l'oggetto *Pure Data* `[fiddle~]` per la stima del *pitch* del segnale audio secondo l'algoritmo di Terhardt [36] e si preoccupa di convertirlo in una scala adatta alla visualizzazione grafica. Le scale utilizzate per la rappresentazione del *pitch* sono due: una assoluta che considera un intervallo di due ottave, da circa 100 Hz a poco più di 400 Hz in modo da adattarsi alle voci di bambini e adulti sia maschili che femminili, e una relativa che prende in esame un intervallo di due quinte, o sette semitoni, intorno al tono medio dell'utente che può essere registrato in una fase di configurazione.

In Figura 2.7 si possono trovare i blocchi relativi all'analisi della durata e dell'ampiezza. L'oggetto *Pure Data* `[env~]` restituisce l'ampiezza *RMS* (*Root Mean Square*) del segnale e questa viene confrontata con una soglia minima che rappresenta il rumore ambientale. Questa analisi identifica la presenza o assenza di un suono da analizzare e viene usata per rappresentare la durata. Inoltre l'ampiezza *RMS* viene convertita in una scala adatta alla sua visualizzazione grafica. La soglia che rappresenta il rumore ambientale viene calcolata nella fase di inizializzazione della *patch* e può essere ricalcolata dall'utente in qualsiasi momento inviando un segnale opportuno alla *subpatch* `[pd cal-colaSoglia]`.

Un altro aspetto al quale porre attenzione nell'uso degli *external* su dispositivi *iOS* riguarda le licenze di distribuzione. Per questioni forse legate alla sicurezza o alle prestazioni e al risparmio energetico, *Apple* impedisce l'uso di librerie di terze parti collegate dinamicamente nelle app per *iOS* [37]. È quindi necessario includere nel progetto i sorgenti delle librerie esterne che si vogliono utilizzare, in questo caso gli *external*, nel progetto in modo che vengano compilati assieme all'applicazione.

Una libreria collegata dinamicamente consiste in un oggetto software che mette a disposizio-

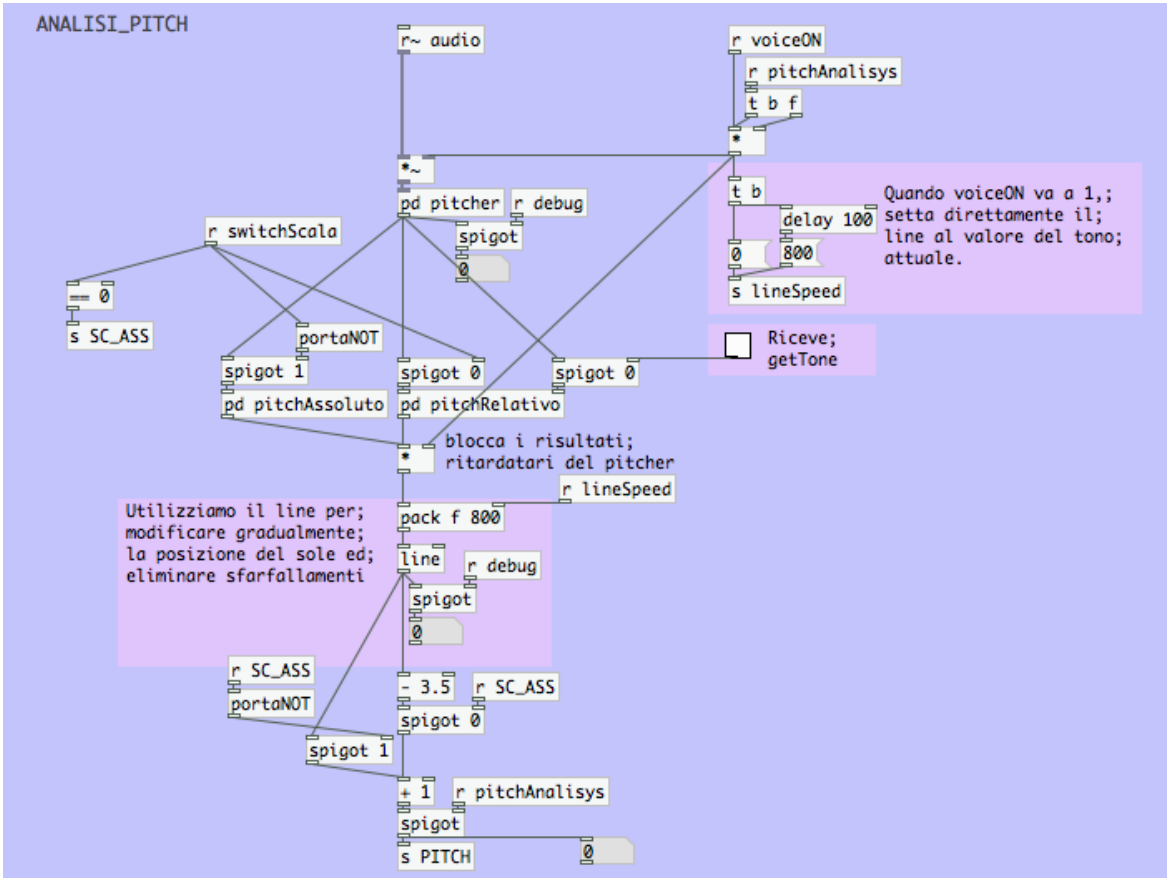


Figura 2.6: *Il blocco di analisi del pitch*

ne determinate funzionalità ed è già compilato per una o più architetture *hardware*. Per utilizzare queste funzionalità basta includerne le definizioni nel proprio codice sorgente e richiamarne i metodi quando necessario. Infine si deve dire al *linker*, il programma che svolge l'ultima fase di una compilazione, dove trovare la libreria compilata per l'architettura verso cui si sta compilando. In fase di distribuzione si dovranno includere tutte le librerie collegate dinamicamente nel pacchetto di distribuzione dell'applicazione.

Questa limitazione pone qualche problema nell'uso di librerie distribuite sotto licenza *GPLv3* (*General Public License v3.0*) [38], in quanto al paragrafo 5 di tale licenza, dove si parla degli obblighi di rilascio dei sorgenti in caso di modifiche, al punto (c) si fa riferimento all'inclusione di un progetto licenziato sotto *GPLv3* in un progetto più grande e all'obbligo in quel caso, di rilasciare il progetto nella sua interezza sotto licenza *GPLv3*.

A questo punto ci si potrebbe chiedere dove stia l'incomodo. Basterebbe seguire queste direttive rispettando la volontà dell'autore della libreria di arricchire la comunità del *software*

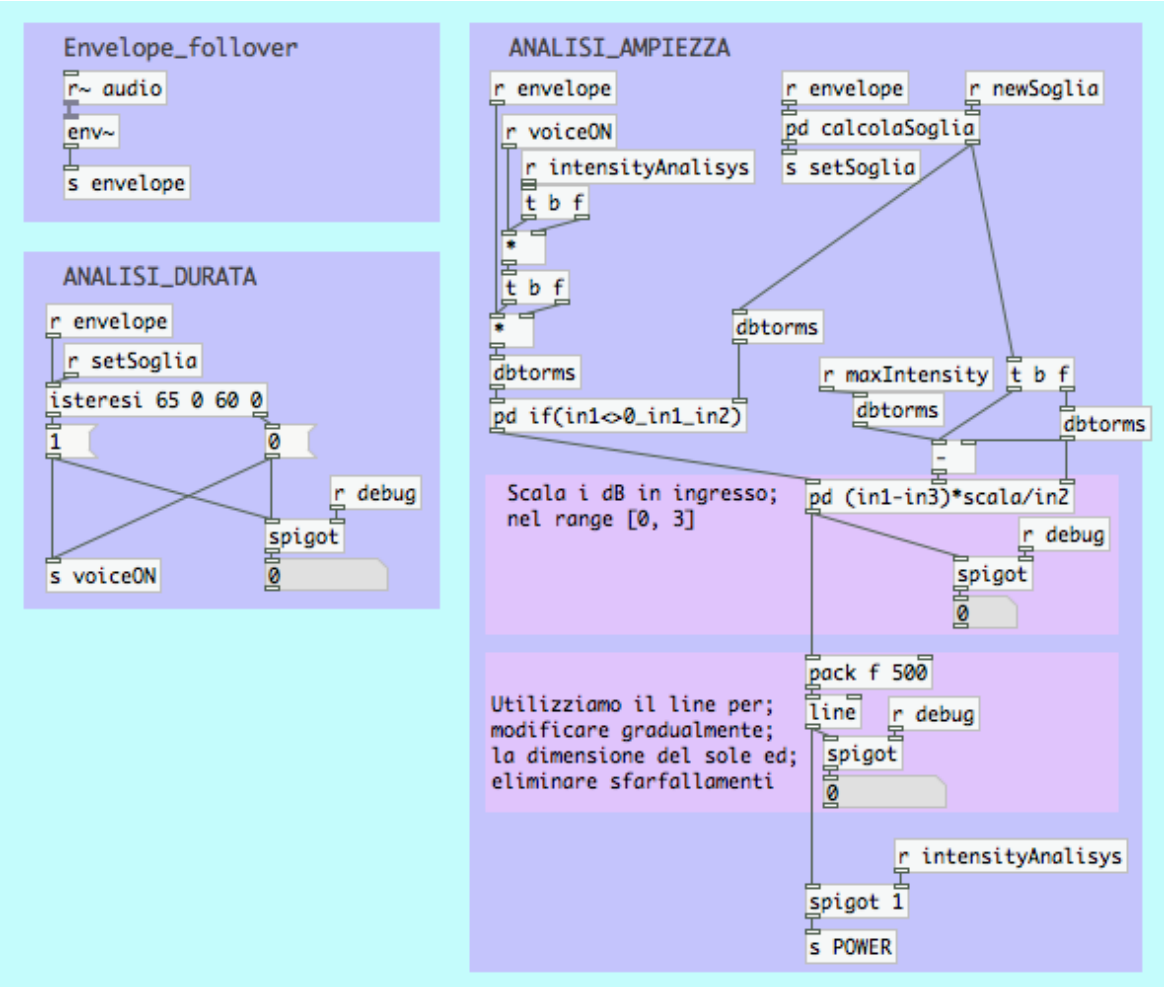
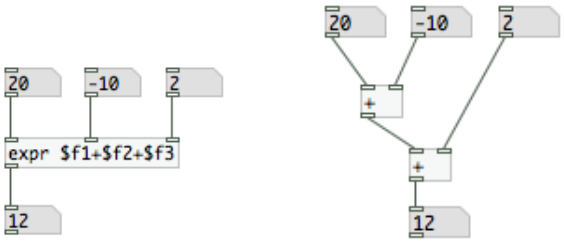


Figura 2.7: *Blocchi di analisi dell'ampiezza e della durata*

libero e arricchirla ulteriormente rilasciando appunto sotto questa licenza. Purtroppo ciò non è possibile per le applicazioni *iOS* dato il particolare sistema di distribuzione tramite *App Store* che si avvale di una licenza definita da *Apple* [39]. I termini di distribuzione definiti in tale documento non sono compatibili con i termini di distribuzione della licenza *GPLv3*, la quale garantisce in primis la piena libertà dell'utente finale di disporre del codice sorgente, modificarlo e ridistribuirlo senza alcuna limitazione, a patto di mantenere i riferimenti al *copyright* e alla licenza stessa.

Consci di queste limitazioni, i responsabili del progetto *Pure Data* e *libpd* hanno cambiato la licenza di distribuzione della versione *vanilla* di *Pure Data* con una licenza meno restrittiva come la licenza *BSD*[3] *2-Clause License* e invitano gli autori di *external* ad utilizzare una licenza simile

[3] *Berkeley Software Distribution.*



(a) (b)

Figura 2.8: *Esempio di espressione calcolata con l'oggetto* `[expr]` *o con gli oggetti matematici di Pure Data*

per garantire la possibilità di distribuire su *Apple App Store* applicazioni che sfruttino *libpd*.

Un *external* molto comodo, utilizzato in vari punti anche nel prototipo, licenziato tuttora sotto *GPLv3* è l'oggetto `[expr]`, il quale permette di applicare ai dati passati ai suoi ingressi espressioni matematiche anche molto complesse, passando semplicemente l'espressione come parametro in fase di costruzione. Ad esempio, se si volessero sommare tre valori di tipo `float` passati in ingresso, basterebbe creare l'oggetto nel seguente modo: `[expr $f1+$f2+$f3]` (Figura 2.8(a)) e collegare le sorgenti dei valori ai tre *inlet* dell'oggetto. I parametri che iniziano con il simbolo $ si riferiscono appunto agli *inlet* e l'oggetto `[expr]` presenterà tanti *inlet* quanti parametri conterrà l'espressione passata.

La versione *vanilla* di *Pure Data* mette a disposizione degli oggetti matematici come `[+]`, `[-]`, `[*]`, `[/]` che accettano due parametri in ingresso e permettono di eseguire le operazioni di base. Altri oggetti permettono operazioni più complesse come le operazioni trigonometriche o il calcolo di logaritmi, ma questi oggetti non sono dinamici e accettano soltanto uno o due parametri in ingresso. Se volessimo sommare tre numeri dovremmo utilizzare due oggetti somma in cascata (Figura 2.8(b)).

Per sostituire l'oggetto `[expr]` e poter quindi distribuire l'applicazione anche tramite il servizio *App Store* mantenendo invariato il diagramma della *patch* sono disponibili due soluzioni:

- creare una *subpatch* che esegua l'espressione usando gli oggetti matematici di *Pure Data*;
- creare un *external* che esegua l'espressione con le funzioni messe a disposizione dalle librerie matematica del linguaggio *C*.

La prima soluzione è molto veloce quando si ha a che fare con espressioni semplici e consiste nell'incorporare una *subpatch* dove si va a ricreare l'espressione come nell'esempio in Figura 2.8, senza che i nuovi oggetti vadano a sconvolgere il diagramma della *patch* principale.

La seconda soluzione è preferibile nel caso di espressioni complesse che richiederebbero molto lavoro per essere ricreate con gli oggetti matematici di *Pure Data*. Al contrario, la sintassi delle espressioni matematiche del linguaggio *C* è abbastanza simile alla sintassi del linguaggio matematico, per cui la traduzione di un'espressione risulta un'operazione molto semplice.

Il compromesso va cercato considerando il fatto che la creazione di un *external* richiede un *overhead* non indifferente dovuta alla struttura di base del codice di un *external* e alla necessità di compilare l'*external* per le architetture sulle quali si vuole mantenere la compatibilità della *patch*.

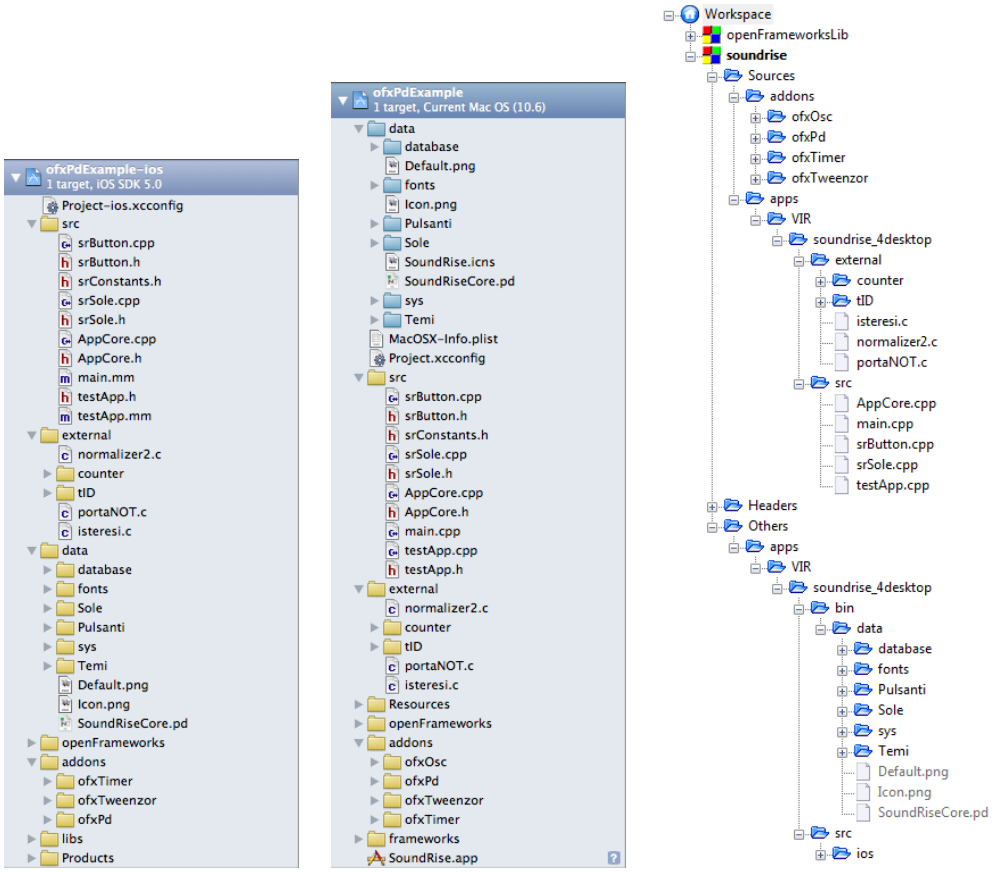
2.7 Descrizione dei progetti software

In Figura 2.9 si possono confrontare gli alberi delle directory dei tre progetti software. Si può notare come siano molto simili, a parte alcune piccole differenze dovute ai diversi sistemi operativi e agli *IDE* utilizzati. L'aspetto più importante riguarda il codice sorgente che differisce soltanto in tre *file* (`main.mm`, `testApp.h` e `testApp.mm`) nel progetto per *iOS*, si tratta dell'involucro più esterno della applicazione, quello che deve interfacciarsi con il sistema operativo, che a differenza degli altri due progetti non usa le librerie *Glut* per la gestione delle finestre, ma l'*addon ofxiPhone* per interfacciarsi al sistema operativo *iOS*.

Altre differenze riguardano aspetti legati alla creazione dei *bundle* su *Mac OS X*, ovvero applicazioni autosufficienti che includono al loro interno tutto quello che serve loro per funzionare e possono essere più facilmente distribuite.

Prendendo a esempio l'albero del progetto *iOS* (Figura 2.9(a)) si può osservare come sia diviso nelle seguenti directory:

- `src`: contiene i sorgenti dell'applicazione;
- `external`: contiene i sorgenti degli *external Pure Data* che vengono usati nella *patch*;
- `data`: contiene le risorse necessarie all'applicazione
  - `database`: contiene i *database* delle caratteristiche sonore delle vocali usate dal classificatore `[timbreID]`;



(a) (b) (c)

Figura 2.9: *L'albero delle directory dei progetti (a) iOS, (b) Mac OS X e (c) Windows*

- `fonts`: contiene i *font TrueType* utilizzati nell'interfaccia grafica;
- `Sole`: contiene le immagini del sole;
- `Pulsanti`: contiene le immagini usate nei pulsanti dell'interfaccia grafica;
- `sys`: contiene le *subpatch* esterne richiamate dalla *patch* `SoundRiseCore.pd`;
- `Temi`: contiene le immagini usate come sfondo dell'interfaccia grafica;
- `Default.png`: l'immagine usata come *splash screen* all'avvio dell'applicazione;
- `Icon.png`: l'immagine usata come icona dell'applicazione;
- `SoundRiseCore.pd`: la *patch* di analisi audio;

- `openFrameworks`: contiene il *toolkit openFrameworks*;

- `addons`: contiene gli *addon openFrameworks* usati nell'applicazione
  - `ofxTimer`: *addon* per la gestione di temporizzatori;
  - `ofxTweenzor`: *addon* che si occupa di variare in modo continuo un qualsiasi argomento numerico (viene utilizzato per le transizioni del colore del sole al variare delle vocali);
  - `ofxPd`: *addon* che permette di utilizzare un'istanza di *Pure Data*;
- `libs`: contiene i *framework* di sviluppo messi a disposizione dall'*SDK* di *iOS*;
- `Products`: contiene il prodotto della compilazione del progetto (su *Xcode 3*, mentre su *Xcode 4* il sistema di *build* dei progetti è cambiato e usa la cartella `Libreria` dell'utente).

La modalità di creazione di un nuovo progetto *openFrameworks* semplifica la configurazione dei progetti, come la configurazione del processo di compilazione, in quanto si prende un progetto di esempio e lo si usa come base di partenza per il proprio. Anche l'*addon ofxPd* mette a disposizione quattro progetti di esempio: uno per il sistema operativo *iOS*, uno per il sistema operativo *Mac OS X*, un altro per il sistema operativo *Windows*, infine uno per il sistema operativo *GNU/Linux*; i primi due per l'*IDE Xcode*, gli ultimi per l'*IDE Code::Blocks*. In sostanza questi progetti includono le *directory* dell'*addon ofxPd*, inseriscono queste *directory* nei percorsi di ricerca degli *header* del progetto e definiscono alcune direttive di compilazione indispensabili per una corretta compilazione dell'*addon ofxPd*.

Per la piattaforma *Apple* devono essere presenti le seguenti direttive al compilatore *C* e *C++*: "`-DHAVE_UNISTD_H -DUSEAPI_DUMMY -DPD -dynamiclib -ldl -lm`". Infine, per l'uso delle funzionalità di *openFrameworks*, devono essere presenti i riferimenti alle librerie precompilate incluse in *openFrameworks* come direttive al *linker* e la direttiva `-ObjC`.

Per la piattaforma *Windows* ci si deve assicurare della presenza delle seguenti direttive al compilatore: `HAVE_UNISTD_H`, `USEAPI_DUMMY`, `MSW`, `PD`, `PD_INTERNAL`, oltre alla direttiva `_WIN32_WINNT=0x502` che risolve alcuni problemi nell'uso delle librerie dinamiche. Inoltre devono essere presenti le direttive al linker `m` e `pthread`.

2.8 Descrizione del codice

Il codice prodotto e riportato in questo lavoro è rilasciato secondo i termini della licenza riportata in Appendice A.1.

Come si può vedere in Figura 2.9 il codice sorgente del progetto è suddiviso in vari *file*, alcuni dovuti alla struttura di un progetto *openFrameworks/ofxPd*, altri in cui vengono definite delle classi di supporto.

I primi comprendono i file `main.cpp`, `testApp.h` e `testApp.cpp` e definiscono l'involucro dell'applicazione, ovvero la parte che si occupa di comunicare con il sistema operativo, creare le finestre e richiedere le risorse di sistema necessarie. Questi sono gli unici *file* presenti in due versioni, quelli appena elencati e `main.mm`, `testApp.h` e `testApp.mm` per il sistema operativo *iOS*.

Il cuore dell'applicazione, ovvero la parte che si occupa di comunicare con la *patch*, disegnare l'interfaccia grafica e rispondere agli *input* dell'utente, è definita nei *file* `AppCore.h` e `AppCore.cpp`.

Infine nel *file* `srConstants.h` vengono definite delle costanti utili alla comprensibilità del codice, nei *file* `srSole.h` e `srSole.cpp` viene definita la classe che si occupa di aggiornare lo stato del sole e di disegnarlo sullo schermo, e nei *file* `srButton.h` e `srButton.cpp` si definisce la classe di controllo e visualizzazione dei pulsanti utilizzati nell'interfaccia grafica.

2.8.1 I *file main* e *testApp*

I *file main* provvedono a includere le librerie necessarie a un programma *openFrameworks*, a inizializzare l'ambiente *OpenGL* e lanciare il *"controller"* dell'applicazione, ovvero quella parte che ne gestisce la logica generale e le comunicazioni con il sistema operativo.

Come si può vedere nei listati di codice 2.1 e 2.2, ci sono alcune differenze tra le due versioni dovute principalmente alla gestione delle finestre e delle eccezioni. Le librerie `ofAppGlutWindow.h` e `Poco/Exception.h` servono proprio a gestire finestre ed eccezioni nei sistemi operativi diversi da *iOS*. Le prime due righe sono comuni a entrambe le versioni e si occupano di includere *openFrameworks* (`#include "ofMain.h"`) e il *file* di definizione dell'applicazione, `testApp.h` (Codice 2.3 e 2.4). La riga 8 del codice 2.1 dichiara la variabile della finestra dell'applicazione, la quale viene inizializzata come finestra *OpenGL* alla riga 11 dove si sceglie la dimensione in pixel e la modalità di visualizzazione tra `OF_WINDOW`, per ottenere una finestra, e `OF_FULL-`

SCREEN, per una visualizzazione a tutto schermo. Infine, alla riga 13, si manda in esecuzione l'applicazione. La stessa operazione viene eseguita alla riga 9 del codice 2.2, mentre alla riga 7 si inizializza la finestra *OpenGL* con un comando un po' diverso, dove la scelta della risoluzione non ha effetto, mentre le costanti OF_WINDOW e OF_FULLSCREEN determinano soltanto se la barra di stato del sistema operativo *iOS* sia visibile o meno.

Codice 2.1: *main.cpp*

```
1  #include "ofMain.h"
2  #include "testApp.h"
3  #include "ofAppGlutWindow.h"
4  #include "Poco/Exception.h"
5
6  //========================================================================
7  int main(){
8      ofAppGlutWindow window;
9
10     // setup the GL context
11     ofSetupOpenGL(&window, 800, 600, OF_FULLSCREEN);
12
13     ofRunApp(new testApp());
14 }
```

Codice 2.2: *main.cpp versione iOS*

```
1  #include "ofMain.h"
2  #include "testApp.h"
3
4  //========================================================================
5  int main() {
6      // setup the GL context
7      ofSetupOpenGL(320, 480, OF_FULLSCREEN);
8
9      ofRunApp(new testApp);
10 }
```

Nei listati di codice 2.3 e 2.4 viene definita l'applicazione vera e propria. Le prime due righe della versione *iOS* richiamano l'*addon openFrameworks* che specifica la struttura delle applicazioni *iOS* e fornisce i metodi per la comunicazione con il sistema operativo. Nelle righe 1 e 4, rispettivamente nei listati di codice 2.3 e 2.4, viene incluso il nucleo dell'applicazione poi dichirato rispettivamente alle righe 23 e 29. Nelle righe 4 e 6 delle due versioni vengono definite le classi dell'applicazione, mentre nelle righe successive vengono dichiarati i metodi che gestiscono il ciclo dell'applicazione, gli eventi di *input* e di sistema, infine i metodi che gestiscono il flusso del segnale audio in ingresso e in uscita. Alla riga 25 del codice 2.3 vengono dichiarate le variabili inputStream e outputStream che servono per inizializzare la gestione del flusso audio nelle piattaforme diverse da *iOS*.

Codice 2.3: *testApp.h*

```
#include "AppCore.h"

// a desktop os app wrapper
class testApp : public ofBaseApp{

    public:
        void setup();
        void update();
        void draw();
        void exit();

        void keyPressed  (int key);
        void mouseMoved(int x, int y );
        void mouseDragged(int x, int y, int button);
        void mousePressed(int x, int y, int button);
        void mouseReleased(int x, int y, int button);
        void windowResized(int w, int h);

        //audio callbacks
        void audioReceived(float * input, int bufferSize, int nChannels);
        void audioRequested(float * output, int bufferSize, int nChannels);

        AppCore core;
        //Definizione dei canali di input e output
        ofSoundStream inputStream, outputStream;
};
```

Nei listati di codice 2.5 e 2.6 vengono riportate le parti più interessanti dell'implementazione di queste classi, ovvero la parte di inizializzazione delle risorse di sistema usate dall'applicazione. Nel primo si usufruisce delle direttive al preprocessore[4] (righe 6, 15 e 21) per inizializzare il flusso del segnale audio in ingresso e in uscita a una frequenza di campionamento di 22050 Hz sulla piattaforma *Microsoft Windows*. Questo aggiustamento deriva da prove sperimentali ed è dovuto a un comportamento poco fluido della rappresentazione dell'altezza del suono su questa piattaforma con una frequenza di campionamento di 44100 Hz. Diversamente da come ci si aspetterebbe, una modifica della frequenza di campionamento non disturba la relazione tra *pitch* e posizione del sole sull'orizzonte. Questo fa pensare ad un problema di comunicazione tra le librerie audio usate da *openFrameworks* e la parte di sistema operativo che gestisce le interfacce audio. Dato il corretto funzionamento dell'applicazione con questa impostazione non sono stati effettuati ulteriori studi a riguardo. Il comando utilizzato per inizializzare lo *stream* audio riceve 6 parametri in ingresso:

- un riferimento alla classe che dichiara e utilizza gli oggetti di tipo `ofSoundStream`;

[4]Programma che aggrega il codice sorgente in un unico file da dare in pasto al compilatore e che può includere o meno determinate parti di codice seguendo le direttive di precompilazione.

Codice 2.4: *testApp.h versione iOS*

```
1  #include "ofxiPhone.h"
2  #include "ofxiPhoneExtras.h"
3
4  #include "../AppCore.h"
5
6  class testApp : public ofxiPhoneApp {
7
8  public:
9      void setup();
10     void update();
11     void draw();
12     void exit();
13
14     void touchDown(ofTouchEventArgs &touch);
15     void touchMoved(ofTouchEventArgs &touch);
16     void touchUp(ofTouchEventArgs &touch);
17     void touchDoubleTap(ofTouchEventArgs &touch);
18     void touchCancelled(ofTouchEventArgs &touch);
19
20     void lostFocus();
21     void gotFocus();
22     void gotMemoryWarning();
23     void deviceOrientationChanged(int newOrientation);
24
25     // audio callbacks
26     void audioReceived(float * input, int bufferSize, int nChannels);
27     void audioRequested(float * output, int bufferSize, int nChannels);
28
29     AppCore core;
30 };
```

- il numero di canali audio in uscita;
- il numero di canali audio in ingresso;
- la frequenza di campionamento;
- la dimensione dei *buffer* di memoria;
- il numero di *buffer* di memoria associati allo *stream*.

Infine, alle righe 14 e 20, si inizializza il nucleo dell'applicazione passando il numero di canali audio di uscita e di ingresso, la frequenza di campionamento e il numero di battiti (*tick*) per *buffer*.

Un'altra impostazione importante, per quanto riguarda l'inizializzazione audio nella piattaforma *Microsoft Windows*, si può vedere alla riga 8, dove si seleziona l'identificativo della interfaccia di *input* audio. Su questa piattaforma tali identificativi variano in funzione dei dispositivi audio installati nel sistema e, a differenza delle altre piattaforme, non esiste un dispositi-

vo virtuale (con identificativo di *default* 0) che rappresenti il dispositivo di *input* predefinito di sistema.

L'accesso a questa impostazione nell'interfaccia grafica per ora è stato tralasciato e verrà aggiunto in una successiva fase di preparazione alla distribuzione. Questo richiede di compilare alcune versioni dell'applicazione con diverse impostazioni dei dispositivi di *input* nel caso di distribuzione su sistemi sconosciuti e provarle a una a una finché non si trova la versione con il dispositivo adatto.

Nel listato di codice 2.6 si inizializza l'applicazione per la piattaforma *iOS*. Alla riga 3 si registra questa classe come *listener* per gli eventi generati dall'*input* dell'utente sullo schermo sensibile al tocco, alla riga 6 si attivano gli accelerometri, anche se non vengono utilizzati in questa applicazione, alla riga 9 si registra questa classe come *listener* per i messaggi del sistema operativo, come l'arrivo di un messaggio di testo, una chiamata telefonica o un avviso di memoria insufficiente, infine alla riga 12 si sceglie l'orientazione della finestra dell'applicazione. Le righe 19 e 22 inizializzano il flusso audio in ingresso e in uscita e il nucleo dell'applicazione, similmente a quanto visto precedentemente per il listato di codice 2.5.

Il resto dell'implementazione di entrambe le versioni si occupa soltanto di inviare al nucleo dell'applicazione (la classe definita nei *file* `AppCore.h` e `AppCore.cpp`) gli eventi che si vogliono gestire e il flusso del segnale audio di ingresso e di uscita, e dunque non vengono qui riportate.

Codice 2.5: *L'inizializzazione dell'applicazione in testApp.cpp*

```
1  void testApp::setup() {
2      // the number if libpd ticks per buffer,
3      // used to compute the audio buffer len: tpb * blocksize (always 64)
4      int ticksPerBuffer = 8; // 8 * 64 = buffer len of 512
5
6  #ifdef TARGET_WIN32
7      // setup win32 default input device
8      inputStream.setDeviceID(2);
9
10     // setup OF sound stream
11     inputStream.setup(this, 0, 2, 22050, PdBase::blockSize()*ticksPerBuffer, 2);
12     outputStream.setup(this, 2, 0, 22050, PdBase::blockSize()*ticksPerBuffer, 3);
13     // setup the app core
14     core.setup(2, 2, 22050, ticksPerBuffer);
15 #else
16     // setup OF sound stream
17     inputStream.setup(this, 0, 2, 44100, PdBase::blockSize()*ticksPerBuffer, 2);
18     outputStream.setup(this, 2, 0, 44100, PdBase::blockSize()*ticksPerBuffer, 3);
19     // setup the app core
20     core.setup(2, 2, 44100, ticksPerBuffer);
21 #endif
22 }
```

Codice 2.6: *L'inizializzazione dell'applicazione in testApp.mm versione iOS*

```
1  void testApp::setup() {
2      // register touch events
3      ofRegisterTouchEvents(this);
4
5      // initialize the accelerometer
6      ofxAccelerometer.setup();
7
8      // iPhoneAlerts will be sent to this
9      ofxiPhoneAlerts.addListener(this);
10
11     // if you want a landscape orientation
12     ofxiPhoneSetOrientation(OFXIPHONE_ORIENTATION_LANDSCAPE_LEFT);
13
14     // the number if libpd ticks per buffer,
15     // used to compute the audio buffer len: tpb * blocksize (always 64)
16     int ticksPerBuffer = 8; // 8 * 64 = buffer len of 512
17
18     // setup OF sound stream
19     ofSoundStreamSetup(2, 1, this, 44100, PdBase::blockSize()*ticksPerBuffer, 3);
20
21     // setup the app core
22     core.setup(2, 1, 44100, ticksPerBuffer);
23 }
```

2.8.2 *AppCore*: il cuore dell'applicazione

In appendice è riportato il *file* `AppCore.h` (Codice A.1) dove si dichiara la classe `AppCore` che si occupa della logica e dell'interfaccia dell'applicazione, di caricare e comunicare con la *patch Pure Data* per l'analisi audio e di rappresentare a schermo le caratteristiche del suono analizzate.

Nelle righe 1-12 si richiamano la classe `ofxPd` che fornisce l'istanza di *libpd*, `srSole` e `srButton` che si occupano della gestione del sole e dei pulsanti dell'interfaccia grafica, `<vector>` che permette l'uso degli *array*, `ofxTimer` per creare e usare dei temporizzatori e, nel caso non si stia compilando per sistemi operativi *Apple iOS* o *Google Android*, la classe `ofxOsc` che si occupa di gestire le comunicazioni su protocollo *OSC*. Infine include il *file* delle costanti definite per l'applicazione (`srConstants.h`) e definisce il *namespace* `pd` per semplificare le chiamate ai metodi della classe `ofxPd`.

Alla riga 14 viene dichiarata la classe come discendente delle classi `PdReceiver` e `PdMidiReceiver` che definiscono l'interfaccia di comunicazione con l'istanza di *libpd*.

Nelle righe 18-25 vengono dichiarati i metodi di gestione del ciclo dell'applicazione e del flusso audio, nelle righe 27-40 vengono dichiarati i metodi di comunicazione con l'istanza di *libpd*, nelle righe 43 e 46 si dichiarano i metodi per gestire gli eventi di *input* dell'utente tramite schermo sensibile al tocco o tramite *mouse*, mentre alla riga 49 viene dichiarato il metodo che

termina la registrazione del tono di riferimento della scala relativa delle altezze.

Le righe successive dichiarano le variabili utilizzate nel codice dell'applicazione, tra cui in particolare `pd` (riga 51), che si riferisce all'istanza di *libpd*, `sole` (riga 57) che si riferisce all'istanza della classe `srSole`, `toneTimer` (riga 59) che si riferisce al temporizzatore che gestisce la memorizzazione del tono di riferimento della scala relativa delle altezze e `OSCreceiver` che si riferisce all'istanza del ricevitore dei messaggi *OSC*.

Nel listato di codice A.2 viene riportata l'implementazione della classe mentre qui si trattano le parti più interessanti. Nelle righe 5-13 si dichiarano le funzioni di configurazione degli *external* che si vogliono usare nella *patch* e il cui codice è stato inserito nella cartella `external` dei progetti (Paragrafo 2.7). Queste funzioni verranno poi richiamate nella inizializzazione della classe (righe 71-79).

Nelle righe 55-69 si provvede alla configurazione dell'istanza di *libpd* e all'iscrizione ai messaggi provenienti dalla *patch*. Le righe 82-93 provvedono ad attivare il *DSP*, a caricare la *patch* e a inviarle i messaggi di configurazione.

I metodi più importanti di un'applicazione *OpenGL* sono il metodo `update()`, dove si possono ricalcolare posizioni, dimensioni e stato degli oggetti tridimensionali che fanno parte della scena[5], e il metodo `draw()`, che si occupa di disegnare la scena nella finestra *OpenGL*. Questi metodi vengono richiamati ad ogni ciclo di *rendering*, in questo caso 60 volte al secondo (come impostato alla riga 30).

Nel metodo `update()` (righe 144-210) si aggiornano le posizioni e dimensioni dei pulsanti per adattarli alla dimensione dello schermo, si richiama il metodo di aggiornamento della classe `srSole` (riga 164) e si elaborano eventuali messaggi *OSC* ricevuti. I pulsanti e il sole vengono posizionati e dimensionati in relazione alle dimensioni della finestra, ma se per il secondo è facile intuire il motivo di dover ricalcolare posizione e dimensione a ogni ciclo, per i primi questo è meno chiaro. Se non si ricalcolassero questi parametri, i pulsanti rimarrebbero dimensionati e posizionati relativamente a una finestra di 800x600 pixel, ovvero la risoluzione scelta nel *file* `main.cpp`. Questo inoltre permette di poter scegliere l'avvio in modalità finestra senza preoccupazioni, utilizzando la costante `OF_WINDOW` in fase di creazione della finestra *OpenGL*.

Nel metodo `draw()` (righe 213-295) si gestiscono le varie pagine (analisi e *setting*) e si disegnano gli oggetti che di volta in volta sono visibili. Peculiare è l'ordine con cui gli oggetti vengono disegnati: si parte con lo sfondo (riga 216), si disegna il sole soltanto se è selezionato uno dei pulsanti sezione (riga 218) così da visualizzarlo soltanto durante l'analisi o la registra-

[5] In un applicazione *OpenGL* è quella parte di ambiente tridimensionale che si vuole visualizzare sullo schermo.

zione del tono di riferimento, si disegna il paesaggio (riga 221) e infine il resto dell'interfaccia grafica. In questo modo si sfrutta la grafica tridimensionale per dare la sensazione della prospettiva anche con immagini bidimensionali, il sole infatti sembra tramontare dietro l'orizzonte quando si trova in basso. Inoltre le barre e i pulsanti dell'interfaccia grafica sono in primo piano. Per quanto riguarda invece la gestione delle pagine, alla riga 271 si gestisce il caso dell'analisi delle altezze, dove si aggiunge alla pagina di analisi (`pageDefault` alla riga 225) soltanto i due pulsanti di selezione tra scala assoluta e scala relativa.

Nelle righe 298-300 si gestisce la chiusura dell'applicazione chiudendo la *patch* e togliendo l'iscrizione agli eventi del temporizzatore.

Nelle righe 304-311 si collega la *patch* al dispositivo audio in ingresso e in uscita.

Nelle righe 323-337 si gestiscono i messaggi di tipo `float` provenienti dalla *patch*, si può vedere come il messaggio inviato dall'oggetto `[s PITCH]` della *patch* vada ad impostare l'altezza del sole, il messaggio dell'oggetto `[s voiceON]` determini la visualizzazione del sole soltanto in presenza di produzione vocale, il messaggio `POWER` controlli la dimensione del sole e il messaggio `sezione` determini lo stato del sole, ovvero come questo si comporterà per rispondere ai vari tipi di analisi audio.

Nelle righe 343-349 si gestisce il messaggio inviato dall'oggetto `[s vocale]` per determinare il colore col quale disegnare il sole secondo la mappa vista al paragrafo **??**. Il messaggio è composto da distanza e confidenza del vettore di caratteristiche audio presente nel *database* dell'oggetto `[timbreID]` più simile al vettore di caratteristiche ricavato dal segnale audio in ingresso [18].

Nelle righe 351-355 si deseleziona il pulsante di ricalcolo della soglia del rumore ambientale quando la *patch* invia il messaggio `setSoglia` al termine del calcolo. Si ricorda che per ricavare questo parametro si effettua una media della potenza del segnale audio in ingresso nell'arco di tre secondi di tempo e durante questa operazione il pulsante viene evidenziato in rosso.

Nelle righe 388-479 e 482-576 si gestiscono le interazioni dell'utente con l'interfaccia grafica rispettivamente attraverso lo schermo sensibile al tocco e attraverso il *mouse*. Si può notare come non si sia speso molto nell'ottimizzare la classe `srButton`, ma si è cercato di ottenere una classe funzionale alle esigenze e funzionante in tempi brevissimi per dedicarsi allo sviluppo della visualizzazione delle caratteristiche del suono. Ai pulsanti sezione, che attivano l'analisi delle caratteristiche audio, è associato un identificativo di sezione che viene inviato alla *patch* tramite il metodo `sendFloat(const std::string& dest, float value)` secondo la seguente mappa:

- 0 $\Rightarrow$ analisi intensità;

- 1 ⇒ analisi altezza;
- 2 ⇒ analisi durata;
- 3 ⇒ analisi vocali;
- 4 ⇒ analisi di tutte le caratteristiche.

Analogamente il metodo `sendBang(const std::string& dest)` permette di inviare alla *patch* un segnale di tipo `bang` il cui destinatario sarà l'oggetto identificato dalla stringa passata come parametro.

2.8.3 Le classi di supporto

La classe di supporto più interessante, riportata in Appendice A.4, è quella che si occupa di gestire le dimensioni, la posizione, lo stato e il colore del sole. Nel metodo di *setup* (righe 11-59 del listato di codice A.4) vengono impostate le dimensioni di base (quelle proposte nelle sezioni diverse dall'analisi dell'ampiezza), caricate le immagini e definiti i colori associati alle vocali.

Nelle righe 61-68 viene definito il metodo `update()` che aggiorna le proprietà del sole ad ogni ciclo *OpenGL*. Si può notare l'uso dell'*addon ofxTweenzor* per la variazione uniforme dei colori.

Nelle righe successive si definiscono i metodi *getter* e *setter* delle varie caratteristiche del sole associate alle caratteristiche del suono.

Degni di menzione i metodi `setVoiceOn`, che comanda la visualizzazione del sole solamente in presenza di produzione vocale, e `setVocale`, dove si impostano i valori di partenza e di arrivo dell'oggetto `Tweenzor` per i vari colori.

Infine il metodo `draw()` (righe 205-300) dove si devono gestire le limitazioni delle specifiche *OpenGL ES* fornite dai sistemi operativi mobili rispetto alle specifiche *OpenGL* presenti nelle altre piattaforme. Queste specifiche non supportano pienamente le immagini con trasparenza, quindi, per disegnare correttamente il sole,si deve ricorrere alle *texture* (righe 239-277). Ciò è dovuto alla necessità di colorare dinamicamente il sole in dipendenza delle vocali prodotte, effetto ottenuto partendo da un'immagine di base con il sole nero e disegnandola nella scena *OpenGL* "sommando" il colore desiderato secondo il modello RGB. Tale modello di colori è appunto un modello additivo dove il nero corrisponde alla terna (0,0,0) e il bianco alla terna (255,255,255).

La classe `srButton` si occupa invece di definire le caratteristiche di base dei pulsanti e i metodi per disegnarli nell'interfaccia grafica e reagire agli *input* dell'utente. Non si ritiene così interessante da meritare un'approfondimento.

Lo stesso vale per il file `srConstants.h` dove vengono definite alcune costanti di supporto per migliorare la comprensione del codice dell'applicazione.

Capitolo 3

L'interfaccia utente di SoundRise

SoundRise è un'applicazione multimodale interattiva per la didattica basata sull'analisi di feature vocali, ovvero permette all'utente di sperimentare le caratteristiche del suono utilizzando la propria voce. Questa applicazione nasce dall'idea di offrire un metodo alternativo per l'insegnamento delle caratteristiche del suono agli allievi della scuola primaria. Le caratteristiche della voce, analizzate in tempo reale, vengono rappresentate attraverso un sole disegnato sullo schermo secondo la mappa presentata al paragrafo 2.1.

All'apertura si presenta come in figura 3.1, mostrando un paesaggio ispirato ai disegni dei bambini e dei semplici pulsanti di controllo. Per non distrarre l'utente o creare confusione, il sole non viene disegnato finché non si sceglie una caratteristica da analizzare attivando il *feedback* visivo.

I pulsanti in basso a sinistra offrono un'anteprima dei temi disponibili e permettono all'utente di scegliere in modo semplice il paesaggio desiderato.

I pulsanti in basso a destra, partendo da destra, permettono di ricalcolare la soglia del rumore ambientale, parametro fondamentale per il corretto funzionamento dell'applicazione, e di accedere alla finestra di *setup* (Figura 3.3) per la configurazione del tono di riferimento della scala relativa delle altezze (Paragrafo 3.2). La soglia del rumore ambientale permette di discriminare il silenzio dai suoni prodotti dall'utente e viene usato come parametro per rappresentare la durata oltre che per determinare quando il sole viene visualizzato nelle sezioni dedicate all'analisi dell'ampiezza, dell'altezza, delle vocali e di tutte le caratteristiche assieme.

Questa soglia corrisponde all'energia media calcolata su tre secondi del segnale audio catturato dal microfono. Durante questo arco di tempo il pulsante viene evidenziato in rosso per indicare all'utente di non emettere produzioni vocali.

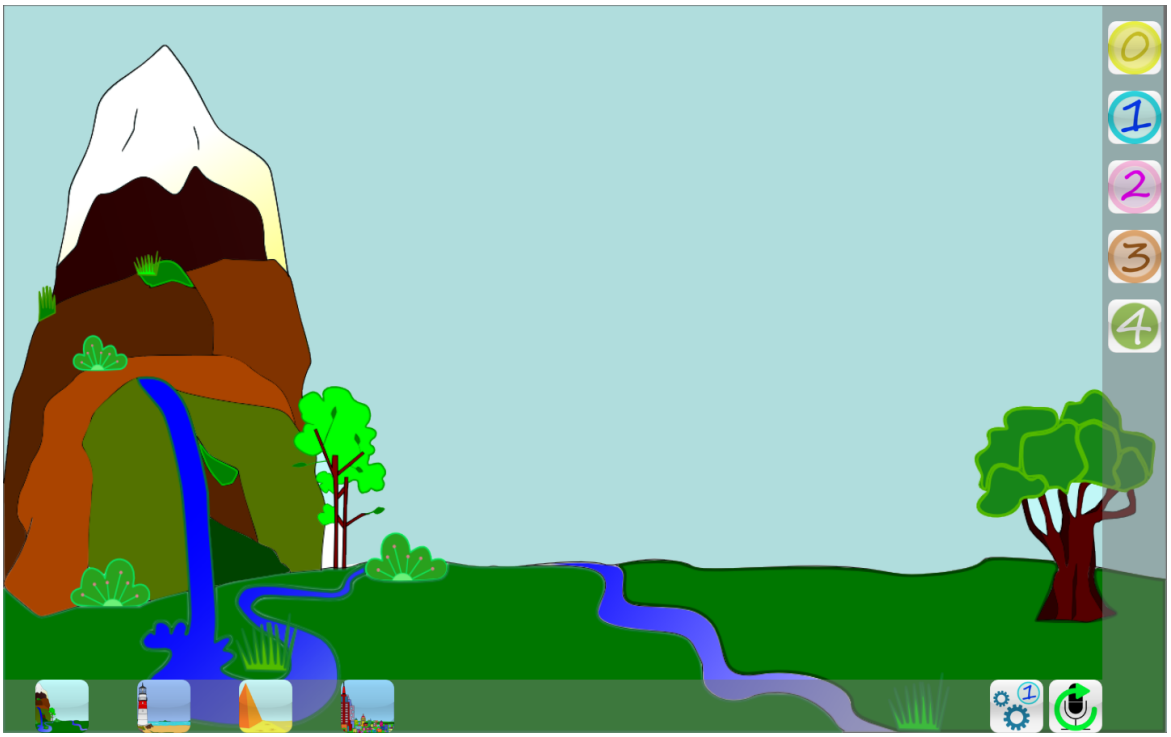


Figura 3.1: *L'applicazione SoundRise all'apertura*

I pulsanti in alto a destra consentono di scegliere le caratteristiche audio che si desidera analizzare attivando quindi il *feedback* visivo.

3.1 La selezione delle caratteristiche da analizzare

I pulsanti in altro a destra, numerati da "0" a "4", controllano le caratteristiche audio che verranno analizzare e rappresentate.

Il pulsante "0" dice all'applicazione di analizzare l'ampiezza dei suoni prodotti dall'utente e di rappresentarli modificando la dimensione del sole, da piccolo a grande, in funzione dell'intensità del suono dal piano al forte (Figura 3.2(a)).

Il pulsante "1" seleziona l'analisi dell'altezza e permette di scegliere una scala assoluta o relativa nella quale disegnare il sole (Figura 3.2(b)). Questo verrà posizionato nel cielo a seconda del tono prodotto dall'utente, rispettivamente in basso per i toni gravi e in alto per i toni acuti. La scala assoluta varia da circa 100 Hz a poco più di 400 Hz per poter rappresentare le produzioni vocali di maschi e femmine indipendentemente dall'età, mentre la scala relativa considera due

quinte (o sette semitoni) intorno a un tono di riferimento che può essere registrato dall'utente usando la finestra di *setup* (Figura 3.3).

Il pulsante "2" permette di scegliere l'analisi della durata (Figura 3.2(c)). Il sole viene mantenuto sempre visibile al centro del cielo con un sorriso a bocca e occhi chiusi nel caso di assenza di produzione vocale da parte dell'utente, oppure con un sorriso a bocca e occhi aperti in caso contrario.

Il pulsante "3" attiva l'analisi delle vocali e dice all'applicazione di rappresentarle variando il colore del sole secondo la mappa presentata al paragrafo 2.1.

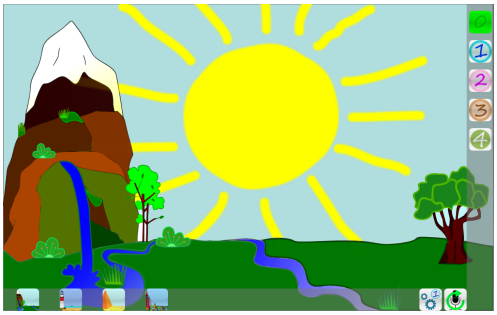
Il pulsante "4" consente di visualizzare tutte le caratteristiche contemporaneamente, offrendo naturalmente la possibilità di scegliere tra una scala assoluta e una scala relativa delle ampiezze.

3.2 La pagina di *setup*

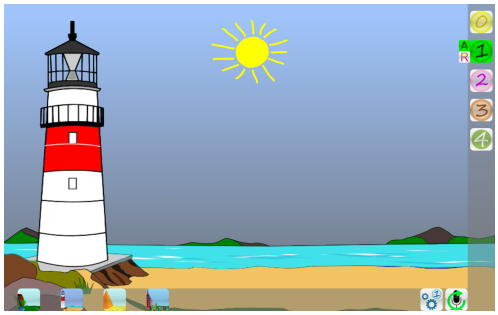
In questa finestra viene data la possibilità all'utente di memorizzare nell'applicazione un tono da utilizzare come riferimento per la scala relativa delle altezze (Figura 3.3). Viene chiesto di premere un pulsante per attivare la memorizzazione e produrre una [a] lunga finché tale pulsante non si "spegne". Alla pressione il pulsante viene evidenziato in verde e rimane evidenziato per due secondi. Durante questo tempo l'applicazione memorizza il valore del *pitch* del tono prodotto dall'utente e ne visualizza l'altezza sulla scala assoluta delle ampiezze.

Il trucco di "spegnere" automaticamente il pulsante assicura con una buona probabilità che ci sia ancora produzione vocale al momento della disattivazione, infatti è proprio l'ultimo valore memorizzato a diventare il *pitch* di riferimento.

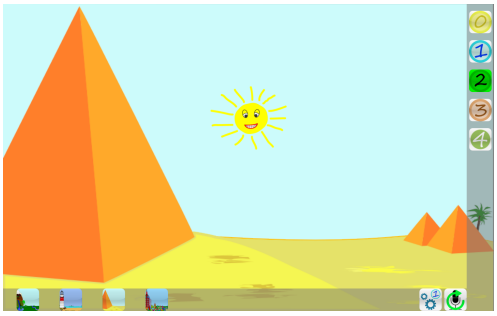
Alla fine dell'operazione premendo il pulsante "Ok" si viene riportati nella finestra principale dell'applicazione.



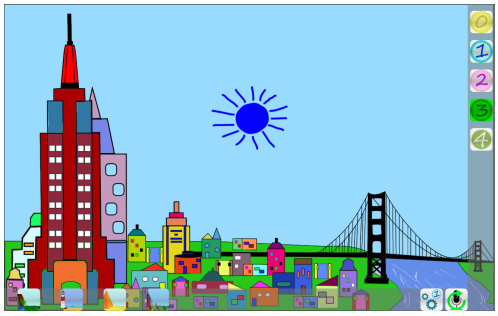
(a) analisi intensità



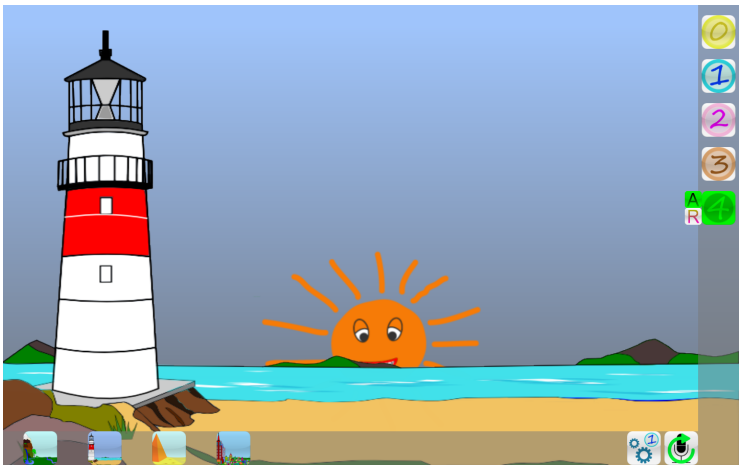
(b) analisi altezza



(c) analisi durata



(d) analisi vocali



(e) analisi di tutte le caratteristiche

Figura 3.2: *Esempi del feedback visivo dell'applicazione SoundRise per l'analisi di intensità, altezza, durata, vocali e di tutte le caratteristiche assieme*

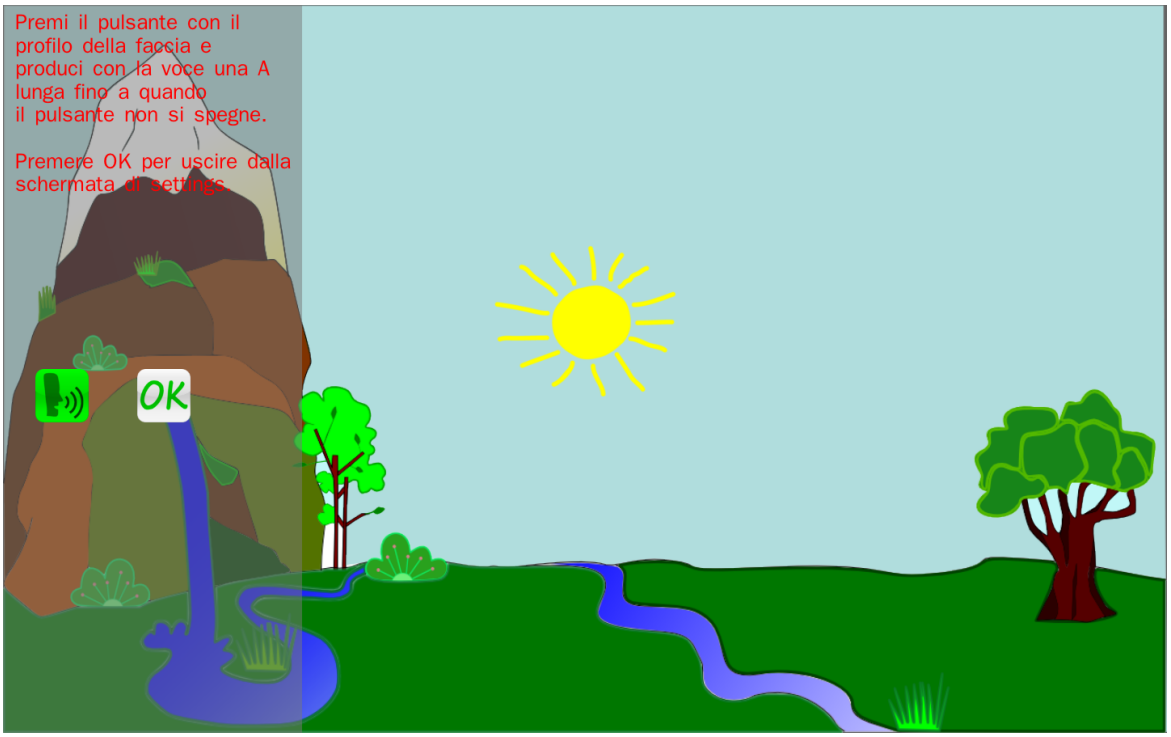


Figura 3.3: *La finestra di setup di SoundRise*

Capitolo 4

Validazione dell'applicazione

4.1 Test sull'usabilità dell'interfaccia e sul funzionamento dell'applicazione

Si è predisposto un test sull'usabilità e intuitività dell'interfaccia e sul funzionamento generale dell'applicazione da sottoporre a volontari di qualsiasi età e sesso. Lo scopo del test è valutare la presenza di eventuali elementi dell'applicazione che possano ostacolare il corretto svolgimento dei compiti da parte di un utente medio. Essendo lo scopo di SoundRise quello di fornire al bambino una modalità di apprendimento delle caratteristiche della voce mediante la loro rappresentazione grafica, è necessario che il sistema produca un chiaro e inequivocabile *feedback* grafico di ognuna di esse. Il test, riportato in Tabella 4.1, si compone di una serie di semplici domande pensate per essere proposte a un pubblico molto giovane e i risultati sono ottenuti riportando se l'applicazione ha risposto correttamente alle azioni intraprese dall'utente.

Le domande poste sono le seguenti:

- **Z4:** "Con questo gioco, che si chiama SoundRise, puoi disegnare il sole sullo schermo del computer usando la tua voce. Prova." In questo caso viene utilizzata la funzione 4;
- **P:** "Con le icone in basso a sinistra puoi scegliere un altro paesaggio su cui disegnare il sole. Prova." In questo caso viene utilizzata la funzione 4;
- **Set:** "Premi il pulsante con il profilo della faccia e produci con la voce una A lunga fino a quando il pulsante non è più verde." In questo caso di utilizza la funzione *Setting*;
- Successivamente viene chiesto di esplorare il funzionamento dei pulsanti a destra:
  - "Proviamo a selezionare lo 0 e vediamo cosa succede al sole."

    - **Ampiezza F:** "Esegui un suono forte."
    - **Ampiezza D:** "Esegui un suono debole."
- "Proviamo a selezionare l'1 e vediamo cosa succede al sole."
    - **Altezza A:** "Esegui un suono acuto/alto"
    - **Altezza Gr:** "Esegui un suono grave/basso"
- "Proviamo a selezionare il 2 e vediamo cosa succede al sole."
    - **Durata L:** "Esegui un suono lungo"
    - **Durata Br:** "Esegui un suono breve/corto"
- "Proviamo a selezionare il 3 e vediamo cosa succede al sole."
- **A:** "Esegui una A aperta[1] lunga"
- **O:** "Esegui una O aperta lunga"
- **E:** "Esegui una E aperta lunga"
- **I:** "Esegui una I lunga"
- **U:** "Esegui una U lunga"

Dai risultati ottenuti si può notare come l'applicazione abbia risposto correttamente in tutte le modalità, eccezion fatta per l'analisi delle vocali. Il test numero 1 riporta un comportamento errato della funzione di *Setting* dovuto a un banale errore subito corretto. In tutti gli altri test si è utilizzata la versione corretta.

Per quanto riguarda la modalità di analisi delle vocali, si deve tener presente che non è stata configurata per un pubblico così eterogeneo: il *database* di caratteristiche utilizzato è infatti ricavato da registrazioni di bambini di 9 e 10 anni. Si può notare come, a parte un singolo caso per nulla rilevante, la [u] non venga mai riconosciuta. Le altre vocali vengono riconosciute correttamente nell'unico test proposto a un bambino di 4 anni (test 4), mentre si può notare un comportamento medio migliore nei test sottoposti al pubblico femminile rispetto che al pubblico maschile. Questo è dovuto alla maggior vicinanza delle caratteristiche della voce femminile a quelle della voce dei bambini, sia per quanto riguarda la frequenza fondamentale, sia per quanto riguarda le caratteristiche del tratto orofaringeo.

[1] In verità la vocale [a] è aperta e non ne esiste una versione chiusa. Si pone così il quesito per aumentare le possibilità di identificazione positiva inducendo gli utenti adulti a produrre una [a] più aperta del consueto. I bambini infatti sono portati a emettere una [a] molto più aperta degli adulti e, di conseguenza, le istanze memorizzate nel *database* hanno questa caratteristica.

Questi risultati, considerati anche i risultati analoghi ottenuti in un test di analisi della comprensione somministrato a bambini di 8 e 9 anni, indicano la necessità di un lavoro ulteriore per quanto riguarda questa modalità, ma non bocciano la tipologia di analisi scelta. Soltanto ponendo maggiore attenzione nella fase di creazione del *database* delle caratteristiche vocali, è probabile che si possano ottenere risultati migliori.

Tabella 4.1: *Risultati dei test sull'usabilità e il funzionamento dell'applicazione*

						Ampiezza		**Altezza**		**Durata**		**Vocali**				
N. test	**Età**	**S**	**Z4**	**P**	**Set**	**F**	**D**	**A**	**Gr**	**L**	**Br**	**A**	**O**	**E**	**I**	**U**
1	27	M	Sì	Sì	No	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
2	28	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
3	30	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	No	No	No	No
4	4	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
5	29	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	No	No	Sì	No
6	27	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	No	No	Sì	Sì
7	30	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
8	27	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
9	23	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
10	23	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	No	Sì	Sì	No
11	30	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
12	27	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
13	19	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	No	Sì	No
14	24	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
15	28	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
16	50	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
17	68	M	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
18	28	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
19	63	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	No	No
20	28	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
21	31	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
22	57	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
23	31	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
24	25	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
25	14	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
26	48	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No
27	17	F	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	Sì	No

Capitolo 5

Conclusioni

In questo lavoro si è visto come l'uso combinato del software *Pure Data* e della libreria *libpd*, una versione dello stesso integrabile in una applicazione *stand-alone*, consenta di velocizzare la creazione di applicazioni di elaborazione o sintesi audio in tempo reale. *Pure Data* è un ambiente di sviluppo visuale che facilita la produzione di applicazioni multimodali che si avvalgono della potenza di un *DSP software* altamente personalizzabile e flessibile.

Sono stati presentati gli strumenti attualmente più validi per fruire nello stesso progetto delle tecnologie di grafica tridimensionale e quelle di elaborazione audio *real time*, in particolare si è visto come *openFrameworks* sia l'unico *toolkit* di sviluppo che permetta questa integrazione.

Si sono delineate delle procedure funzionali alla realizzazione di una applicazione *stand-alone* multipiattaforma a partire da un prototipo costruito in *Pure Data*. Tra queste si sono presentati gli aspetti da considerare per potersi avvalere di un mercato vasto come quello fornito dall'*Apple App Store*, ovvero i problemi di compatibilità tra diverse licenze di distribuzione.

Il risultato di questo lavoro propone un'interfaccia semplice e intuitiva, utilizzabile senza supervisione dai bambini della scuola materna e primaria, una rappresentazione delle caratteristiche audio coerente e facilmente comprensibile, infine un funzionamento sufficientemente corretto nelle varie condizioni d'uso. Inoltre si presenta come una piattaforma facilmente adattabile ad altri usi o altri tipi di analisi audio. A questo proposito i test effettuati evidenziano un riconoscimento non ottimale delle vocali e quindi la necessità di migliorare la soluzione adottata, ad esempio con la creazione di un *database* di riferimento migliore, o di adottare soluzioni alternative, come un algoritmo *LPC*[1].

Infine si è visto come l'uso dei dispositivi mobili nell'ambito del trattamento di patologie che possono ostacolare la comunicazione sociale possa rappresentare un campo molto interessante

[1] *Linear Predictive Coding*.

per la ricerca di nuovi strumenti e tecnologie.

Appendici

Appendice A

Codice

A.1 Licenza

CULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

A.2 La dichiarazione della classe AppCore

Codice A.1: *AppCore.h*

```
#include "ofxPd.h"
#include "srSole.h"
#include "srButton.h"
#include "srConstants.h"
#include <vector>
#include "ofxTimer.h"

#ifndef TARGET_OF_IPHONE || TARGET_ANDROID
#include "ofxOsc.h"
#endif

using namespace pd;

class AppCore : public PdReceiver, public PdMidiReceiver {

    public:
        // main
        void setup(const int numOutChannels, const int numInChannels, const int
            ↪ sampleRate, const int ticksPerBuffer);
        void update();
        void draw();
        void exit();

        // audio callbacks
        void audioReceived(float * input, int bufferSize, int nChannels);
        void audioRequested(float * output, int bufferSize, int nChannels);

        // pd callbacks
        void print(const std::string& message);
        void receiveBang(const std::string& dest);
        void receiveFloat(const std::string& dest, float value);
        void receiveSymbol(const std::string& dest, const std::string& symbol);
        void receiveList(const std::string& dest, const List& list);
        void receiveMessage(const std::string& dest, const std::string& msg, const
            ↪ List& list);
        void receiveNoteOn(const int channel, const int pitch, const int velocity);
        void receiveControlChange(const int channel, const int controller, const int
            ↪ value);
        void receiveProgramChange(const int channel, const int value);
        void receivePitchBend(const int channel, const int value);
        void receiveAftertouch(const int channel, const int value);
        void receivePolyAftertouch(const int channel, const int pitch, const int
            ↪ value);
        void receiveMidiByte(const int port, const int byte);

        // touch callbacks
        void touchDown(ofTouchEventArgs &touch);

        // mouse callbacks
        void mousePressed(int x, int y, int button);

        // tone callback
```

```
49          void onToneTimerReached(ofEventArgs &args);
50
51          ofxPd pd;
52          ofTrueTypeFont font;
53          ofImage imgSfondoBack;
54          vector<ofImage> imgSfondoFront;
55          int indexSfondo;
56
57          srSole sole;
58          //Timer setting tono medio
59          ofxTimer toneTimer;
60
61          //-----Parte relativa all'interfaccia (pulsanti)-----//
62          vector<srButton> pulsanti; //qui il vettore non puo' essere inizializzato
63          //costruttore. In AppCore.cpp viene poi inizializzato il vettore nel
                ↪ costruttore
64          AppCore();
65          //distanza pulsante dal bordo destro della finestra
66          int distX_Pulsanti;
67          //distanza aggiuntiva tra un pulsante e l'altro
68          int distY_Pulsanti;
69          //dimensioni pulsanti
70          int pulsanteWidth, pulsanteHeight;
71          //distanza tra le anteprime dei temi
72          int distX_Temi;
73          //ampiezza barra pulsanti verticale
74          int widthVertBar;
75          //ampiezza barra pulsanti orizzontale
76          int widthOrizBar;
77          //altezza barra pulsanti orizzontale
78          int heightOrizBar;
79          //indice pulsante selezionato
80          int selectedButton;
81          //variabile indice vista
82          int page;
83          //variabile di memorizzazione scala altezza
84          int switchScala;
85          //variabile per spostare i pulsanti switchScala tra il pulsante pitchAnalisys
                ↪ e allAnalisys
86          int offSetBtnScala;
87          //memorizzo lo stato di voiceON
88          bool voiceON;
89
90  #ifndef TARGET_OF_IPHONE || TARGET_ANDROID
91          // OSC/UDP
92          ofxOscReceiver OSCreceiver;
93  #endif
94  };
```

A.3 L'implementazione della classe AppCore

Codice A.2: *AppCore.cpp*

```
1   #include "AppCore.h"
2   #include <Poco/Path.h>
3
4   //diciamo al compilatore di cercare funzioni C (non C++)
5   extern "C"{
6       void counter_setup();
7       void bfcc_tilde_setup();
```

```
8       void mfcc_setup();
9       void timbreID_setup();
10      void isteresi_setup();
11      void fiddle_setup();
12      void portaNOT_setup();
13      void normalizer2_setup();
14  }
15
16  // costanti
17  const bool filtroON = false; //filtroON=false ==> filtro attivo
18  const int scalaAssoluta = 0;
19  const int scalaRelativa = 1;
20
21  //Si inizializzano le dimensioni dei vettori nel costruttore
22  AppCore::AppCore() {
23      pulsanti = vector<srButton>(16);
24      imgSfondoFront = vector<ofImage>(4);
25  }
26
27  //--------------------------------------------------------------
28  void AppCore::setup(const int numOutChannels, const int numInChannels, const int
        ↪ sampleRate, const int ticksPerBuffer) {
29      // si attiva il ciclo OpenGL a 60fps
30      ofSetFrameRate(60);
31      ofSetVerticalSync(true);
32
33      // si seleziona la view
34      page = pageDefault;
35
36      // si imposta la scala assoluta delle altezze all'avvio
37      switchScala = scalaAssoluta;
38
39      // si inizializza voiceON
40      voiceON = false;
41
42      // si imposta il percorso della cartella delle risorse nel bundle applicazione di
            ↪ Mac OS X
43  #ifdef TARGET_OSX
44      ofSetDataPathRoot("../Resources/data/");
45  #endif
46
47      // si impostano dimensioni diverse dei font per i dispositivi mobili e non
48  #ifdef TARGET_OF_IPHONE || TARGET_ANDROID
49      font.loadFont("fonts/UnDotum.ttf", 11);
50  #else
51      font.loadFont("fonts/UnDotum.ttf", 20);
52  #endif
53
54      // si inizializza l'istanza di libpd
55      if(!pd.init(numOutChannels, numInChannels, sampleRate, ticksPerBuffer)) {
56          ofLog(OF_LOG_ERROR, "Could not init pd");
57          OF_EXIT_APP(1);
58      }
59
60      // add receive source names
61      pd.subscribe("vocale");
62      pd.subscribe("PITCH");
63      pd.subscribe("voiceON");
64      pd.subscribe("POWER");
65      pd.subscribe("sezione");
66      pd.subscribe("setSoglia");
67
68      // add listener
69      pd.addReceiver(*this);
70
```

```
71      // setup degli external
72      counter_setup();
73      bfcc_tilde_setup();
74      mfcc_setup();
75      timbreID_setup();
76      isteresi_setup();
77      fiddle_setup();
78      portaNOT_setup();
79      normalizer2_setup();
80
81      // audio processing on
82      pd.start();
83
84      // open patch
85      Patch patch = pd.openPatch("SoundRiseCore.pd");
86      cout << patch << endl;
87
88      // inizializzazione patch
89      pd.sendBang("setCore");
90      pd.sendFloat("filtroON", !filtroON);
91      pd.sendFloat("maxPitch", 68);
92      pd.sendFloat("maxIntensity", 140);
93      pd.sendFloat("switchScala", 0);
94
95      // si caricano le immagini di sfondo
96      imgSfondoBack.loadImage("Temi/MontagnaBack.png");
97      imgSfondoFront[0].loadImage("Temi/MontagnaFront.png");
98      imgSfondoFront[1].loadImage("Temi/MareFront.png");
99      imgSfondoFront[2].loadImage("Temi/EgittoFront.png");
100     imgSfondoFront[3].loadImage("Temi/Citta'Front.png");
101
102     // si inizializza il sole con le relative immagini
103     sole.setup("Sole/Sole.png", "Sole/Sole_sorrisoAperto.png",
            ↪ "Sole/Sole_sorrisoChiuso.png");
104
105 #ifndef TARGET_OF_IPHONE || TARGET_ANDROID
106     // attivazione server OSC per piattaforme diverse da iOS e Android
107     OSCreceiver.setup(OSCport);
108 #endif
109
110     /********************** Inizializzazione pulsanti **********************/
111     selectedButton = noSelectedButton;
112     //dimensione pulsanti e barre interfaccia
113     pulsanteWidth = ofGetWidth() / 12;
114     pulsanteHeight = pulsanteWidth;
115     widthVertBar = pulsanteWidth + 14;
116     distX_Pulsanti = (widthVertBar - pulsanteWidth) / 2;
117     distY_Pulsanti = ofGetHeight() / 30;
118     distX_Temi = ofGetWidth() / 20;
119     heightOrizBar = pulsanteHeight;
120     widthOrizBar = ofGetWidth() - widthVertBar;
121
122     pulsanti[0].setup("Pulsanti/Sezione0.png", ofGetWidth() - pulsanteWidth -
            ↪ distX_Pulsanti, distY_Pulsanti, pulsanteWidth, pulsanteHeight,
            ↪ ofColor::green, intensityAnalisys);
123     pulsanti[1].setup("Pulsanti/Sezione1.png", ofGetWidth() - pulsanteWidth -
            ↪ distX_Pulsanti, pulsanteHeight + 2*distY_Pulsanti, pulsanteWidth,
            ↪ pulsanteHeight, ofColor::green, pitchAnalisys);
124     pulsanti[2].setup("Pulsanti/Sezione2.png", ofGetWidth() - pulsanteWidth -
            ↪ distX_Pulsanti, 2*pulsanteHeight + 3*distY_Pulsanti, pulsanteWidth,
            ↪ pulsanteHeight, ofColor::green, durationAnalisys);
125     pulsanti[3].setup("Pulsanti/Sezione3.png", ofGetWidth() - pulsanteWidth -
            ↪ distX_Pulsanti, 3*pulsanteHeight + 4*distY_Pulsanti, pulsanteWidth,
            ↪ pulsanteHeight, ofColor::green, vowelAnalisys);
```

```
126     pulsanti[4].setup("Pulsanti/Sezione4.png", ofGetWidth() - pulsanteWidth -
            ↪ distX_Pulsanti, 4*pulsanteHeight + 5*distY_Pulsanti, pulsanteWidth,
            ↪ pulsanteHeight, ofColor::green, allAnalisys);
127     pulsanti[5].setup("Temi/MontagnaBack.png", distX_Temi, ofGetHeight() -
            ↪ pulsanteHeight, pulsanteWidth, pulsanteHeight);
128     pulsanti[6].setup("Temi/MareBack.png", pulsanteWidth + 2.5*distX_Temi,
            ↪ ofGetHeight() - pulsanteHeight, pulsanteWidth, pulsanteHeight);
129     pulsanti[7].setup("Temi/EgittoBack.png", 2*pulsanteWidth + 4*distX_Temi,
            ↪ ofGetHeight() - pulsanteHeight, pulsanteWidth, pulsanteHeight);
130     pulsanti[8].setup("Temi/Citta'Back.png", 3*pulsanteWidth + 5.5*distX_Temi,
            ↪ ofGetHeight() - pulsanteHeight, pulsanteWidth, pulsanteHeight);
131     pulsanti[9].setup("Pulsanti/Refresh.png", widthOrizBar - pulsanteWidth,
            ↪ ofGetHeight() - pulsanteHeight, pulsanteWidth, pulsanteHeight,
            ↪ ofColor::red);
132     //pulsanti[10].setup("", widthOrizBar, ofGetHeight() - pulsanteHeight,
            ↪ pulsanteWidth, pulsanteHeight);
133     pulsanti[11].setup("Pulsanti/PitchSetting.png", widthOrizBar - 2*pulsanteWidth -
            ↪ distX_Pulsanti, ofGetHeight() - pulsanteHeight, pulsanteWidth,
            ↪ pulsanteHeight);
134     pulsanti[12].setup("Pulsanti/Tono.png",distX_Temi, ofGetHeight() / 2,
            ↪ pulsanteWidth, pulsanteHeight);
135     pulsanti[13].setup("Pulsanti/Ok.png", pulsanteWidth + 2.5*distX_Temi,
            ↪ ofGetHeight() / 2, pulsanteWidth, pulsanteHeight);
136     pulsanti[14].setup("Pulsanti/ScalaA.png", ofGetWidth() - 1.6*pulsanteWidth,
            ↪ pulsanteHeight + 2*distY_Pulsanti, pulsanteWidth/2, pulsanteHeight/2);
137     pulsanti[15].setup("Pulsanti/ScalaR.png", ofGetWidth() - 1.6*pulsanteWidth,
            ↪ 1.5*pulsanteHeight + 2*distY_Pulsanti, pulsanteWidth/2, pulsanteHeight/2);
138
139     // Iscrizione all'evento TIMER_REACHED
140     ofAddListener(toneTimer.TIMER_REACHED, this, &AppCore::onToneTimerReached);
141 }
142
143 //--------------------------------------------------------------
144 void AppCore::update() {
145     // update posizione e dimensioni pulsanti
146     pulsanti[0].update(ofGetWidth() - pulsanteWidth - distX_Pulsanti, distY_Pulsanti,
            ↪ pulsanteWidth, pulsanteHeight);
147     pulsanti[1].update(ofGetWidth() - pulsanteWidth - distX_Pulsanti, pulsanteHeight +
            ↪ 2*distY_Pulsanti, pulsanteWidth, pulsanteHeight);
148     pulsanti[2].update(ofGetWidth() - pulsanteWidth - distX_Pulsanti, 2*pulsanteHeight
            ↪ + 3*distY_Pulsanti, pulsanteWidth, pulsanteHeight);
149     pulsanti[3].update(ofGetWidth() - pulsanteWidth - distX_Pulsanti, 3*pulsanteHeight
            ↪ + 4*distY_Pulsanti, pulsanteWidth, pulsanteHeight);
150     pulsanti[4].update(ofGetWidth() - pulsanteWidth - distX_Pulsanti, 4*pulsanteHeight
            ↪ + 5*distY_Pulsanti, pulsanteWidth, pulsanteHeight);
151     pulsanti[5].update(distX_Temi,  ofGetHeight() - pulsanteHeight, pulsanteWidth,
            ↪ pulsanteHeight);
152     pulsanti[6].update(pulsanteWidth + 2.5*distX_Temi, ofGetHeight() - pulsanteHeight,
            ↪ pulsanteWidth, pulsanteHeight);
153     pulsanti[7].update(2*pulsanteWidth + 4*distX_Temi, ofGetHeight() - pulsanteHeight,
            ↪ pulsanteWidth, pulsanteHeight);
154     pulsanti[8].update(3*pulsanteWidth + 5.5*distX_Temi, ofGetHeight() -
            ↪ pulsanteHeight, pulsanteWidth, pulsanteHeight);
155     pulsanti[9].update(ofGetWidth() - widthVertBar - pulsanteWidth, ofGetHeight() -
            ↪ pulsanteHeight, pulsanteWidth, pulsanteHeight);
156     //pulsanti[10].update(ofGetWidth() - widthVertBar + distX_Pulsanti, ofGetHeight()
            ↪ - pulsanteHeight, pulsanteWidth, pulsanteHeight);
157     pulsanti[11].update(ofGetWidth() - widthVertBar - 2*pulsanteWidth -
            ↪ distX_Pulsanti, ofGetHeight() - pulsanteHeight, pulsanteWidth,
            ↪ pulsanteHeight);
158     pulsanti[12].update(distX_Temi, ofGetHeight() / 2, pulsanteWidth, pulsanteHeight);
159     pulsanti[13].update(pulsanteWidth + 2.5*distX_Temi, ofGetHeight() / 2,
            ↪ pulsanteWidth, pulsanteHeight);
160     pulsanti[14].update(ofGetWidth() - 1.6*pulsanteWidth, pulsanteHeight +
            ↪ 2*distY_Pulsanti + offSetBtnScala, pulsanteWidth/2, pulsanteHeight/2);
```

```
161     pulsanti[15].update(ofGetWidth() - 1.6*pulsanteWidth, 1.5*pulsanteHeight +
            ↪ 2*distY_Pulsanti + offSetBtnScala, pulsanteWidth/2, pulsanteHeight/2);
162
163     // update posizione, dimensione, colore e stato del sole
164     sole.update();
165
166     // logica ricezione messaggi OSC
167 #ifndef TARGET_OF_IPHONE || TARGET_ANDROID
168     while(OSCreceiver.hasWaitingMessages()){
169         ofxOscMessage OSCmsg;
170         OSCreceiver.getNextMessage(&OSCmsg);
171         if(strcmp(OSCmsg.getAddress().c_str(), "/num") == 0){
172             int sez = -1;
173             switch(OSCmsg.getArgType(0)){
174             case OFXOSC_TYPE_INT32:
175                 sez = OSCmsg.getArgAsInt32(0);
176                 break;
177             case OFXOSC_TYPE_FLOAT:
178                 sez = (int)OSCmsg.getArgAsFloat(0);
179                 break;
180             default:
181                 cout << "Messaggio OSC non riconosciuto!" << endl;
182                 break;
183             }
184             // mappa zone stanza LogoMotoria <==> caratteristiche audio analizzate
185             switch(sez){
186             case 2:
187                 sez = 0;
188                 break;
189             case 4:
190                 sez = 1;
191                 break;
192             case 6:
193                 sez = 2;
194                 break;
195             case 8:
196                 sez = 3;
197                 break;
198             case 9:
199                 sez = 4;
200                 break;
201             default:
202                 sez = -1;
203                 break;
204             }
205             // se il messaggio e' valido imposto la sezione nella patch
206             if (sez != -1) pd.sendFloat("sezione", sez);
207         }
208     }
209 #endif
210 }
211
212 //--------------------------------------------------------------
213 void AppCore::draw() {
214
215     ofSetColor(255, 255, 255);
216     imgSfondoBack.draw(0, 0, ofGetWidth(), ofGetHeight());
217
218     if (selectedButton != noSelectedButton) sole.draw();
219
220     ofEnableAlphaBlending();
221     imgSfondoFront[indexSfondo].draw(0, 0, ofGetWidth(), ofGetHeight());
222
223     //gestione delle view
224     switch(page){
```

```
225         case pageDefault:
226             //barra grigia che contiene i pulsanti
227             ofEnableAlphaBlending();
228             ofSetColor(120, 120, 120, 130);
229             //barra pulsanti verticale
230             ofRect(ofGetWidth() - widthVertBar, 0, widthVertBar, ofGetHeight());
231             //barra pulsanti orizzontale
232             ofRect(0, ofGetHeight() - pulsanteHeight, ofGetWidth() - widthVertBar,
                    ↪ heightOrizBar);
233             ofDisableAlphaBlending();
234             pulsanti[0].draw();
235             pulsanti[1].draw();
236             pulsanti[2].draw();
237             pulsanti[3].draw();
238             pulsanti[4].draw();
239             pulsanti[5].draw();
240             pulsanti[6].draw();
241             pulsanti[7].draw();
242             pulsanti[8].draw();
243             pulsanti[9].draw();
244             //pulsanti[10].draw();
245             pulsanti[11].draw();
246             break;
247         case pageSetting:
248             ofEnableAlphaBlending();
249             ofSetColor(120, 120, 120, 130);
250 
251             //barra desrizione funzionamento setting
252 #ifdef TARGET_OF_IPHONE || TARGET_ANDROID
253 #define FONT_MARGIN 20
254 #else
255 #define FONT_MARGIN 30
256 #endif
257             ofRect(0, 0, FONT_MARGIN + font.stringWidth("Premere OK per uscire
                    ↪ dalla\n"), ofGetHeight());
258             ofDisableAlphaBlending();
259             ofSetColor(255, 0, 0);
260             font.drawString("Premi il pulsante con il\n"
261                             "profilo della faccia e\n"
262                             "produci con la voce una A\n"
263                             "lunga fino a quando\n"
264                             "il pulsante non si spegne.\n\n"
265                             "Premere OK per uscire dalla\n"
266                             "schermata di settings.", FONT_MARGIN / 2, FONT_MARGIN);
267 
268             pulsanti[12].draw();
269             pulsanti[13].draw();
270             break;
271         case pagePitchPressed:
272             //barra grigia che contiene i pulsanti
273             ofEnableAlphaBlending();
274             ofSetColor(120, 120, 120, 130);
275             //barra pulsanti verticale
276             ofRect(ofGetWidth() - widthVertBar, 0, widthVertBar, ofGetHeight());
277             //barra pulsanti orizzontale
278             ofRect(0, ofGetHeight() - pulsanteHeight, ofGetWidth() - widthVertBar,
                    ↪ heightOrizBar);
279             ofDisableAlphaBlending();
280             pulsanti[0].draw();
281             pulsanti[1].draw();
282             pulsanti[2].draw();
283             pulsanti[3].draw();
284             pulsanti[4].draw();
285             pulsanti[5].draw();
286             pulsanti[6].draw();
```

```
287             pulsanti[7].draw();
288             pulsanti[8].draw();
289             pulsanti[9].draw();
290             //pulsanti[10].draw();
291             pulsanti[11].draw();
292             pulsanti[14].draw();
293             pulsanti[15].draw();
294     }
295 }
296
297 //--------------------------------------------------------------
298 void AppCore::exit() {
299     pd.closePatch("SoundRiseCore.pd");
300     ofRemoveListener(toneTimer.TIMER_REACHED, this, &AppCore::onToneTimerReached);
301 }
302
303 //--------------------------------------------------------------
304 void AppCore::audioReceived(float * input, int bufferSize, int nChannels) {
305     pd.audioIn(input, bufferSize, nChannels);
306 }
307
308 //--------------------------------------------------------------
309 void AppCore::audioRequested(float * output, int bufferSize, int nChannels) {
310     pd.audioOut(output, bufferSize, nChannels);
311 }
312
313 //--------------------------------------------------------------
314 void AppCore::print(const std::string& message) {
315     cout << message << endl;
316 }
317
318 //--------------------------------------------------------------
319 void AppCore::receiveBang(const std::string& dest) {
320     cout << "OF: bang " << dest << endl;
321 }
322
323 void AppCore::receiveFloat(const std::string& dest, float value) {
324     cout << "OF: float " << dest << ": " << value << endl;
325     // si aggiorna lo stato dell'interfaccia grafica alla ricezione di valori dalla
        ↪ patch
326     if (dest.compare("PITCH") == 0) {
327         sole.setAltezza(value);
328     } else if (dest.compare("voiceON") == 0) {
329         sole.setVoiceOn(value);
330         voiceON = value;
331     } else if (dest.compare("POWER") == 0) {
332         sole.setIntensita(value);
333     } else if (dest.compare("sezione") == 0) {
334         sole.setSezione(value);
335         sole.setVoiceOn(voiceON);
336     }
337 }
338
339 void AppCore::receiveSymbol(const std::string& dest, const std::string& symbol) {
340     cout << "OF: symbol " << dest << ": " << symbol << endl;
341 }
342
343 void AppCore::receiveList(const std::string& dest, const List& list) {
344     cout << "OF: list " << dest << ": " << list << endl;
345     // ricezione della vocale riconosciuta dalla patch e comunicazione del suo valore
        ↪ al sole
346     if (dest.compare("vocale") == 0) {
347         sole.setVocale(list.getFloat(0), list.getFloat(1));
348     }
349 }
```

```
350
351 void AppCore::receiveMessage(const std::string& dest, const std::string& msg, const
        ↪ List& list) {
352     cout << "OF: msg " << dest << ": " << msg << " " << list.toString() <<
            ↪ list.types() << endl;
353     // deselezione pulsante di ricalcolo soglia quando la patch ha terminato il calcolo
354     if (dest.compare("setSoglia") == 0) {
355         pulsanti[9].setSelected(false);
356     }
357 }
358
359 //--------------------------------------------------------------
360 void AppCore::receiveNoteOn(const int channel, const int pitch, const int velocity) {
361     cout << "OF: note: " << channel << " " << pitch << " " << velocity << endl;
362 }
363
364 void AppCore::receiveControlChange(const int channel, const int controller, const int
        ↪ value) {
365     cout << "OF: ctl: " << channel << " " << controller << " " << value << endl;
366 }
367
368 void AppCore::receiveProgramChange(const int channel, const int value) {
369     cout << "OF: pgm: " << channel << " " << value << endl;
370 }
371
372 void AppCore::receivePitchBend(const int channel, const int value) {
373     cout << "OF: bend: " << channel << " " << value << endl;
374 }
375
376 void AppCore::receiveAftertouch(const int channel, const int value) {
377     cout << "OF: touch: " << channel << " " << value << endl;
378 }
379
380 void AppCore::receivePolyAftertouch(const int channel, const int pitch, const int
        ↪ value) {
381     cout << "OF: polytouch: " << channel << " " << pitch << " " << value << endl;
382 }
383
384 void AppCore::receiveMidiByte(const int port, const int byte) {
385     cout << "OF: midibyte: " << port << " " << byte << endl;
386 }
387
388 /************************ Logica pulsanti touch ***************************/
389 void AppCore::touchDown(ofTouchEventArgs &touch) {
390     if (pulsanti[0].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
391         page = pageDefault;
392         pd.sendFloat("sezione", pulsanti[0].getId());
393     }
394     else if (pulsanti[1].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
395         if (pulsanti[1].isSelected()) {
396             page = pagePitchPressed;
397             pulsanti[14].setSelected(!switchScala);
398             pulsanti[15].setSelected(switchScala);
399             offSetBtnScala = 0;
400         }
401         else {
402             page = pageDefault;
403         }
404         pd.sendFloat("sezione", pulsanti[1].getId());
405     }
406     else if (pulsanti[2].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
407         page = pageDefault;
```

```
408         pd.sendFloat("sezione", pulsanti[2].getId());
409     }
410     else if (pulsanti[3].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
411         page = pageDefault;
412         pd.sendFloat("sezione", pulsanti[3].getId());
413     }
414     else if (pulsanti[4].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
415         if (pulsanti[4].isSelected()) {
416             page = pagePitchPressed;
417             pulsanti[14].setSelected(!switchScala);
418             pulsanti[15].setSelected(switchScala);
419             offSetBtnScala = 3 * (pulsanteHeight + distY_Pulsanti);
420         }
421         else {
422             page = pageDefault;
423         }
424         pd.sendFloat("sezione", pulsanti[4].getId());
425     }
426     else if (pulsanti[5].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
427         imgSfondoBack = pulsanti[5].getImg();
428         indexSfondo = Montagna;
429     }
430     else if (pulsanti[6].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
431         imgSfondoBack = pulsanti[6].getImg();
432         indexSfondo = Mare;
433     }
434     else if (pulsanti[7].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
435         imgSfondoBack = pulsanti[7].getImg();
436         indexSfondo = Egitto;
437     }
438     else if (pulsanti[8].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
439         imgSfondoBack = pulsanti[8].getImg();
440         indexSfondo = Citta;
441     }
442     else if (pulsanti[9].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
443         if (!pulsanti[9].isSelected()) {
444             pd.sendBang("newSoglia");
445             pulsanti[9].setSelected(true);
446         }
447     }
448     else if (pulsanti[11].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
449         if (selectedButton != noSelectedButton) {
450             pulsanti[selectedButton].setSelected(false);
451             sole.setIntensita(0);
452             selectedButton = noSelectedButton;
453         }
454         page = pageSetting;
455     }
456     else if (pulsanti[12].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
457         pulsanti[12].setSelected(true);
458         pd.sendBang("getTone");
459         toneTimer.setup(2000, false);
460         selectedButton = pitchAnalisys;
461         pd.sendFloat("sezione", pitchAnalisys);
462         pd.sendFloat("switchScala", scalaAssoluta);
463     }
```

```
464     else if (pulsanti[13].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
465         page = pageDefault;
466     }
467     else if (pulsanti[14].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
468         switchScala = scalaAssoluta;
469         pd.sendFloat("switchScala", switchScala);
470         pulsanti[14].setSelected(!switchScala);
471         pulsanti[15].setSelected(switchScala);
472     }
473     else if (pulsanti[15].pressedButton(touch.x, touch.y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
474         switchScala = scalaRelativa;
475         pd.sendFloat("switchScala", switchScala);
476         pulsanti[14].setSelected(!switchScala);
477         pulsanti[15].setSelected(switchScala);
478     }
479 }
480
481 /*************************** Logica pulsanti ****************************/
482 void AppCore::mousePressed(int x, int y, int button) {
483     if (pulsanti[0].pressedButton(x, y, &selectedButton, pulsanti[selectedButton], pd,
            ↪ sole)){
484         page = pageDefault;
485         pd.sendFloat("sezione", pulsanti[0].getId());
486     }
487     else if (pulsanti[1].pressedButton(x, y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
488         if (pulsanti[1].isSelected()) {
489             page = pagePitchPressed;
490             pulsanti[14].setSelected(!switchScala);
491             pulsanti[15].setSelected(switchScala);
492             offSetBtnScala = 0;
493         }
494         else {
495             page = pageDefault;
496         }
497         pd.sendFloat("sezione", pulsanti[1].getId());
498     }
499     else if (pulsanti[2].pressedButton(x, y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
500         page = pageDefault;
501         pd.sendFloat("sezione", pulsanti[2].getId());
502     }
503     else if (pulsanti[3].pressedButton(x, y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
504         page = pageDefault;
505         pd.sendFloat("sezione", pulsanti[3].getId());
506     }
507     else if (pulsanti[4].pressedButton(x, y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
508         if (pulsanti[4].isSelected()) {
509             page = pagePitchPressed;
510             pulsanti[14].setSelected(!switchScala);
511             pulsanti[15].setSelected(switchScala);
512             offSetBtnScala = 3 * (pulsanteHeight + distY_Pulsanti);
513         }
514         else {
515             page = pageDefault;
516         }
517         pd.sendFloat("sezione", pulsanti[4].getId());
518     }
519     else if (pulsanti[5].pressedButton(x, y, &selectedButton,
            ↪ pulsanti[selectedButton], pd, sole)){
```

```
520         imgSfondoBack = pulsanti[5].getImg();
521         indexSfondo = Montagna;
522     }
523     else if (pulsanti[6].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
524         imgSfondoBack = pulsanti[6].getImg();
525         indexSfondo = Mare;
526     }
527     else if (pulsanti[7].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
528         imgSfondoBack = pulsanti[7].getImg();
529         indexSfondo = Egitto;
530     }
531     else if (pulsanti[8].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
532         imgSfondoBack = pulsanti[8].getImg();
533         indexSfondo = Citta;
534     }
535     else if (pulsanti[9].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
536         if (!pulsanti[9].isSelected()) {
537             pd.sendBang("newSoglia");
538             pulsanti[9].setSelected(true);
539         }
540     }
541     /*else if (pulsanti[10].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
542         if (ofGetWindowMode() == OF_WINDOW) ofSetFullscreen(true);
543         else ofSetFullscreen(false);
544     }*/
545     else if (pulsanti[11].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
546         if (selectedButton != noSelectedButton) {
547             pulsanti[selectedButton].setSelected(false);
548             sole.setIntensita(0);
549             selectedButton = noSelectedButton;
550         }
551         page = pageSetting;
552     }
553     else if (pulsanti[12].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
554         pulsanti[12].setSelected(true);
555         pd.sendBang("getTone");
556         toneTimer.setup(2000, false);
557         selectedButton = pitchAnalisys;
558         pd.sendFloat("sezione", pitchAnalisys);
559         pd.sendFloat("switchScala", scalaAssoluta);
560     }
561     else if (pulsanti[13].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
562         page = pageDefault;
563     }
564     else if (pulsanti[14].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
565         switchScala = scalaAssoluta;
566         pd.sendFloat("switchScala", switchScala);
567         pulsanti[14].setSelected(!switchScala);
568         pulsanti[15].setSelected(switchScala);
569     }
570     else if (pulsanti[15].pressedButton(x, y, &selectedButton,
        ↪ pulsanti[selectedButton], pd, sole)){
571         switchScala = scalaRelativa;
572         pd.sendFloat("switchScala", switchScala);
573         pulsanti[14].setSelected(!switchScala);
574         pulsanti[15].setSelected(switchScala);
```

```
575     }
576 }
577
578 void AppCore::onToneTimerReached(ofEventArgs &args) {
579     // allo scadere del temporizzatore si termina la memorizzazione del tono
580     pulsanti[12].setSelected(false);
581     pd.sendBang("getTone");
582     selectedButton = noSelectedButton;
583     pd.sendFloat("switchScala", switchScala);
584 }
```

A.4 La classe di supporto srSole

Codice A.3: *srSole.h*

```
1  #ifndef __SOUNDRISE_SOLE_
2  #define __SOUNDRISE_SOLE_
3
4  #include <math.h>
5  #include "ofMain.h"
6  #include "ofxTweenzor.h"
7  #include "srConstants.h"
8  #include "ofxTimer.h"
9
10 // costanti delle vocali
11 const int vocaleA = 0;
12 const int vocaleE = 1;
13 const int vocaleI = 2;
14 const int vocaleO = 3;
15 const int vocaleU = 4;
16 const int silenzio = 5;
17
18 //costanti dei colori
19 const int verde = 0;
20 const int rosso = 1;
21 const int arancione = 2;
22 const int blu = 3;
23 const int grigio = 4;
24 const int giallo = 5;
25
26 //Tempo di transizione da un colore ad un altro
27 const float tempo_TransColor = 1.0f;
28
29 class srSole {
30     int x, y, r, colore;
31     float alpha;
32     int sezione;
33     bool voiceOn;
34     int scalaAltezza;
35     ofImage imgSole, imgSoleSorrisoAperto, imgSoleSorrisoChiuso;
36     float sunColor_r, sunColor_g, sunColor_b;
37     ofColor E_color;
38     ofColor A_color;
39     ofColor O_color;
40     ofColor I_color;
41     ofColor U_color;
42     ofColor default_color;
43     //Timer ritardo visualizzazione sole in pitch analisys per far apparire il sole
           ↪ direttamente alla posizione del tono emesso
44     ofxTimer pitchTimer;
```

```
45      bool isPichAnalysed;
46
47  public:
48      srSole();
49      ~srSole();
50      void setup(string sole, string soleSorrisoAperto, string soleSorrisoChiuso);
51      float getAltezza();
52      float getIntensita();
53      float getSezione();
54      void setAltezza(float altezza);
55      void setIntensita(float intensita);
56      void setSezione(float sezione);
57      void setVoiceOn(float voiceOn);
58      void setVocale(int vocale);
59      void setVocale(int vocale, float confidenza);
60      void draw();
61      void update();
62      // pitch callback
63      void onPitchTimerReached(ofEventArgs &args);
64  };
65
66  #endif /*__SOUNDRISE_SOLE_*/
```

Codice A.4: *srSole.cpp*

```
1   #include "srSole.h"
2
3   srSole::srSole() {
4   }
5
6   srSole::~srSole() {
7       ofRemoveListener(pitchTimer.TIMER_REACHED, this, &srSole::onPitchTimerReached);
8       Tweenzor::destroy();
9   }
10
11  void srSole::setup(string sole, string soleSorrisoAperto, string soleSorrisoChiuso) {
12      this->scalaAltezza = (int)round(ofGetHeight() / 8.0);
13      this->x = ofGetWidth() / 2.0;
14      this->y = ofGetHeight() / 2.0 + this->scalaAltezza;
15      this->r = 0;
16      this->sezione = intensityAnalisys;
17      this->voiceOn = false;
18      this->imgSole.loadImage(sole);
19      this->imgSole.resize(256, 256);
20      this->imgSoleSorrisoAperto.loadImage(soleSorrisoAperto);
21      this->imgSoleSorrisoAperto.resize(256, 256);
22      this->imgSoleSorrisoChiuso.loadImage(soleSorrisoChiuso);
23      this->imgSoleSorrisoChiuso.resize(256, 256);
24
25      //Utilizzo addon Tweenzor per effetto transizione colore sole
26      Tweenzor::init();  // must call this before adding any tweens
27      //giallo del sole
28      sunColor_r = 255.f;
29      sunColor_g = 255.f;
30      sunColor_b = 0.f;
31      //verde per la vocale E
32      E_color.r = 0.f;
33      E_color.g = 255.f;
34      E_color.b = 0.f;
35      //rosso per la vocale A
36      A_color.r = 255.f;
37      A_color.g = 0.f;
38      A_color.b = 0.f;
39      //arancione per la vocale O
```

```
40      O_color.r = 255.f;
41      O_color.g = 120.f;
42      O_color.b = 0.f;
43      //blu per la vocale I
44      I_color.r = 0.f;
45      I_color.g = 0.f;
46      I_color.b = 255.f;
47      //grigio per la vocale U
48      U_color.r = 120.f;
49      U_color.g = 120.f;
50      U_color.b = 120.f;
51      //colore giallo per il default
52      default_color.r = 255.f;
53      default_color.g = 255.f;
54      default_color.b = 0.f;
55
56      //Iscrizione all'evento TIMER_REACHED
57      ofAddListener(pitchTimer.TIMER_REACHED, this, &srSole::onPitchTimerReached);
58
59  }
60
61  void srSole::update(){
62      Tweenzor::update(ofGetElapsedTimeMillis());
63      //ricalcolo posizionamento e dimensione del sole
64      this->scalaAltezza = (int)round(ofGetHeight() / 8.0);
65      this->x = ofGetWidth() / 2.0;
66      if (!isPichAnalysed) this->y = ofGetHeight() / 2.0 - this->scalaAltezza;
67      //this->r = this->scalaAltezza;
68  }
69
70  float srSole::getAltezza() { //valori uscita nel range [-1, 3]
71      return (float)((this->y - (ofGetHeight() / 2.0)) / (2.0 * this->scalaAltezza));
72  }
73
74  float srSole::getIntensita() { //valori uscita nel range [1, 3]
75      return (float)this->r;
76  }
77
78  float srSole::getSezione() {
79      return (float)this->sezione;
80  }
81
82  void srSole::setAltezza(float altezza) { //valori ingresso nel range [-1, 3]
83      this->y = (ofGetHeight() / 2.0) - (altezza * this->scalaAltezza);
84  }
85
86  void srSole::setIntensita(float intensita) { //valori ingresso nel range [1, 3]
87      this->r = intensita * (this->scalaAltezza / 2.0);
88  }
89
90  void srSole::setSezione(float sezione) {
91      this->sezione = (int)trunc(sezione);
92      //resetto il sole (posizione e dimensione iniziali)
93      this->y = ofGetHeight() / 2.0;
94      if (sezione == durationAnalisys) this->r = this->scalaAltezza; //sole a meta'
            ↪ altezza
95      else this->r = 0; //sole non visibile
96  }
97
98  void srSole::setVoiceOn(float voiceOn) {
99      switch (this->sezione) {
100         case allAnalisys:
101             isPichAnalysed =true;
102             this->voiceOn = voiceOn;
103             break;
```

```
104         case durationAnalisys:
105             isPichAnalysed =false;
106             this->voiceOn = voiceOn;
107             break;
108         case vowelAnalisys:
109             cout << "vowelAnalisys" << endl;
110         case pitchAnalisys:
111             cout << "pitchAnalisys" << endl;
112             isPichAnalysed =true;
113             if (voiceOn == true) {
114                 pitchTimer.setup(100, false);
115             }
116             else this->r = 0;
117             break;
118         case intensityAnalisys:
119             isPichAnalysed =false;
120             cout << "intensityAnalisys" << endl;
121             this->r = this->r * (int)voiceOn;
122             break;
123         default:
124             break;
125     }
126 }
127
128 void srSole::setVocale(int vocale) {
129     setVocale(vocale, 1.0);
130 }
131
132 void srSole::setVocale(int vocale, float confidenza) {
133     this->alpha = confidenza;
134     switch (vocale){
135         case vocaleE:
136             this->colore = verde;
137             Tweenzor::removeAllTweens();
138             // add a tween that uses frames as time
139             Tweenzor::add(&sunColor_r, sunColor_r, E_color.r, 0.f, tempo_TransColor);
140             Tweenzor::add(&sunColor_g, sunColor_g, E_color.g, 0.f, tempo_TransColor);
141             Tweenzor::add(&sunColor_b, sunColor_b, E_color.b, 0.f, tempo_TransColor);
142             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
143             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
144             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
145             break;
146         case vocaleA:
147             this->colore = rosso;
148             Tweenzor::removeAllTweens();
149             // add a tween that uses frames as time
150             Tweenzor::add(&sunColor_r, sunColor_r, A_color.r, 0.f, tempo_TransColor);
151             Tweenzor::add(&sunColor_g, sunColor_g, A_color.g, 0.f, tempo_TransColor);
152             Tweenzor::add(&sunColor_b, sunColor_b, A_color.b, 0.f, tempo_TransColor);
153             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
154             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
155             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
156             break;
157         case vocaleO:
158             this->colore = arancione;
159             Tweenzor::removeAllTweens();
160             // add a tween that uses frames as time
161             Tweenzor::add(&sunColor_r, sunColor_r, O_color.r, 0.f, tempo_TransColor);
162             Tweenzor::add(&sunColor_g, sunColor_g, O_color.g, 0.f, tempo_TransColor);
163             Tweenzor::add(&sunColor_b, sunColor_b, O_color.b, 0.f, tempo_TransColor);
164             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
165             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
166             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
167             break;
168         case vocaleI:
```

```
169             this->colore = blu;
170             Tweenzor::removeAllTweens();
171             // add a tween that uses frames as time
172             Tweenzor::add(&sunColor_r, sunColor_r, I_color.r, 0.f, tempo_TransColor);
173             Tweenzor::add(&sunColor_g, sunColor_g, I_color.g, 0.f, tempo_TransColor);
174             Tweenzor::add(&sunColor_b, sunColor_b, I_color.b, 0.f, tempo_TransColor);
175             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
176             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
177             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
178             break;
179         case vocaleU:
180             this->colore = grigio;
181             Tweenzor::removeAllTweens();
182             // add a tween that uses frames as time
183             Tweenzor::add(&sunColor_r, sunColor_r, U_color.r, 0.f, tempo_TransColor);
184             Tweenzor::add(&sunColor_g, sunColor_g, U_color.g, 0.f, tempo_TransColor);
185             Tweenzor::add(&sunColor_b, sunColor_b, U_color.b, 0.f, tempo_TransColor);
186             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
187             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
188             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
189             break;
190         case silenzio:
191             this->colore = giallo;
192             Tweenzor::removeAllTweens();
193             // add a tween that uses frames as time
194             Tweenzor::add(&sunColor_r, sunColor_r, default_color.r, 0.f,
                    ↪ tempo_TransColor);
195             Tweenzor::add(&sunColor_g, sunColor_g, default_color.g, 0.f,
                    ↪ tempo_TransColor);
196             Tweenzor::add(&sunColor_b, sunColor_b, default_color.b, 0.f,
                    ↪ tempo_TransColor);
197             Tweenzor::getTween(&sunColor_r)->setRepeat(1, true);
198             Tweenzor::getTween(&sunColor_g)->setRepeat(1, true);
199             Tweenzor::getTween(&sunColor_b)->setRepeat(1, true);
200             break;
201     }
202 }
203
204
205 void srSole::draw() {
206
207     ofPushStyle();
208     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_ADD);
209     ofEnableBlendMode(OF_BLENDMODE_ADD);
210     //Abilito la trasparenza per poter settare a 255 l'alpha
211     ofEnableAlphaBlending();
212     ofSetColor(255, 255, 0, 255);
213     if (this->sezione == vowelAnalisys) isPichAnalysed = false;
214     if ((this->sezione == vowelAnalisys) || (this->sezione == allAnalisys)) {
215         switch(this->colore){
216             case verde:
217                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
218                 break;
219             case rosso:
220                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
221                 break;
222             case arancione:
223                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
224                 break;
225             case blu:
226                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
227                 break;
228             case grigio:
229                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
230                 break;
```

```
231             case giallo:
232                 ofSetColor(sunColor_r, sunColor_g, sunColor_b);
233                 break;
234         }
235     }
236 
237 #ifdef TARGET_OPENGLES
238 
239     this->imgSole.getTextureReference().bind();
240     GLfloat vertici[] = {
241         x - r, y - r,
242         x + r, y - r,
243         x - r, y + r,
244         x + r, y - r,
245         x + r, y + r,
246         x - r, y + r
247      };
248      GLfloat texCoords[] = {
249      0.0, 0.0,
250      1.0, 0.0,
251      0.0, 1.0,
252      1.0, 0.0,
253      1.0, 1.0,
254      0.0, 1.0
255      };
256 
257     glEnableClientState(GL_TEXTURE_COORD_ARRAY);
258     glTexCoordPointer(2, GL_FLOAT, 0, texCoords);
259     glEnableClientState(GL_VERTEX_ARRAY);
260     glVertexPointer(2, GL_FLOAT, 0, vertici);
261     glDrawArrays(GL_TRIANGLES, 0, 6);
262     this->imgSole.getTextureReference().unbind();
263 
264     if ((this->sezione == durationAnalisys) || (this->sezione == allAnalisys)) {
265         ofSetColor(0, 0, 0, 255);
266         if (this->voiceOn) {
267             this->imgSoleSorrisoAperto.getTextureReference().bind();
268             glTexCoordPointer(2, GL_FLOAT, 0, texCoords);
269             glDrawArrays(GL_TRIANGLES, 0, 6);
270             this->imgSoleSorrisoAperto.getTextureReference().unbind();
271         } else {
272             this->imgSoleSorrisoChiuso.getTextureReference().bind();
273             glTexCoordPointer(2, GL_FLOAT, 0, texCoords);
274             glDrawArrays(GL_TRIANGLES, 0, 6);
275             this->imgSoleSorrisoChiuso.getTextureReference().unbind();
276         }
277     }
278 
279     glDisableClientState(GL_VERTEX_ARRAY);
280     glDisableClientState(GL_TEXTURE_COORD_ARRAY);
281 
282 #else //OPENGL
283 
284     this->imgSole.draw(x-r, y-r, 2*r, 2*r);
285     if ((this->sezione == durationAnalisys) || (this->sezione == allAnalisys)) {
286         ofSetColor(0, 0, 0, 255);
287         if (this->voiceOn) {
288             this->imgSoleSorrisoAperto.draw(x-r, y-r, 2*r, 2*r);
289         } else {
290             this->imgSoleSorrisoChiuso.draw(x-r, y-r, 2*r, 2*r);
291         }
292     }
293 
294 #endif
295 
```

```
296     ofDisableAlphaBlending();
297     ofDisableBlendMode();
298     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
299     ofPopStyle();
300 }
301
302 void srSole::onPitchTimerReached(ofEventArgs &args) {
303     this->r = this->scalaAltezza;
304 }
```

A.5 La classe di supporto srButton

Codice A.5: *srButton.h*

```
1  #ifndef SRBUTTON_H
2  #define SRBUTTON_H
3
4  #include <math.h>
5  #include "ofMain.h"
6  #include "ofxPd.h"
7  #include "srConstants.h"
8  #include "srSole.h"
9
10 const string button = "Pulsanti/button.png";
11
12 class srButton
13 {
14     ofImage imgPulsante;
15     ofImage imgButton;
16     ofImage imgButtonSelected;
17     bool selected;
18     int x, y, width, height, index;
19
20     public:
21     void setup(string pulsante, int xCord, int yCord, int widthP, int heightP, ofColor
           ↪ selectedColor = ofColor::green, int indexSezione = defaultButton);
22         bool pressedButton(int xCord, int yCord, int *selectedButton, srButton
               ↪ &pulsanteSelezionato, ofxPd &pd, srSole &sole);
23         bool isSelected();
24         void setSelected(bool selected);
25         void draw();
26         int getId();
27         ofImage getImg();
28         void update(int xCord, int yCord, int widthP, int heightP);
29 };
30
31 #endif // SRBUTTON_H
```

Codice A.6: *srButton.cpp*

```
1  #include "srButton.h"
2
3  void srButton::setup(string pulsante, int xCord, int yCord, int widthP, int heightP,
       ↪ ofColor selectedColor, int indexSezione){
4
5      //identificativo sezione
6      this->index = indexSezione;
7      //dimensione pulsante
8      this->width = widthP;
9      this->height = heightP;
```

```
10      //coordinate pulsante
11      this->x = xCord;
12      this->y = yCord;
13      //il pulsante viene settato come non selezionato
14      this->selected = false;
15      //carico l'immagine del pulsante
16      this->imgPulsante.loadImage(pulsante);
17      //carico l'immagine button che crea l'effetto del pulsante
18      this->imgButton.loadImage(button);
19      ofImage imgTemp = this->imgPulsante;
20      imgTemp.crop((imgTemp.getWidth() - imgTemp.getHeight()) / 4, 0,
            ↪ imgTemp.getHeight(), imgTemp.getHeight());
21      imgTemp.resize(this->imgButton.getWidth(), this->imgButton.getHeight());
22      imgTemp.update();
23      ofColor buttonColor;
24      for(int i = 0; i < this->imgButton.getWidth(); i++){
25          for(int j = 0; j < this->imgButton.getHeight(); j++){
26              buttonColor = this->imgButton.getColor(i, j) * imgTemp.getColor(i, j);
27              buttonColor.a = this->imgButton.getColor(i, j).a;
28              this->imgButton.setColor(i, j, buttonColor);
29          }
30      }
31      this->imgButton.update();
32      // immagine che crea l'effetto della selezione del pulsante
33      this->imgButtonSelected = this->imgButton;
34      ofColor buttonSelectedColor;
35      for(int i = 0; i < this->imgButtonSelected.getWidth(); i++){
36          for(int j = 0; j < this->imgButtonSelected.getHeight(); j++){
37              buttonSelectedColor = this->imgButtonSelected.getColor(i, j) *
                    ↪ selectedColor;
38              buttonSelectedColor.a = this->imgButtonSelected.getColor(i, j).a;
39              this->imgButtonSelected.setColor(i, j, buttonSelectedColor);
40          }
41      }
42      this->imgButtonSelected.update();
43  }
44
45  void srButton::update(int xCord, int yCord, int widthP, int heightP){
46      //dimensioni pulsante
47      this->width = widthP;
48      this->height = heightP;
49      //coordinate pulsante
50      this->x = xCord;
51      this->y = yCord;
52  }
53
54  //controlla quale pulsante relativo alla sezione e' stato cliccato
55  bool srButton::pressedButton(int xCord, int yCord, int *selectedButton, srButton
        ↪ &pulsanteSelezionato, ofxPd &pd, srSole &sole){
56      if(xCord > this->x && xCord < this->x+this->width && yCord > this->y && yCord <
            ↪ this->y+this->height){ //se sono stato premuto
57          if (this->index != defaultButton){ //se sono un pulsante sezione
58              if (*selectedButton == this->index){ //se ero gia' selezionato mi
                    ↪ deseleziono
59                  sole.setIntensita(0);
60                  this->selected = false;
61                  *selectedButton = noSelectedButton;
62                  return true;
63              }
64              if (*selectedButton == noSelectedButton){ //se non c'era un altro pulsante
                    ↪ selezionato
65                      this->selected = true;
66                      *selectedButton = this->index;
67              }
68              else{ //se c'era un altro pulsante selezionato
```

```
                pulsanteSelezionato.setSelected(false);
                this->selected = true;
                *selectedButton = this->index;
            }
            return true;
        }
        else{ //sono un pulsante generico
            return true;
        }
    }
    else return false;
}

void srButton::setSelected(bool selected){
    this->selected = selected;
}


bool srButton::isSelected(){
    return this->selected;
}

void srButton::draw(){
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_ADD);
    ofEnableBlendMode(OF_BLENDMODE_ADD);
    //Abilito la trasparenza per poter settare a 255 l'alpha
    ofEnableAlphaBlending();

    ofSetColor(0, 0, 0, 255);
    if (!selected) this->imgButton.draw(x, y, height, height);
    else this->imgButtonSelected.draw(x, y, height, height);//evidenzio il pulsante
        ↪ selezionato

    ofDisableAlphaBlending();
    ofDisableBlendMode();
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
}

int srButton::getId() {
    return this->index;
}

ofImage srButton::getImg() {
    return this->imgPulsante;
}
```

A.6 Le definizioni delle costanti

Codice A.7: *srConstants.h*

```
#ifndef SRCONSTANTS_H_INCLUDED
#define SRCONSTANTS_H_INCLUDED

// costanti delle sezioni
const int intensityAnalisys = 0;
const int pitchAnalisys = 1;
const int durationAnalisys = 2;
const int vowelAnalisys = 3;
const int allAnalisys = 4;

```

```
11 //costanti dei temi
12 const int Montagna = 0;
13 const int Mare = 1;
14 const int Egitto = 2;
15 const int Citta = 3;
16
17 //costante porta OSC
18 const int OSCport = 9997;
19
20 //costante nessun pulsante selezionato
21 const int noSelectedButton = -2;
22 //costante pulsante generico
23 const int defaultButton = -1;
24
25 //costante delle viste
26 const int pageDefault = 0;
27 const int pageSetting = 1;
28 const int pagePitchPressed = 2;
29
30 #endif // SRCONSTANTS_H_INCLUDED
```

Bibliografia

[1] A. M. Oster. Clinical applications of computer-based speech training for children with hearing impairment. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP*, volume 1, pp. 157–160. IEEE, 1996.

[2] M. Russell, C. Brown, A. Skilling, R. Series, J. Wallace e P. Barker. Applications of automatic speech recognition to speech and language development in young children. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP*, volume 1 di *International Conference on Spoken Language Processing, ICSLP, Proceedings*, pp. 176–179. IEEE, 1996.

[3] N. Al-Ghamdi, A. Al-Nafjan e Y. Al-Ohali. A Computer-Based Aural Rehabilitation Assistant for Cochlear Implanted Children. In *Proceeding of International Conference on Future Information Technology*, volume 13, pp. 279–284, 2011.

[4] T. Caplan e F. Caplan. *The Early Childhood Years: The 2 to 6 Year Old*. SOS Free Stock, 1984.

[5] D. Ling. *Speech and the Hearing-Impaired Child: Theory and Practice*. Alex Graham Bell Assn for Deaf, 2002.

[6] K. Vicsi, P. Roach, A. Oster, Z. Kacic, F. Csatàri, A. Sfakianaki e R. Veronic. A multilingual, multimodal, speech training system, SPECO. In *Proceeding of EUROSPEECH 2001 Scandinavia*, pp. 2807 – 2810, 2001.

[7] K. Vicsi e À. Vàry. Distinctive training methods and Evaluation of a Multilingual , Multimodal Speech Training System. In *Proceeding of Fourth International Conference on Disability, Virtual Reality and Associated Technologies, Veszprém*, pp. 47–52, 2002.

[8] D. Jones. *The Pronunciation of English*. Cambridge University Press, 1966.

[9] L. Rabiner e B. Juang. An introduction to hidden Markov models. *Current protocols in bioinformatics editoral board Andreas D Baxevanis et al*, 3:4–16, 1986.

[10] A. J. A. Soleymani, M. J. Mccutcheon e M. H. Southwood. Design Of Speech Illumina Mentor (SIM) For Teaching Speech To The Hearing Impaired. In *In Proceedings of The Sixteenth Southern Biomedical Engineering Conferences*, pp. 425–428, 1997.

[11] O. Bälter, O. Engwall, A. M. Öster e H. Sidenbladh-kjellström. Wizard-of-Oz Test of ARTUR- a Computer-Based Speech Training System with Articulation Correction. In *Proceedings of the Seventh International ACM SIGACCESS Conference on Computers and Accessibility, Baltimora*, pp. 36–43, 2005.

[12] H. Fell, C. J. Cress, J. MacAuslan e L. Ferrier. visiBabble for reinforcement of early vocalization. *Proceedings of the ACM SIGACCESS conference on Computers and accessibility ASSETS 04*, (77-78):161, 2003.

[13] Autismo - Wikipedia. Sito web: `http://it.wikipedia.org/wiki/Autismo` (Visitato il: 09 July 2012).

[14] K. Dautenhahn e I. Werry. Towards Interactive Robots in Autism Therapy: Background, Motivation and Challenges. *Pragmatics Cognition*, 12(1):1–35, 2004.

[15] Proloquo2Go. Sito web: `http://www.assistiveware.com/product/proloquo2go` (Visitato il: 09 July 2012).

[16] Apps for Autism, aired on Oct. 23, 2011. Sito web: `http://www.cbsnews.com/video/watch/?id=7385686n` (Visitato il: 09 July 2012).

[17] Touch Autism. Sito web: `http://touchautism.com/` (Visitato il: 09 July 2012).

[18] S. Giusto. SoundRise: studio e progettazione di un'applicazione multimodale interattiva per la didattica basata sull'analisi di feature vocali. Tesi magistrale, 2012.

[19] G. De Poli e F. Avanzini. *Algorithms for Sound and Music Computing*. 2010.

[20] J. Simner, J. Ward, M. Lanz, A. Jansari, K. Noonan, L. Glover e David a Oakley. Non-random associations of graphemes to colours in synaesthetic and non-synaesthetic populations. *Cognitive neuropsychology*, 22(8):1069–85, gennaio 2005.

[21] P. Akhter. An extensive review on children's learning process through their use of digital technology at home. In *2011 5th International Conference on Software, Knowledge Information, Industrial Management and Applications (SKIMA)*, pp. 1–7. IEEE, settembre 2011.

[22] Sparrow Framework - The Open Source Game Engine for iOS. Sito web: `http://gamua.com/sparrow/` (Visitato il: 09 July 2012).

[23] iSGL3D. Sito web: `http://isgl3d.com/` (Visitato il: 09 July 2012).

[24] COCOS2D. Sito web: `http://www.cocos2d-iphone.org/` (Visitato il: 09 July 2012).

[25] Moai |The mobile platform for pro game developers. Sito web: `http://getmoai.com/` (Visitato il: 09 July 2012).

[26] emo-framework - open source game engine for Android and iOS. Sito web: `http://www.emo-framework.com/` (Visitato il: 09 July 2012).

[27] openFrameworks. Sito web: `http://www.openframeworks.cc/` (Visitato il: 09 July 2012).

[28] Xcode 4 - Apple Developer. Sito web: `https://developer.apple.com/xcode/` (Visitato il: 09 July 2012).

[29] Code::Blocks. Sito web: `http://www.codeblocks.org/` (Visitato il: 09 July 2012).

[30] MinGW. Sito web: `http://www.mingw.org/` (Visitato il: 09 July 2012).

[31] P. Brinkmann, C. Mccormick, P. Kirn, M. Roth e R. Lawler. Embedding Pure Data with libpd. *of the Pure Data*, 2011.

[32] Filtro Butterworth - Wikipedia. Sito web: `http://it.wikipedia.org/wiki/Filtro_Butterworth` (Visitato il: 09 July 2012).

[33] A. Camurri, G. Volpe, S. Canazza e C. Canepa. The 'Stanza Logo– Motoria': an interactive environment for learning and communication. *Proceedings of SMC*, 2010.

[34] W. Brent. A timbre analysis and classification toolkit for pure data. *Proceedings of the International Computer*, pp. 2–7, 2010.

[35] Lawrence Rabiner e Biing-Hwang Juang. *Fundamentals of Speech Recognition*, volume 103 di *Prentice Hall signal processing series*. Prentice Hall, 1993.

[36] E. Terhardt, S. Gerhard e S. Manfred. Algorithm for extraction of pitch and pitch salience from complex tonal signals. *The Journal of the Acoustical Society of America*, 71(3):679, marzo 1982.

[37] iOS Technology Overview - About iOS Development. Sito web: `http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html` (Visitato il: 09 July 2012).

[38] The GNU General Public License v3.0. Sito web: `http://www.gnu.org/copyleft/gpl.html` (Visitato il: 09 July 2012).

[39] iTUNES STORE - TERMS AND CONDITIONS. Sito web: `http://www.apple.com/legal/itunes/us/terms.html` (Visitato il: 09 July 2012).