UNIVERSITÀ DI PADOVA **FACOLTÀ DI INGEGNERIA**

TESI DI LAUREA

SVILUPPO IN AMBIENTE ANDROID DI UN'INTERFACCIA GRAFICA PER STRUMENTI MUSICALI ELETTRONICI VIRTUALI

Laureando: Laura Nao

Relatore: Prof. Federico Avanzini
Correlatore: Ing. Michele Geronazzo

Corso di Laurea Triennale in Ingegneria dell'informazione

Anno accademico 2012/2013

Sommario

In questa tesi verrà sviluppata l'interfaccia grafica di un'applicazione Android che simuli una parte della strumentazione acustica dello Studio di Fonologia RAI di Milano, uno studio fondato negli anni '50 che ha avuto un ruolo di fondamentale importanza nella nascita e nello sviluppo della musica elettroacustica in Europa. Il lavoro fa parte del progetto europeo D.R.E.A.M., a cui si deve l'installazione al Museo del Castello Sforzesco di Milano che ospita la ricostruzione di parte degli strumenti dello Studio di Fonologia. L'interfaccia grafica dell'applicazione è stata sviluppata utilizzando il framework JUCE, una libreria C++ ideata per lo sviluppo di software multipiattaforma. Il lavoro svolto funge da punto di partenza per un'applicazione Android da integrare successivamente in un plug-in VST utilizzabile nei più comuni software di sintesi audio. Di seguito una breve descrizione dei capitoli:

Primo capitolo
In questo capitolo viene percorsa la storia dello Studio di Fonologia di Milano e descritti brevemente i dispositivi presenti. Sono poi esposti i risultati e gli obiettivi del progetto D.R.E.A.M. .

Secondo capitolo
In questo capitolo è presentato l'ambiente di sviluppo del lavoro di tesi. Viene descritta la libreria JUCE utilizzata e gli strumenti di scrittura e manipolazione del codice; ci si sposta poi alla configurazione dei software per la compilazione dell'applicazione Android. Infine è introdotto il software ausiliario KnobMan, utilizzato per realizzare le grafiche per l'applicazione.

Terzo capitolo
Descrive in dettaglio lo sviluppo del codice per l'interfaccia grafica dell'applicazione, utilizzando il software Introjucer. Sono poi spiegate le operazioni svolte per la compilazione e l'esecuzione sul tablet dell'applicazione Android.

Quarto capitolo
Presenta gli obiettivi e gli sviluppi futuri dell'applicazione.

Indice

Sommario

Capitolo 1

Lo studio di Fonologia Musicale RAI

1.1 Storia

Lo Studio di Fonologia Musicale venne fondato nel 1955 negli uffici della “RAI – Radiotelevisione Italiana” di Milano, accogliendo l'iniziativa dei compositori italiani Luciano Berio e Bruno Maderna. Nel maggio 1956 lo Studio venne inaugurato ufficialmente a Milano, presentandosi come uno dei laboratori più equipaggiati nel panorama della musica elettroacustica internazionale. In pochi anni lo Studio divenne un punto di riferimento per molti compositori contemporanei, affiancando i già esistenti Studio für Elektronische Musik (WDR) di Colonia e il Groupe de Recherches Musicales (GRM) di Parigi. Fra i tanti compositori che lavorarono nello studio vi sono gli italiani Luigi Nono, Nicolo` Castiglioni, Aldo Clementi, Franco Donatoni, Armando Gentilucci, Giacomo Manzoni, Gino Marinuzzi Jr., Angelo Paccagnini, Salvatore Sciarrino, Camillo Togni, Roman Vlad. Fra gli anni '50 e '60 lo Studio attirò l'attenzione anche di artisti internazionali come Henry Pousseur e John Cage.
Lo Studio si differenzia musicalmente, tecnologicamente e tecnicamente dai due poli europei di Parigi e Colonia, che si distinguevano per la scelta delle sorgenti sonore (nella “musica concreta” del GRM i suoni e i rumori naturali venivano ripresi da microfoni mentre nella “musica elettronica” del WDR i suoni erano generati da oscillatori o ricavati dal rumore bianco con filtri) ma utilizzavano gli stessi mezzi elettronici per la successiva elaborazione dei suoni. Lo spazio del nuovo Studio è concepito come un unico strumento musicale, dando la possibilità ai compositori di utilizzare entrambi i metodi in funzione delle esigenze artistiche. Lo Studio è stato vissuto dai compositori come mezzo di emancipazione dagli strumenti tradizionali, con i suoi nove oscillatori,generatori di suono bianco, selezionatore d'ampiezza,modulatori,filtri. Questi strumenti permettevano di generare,manipolare,misurare e controllare il suono in tempo reale prima di registrare su nastro magnetico ed erano pezzi unici, ideati e creati dal fisico Alfredo Lietti per incontrare le esigenze degli artisti che frequentavano lo Studio. Il fatto di possedere 9 oscillatori intonati su frequenze diverse (rispetto all'unico di Colonia) e di poter connettere gli apparati senza uno schema prefissato fornivano ai compositori la massima libertà espressiva.
A sovrintendenza delle apparecchiature elettroniche e a sostegno dei compositori che per la prima volta si approcciavano ai nuovi strumenti era Marino Zuccheri, tecnico e unico impiegato stabile dello Studio dalla sua apertura alla chiusura avvenuta con il suo pensionamento. Avendo lavorato alla RAI (che all'epoca si chiamava EIAR, Ente Italiano Audizioni Radiofoniche) fin dal 1942 fu la vera anima dello Studio, non solo guidava gli artisti a comprendere e sfruttare al meglio le potenzialità dei nuovi strumenti ma collaborava con loro in modo creativo dalla ripresa e registrazione del suono alla sua trasformazione e infine al montaggio dei nastri magnetici e al missaggio.
“…Mi piace ricordare Marino nel suo Studio di Fonologia, maestro tra i maestri, maestro del suono tra i maestri della musica, perché per lui il suono non aveva segreti, dato che aveva fatto la gavetta negli

auditori lavorando alla Radio con i più famosi registi dell'epoca. Lui ricordava sempre che aveva inziato per caso il lavoro in Fonologia, ma certamente non per caso ha continuato negli anni, dato che è stato l'unico titolare dello Studio dalla fondazione (1955) alla chiusura (1983)"[1].
Nel 1957 lo Studio subì un sostanziale rinnovamento in cui parte dei dispositivi andarono persi, alcuni vennero ricostruiti usando la tecnologia a transistor e molti degli apparati costruiti fino al 1950 vennero riutilizzati. Negli archivi dello Studio vennero conservate fotografie, disegni, articoli e gli schemi originali degli apparati di Lietti, permettendo oggi di comprendere le funzionalità e le caratteristiche anche dei dispositivi andati persi.
Dopo la chiusura dello studio nel 1983, gli apparati vennero disassemblati e conservati al Museo della Radio della Rai di Torino. Nel 2003 le apparecchiature vennero riportate a Milano e rese accessibili al pubblico in un'esibizione permanente del Museo degli strumenti musicali. Nonostante gli strumenti originali non siano oggi funzionanti è possibile ascoltare le registrazioni delle composizioni musicali che videro la luce nello Studio [2].

1.2 Strumenti in Studio

Di seguito verranno elencati gli strumenti più significativi presenti in Studio, la maggior parte disegnati e costruiti da Lietti stesso [3].

Oscillatori
Gli oscillatori, costruiti tra il 1955 e il 1956, generano un'onda sinusoidale in grado di oscillare con una stabilità di uno su trentamila. L'interfaccia consiste di un display e due manopole, la prima da sinistra per selezionare fra sei bande di frequenza e la seconda per il controllo fine della frequenza nel range della banda desiderata. Il display è composto di sei diverse scale, una per ciascuna delle bande. Lo Studio disponeva di 9 oscillatori con range di frequenze diverse.

Figura 1.1: Martino Berio e uno degli oscillatori

Miscelatore a 9 ingressi
Permette la miscelazione di nove segnali, mediante nove dosatori rotativi. Serviva principalmente a miscelare i suoni prodotti dai nove oscillatori.

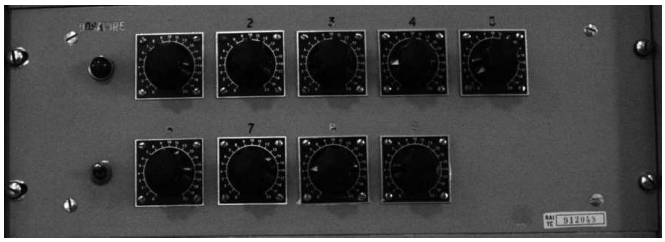


Figura 1.2: Mixer a 9 ingressi

Miscelatore a 8 ingressi

Era utilizzato principalmente per miscelare i suoni risultanti dai banchi di filtri.

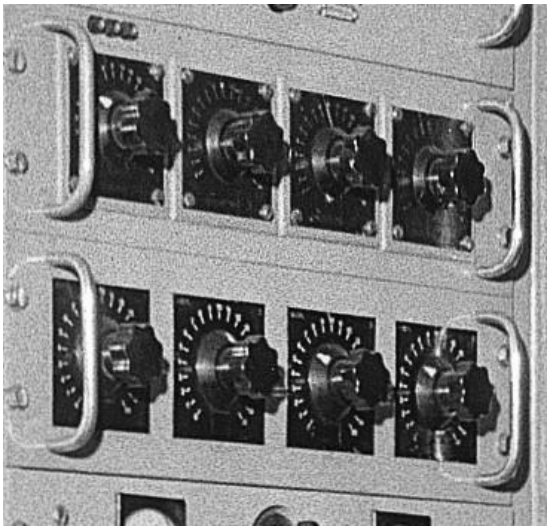


Figura 1.3: Mixer a 8 ingressi

Generatore di suono bianco

Genera rumore bianco amplificando il segnale di tensione generato da una valvola a vuoto. L'interfaccia è composta di due display, un controllo di volume e l'interruttore per l'accensione.

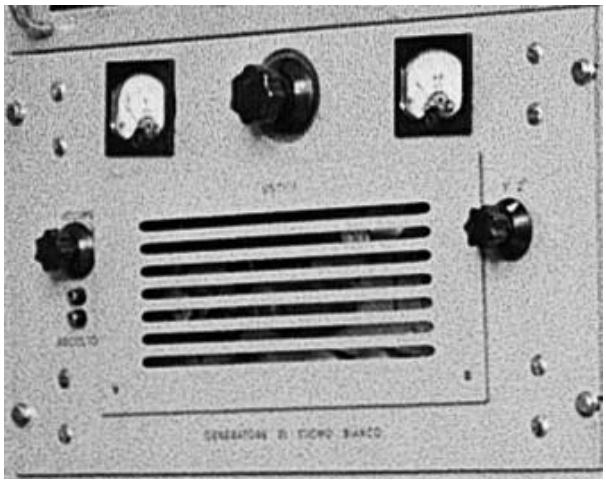


Figura 1.4: Generatore di rumore bianco

Selezionatore d'ampiezza

Messo in esercizio fra il 1956 e il 1957, è un dispositivo che trasforma il segnale in ingresso lasciando inalterate le porzioni di suono con ampiezza maggiore di una determinata soglia e sopprimendo le altre. L'interfaccia è composta di un'interruttore per l'accensione, un Vu-meter e due manopole, la prima a sinistra per impostare la soglia al di sotto della quale il segnale viene soppresso, la seconda per selezionare la velocità con cui entra in funzione il circuito di selezione.

Figura 1.5: Selezionatore d'ampiezza

Filtri d'ottava
E' un banco di 6 filtri passa-banda, dimensionati sulle bande di frequenza 200-400 Hz, 400-800 Hz, 800-1600 Hz, 1600-3200 Hz, 3200-6400 Hz, 6400-12800 Hz.

Figura 1.6: Filtri d'ottava

Filtri Passa-alto
Banco di sette filtri passa-alto con frequenza di taglio in Hz pari a 280, 560, 1100, 2200, 4500, 8500, 160.

Figura 1.7: Filtri passa-alto

Filtri Passa-basso
Banco di sette filtri passa-basso con frequenza di taglio in Hz pari a 280, 560, 1100, 2200, 4500, 8500, 6000.

Figura 1.8: Filtri passa-basso

Filtro passa-banda variabile
Presente nello Studio fin dalla sua aperture,Il Krohn-Hite 310-A è composto dalla serie di un filtro passa-basso e di uno passa-alto. Il rotore di sinistra seleziona la frequenza di taglio per il filtro passa-basso, quello di destra la frequenza di taglio del passa-alto. Il rotore inferiore imposta il fattore moltiplicativo (x1, x10, x100, x1000), mentre quello superiore permette di scegliere un valore tra 20 e 200. In questo modo è possibile selezionare una frequenza di taglio tra 20Hz e 200kHz.

Figura 1.9: Filtro passa-banda variabile

Generatore di "Toc"
Gli impulsi generati dal dispositivo potevano essere utilizzati per controllare l'inviluppo di ampiezza dei suoni presenti all'ingresso del modulatore di ampiezza.

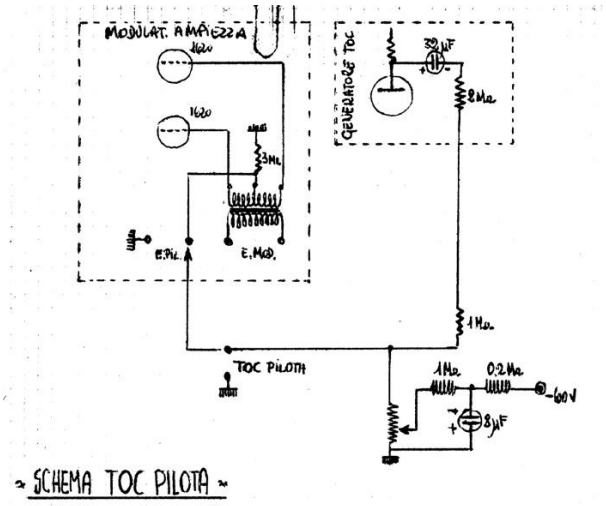


Figura 1.10: Schema circuitale del generatore di Toc

Modulatore ad anello

E' un dispositivo in grado di moltiplicare tra loro i segnali in ingresso.

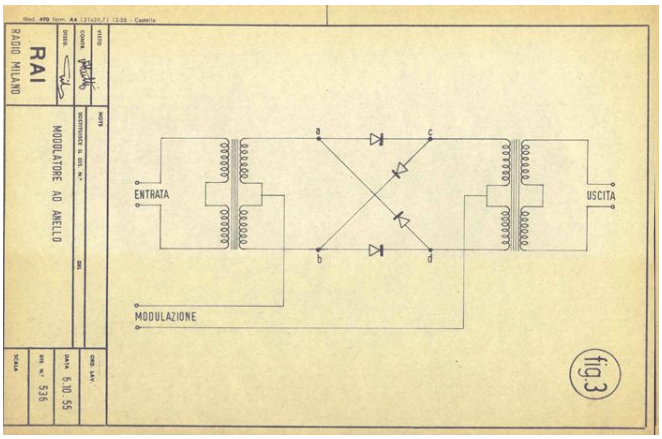


Figura 1.11: Schema circuitale del modulatore ad anello

Modulatore di ampiezza

Consente di modulare l'ampiezza del segnale in ingresso mediante un segnale sinusoidale a bassa frequenza,ottenendo un effetto di tremolo.

Figura 1.12: Modulatore di ampiezza

Modulatore dinamico
Il modulatore dinamico ha due segnali di ingresso e uno di uscita, l'ampiezza del primo segnale in ingresso viene modulata dall'ampiezza del segnale al secondo ingresso. E' dotato di potenziometri che ne regolano il funzionamento. Venne messo in esercizio nel 1958.

Figura 1.13: Modulatore dinamico

Traspositore di frequenza
Permette di modulare in frequenza il segnale in ingresso mediante l'utilizzo di due modulatori ad anello, due oscillatori a 20kHz, un filtro passa-alto con frequenza di taglio a 20kHz e un filtro passa-basso. Venne messo in esercizio nel 1959.

Figura 1.14: Traspositore di frequenza

1.3 Progetto D.R.E.A.M.

Lo Studio giocò un ruolo fondamentale nella nascita e nello sviluppo della musica elettronica sperimentale e costituisce oggi un patrimonio tecnologico imprescindibile.
La salvaguardia dei lavori di musica elettroacustica è complessa, ogni produzione si compone di materiale grafico e testuale (schemi, indicazioni, suggerimenti), documenti audio (l'intera composizione o singole parti), dispositivi software (per la sintesi del suono o la registrazione) e strumenti musicali elettrofoni (l'insieme delle apparecchiature è comunemente detto sistema). La conservazione di questi materiali solleva problematiche di natura diversa, essendo documentata non solo su carta ma facendo uso anche di supporti magnetici volatili. Per fornire una testimonianza completa di come un compositore operava in un dato periodo non è sufficiente conservare un singolo elemento di un sistema. Mantenere in vita l'eredità della musica elettroacustica dello Studio richiede

da un lato la preservazione dei pezzi originali e dall'altro un processo di restauro, atto a ricreare l'ambiente in cui gli artisti lavoravano. È necessario che i documenti audio originali vengano protetti dagli agenti esterni mantenendo inalterati i componenti elettronici (preservazione “passiva”) e vengano trasferiti nel dominio digitale (presevazione “attiva”)[4].
A testimonianza dell'importanza di conservare la memoria dello Studio di Fonologia, il 17 Settembre 2008 venne inaugurato uno spazio dedicato presso il Museo degli Strumenti Musicali di Milano. Il Museo ospita la ricostruzione dell'ambiente di lavoro originale e include le apparecchiature per la generazione di suoni elettronici, i dispositivi per la trasformazione e combinazione di suoni, per la registrazione,riproduzione e per l'ascolto. Il progetto, denominato D.R.E.A.M. (Digital Re-working /Re-appropriation of Electro-Acoustic Music) e sostenuto dall'Unione Europea, venne avviato dalla consulente RAI Maria Maddalena Novati ed è frutto della collaborazione dell’Università di Padova, della Aalborg University di Copenhagen, la Middlesex University, il comune di Milano e la RAI. Il progetto ha come obiettivo la creazione di un'installazione interattiva con sistemi software e hardware che ricreino la liuteria elettronica dello Studio. Sono stati ricostruiti, usando la tecnologia digitale, tre dei 9 oscillatori e il relativo mixer, il generatore di rumore bianco, il selezionatore d'ampiezza con il banco di filtri d'ottava e il relativo mixer a 8 ingressi. La parte di sintesi audio è stata sviluppata utilizzando il linguaggio Pure Data [6] e l'interazione con le interfacce tangibili, costruite per rispecchiare al massimo gli strumenti originali, è stata realizzata con l'hardware open-source Arduino. Il progetto si pone come obiettivo anche la creazione di un'applicazione Android che simuli il funzionamento degli strumenti sopra elencati; lo sviluppo dell'interfaccia è oggetto di studio di questa tesi.
L’installazione con gli strumenti ricostruiti è stata presentata il 15 giugno 2012 nel Museo, all’interno del Castello Sforzesco a Milano.

Capitolo 2

Ambiente di sviluppo

In questo capitolo verranno descritti brevemente i software utilizzati per realizzare l'interfaccia dell'applicazione. Il primo paragrafo tratta della libreria JUCE e degli strumenti The Jucer e Introjucer utilizzati per la scrittura del codice; viene poi esposto il procedimento per creare un primo progetto di applicazione in Introjucer. Il secondo paragrafo espone brevemente la configurazione del software Eclipse e degli strumenti necessari al build dell'applicazione Android. Il terzo paragrafo descrive infine il software KnobMan, utilizzato per creare le immagini delle manopole, degli interruttori e di tutte le altre parti dell'interfaccia grafica.

2.1 Juce Framework

2.1.1 Introduzione alla libreria JUCE

JUCE (Jules' Utility Class Extensions) è una libreria C++ ideata per lo sviluppo di software multipiattaforma. La libreria include una moltitudine di classi, che implementano la maggior parte delle funzionalità richieste nello sviluppo di un'applicazione. In particolar modo permette di creare interfacce grafiche altamente personalizzate e facilita la gestione e l'integrazione di risorse multimediali quali immagini, audio e file MIDI. La libreria nasce come derivato dello sviluppo di vari software audio della Raw Material Software ltd (come Tracktion, PPMulator, SoundBasket), azienda fondata nel 1999 da Julian Storer. Una delle caratteristiche più importanti di JUCE è il supporto cross-platform. Gli strumenti inclusi nella libreria si occupano della creazione del progetto specifico per ogni piattaforma, lasciando all'utente solo la scrittura del codice relativo all'applicazione. In questo modo, se il codice C++ scritto è portabile, è sufficiente ricompilare l'applicazione per eseguirla su diversi sistemi operativi. Le piattaforme supportate da JUCE sono Mac OSX, Windows, Linux, iOS, Android. La maggior parte dei moduli della libreria è rilasciata sotto licenza GPL, ma sono disponibili anche delle licenze a pagamento per la commercializzazione di applicazioni proprietarie.
La libreria è suddivisa in sotto-moduli, ovvero in cartelle ciascuna contenente classi che implementano i vari componenti e la loro interazione:

- *juce_core* – classi fondamentali per il raggruppamento e il collegamento dei componenti
- *juce_events* – classi per la gestione e l'invio di messaggi
- *juce_data_structures* – classi per la gestione di strutture dati

- *juce_graphics* – motore di render 2D, gestione del formato delle immagini e dei font
- *juce_gui_basics* – classi con le GUI (Graphical User Interface) fondamentali
- *juce_gui_extra* – classi con altre GUI più particolari ed elaborate
- *juce_audio_basics* – classi per la gestione dei dati audio e MIDI
- *juce_audio_devices* – classi per la gestione su più piattaforme dell'audio in input/output
- *juce_audio_formats* – classi per la scrittura e lettura del formato dei file audio
- *juce_audio_processors* – classi per la gestione di plugin audio
- *juce_audio_plugin_client* – classi per la creazione di client per plugin audio
- *juce_audio_utils* – classi aggiuntive per file audio e GUI
- *juce_cryptography* – classi per crittografare i dati con RSA, Blowfish, MD5, SHA, ecc
- *juce_opengl* – classi per la gestione di openGL
- *juce_video* – classi per la cattura e la riproduzione video
- *juce_box2d* – classi per l'aggiunta di librerie box2d[1]

Al fine di ricreare l'aspetto e le funzionalità dei vari strumenti è necessario che i controlli che realizzano l'interfaccia siano gestibili dall'utente tramite touchscreen. Per quanto riguarda l'implementazione delle varie parti che costituiscono ogni strumento in JUCE sono presenti classi per la realizzazione di bottoni, slider, editor di testo, tabelle, finestre con schede, alberi e molti altri. L'interazione con l'utente di questi componenti (tramite mouse, tastiera o touchscreen) è già implementata ed è possibile modificare il comportamento di ciascuno tramite metodi dedicati. Tutte le classi che implementano questi oggetti sono contenute nella classe `Component`[2], situata nel modulo *juce_gui_basics* (Figura 2.1). Per quanto riguarda invece l'aspetto grafico dei controlli è stata utilizzata la classe `Look And Feel`, situata anch'essa nel modulo *juce_gui_basics.* Questa classe contiene i metodi responsabili di disegnare le varie parti di ogni componente; sovrascrivendo tali metodi è possibile personalizzare la forma, lo sfondo, la posizione e i colori di ciascun controllo.

1 La documentazione per box2d è disponibile all'indirizzo http://box2d.org/.

2 La classe Component eredita la classe MouseListener, responsabile di gestire l'interazione del mouse con ogni oggetto.

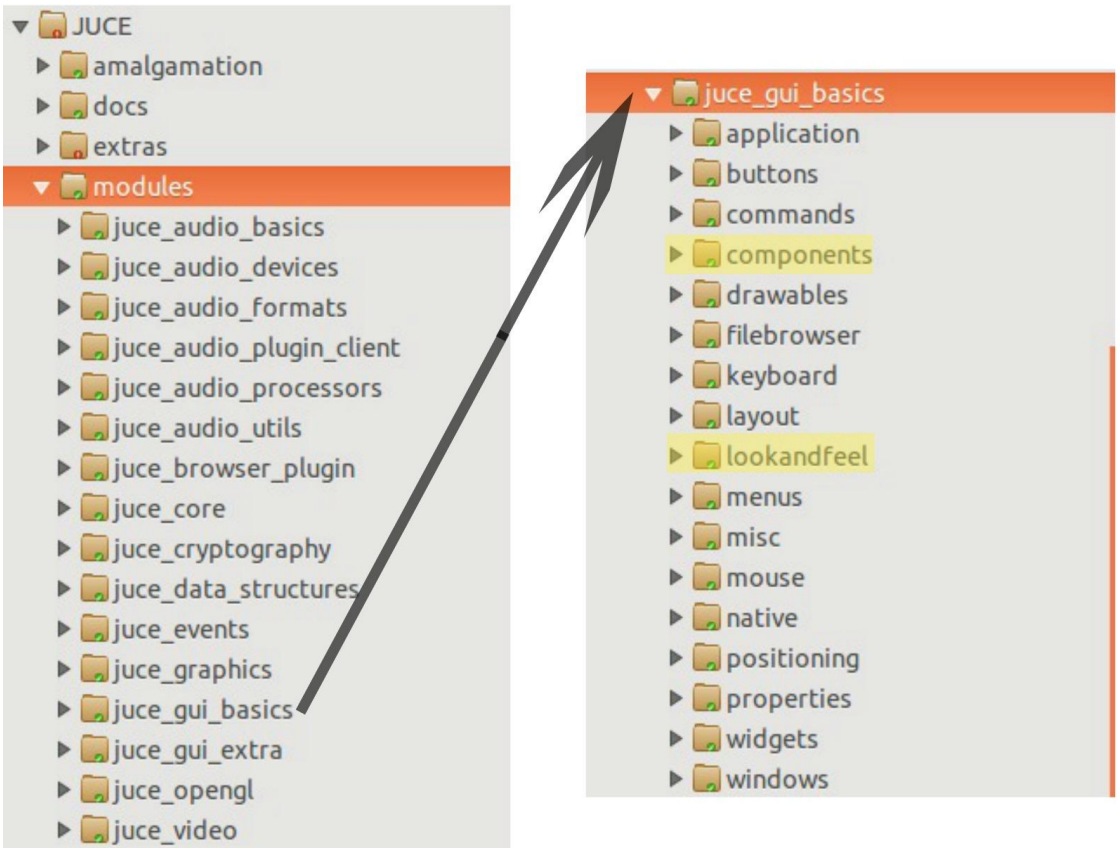


Figura 2.1: Struttura dei moduli. Le cartelle evidenziate contengono la classe Component e la classe LookAndFeel

2.1.2 Editor grafico: The Jucer

The Jucer è un software che permette di aggiungere e modificare oggetti della classe `Component` tramite interfaccia grafica (Figura 2.2). Il programma è indipendente da Introjucer ed è contenuto in [JUCE/extras/the jucer]. Con questo software è possible modificare visivamente l'aspetto e le funzionalità di componenti come slider, pulsanti, etichette e vedere un'anteprima del codice generato. Nello sviluppo di questa tesi The Jucer è stato utilizzato solamente in fase iniziale per prendere confidenza con il codice e i metodi principali della classe `Component`. A partire dalla versione di JUCE 2.1.1 The Jucer è stato integrato in Introjucer, rendendo possibile creare e modificare i componenti di base tramite interfaccia grafica direttamente da quest'ultimo.

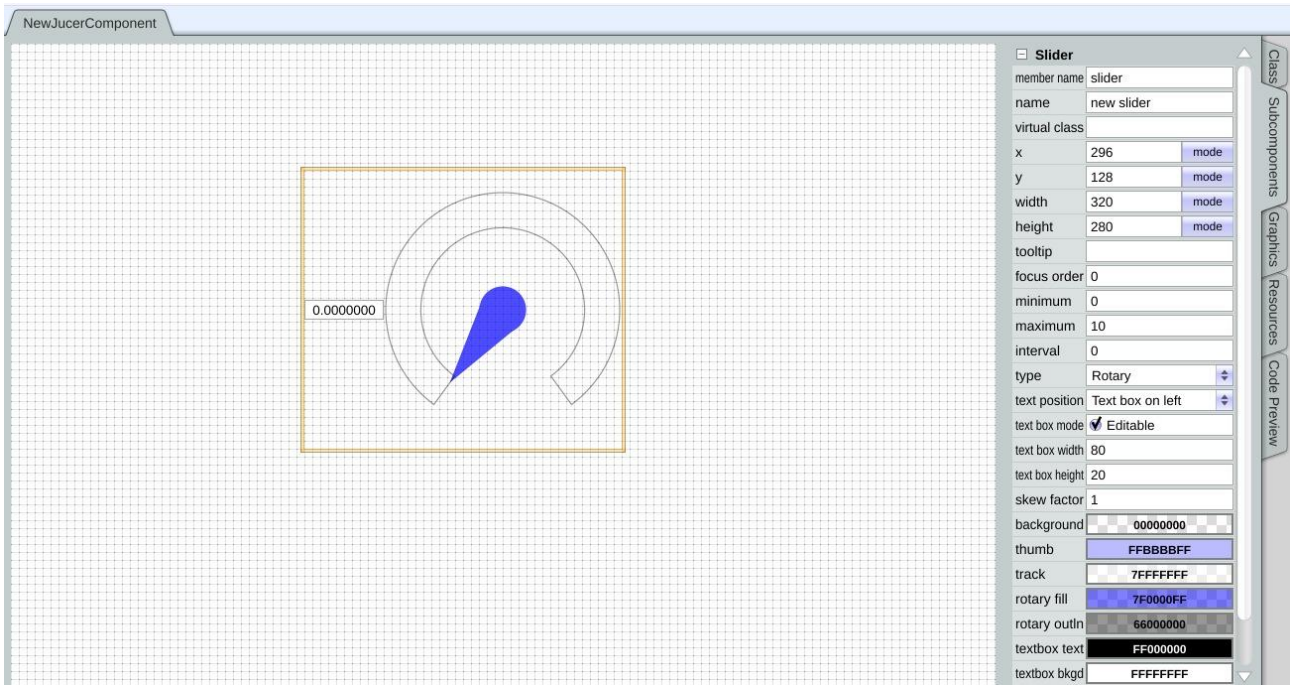


Figura 2.2: Schermata di The Jucer con uno slider e i relativi parametri

2.1.3 Ambiente di progettazione integrato: Introjucer

Introjucer è lo strumento principale per la scrittura del codice C++ e la gestione globale del progetto. Diversamente dai tradizionali ambienti di progettazione integrata (IDE-Integrated development environment) non dispone di un compilatore interno ma permette di esportare un progetto pronto da compilare su ogni piattaforma (utilizzando i compilatori dedicati come ad esempio Xcode per iOS, MS Visual Studio per Windows, Ant ed Eclipse per Android) (Figura 2.3).

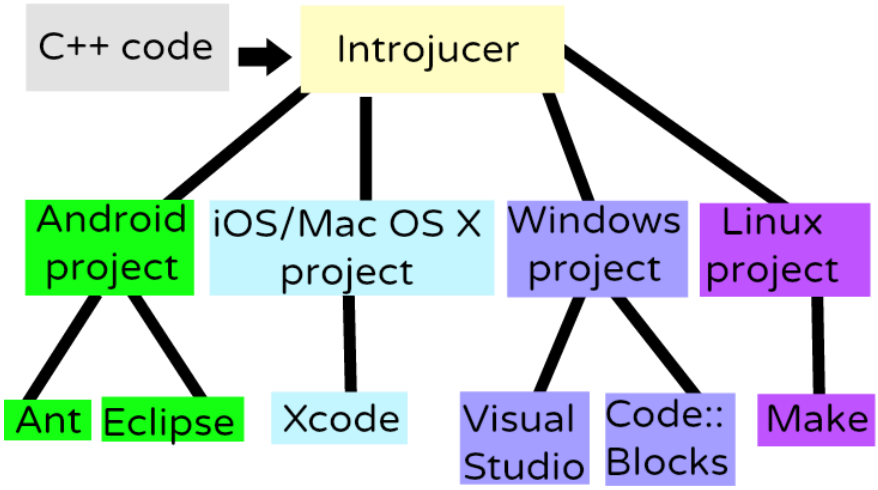


Figura 2.3: Progetti esportabili da Introjucer e relativi compilatori

Il controllo delle impostazioni dei compilatori per ciascuna piattaforma è affidato ad Introjucer, in questo modo la modifica o l'aggiunta di un file avviene una sola volta e l'intero progetto è centralizzato in un unico file.
L'interfaccia di Introjucer si compone di due schede: “Files” e “Config”. Nella prima è presente la lista di tutti i file di progetto e l'editor di testo per il codice (Figura 2.4). Nella seconda è presente la lista dei moduli inclusi nel progetto e l'elenco dei target per la compilazione (Figura 2.5).

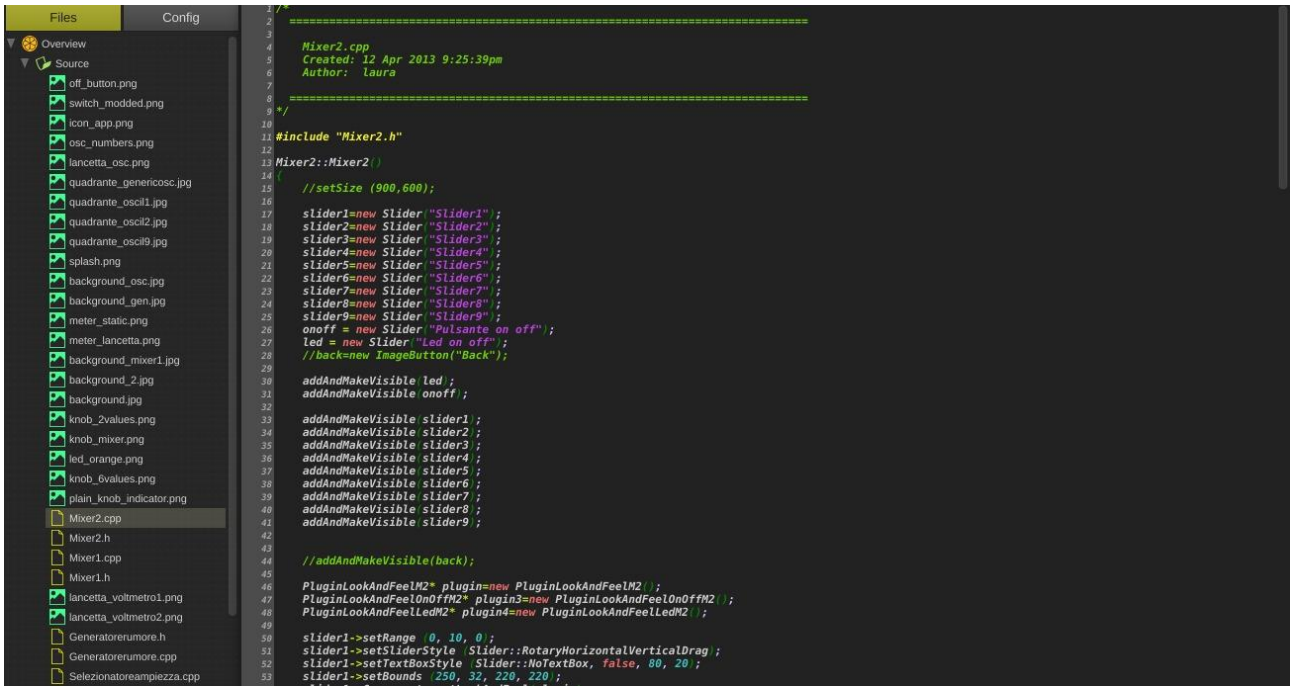


Figura 2.4: Schermata principale di Introjucer: file contenuti nel progetto e codice C++

Una funzione cruciale di Introjucer è la gestione dei moduli JUCE. Dalla scheda “Config” è possibile selezionare i moduli che si desidera utilizzare e, una volta salvato il progetto, Introjucer genera automaticamente tutto il codice richiesto per compilarli e includere i rispettivi file header (Figura 2.5).

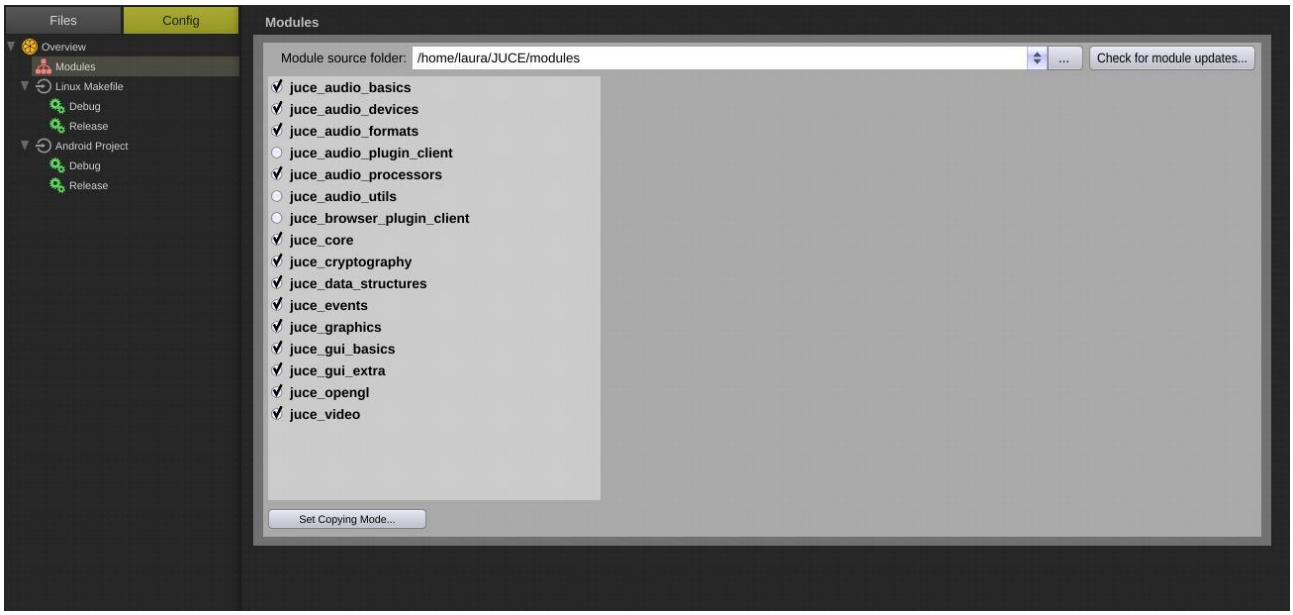


Figura 2.5: Schermata di Introjucer: selezione dei moduli e piattaforme di destinazione

2.1.4 Configurazione iniziale e creazione di un nuovo progetto

Il framework JUCE è distribuito sotto forma di file compresso ZIP. All'interno della cartella principale sono presenti varie sotto cartelle, fra cui la cartella extras (Figura 2.6). Al suo interno si trovano programmi di esempio (nella cartella [extras/example projects]), una demo delle funzionalità di Juce (in [extras/JuceDemo]) e i sorgenti pronti alla compilazione di Introjucer e The Jucer per vari sistemi operativi.

Lo sviluppo di questo lavoro di tesi è avvenuto in ambiente Linux, quindi è innanzitutto necessario compilare (tramite il comando *make* da terminale) i file sorgenti di Introjucer e The Jucer (contenuti in [extras/Introjucer/Builds] e [extras/the jucer/Builds]). Terminata l'operazione, si hanno a disposizione i file eseguibili dei due programmi e si può procedere con la compilazione della demo per verificare che tutte le dipendenze per JUCE siano installate correttamente.

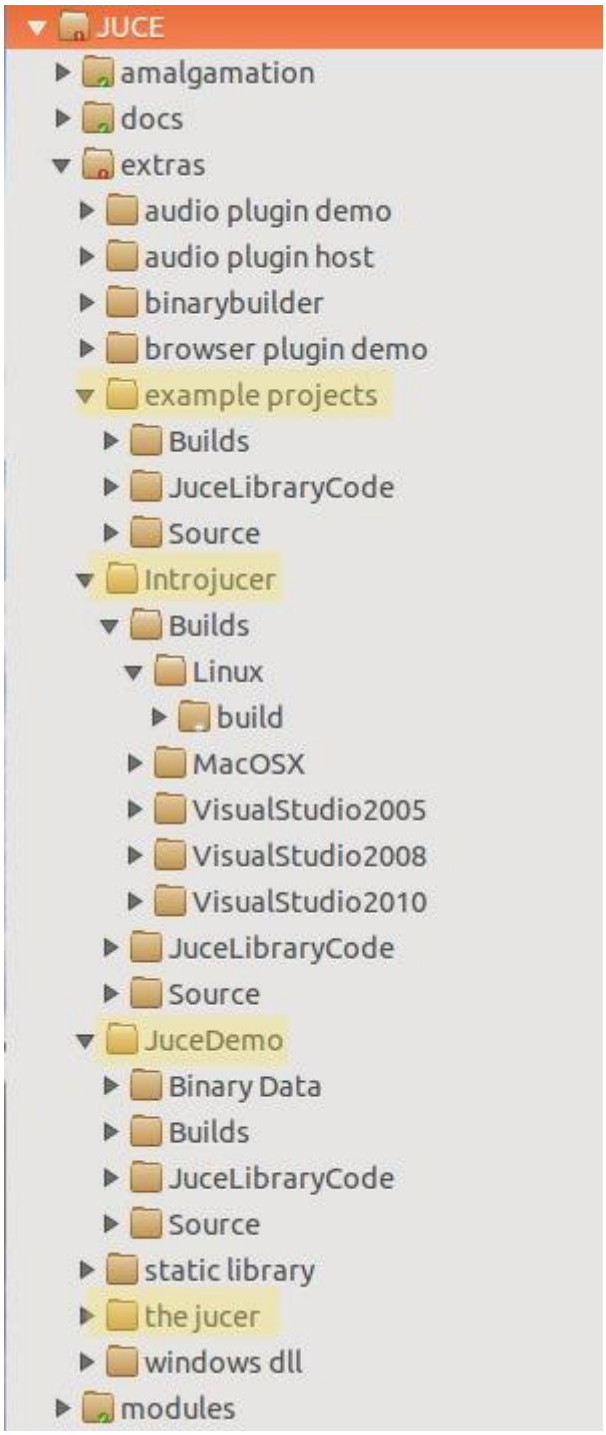


Figura 2.6: Struttura della cartella JUCE

La compilazione e l'esecuzione di un progetto con JUCE in ambiente Linux avviene secondo i seguenti passi:

- Creazione e salvataggio di un nuovo progetto in Introjucer

  Seguendo il percorso *File->New Project* si apre una finestra in cui specificare il nome del progetto, il tipo (GUI Application, Console Application oppure Audio Plug-In), la cartella di destinazione e i file che verranno auto-generati. In questo caso è stato scelto “Create a Main.cpp file and a basic window”. Scegliendo questa

opzione vengono creati tre file:

1. *MainComponent.h*: qui è dichiarata la classe `MainContentComponent`, che eredita i metodi della classe `Component` e rappresenta il “contenitore” in cui poi saranno messi tutti i componenti (pulsanti, manopole, lancette, etc.) dell'interfaccia.
2. *MainComponent.cpp*: vengono specificati il costruttore,il distruttore di `MainContentComponent` e i metodi `paint` e `resized`. In questo esempio è presente una semplice implementazione del metodo paint, in cui viene impostato il colore di sfondo, viene scritta la frase “Hello World!” e ne vengono dichiarati colore e font. Successivamente il codice di esempio di questo metodo verrà sostituito con l'implementazione di questo progetto di tesi.
3. *Main.cpp*: contiene il codice necessario a realizzare una finestra di base per l'applicazione. Sono presenti titolo e pulsanti per massimizzare, minimizzare e chiudere la finestra. La classe `MainWindow` si occupa di disegnare questa finestra e di mettere al suo interno un'istanza della classe `MainContentComponent`.

Nella cartella di destinazione specificata troviamo il file principale di Introjucer (con estensione .jucer), la cartella Builds (che conterrà i progetti da compilare per le varie piattaforme), la cartella JuceLibraryCode (con le librerie JUCE e i moduli necessari), la cartella Source (contenente i file .cpp e i file .h con il codice C++) (Figura 2.7).

Figura 2.7: Struttura della cartella di progetto

In questo lavoro di tesi l'applicazione è stata testata passo-passo su Linux e solo successivamente è stata eseguita su un dispositivo Android. Eseguire un file su Linux infatti risulta più immediato in quanto è sufficiente aprire il file compilato, mentre per eseguire su un terminale Android è necessario utilizzare Eclipse (vedi paragrafo successivo) poiché non è presente in Introjucer un compilatore integrato. Nella scheda “Config” di Introjucer è possibile specificare i target per la compilazione, in questo caso sono stati aggiunti Linux e Android (Figura 2.5). Nella scheda relativa al target Android è di cruciale importanza specificare i percorsi di SDK ed NDK (vedi paragrafo successivo).

Infine,per quanto riguarda i moduli, è stata lasciata la configurazione di default di

Introjucer (nonostante alcuni moduli come quelli necessari all'elaborazione audio e video non siano necessari in questo specifico progetto, saranno utili alla futura realizzazione della sintesi audio e all'aggiunta di materiali multimediali).

- Compilazione tramite terminale

  Per compilare il progetto su Linux è sufficiente usare il comando *make* dal terminale dopo essersi spostati nella cartella [Builds/Linux].

  La compilazione per Android avviene invece dando il comando da terminale *ant debug* o *ant release* nella cartella [Builds/Android] (vedi paragrafo successivo).

- Esecuzione dell'applicazione

  Il file eseguibile per Linux è contenuto in [Builds/Linux/builds]. In Figura 2.8 viene fotografato il progetto visto finora, una semplice finestra con la scritta “Hello World!”.

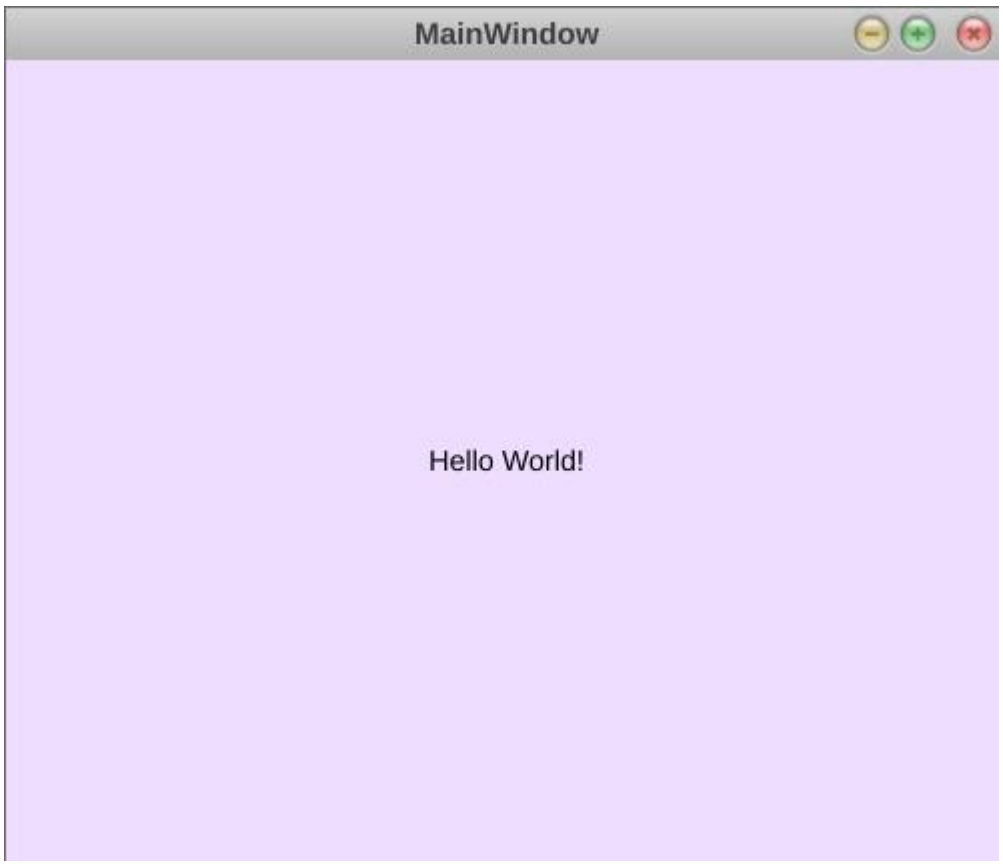


Figura 2.8: Semplice progetto di partenza

2.2 Android OS

Per la compilazione e l'esecuzione di applicazioni per Android sono necessari i seguenti strumenti[3]:

- **Ant**

Apache Ant[4] è un software per l'automazione del processo di build. La sua funzione è simile a quella del comando *make* ma è principalmente orientato allo sviluppo Java. Questo strumento può essere utilizzato per compilare, installare ed eseguire applicazioni su un terminale Android tramite riga di comando. In questo caso è stato utilizzato il comando *ant debug* e successivamente *ant release* per compilare il progetto Android, mentre per l'esecuzione è stato scelto di passare per il software Eclipse. La compilazione da terminale è stata preferita a quella tramite Eclipse perchè segnala più dettagliatamente l'eventuale mancanza di moduli JUCE necessari nel progetto. L'esecuzione tramite Eclipse risulta infine più veloce rispetto al trasferimento e all'installazione manuale del file compilato sul dispositivo Android.

- **Eclipse**

Eclipse[5] è un ambiente di sviluppo integrato (IDE) open-source, ovvero una piattaforma integrata che consente di gestire interamente il processo di sviluppo di applicazioni Java. Il programma è disponibile in varie versioni anche per altri linguaggi di programmazione, in questo caso è stata scelta “Eclipse IDE for C/C++ Developers”.
Il funzionamento del software è incentrato sull'uso di plug-in. Di particolare interesse in questo caso di studio è il plug-in ADT (Android Development Tools), ideato per facilitare la creazione di applicazioni Android, realizzarne l'interfaccia ed eseguirne il debug.

- **Android SDK e NDK**

  Android SDK[6] (Software Development Kit) è un pacchetto che contiene un set completo di API (Application Programming Interface), librerie e strumenti di sviluppo utili a compilare, testare e debuggare applicazioni per Android.
  Android NDK[7] (Native Development Kit) è un set di strumenti che permette di implementare parti di un'applicazione Android utilizzando linguaggi di programmazione nativi come C e C++. In questo caso è necessario utilizzarlo, in quanto la libreria JUCE e il codice che realizzerà l'interfaccia sono scritti in C++.

  Una volta scaricati entrambi i pacchetti (SDK ed NDK sono entrambi distribuiti sotto forma di file ZIP) ed estratti, è di fondamentale importanza impostare correttamente sia in Eclipse che in Introjucer i percorsi in cui vengono scompattati.

Terminata l'installazione di Eclipse e del plug-in ADT e la configurazione dei path per l'SDK

3 Si assume d'ora in avanti che l'ambiente di lavoro sia Linux.
4 Link al sito di Apache ant: http://ant.apache.org/
5 Link al sito di Eclipse: http://www.eclipse.org/
6 Link ad Android SDK: http://developer.android.com/sdk/
7 Link ad Android NDK: http://developer.android.com/tools/sdk/ndk/

e l'NDK, è possibile importare un progetto Android creato con Introjucer ed eseguirlo su un terminale Android connesso al pc (oppure in alternativa su un emulatore eventualmente creato in Eclipse[8]). Nel lavoro di questa tesi Eclipse è stato utilizzato unicamente per eseguire l'applicazione, già precedentemente debuggata da terminale con Ant, e verificarne il funzionamento.

2.3 KnobMan

KnobMan[9] è un programma sviluppato dall'azienda g200kg, produttrice di diversi software audio e plug-in VST (Virtual Studio Technology, ovvero plugin musicali). Il software permette di disegnare graficamente un oggetto, come può essere una manopola, e di indicare il tipo di animazione che si desidera dare a tale oggetto, nel caso della manopola un movimento rotativo. KnobMan esporta poi un'immagine in formato .png sotto forma di striscia con i fotogrammi dell'animazione. È possibile impostare il numero di fotogrammi desiderati e disporli orizzontalmente o verticalmente.

Figura 2.9: Esempio di immagine creata con KnobMan

Il disegno dell'oggetto avviene su più layer (ovvero livelli) di tipo bitmap. Ogni layer può contenere un'immagine primitiva come una linea, un cerchio, una sfera, un'immagine importata, del testo e così via. Per ogni primitiva si possono specificare diversi parametri e aggiungere effetti come zoom, rotazioni, effetti sui colori, maschere e ombreggiature. I layer sovrapposti formano l'immagine finale (Figura 2.10).

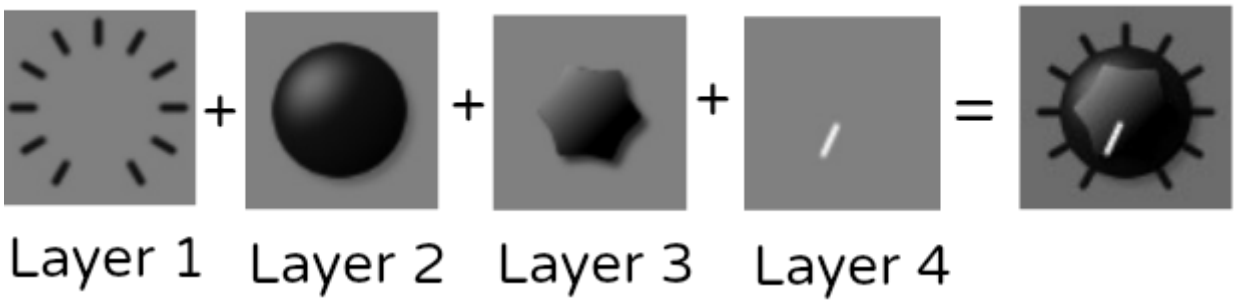


Figura 2.10: Layer che formano l'immagine di una manopola

8 In Eclipse, se non è connesso alcun terminale al PC, al momento dell'esecuzione del progetto (dal menu Run → Run) viene richiesto se si desidera configurare una nuova macchina virtuale Android. È sufficiente poi selezionare Launch a new Android Virtual Device → Manager → New e configurare il nuovo dispositivo virtuale.

9 Link al software Knobman: http://www.g200kg.com/en/software/knobman.html

E' poi possibile specificare il tipo di animazione dell'oggetto (rotazione, traslazione, etc.).[10]

In Figura 2.11 è mostrata la schermata principale di KnobMan. La prima colonna mostra i layer dell'immagine. Nella seconda colonna è specificato il tipo di immagine primitiva contenuta nel layer corrente (linea, cerchio, sfera, rettangolo, etc.) e i suoi parametri di base (dimensioni, colore, etc.). La terza colonna mostra ulteriori effetti che si possono applicare all'immagine primitiva corrente (ombreggiature, rotazioni, etc.). Nella scheda “Rotate at” al suo interno è possibile specificare gli angoli iniziale e finale nel caso in cui l'animazione dell'oggetto sia una rotazione. L'ultima colonna mostra un'anteprima dell'immagine finale, con i fotogrammi dell'oggetto. Oltre a questa anteprima “statica” è presente anche una finestra “dinamica”, in cui è possibile interagire direttamente con l'oggetto e verificarne l'animazione utilizzando il mouse.
Immagini con fotogrammi così disposti sono utili allo sviluppo dell'interfaccia grafica dei plug-in VST e il loro utilizzo in questo lavoro verrà successivamente spiegato (al paragrafo 3.2.3).

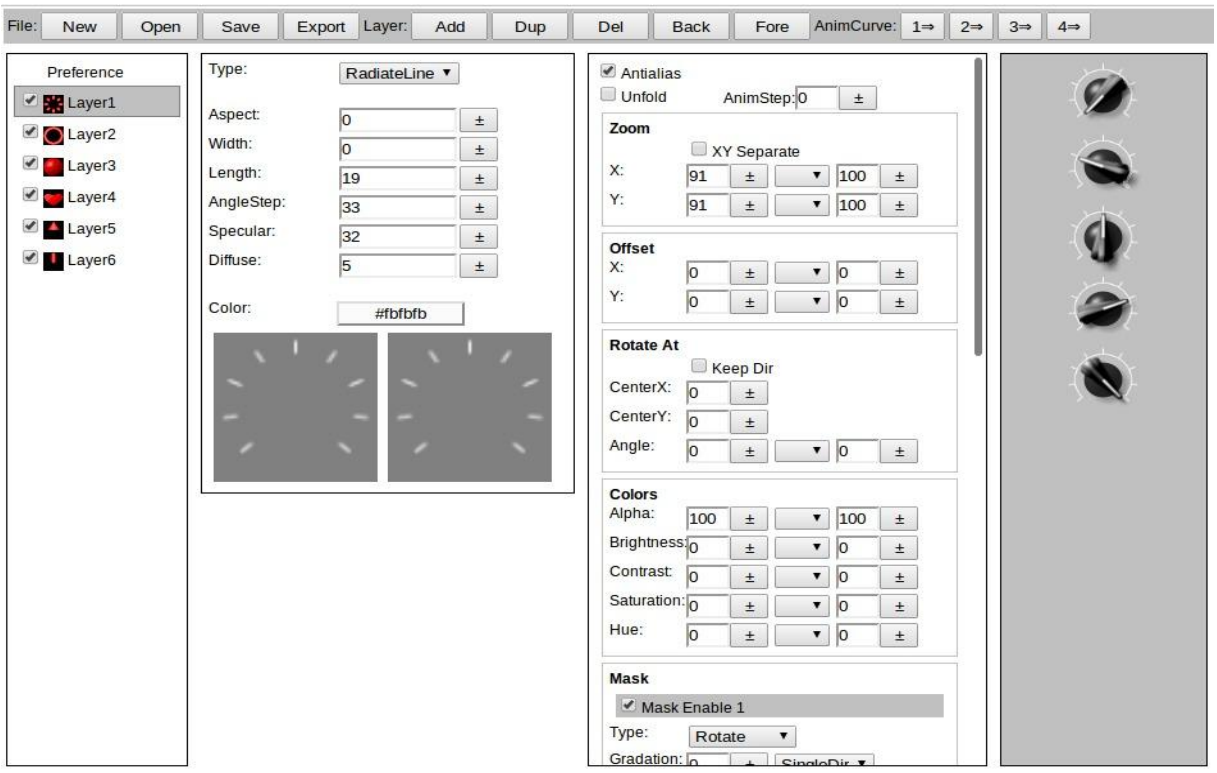


Figura 2.11: Schermata del programma WebKnobMan

Nella scheda “Preference” della prima colonna è possibile specificare i parametri per l'esportazione dell'immagine finale: le dimensioni della striscia, il numero di fotogrammi visualizzati nell'anteprima, il numero effettivo di fotogrammi esportati, la disposizione verticale o orizzontale dei frame e infine il colore di sfondo (Figura 2.12).

10 Un manuale dettagliato sull'uso e di KnobMan è disponibile al seguente indirizzo:http://www.g200kg.com/en/docs/knobman/

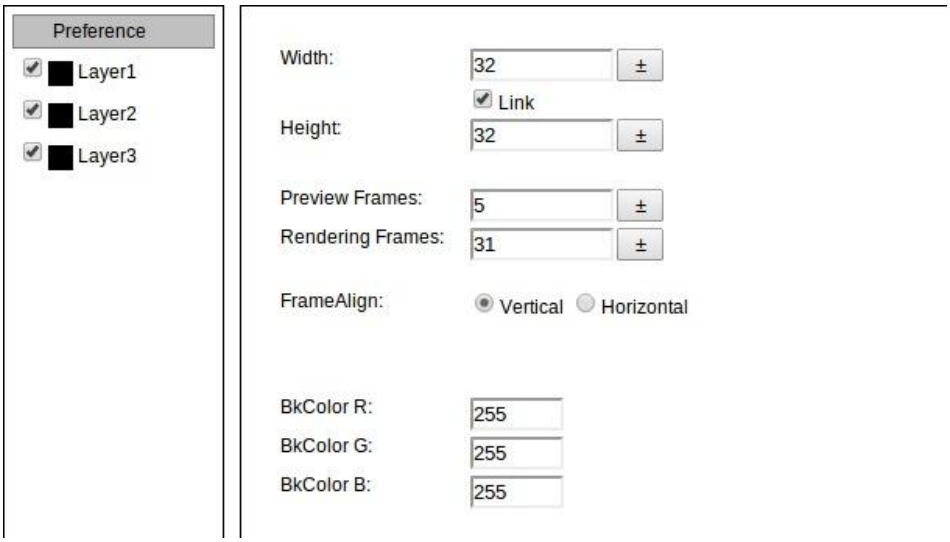


Figura 2.12: Finestra di KnobMan per l'esportazione dell'immagine finale

È disponibile sul sito del programma una galleria di progetti caricati dagli utenti con le relative licenze di utilizzo. È stata sviluppata inoltre una versione con funzioni limitate del programma, WebKnobMan, utilizzabile direttamente da browser.

Capitolo 3

Progetto

In questo capitolo verrà descritta nel dettaglio l'implementazione del codice per l'interfaccia grafica dell'applicazione Android. L'esposizione inizia con la scelta dei singoli componenti di JUCE adatti allo scopo, per poi passare alla costruzione del layout dei vari strumenti e poi alla personalizzazione di tutti gli aspetti grafici. Infine verrà descritta la costruzione del progetto Android in Introjucer, la compilazione e infine l'esecuzione dell'applicazione sul dispositivo Android.

3.1 Obiettivi

Il lavoro svolto si è posto come obiettivo la realizzazione dell'interfaccia grafica di un'applicazione Android che simuli una selezione degli gli strumenti musicali dello Studio di Fonologia. Gli strumenti presi in considerazione sono:

- I 9 oscillatori
- Il selezionatore d'ampiezza
- Il generatore di rumore bianco
- Il mixer degli oscillatori
- Il mixer a 8 ingressi

La scelta degli strumenti è stata fatta in funzione del futuro sviluppo della parte audio che completerà l'applicazione, rendendo l'applicazione un VST utilizzabile in qualsiasi produzione musicale. Prendendo come riferimento le foto dello Studio di Fonologia è stato possibile ricreare l'aspetto degli strumenti originali,per poi curare l'usabilità dell'applicazione e per fornire all'utente un'esperienza più fluida possibile. Vista la natura dei controlli degli strumenti, costituiti per lo più da manopole e interruttori, è stato scelto di sviluppare l'applicazione per tablet. Uno schermo grande permette infatti di interagire con i controlli più facilmente. Di seguito verrà spiegato com'è stato possibile ricreare i controlli e la grafica degli strumenti utilizzando i componenti della libreria JUCE, precedentemente descritta nel Capitolo 2.

3.2 Implementazione

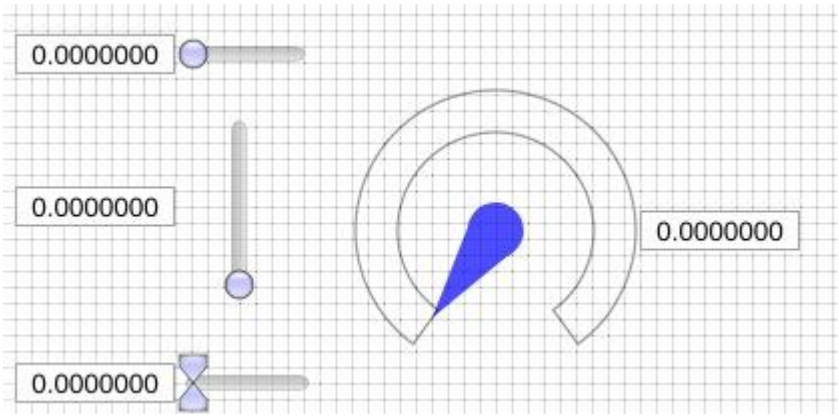
3.2.1 Realizzazione dei componenti di base

I componenti principali che compongono gli strumenti oggetto di studio sono:

- *Controlli rotativi*: si tratta delle manopole che controllano i valori di frequenza, soppressione, volume, etc. Sono presenti sia controlli a rotazione continua sia controlli con rotazione a scatti.

- *Lancette*: i valori sulle scale degli oscillatori, del selezionatore di ampiezza e del generatore di rumore sono indicati da lancette.
- *Interruttori*: su ogni oscillatore,sul selezionatore d'ampiezza e sul mixer a 9 ingressi è presente un interruttore per l'accensione e lo spegnimento dello strumento.
- *Led*: indica quando lo strumento è acceso.

La libreria JUCE offre dei componenti di base con funzionalità simili a quelle dei controlli desiderati. È stato utilizzato The Jucer per testare velocemente i componenti standard di JUCE, individuando infine quelli più adatti allo scopo.
JUCE mette a disposizione una serie di slider, ovvero controlli a trascinamento, con diverse funzioni e aspetto. Sono presenti numerosi parametri per modificarne dimensioni e posizione, il tipo di controllo (lineare orizzontale, lineare verticale, rotativo con trascinamento circolare/orizzontale/verticale/orizzontale e verticale, con due o tre indicatori e altri ancora), il fattore di scala e il colore dello slider. È possibile visualizzare anche un box di testo con il valore indicato dallo slider (Figura 3.1).



Nel nostro caso lo slider più interessante è quello rotativo. Questo componente si comporta esattamente come una manopola ed è stato scelto per implementare tutti i controlli rotativi. È possibile impostare come parametri il minimo e il massimo valore rappresentato e l'intervallo fra un valore e l'altro, rendendo lo slider rotativo adatto a realizzare sia la manopola a rotazione continua sia quella a scatti. Un esempio di manopola a scatti è quella per la selezione di una delle 6 bande di frequenza nell'oscillatore; un esempio di manopola continua è quella per il controllo fine della frequenza (v. Paragrafo 1.2).

Lo slider si presta anche all'implementazione della lancetta. L'aspetto di interesse in questo caso è la struttura stessa del componente, costituita da un oggetto che in base alla posizione assunta rappresenta un valore diverso. Per implementare la lancetta è stato scelto, come verrà più avanti specificato, uno slider rotativo.

Anche per implementare l'interruttore di accensione degli strumenti si è scelto uno slider. Per simulare il movimento della mano nell'accensione dell'interruttore è stato scelto uno slider a trascinamento verticale, impostando solo due valori (che rappresentano gli stati acceso e spento).

Il led di stato è stato implementato con uno slider rotativo a due valori. Anche in questo caso il tipo di slider è ininfluente, è sufficiente disporre di un componente con due diversi stati. L'implementazione con un bottone (classe `Button`) è stata scartata perchè risulta più scomoda; il bottone infatti assume quattro diversi stati a seconda che sia attivo (ovvero sia stato cliccato dal mouse), disattivo (ovvero non cliccato dal mouse), che il mouse sia sopra l'oggetto oppure al suo esterno. A ciascuno stato corrisponde un aspetto grafico diverso. Per modificare quindi l'aspetto del bottone è necessario sovrascrivere più metodi rispetto al caso dello slider.
I componenti realizzabili con le classi di JUCE, come slider e bottoni, sono detti “widget”.

3.2.2 Layout di base dei singoli strumenti

Dopo aver individuato i componenti ideali a rappresentare le varie parti degli strumenti si può procedere abbozzando il layout, ovvero la collocazione dei controlli all'interno della finestra.
In fase iniziale gli strumenti sono tenuti separati creando un progetto per ognuno.
Per prendere confidenza con il codice dei componenti è stata sfruttata l'anteprima del listato fornita da The Jucer e sono state analizzate le caratteristiche e i metodi delle classi relative [4].

Una volta creato un nuovo progetto in Introjucer dichiariamo fra le variabili private della classe `MainContentComponent` (la classe che rappresenta il “contenitore” per i vari componenti) le nuove variabili che rappresentano i componenti che vogliamo aggiungere. Per esempio, nel caso dell'oscillatore dovremo aggiungere due manopole (una continua ed una a scatti), una lancetta, un interruttore e il relativo led di accensione (v. Paragrafo 1.2):

```
//nel file MainComponent.h
class MainContentComponent   : public Component
{
public:

      MainContentComponent();
      ~MainContentComponent();

      void resized();
      void paint(Graphics& g);

private:

Slider* lancetta;
Slider* manopolascatti;
Slider* manopola;
Slider* onoff;
Slider* led;

};
```

La stringa *public Component* indica che `MainContentComponent` eredita i metodi della

classe `Component`, che si possono quindi utilizzare.

Nel costruttore di `MainContentComponent` si possono ora aggiungere i componenti, creando una nuova istanza di ciascuna classe (per esempio un'istanza della classe `Slider`) con i parametri opportuni e assegnandola alle variabili appena create. Dopodichè è necessario invocare il metodo `Component::addAndMakeVisible(Component* child)` su ciascuno dei componenti. Il componente passato come parametro viene così dichiarato componente figlio di `MainContentComponent` e viene reso visibile nella finestra. Nel caso dell'oscillatore, si otterrà il seguente corpo del costruttore di `MainContentComponent`:

```
//nel file MainComponent.cpp
MainContentComponent::MainContentComponent() //costruttore
{

    lancetta = new Slider ("Lancetta");
    manopolascatti = new Slider ("Manopola a scatti");
    manopola = new Slider("Manopola");
    onoff = new Slider("Pulsante on off");
    led = new Slider("Led on off");

    addAndMakeVisible(lancetta);
    addAndMakeVisible(manopolascatti);
    addAndMakeVisible(manopola);
    addAndMakeVisible(onoff);
    addAndMakeVisible(led);

}
```

I componenti vanno poi collocati nello spazio della finestra ed eventualmente personalizzati settando i vari parametri. I metodi principali di base utilizzati sono:

- `Component::setBounds (int x, int y, int height, int weight)`

Specifica la posizione e le dimensioni del componente.

- `Slider::setSliderStyle (SliderStyle newStyle)`

Specifica il tipo di interfaccia dello slider. Il parametro newStyle può essere ad esempio `LinearVertical` per lo slider lineare a trascinamento verticale oppure `RotaryHorizontalDrag` per lo slider rotativo a trascinamento orizzontale.

- `Slider::setRange (double newMinimum, double newMaximum, double newInterval = 0)`
  Imposta il minimo e massimo valore dello slider e l'intervallo fra un valore e l'altro.

- `Slider::setTextBoxStyle (TextEntryBoxPosition newPosition, bool isReadOnly, int textEntryBoxWidth, int textEntryBoxHeight)`
  Specifica la posizione del box di testo con il valore dello slider (in particolare newPosition può essere `NoTextBox` per nasconderlo), indica se il valore del box può essere inserito manualmente dall'utente, imposta altezza e larghezza del box.

- `Slider::setRotaryParameters (float startAngleRadians, float endAngleRadians, bool stopAtEnd)`

Specifica i parametri dello slider rotativo. Di particolare interesse è il parametro stopAtEnd, che indica se lo slider deve fermarsi o tornare al valore iniziale alla fine della rotazione. Se questo parametro è posto uguale a 'false' lo slider non si ferma e ruota continuamente.

- `Component::setEnabled (bool shouldBeEnabled)`

Indica se il componente dev'essere abilitato o meno. Settando il parametro su 'false' il componente diventa insensibile ai movimenti del mouse su di esso. Questo metodo è indicato per disabilitare il led e la lancetta, in quanto non devono essere controllabili direttamente dall'utente. Momentaneamente la lancetta è abilitata, per testarne il funzionamento.

Un esempio di utilizzo di questi metodi è il seguente:

```
    MainContentComponent::MainContentComponent() //costruttore
{
      ...

    manopolascatti->setRange (0, 6, 1); //manopola che ruota da 1 a 6 a scatti
    manopolascatti->setSliderStyle (Slider::RotaryHorizontalVerticalDrag);
    manopolascatti->setTextBoxStyle (Slider::NoTextBox, false, 80, 20);
    manopolascatti->setBounds (120, 400, 200, 200);

    manopola->setRange(0,100,0);//manopola che ruota da 0 a 100%
    manopola->setRotaryParameters(0,6.28,false); //rotazione continua
    manopola->setSliderStyle (Slider::RotaryHorizontalVerticalDrag);
    manopola->setTextBoxStyle (Slider::NoTextBox,false,80,20);
    manopola->setBounds (750, 420, 160, 160);

    led->setSliderStyle(Slider::Rotary);//slider rotativo a due valori
    led->setRange(0,1,1);
    led->setBounds(1135,85,50,50);
    led->setTextBoxStyle(Slider::NoTextBox,false,10,10);
    led->Component::setEnabled(false);


      ...
}
```

Il layout di base dell'oscillatore, una volta aggiunti e disposti i componenti con i metodi fin'ora descritti, è mostrato in Figura 3.2.

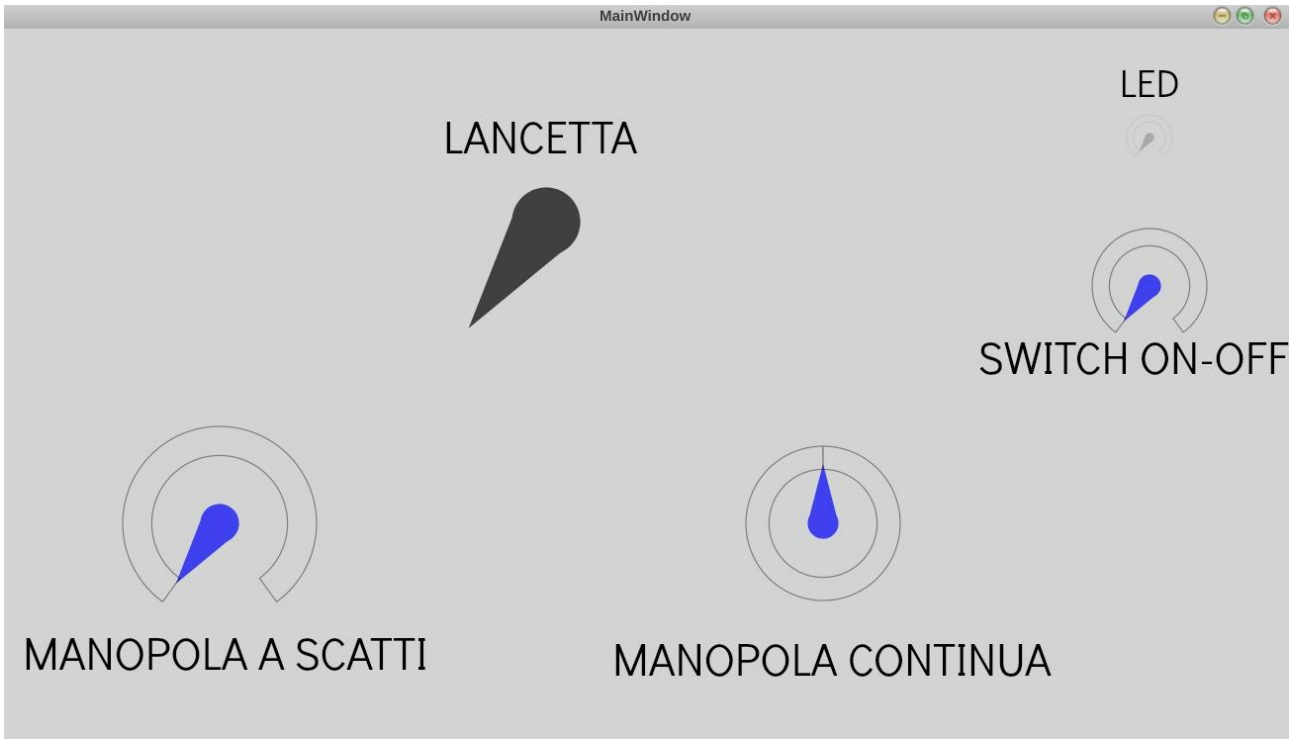


Figura 3.2: Layout di base dell'oscillatore

Finora è stato modificato il costruttore di `MainContentComponent`. Per quanto riguarda il distruttore è sufficiente invocare il metodo `Component::deleteAllChildren()`, che si occupa autonomamente di eliminare tutti i componenti figli quando `MainContentComponent` viene rimosso.

Una volta terminata la disposizione dei vari controlli se ne possono definire il comportamento e l'interazione con gli altri componenti. L'obiettivo ora è fare in modo che all'accensione dell'interruttore corrisponda l'accensione del led. Per accensione si intende qui il passaggio dal valore 0 al valore 1. Successivamente tali valori avranno una corrispondenza con le immagini del led/interruttore spento/acceso. Verrà spiegata l'implementazione solo per l'interruttore e il led in quanto il valore degli altri slider (controlli rotativi e lancette) è direttamente collegato ai valori risultanti dalla sintesi audio, che non è oggetto di questa tesi. Il procedimento per fare in modo che le manopole cambino il loro valore in corrispondenza di altri eventi è del tutto analogo a quello che verrà descritto per il led, in quanto tutti i componenti sono stati implementati come slider.
La classe `Slider`, assieme a molte altre, invia dei messaggi in corrispondenza di un evento, come la variazione del suo valore [5]. Per intercettare questi messaggi e utilizzarli per svolgere la corrispondente operazione è necessario rendere la classe contenitore `MainContentComponent` un “Listener”, ovvero ereditare i metodi della classe `Slider::Listener` che permettono di “ascoltare” i messaggi lanciati dagli slider e riutilizzarli.

Per registrare “onoff come “Listener” si utilizza il metodo `Slider::addListener(Listener* listener)`:

```
//nel file MainComponent.cpp
 class MainContentComponent::MainContentComponent() //costruttore
{     …
      onoff->Slider::addListener(this);
      ...
}
```

Il nostro obiettivo è fare in modo che quando lo slider “onoff” assume il valore 0 (corrispondente allo stato “spento”) lo slider “led” assuma il valore 0 (corrispondente al led spento) e quando “onoff” assume il valore 1 anche “led” assuma il valore 1. Per fare ciò basta sovrascrivere il metodo virtuale `Slider::Listener::sliderValueChanged (Slider* slider)` e inserire un semplice ciclo if-else:

```
//nel file MainComponent.h
class MainContentComponent   : public Component, public Slider::Listener
{     …
void sliderValueChanged (Slider* slider)
{

    if (slider==onoff)
    {double value2 = onoff->getValue();
        if (value2 ==0)
        {led->setValue(value2,dontSendNotification);}
        else if (value2==1)
        {led->setValue(value2,dontSendNotification);}
    }
}

…
}
```

3.2.3 Personalizzazione dei componenti

JUCE dispone di una classe ideata appositamente per personalizzare l'aspetto di tutti i “widget” (ovvero degli oggetti che si possono implementare con le classi della libreria): la classe LookAndFeel. Le sue sottoclassi possono essere usate per applicare diverse “skin” all'applicazione. L'obiettivo ora è quello di modificare l'aspetto degli slider e renderli simili alle manopole originali.
Le immagini di riferimento sono state create con il software KnobMan (v. Paragrafo 2.3) .
Di seguito verrà spiegato il procedimento adottato per modificare l'aspetto di una particolare manopola, il metodo per modificare gli altri componenti è del tutto analogo.

1. Creazione del plug-in

   Iniziamo con la creazione del plug-in `PluginLookAndFeelOsc`, ovvero di una classe che eredita i metodi di `LookAndFeel` e che definisce l'aspetto della manopola continua dell'oscillatore (identificata dalla variabile “manopola”) (Figura 3.3).

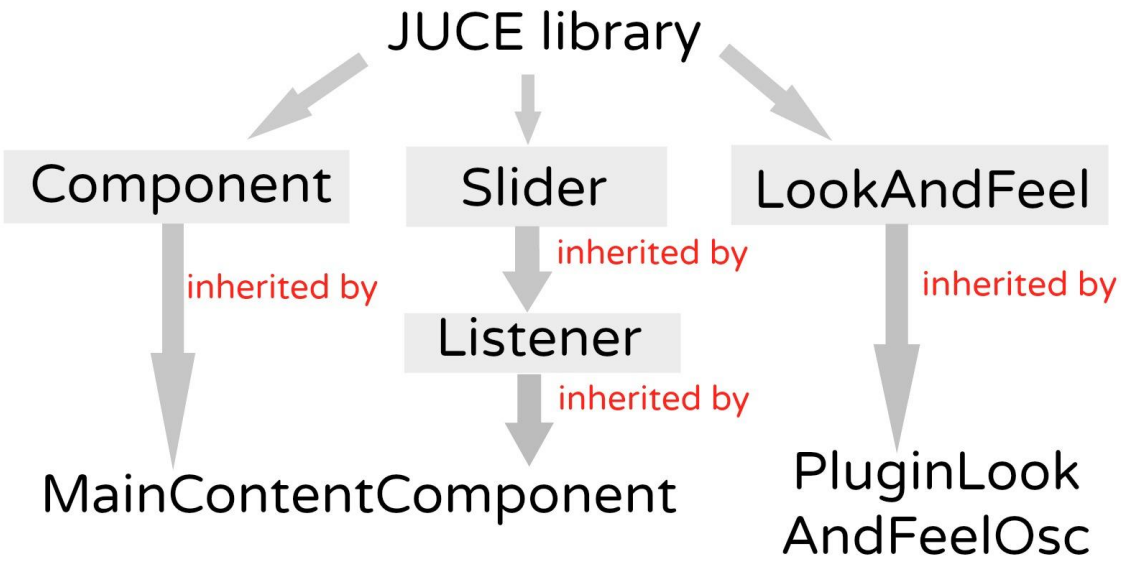


Figura 3.3: Struttura delle classi fin'ora utilizzate

L'approccio utilizzato prevede di caricare un'immagine con i fotogrammi della manopola (preparata con KnobMan) e poi sovrascrivere il metodo `drawRotarySlider` della classe `LookAndFeel`, responsabile di disegnare lo slider.

Il metodo più efficiente per caricare un'immagine nel progetto è trascinarla con il mouse all'interno della finestra Introjucer e salvare. In questo modo l'immagine viene automaticamente convertita in un file binario (detto “BinaryData”) e aggiunta fra i file di progetto. È possibile così utilizzare l'immagine importandola con il metodo `getFromMemory` della classe `ImageCache`. Con altri metodi è possibile importare l'immagine indicando manualmente il percorso in cui è salvata. Il primo metodo è preferibile perché velocizza il disegno delle immagini, requisito indispensabile se si usa, come in questo caso, un'immagine composta da molti fotogrammi per animare l'oggetto. Di seguito è mostrato un esempio del tipo di immagine utilizzata:

Figura 3.4: Porzione dell'immagine per la manopola continua con i primi 5 fotogrammi

Un esempio del plug-in è il seguente[11]:

```
//nel file MainComponent.h
class PluginLookAndFeelOsc : public LookAndFeel // LOOK MANOPOLA CONTINUA

{
    public:
    PluginLookAndFeelOsc() //costruttore

    {}

        void drawRotarySlider (Graphics &  g, int x, int y, int width,
                               int height,
                               float sliderPosProportional,
                               float  rotaryStartAngle,
                               float  rotaryEndAngle,
                               Slider &  slider ) //metodo per il disegno dello slider

    {
    Image myStrip =
ImageCache::getFromMemory(BinaryData::plain_knob_indicator_png,BinaryData::plain_knob_
indicator_pngSize);

      const double fractRotation =(slider.getValue()-
slider.getMinimum())/(slider.getMaximum() - slider.getMinimum());//value between 0 and
1 for current amount of rotation

      const int nFrames = myStrip.getHeight()/myStrip.getWidth(); // number of frames
for vertical film strip

      const int frameIdx = (int)ceil(fractRotation * ((double)nFrames-1.0) ); //
current index from 0 --> nFrames-1

      const float radius = jmin (width / 2.0f, height / 2.0f) ;
      const float centreX = x + width * 0.5f;
      const float centreY = y + height * 0.5f;
      const float rx = centreX - radius - 1.0f;
      const float ry = centreY - radius - 1.0f;

      g.drawImage(myStrip, (int)rx, (int)ry, 200, 200, 0, frameIdx*myStrip.getWidth(),
      myStrip.getWidth(), myStrip.getWidth());
    }

~PluginLookAndFeelOsc() //distruttore

    {}

    private:
    Image myStrip;
};
```

11 Per produrre il codice seguente è stato preso spunto dall'esempio trovato all'indirizzo https://github.com/audioplastic/Juce-look-and-feel-examples.

Il metodo che indica l'immagine sorgente su cui basare il disegno dello slider è:

- `Graphics::drawImage (const Image& imageToDraw, int destX, int destY, int destWidth, int destHeight, int sourceX, int sourceY, int sourceWidth, int sourceHeight)`

Il primo parametro indica l'immagine sorgente da utilizzare. I sucessivi quattro parametri indicano rispettivamente le coordinate X e Y, la larghezza e l'altezza del rettangolo di destinazione che conterrà l'immagine. I successivi quattro parametri indicano le coordinate, la larghezza e l'altezza del rettangolo di immagine da ritagliare dall'immagine sorgente.
La porzione ritagliata dall'immagine sorgente (nel nostro caso un fotogramma) viene utilizzata per disegnare lo slider (l'immagine avrà le dimensioni e la posizione del rettangolo di destinazione)[12]. Nell'esempio l'indice “frameIdx” serve a spostarsi di volta in volta al fotogramma successivo da ritagliare (Figura 3.5). Per calcolarlo è stata prima dichiarata la variabile “fractRotation”, che indica con un valore da 0 a 1 di quanto è ruotato lo slider e si calcola sottraendo al valore corrente il minimo valore e dividendo per il range di valori. “frameIdx” si ottiene poi moltiplicando “fractRotation” per il numero di frame dell'immagine sorgente meno uno (in quanto il primo fotogramma viene considerato con indice 0).

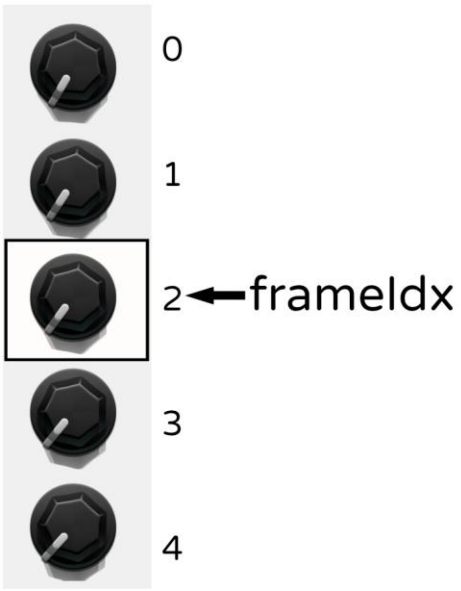


Figura 3.5: Funzionamento dell'indice "frameIdx" su una porzione di striscia

2. Utilizzo del plug-in

Per usare il plug-in appena creato e modificare l'aspetto della manopola si usa il metodo `Component::setLookAndFeel (LookAndFeel* newLookAndFeel)`, dopo aver

12 Nel codice di esempio vengono utilizzati diversi indici per calcolare il numero di fotogrammi nell'immagine sorgente e per posizionare e dimensionare l'immagine di destinazione. Nei plug-in di altri componenti il codice è stato semplificato, indicando manualmente il numero di fotogrammi e le dimensioni.

creato un'istanza dello stesso:

```
//nel file MainComponent.cpp
class MainContentComponent::MainContentComponent() //costruttore
{     …
      PluginLookAndFeelOsc* plugin=new PluginLookAndFeelOsc();
      …
      manopola->Component::setLookAndFeel(plugin);
      …
}
```

Il risultato dell'applicazione del plug-in alla manopola continua dell'oscillatore è il seguente:

Figura 3.6: Layout dell'oscillatore con manopola modificata

In modo del tutto analogo al precedente sono stati realizzati i plug-in per gli slider di tutti gli strumenti, creando ad hoc con KnobMan le immagini con i relativi fotogrammi (Figura 3.7).

Figura 3.7: fotogrammi utilizzati per gli altri componenti

Il metodo fin qui esposto è indicato per componenti di dimensioni limitate. Per quanto riguarda le lancette si è scelta un'implementazione diversa a seconda degli strumenti:

- **Oscillatori**

  Il display degli oscillatori è composto da sei diverse scale, ciascuna relativa alla banda di frequenza corrispondente. Avendo a disposizione le copie delle scale originali, è stata realizzata un'immagine abbastanza grande del quadrante ed è stata importata nel progetto (Figura 3.8). Il quadrante è stato poi disegnato all'interno della finestra come immagine a sé stante, sovrascrivendo il metodo `Component::paint (Graphics& g)`:

```
//nel file MainComponent.cpp
void MainContentComponent::paint (Graphics& g)
{ …
Image
quad1=ImageCache::getFromMemory(BinaryData::quadrante_oscil1_jpg,BinaryData::quadran
te_oscil1_jpgSize);
g.drawImage(quad1,250,20,612,359,0,0,1001,576);
 …
}
```

E' stato scelto di utilizzare il metodo `drawImage` perchè permette di spostare e ridimensionare l'immagine agevolmente attraverso i suoi parametri. È stato così possibile regolare in modo preciso la posizione del quadrante rispetto alla lancetta.

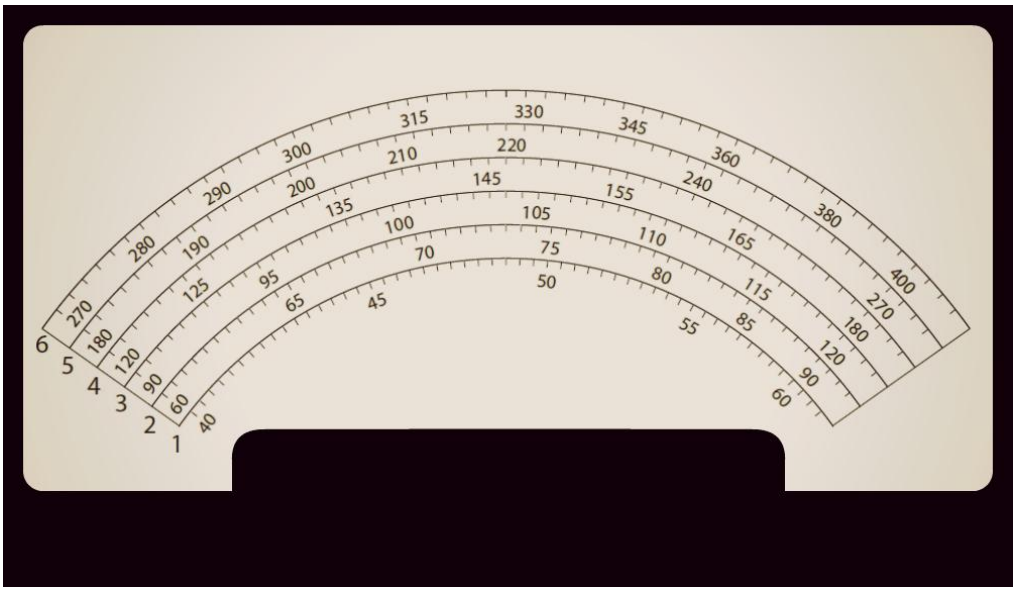


Figura 3.8: Quadrante di uno degli oscillatori

La lancetta è stata implementata secondo l'approccio precedentemente descritto e poi posizionata opportunamente sopra la scala (Figura 3.9).

Figura 3.9: Alcuni fotogrammi della lancetta per l'oscillatore

- **Selezionatore d'ampiezza**
  Anche in questo caso si è scelto di implementare solamente la lancetta. Il posizionamento della lancetta sulla scala del selezionatore d'ampiezza risulta più semplice rispetto al caso dell'oscillatore, in quanto il quadrante risulta più piccolo. Pertanto il display con la scala del selezionatore d'ampiezza è stato integrato direttamente nell'immagine usata poi come sfondo della finestra (v. Paragrafo 3.2.4).

- **Generatore di rumore**
  I due display del generatore di rumore sono stati invece implementati allo stesso modo delle manopole, essendo di dimensioni più ridotte e con scale di più facile lettura (Figura 3.10).

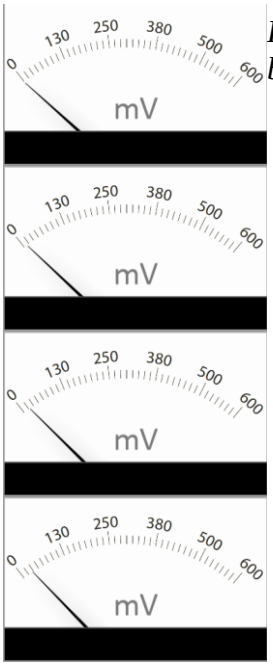


Figura 3.10: Alcuni fotogrammi del voltmetro del generatore di rumore bianco

3.2.4 Grafiche di sfondo

Per rendere gli strumenti virtuali più simili possibile a quelli originali sono stati aggiunti degli elementi decorativi sullo sfondo.
Le grafiche sono state create con il software Gimp[13]. Per impostarle come sfondo della finestra è sufficiente importarle e disegnarle nel metodo `paint`:

```
//nel file MainComponent.cpp
void MainContentComponent::paint (Graphics& g)
{…
Image background =
ImageCache::getFromMemory(BinaryData::background_osc_jpg,BinaryData::background_osc_
jpgSize);


   g.drawImage (background,
                0, 0, getWidth(), getHeight(),
                0, 0, background.getWidth(), background.getHeight());


 …
}
```

A titolo di esempio, fra gli elementi aggiunti ci sono una griglia per il generatore di rumore, scritte, ingressi audio, viti decorative agli angoli e le maniglie del mixer a 8 ingressi (Figura 3.11).

Figura 3.11: Grafica di sfondo del generatore di rumore bianco

13 Link al software Gimp: http://www.gimp.org/

3.2.5 Creazione del progetto DREAMVst

Dopo aver completato il singolo progetto per ogni dispositivo possiamo procedere con la creazione del progetto Introjucer che ospiterà tutti gli strumenti. Nella cartella Source del nuovo progetto saranno presenti i seguenti file (Figura 3.12):

- I file sorgenti degli strumenti, ovvero i file .cpp e .h di oscillatore, selezionatore d'ampiezza, generatore di rumore bianco, mixer 1 (a 8 ingressi) e mixer 2 (a 9 ingressi). Questi file sono stati importati nel progetto semplicemente trascinandoli al suo interno e sono stati rinominati secondo il formato "*nomestrumento.cpp*" per agevolarne la distinzione.

- *MainComponent.cpp* e *MainComponent.h*. In questi file ci sono la classe `MainContentComponent`, che contiene tutti gli elementi dell'applicazione finale, più eventuali altre classi aggiuntive. I vari strumenti andranno inseriti e gestiti come componenti figli di `MainContentComponent`. Per accedere ai file header delle altre classi è necessario usare la direttiva *#include*:

```
//nel file MainComponent.h

#include "Oscillatore.h"
#include "Selezionatoreampiezza.h"
#include "Generatorerumore.h"
#include "Mixer1.h"
#include "Mixer2.h"
```

- *Main.cpp*, ovvero il file che si occupa della gestione della finestra che contiene l'applicazione.

Figura 3.12: Struttura della cartella Source di progetto per l'applicazione

3.2.6 Realizzazione dell'interfaccia utente

Come struttura di base dell'interfaccia si è scelto di utilizzare una barra a schede, che occupa poco spazio nella finestra e permette di passare agevolmente da uno strumento all'altro. JUCE dispone di un componente apposito:

- `TabbedComponent (TabbedButtonBar::Orientation orientation)`

Si tratta di una barra di navigazione, il cui orientamento verticale o orizzontale è specificato dal parametro orientation. Alla barra si possono facilmente aggiungere delle schede con il metodo `addTab`. I parametri di questo metodo descrivono il nome della scheda, il colore di sfondo, il componente che andrà ad ospitare e il comportamento da adottare in caso la scheda venga chiusa (indica se eliminare anche il relativo componente o meno).

Nel nostro caso creiamo un'istanza di TabbedComponent in `MainContentComponent` e aggiungiamo una scheda per ogni strumento, passando come parametro al metodo `addTab` le istanze delle relative classi.

```
//nel file MainComponent.cpp
MainContentComponent::MainContentComponent() //costruttore

{     …

      tab=new TabbedComponent(TabbedButtonBar::TabsAtTop); //nuova barra

      addAndMakeVisible(tab);

      tab->addTab("HOME",Colour(0xff515356),new Menu(),false);//scheda per homepage

Oscillatore* osc1=new Oscillatore();
osc1->setComponentID("osc1");
      tab->addTab ("OSC1",Colour(0xff515356),osc1,false);
Oscillatore* osc2=new Oscillatore();
osc2->setComponentID("osc2");
      tab->addTab ("OSC2",Colour(0xff515356),osc2,false);

      … //schede per gli altri oscillatori

      tab->addTab("SELEZIONATORE",Colour(0xff515356),new
Selezionatoreampiezza(),false);
      tab->addTab("GENERATORE",Colour(0xff515356),new Generatorerumore(),false);
      tab->addTab("MIX 1",Colour(0xff515356),new Mixer1(),false);
      tab->addTab("MIX 2",Colour(0xff515356),new Mixer2(),false);



      …

}
```

Come si vede, la prima scheda ospita un'istanza della classe `Menu`. Questa classe crea la pagina iniziale per l'applicazione e contiene il link al sito del progetto D.R.E.A.M., aggiunto con l'apposito componente JUCE `HyperlinkButton` (Figure 3.13 e 3.14) .

Le successive 9 schede ospitano ciascuna una nuova istanza della classe `Oscillatore`. Per distinguerle è stata assegnata ad ognuna un identificatore (detto ID) con il metodo `Component::setID (const string& newID)`.
Sfruttando questi identificatori si possono poi disegnare le scale opportune e il numero corrispondente ad ogni oscillatore. Le scale del primo, del secondo e del nono oscillatore sono infatti diverse; per gli altri è stata lasciata come riferimento la scala del primo.

Figura 3.13: Homepage dell'applicazione

```
classe MainContentComponent
{
    istanza di TabbedComponent
    { tab1 ← istanza di Menu()
      tab2 ← istanza di oscillatore()
      tab3 ← istanza di oscillatore()
        ...
      tab10 ← istanza di oscillatore()
      tab 11←istanza di selezionatoreampiezza()
      tab 12←istanza di generatorerumore()
      tab 13←istanza di mixer1()
      tab 14←istanza di mixer2()
    }
}
```

Figura 3.14: Struttura del contenuto di MainContentComponent

Si è poi scelto di eliminare la barra superiore con il titolo dell'applicazione e i pulsanti per chiudere, massimizzare e minimizzare la finestra. La gestione della finestra viene così interamente affidata al sistema Android, una volta compilata l'applicazione. Per eliminare i pulsanti rendiamo la finestra principale una `ResizableWindow`, invece della `DocumentWindow` dichiarata di default.

```
//nel file Main.cpp

class MainWindow    : public ResizableWindow
{

…

}
```

È stato infine aggiunto uno splashscreen, ovvero un'immagine con il logo D.R.E.A.M. che compare per qualche secondo all'apertura dell'applicazione. Per farlo si è ricorso al componente JUCE `SplashScreen`:

```
//nel file Main.cpp
class OverviewApplication  : public JUCEApplication#
{
      …

void initialise (const String& commandLine)
    {
      …
       SplashScreen* splash = new SplashScreen();
       splash->show ("DreamVST",
       ImageCache::getFromMemory(BinaryData::splash_png,BinaryData::splash_pngSize),
       4000, false);
      …
    }


      …
}
```

3.2.7 Build per piattaforma Android

Prima di procedere alla compilazione è necessario verificare che tutti i parametri della finestra “Config” di Introjucer (quella con le proprietà del progetto) siano impostati correttamente, in particolare bisogna controllare i percorsi di SDK e NDK per il target Android. Sempre dalla scheda “Config” si deve inoltre impostare l'icona dell'applicazione, nel nostro caso è stato scelto il logo del progetto D.R.E.A.M..

Per eseguire il debug dell'applicazione si può eseguire il comando *ant debug* da terminale, dopo essersi spostati nella cartella di progetto [Builds/Android]. Si può poi procedere con il comando *ant release*, alla fine del quale l'applicazione sarà disponibile nella cartella [Builds/Android/bin] con estensione “.apk”. Questo file può essere trasferito al dispositivo Android e installato manualmente, tuttavia si è scelto di passare per il software Eclipse per monitorare eventuali errori o malfunzionamenti dalla finestra di LogCat. Dopo aver importato il progetto in Eclipse è sufficiente collegare il terminale Android al pc e premere il pulsante “Run” (o dal menu Run → Run).

Nella cartella radice di ogni progetto Android è presente il file “AndroidManifest.xml”, in cui sono raccolte le informazioni che il sistema Android deve necessariamente avere prima di eseguire il codice dell'applicazione. Fra queste è indicato l'orientamento dell'applicazione (“landscape”, ovvero orizzontale , “portrait”, verticale, “sensor” per l'orientamento automatico in base all'accelerometro del dispositivo, etc.) e il tema della finestra. Nel nostro caso si desidera che l'applicazione sia sempre in modalità “landscape” e sia avviata a schermo intero. Per fare ciò aggiungiamo i seguenti attributi alla sottoclasse `activity`, che implementa i vari aspetti dell'interfaccia utente:

```
//nel file AndroidManifest.xml

<manifest …
…
<activity …
android:screenOrientation="landscape"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
…
</activity>
…
</manifest>
```

3.3 Usabilità

Dopo aver testato il funzionamento dell'applicazione sul tablet sono stati modificati alcuni aspetti per migliorarne l'usabilità:

1. Trascinamento delle manopole

   Dopo aver testato i vari tipi di trascinamento per gli slider si è scelto infine il trascinamento orizzontale e verticale. Questa tipologia rende la manopola più reattiva e l'animazione più fluida rispetto al trascinamento rotativo e permette un controllo fine del valore delle manopole continue.

2. Sensibilità delle manopole

   La sensibilità degli slider a trascinamento orizzontale/verticale è regolabile tramite il metodo `Slider::setMouseDragSensitivity (int distanceForFullScaleDrag)`, che imposta la distanza che deve percorrere il mouse per passare dal valore minimo al valore massimo. Dopo aver fatto varie prove è stato scelto per il parametro `distanceForFullScaleDrag` un valore pari circa alla metà della larghezza della manopola.

3. Popup valore dello slider

   Per visualizzare il valore corrente dello slider è stato usato il metodo `Slider::setPopupDisplayEnable (bool isEnabled)` su tutte le manopole a rotazione continua. Questo metodo fa in modo che durante il trascinamento compaia un popup con il valore corrente.

Di seguito vengono riportate le schermate dell'applicazione finale (Figure 3.15-3.19).

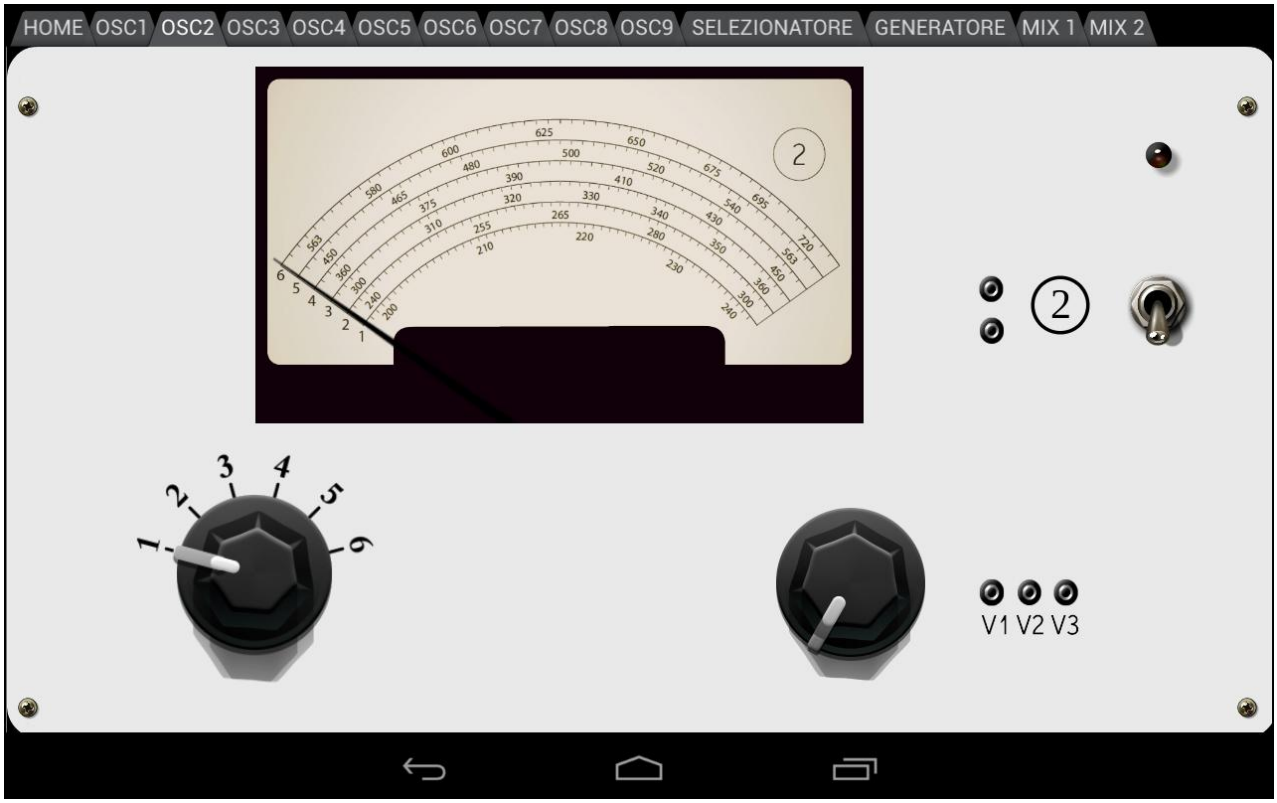


Figura 3.15: Secondo oscillatore (screenshot catturato dal tablet)

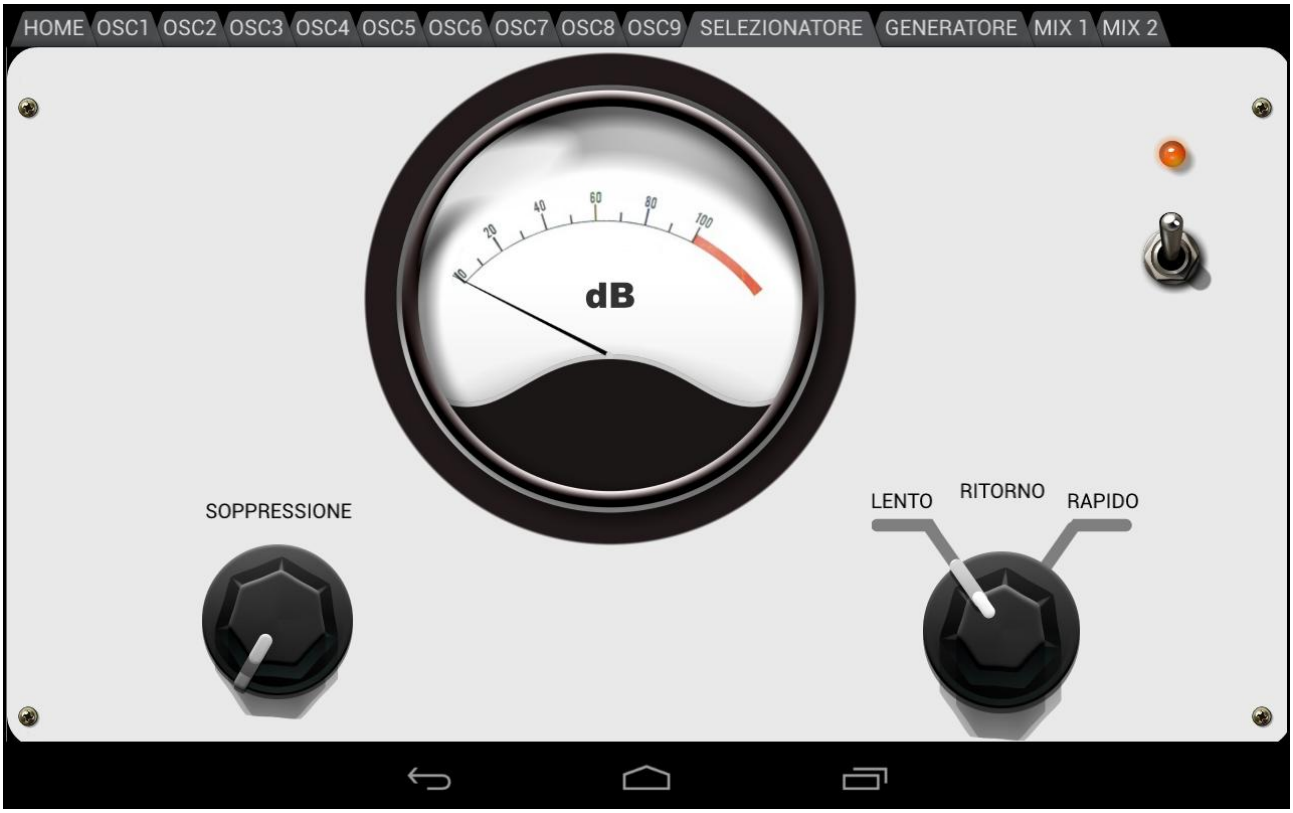


Figura 3.16: Selezionatore d'ampiezza (screenshot catturato dal tablet)

Figura 3.17: Generatore di rumore bianco (screenshot catturato dal tablet)

Figura 3.18: Mixer a 8 ingressi (screenshot catturato dal tablet)

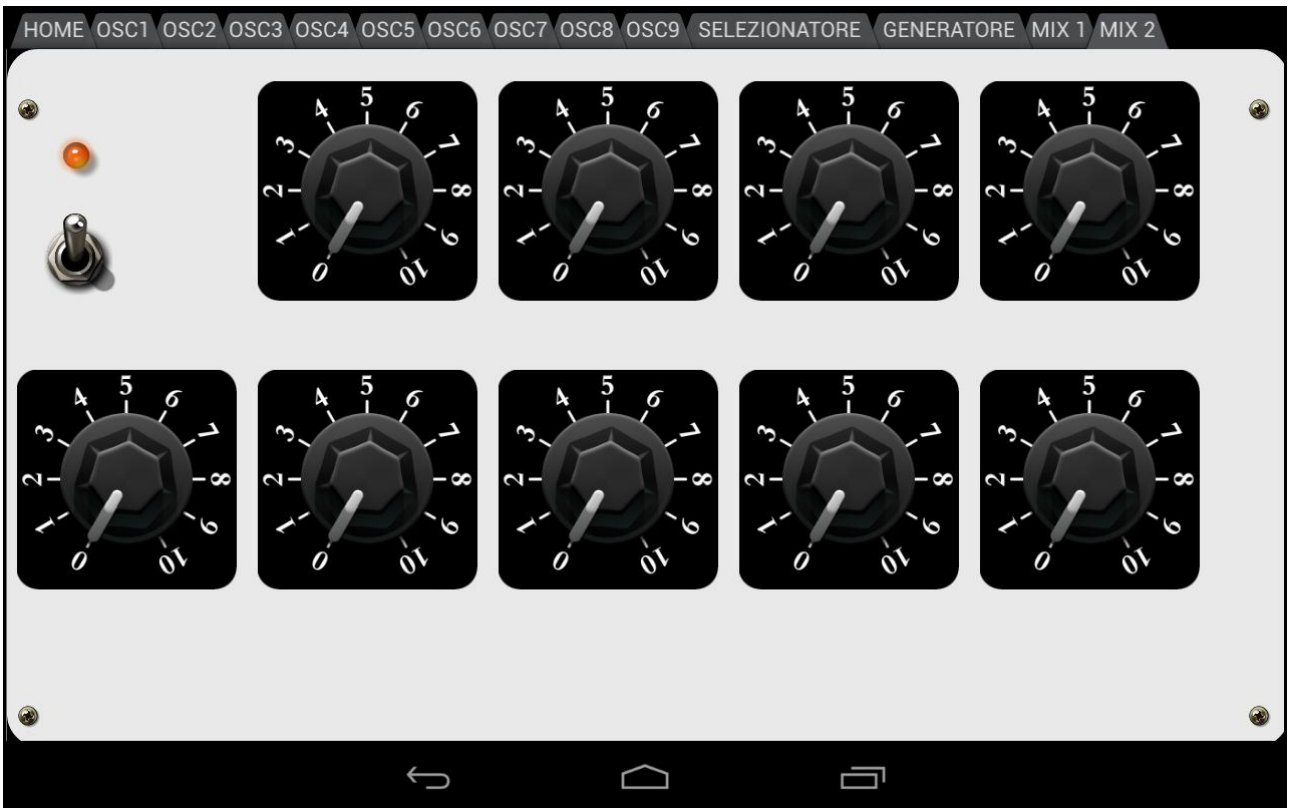


Figura 3.19: Mixer a 9 ingressi (screenshot catturato dal tablet)

Capitolo 4

Conclusioni e sviluppi futuri

Il lavoro svolto funge da front-end e punto di partenza per terminare l'applicazione Android DREAM Vst. Per completarla è necessario integrare nel progetto JUCE la parte audio che simuli il comportamento acustico degli strumenti. L'installazione al museo del Castello Sforzesco di Milano ospita le ricostruzioni di tre oscillatori e il relativo mixer, il generatore di rumore bianco, il selezionatore d'ampiezza con il banco di filtri d'ottava e il relativo mixer. Per la simulazione audio degli strumenti sono state realizzate delle patch in ambiente Pure Data e sono state utilizzate delle librerie in C per la modellazione fisica dei singoli componenti elettronici; il software è stato poi opportunamente esteso per essere controllato dalle interfacce tangibili ricostruite.
Lo sviluppo futuro di questo lavoro prevede di completare l'applicazione collegando i controlli alle funzionalità audio corrispondenti, sfruttando le patch già esistenti. Vi sono due strade:

1. Integrare in JUCE libpd, una libreria concepita per agevolare l'utilizzo del codice Pure Data nello sviluppo di applicazioni mobile, di giochi e molte altre e di rendere compatibili le patch con altri ambienti di sviluppo.

2. Ricreare il comportamento acustico degli strumenti utilizzando le librerie audio di JUCE, facendo un porting delle patch Pure Data.

Successivamente si prevede di incapsulare l'applicazione in un plug-in Steinberg VST[14] e in un Audio Unit[15] (l'equivalente Apple di quest'ultimo) per utilizzare gli strumenti dello Studio di Fonologia nei più comuni sequencer (Cubase, ProTools, etc).

14 Link al sito di Steinberg: http://www.steinberg.net/en/products/vst.html
15 Link alla documentazione di Audio Unit: https://developer.apple.com/library/mac/#documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/TheAudioUnit/TheAudioUnit.html#//apple_ref/doc/uid/TP40003278-CH12-SW1

Bibliografia

[1] Giovanni Belletti. *Marino Zuccheri in Fonologia,*2008.

[2] Maria Maddalena Novati and John Dack, editors. *The studio di fonologia, a musical journey 1954-1983*. Ricordi, 2012.

[3] pages 304-309 in S. Canazza, A. Rodà, M. Novati, and F. Avanzini, *Active preservation of electrophone musical instruments. the case of the "Liettizzatore" of "Studio di Fonologia Musicale" (RAI, Milano)*. In *Proc. Int. Conf. Sound and Music Computing (SMC2011)*. Padova, July 2011. Padova University Press.

[4] Raw Material Software ltd. *JUCE complete doxygen-generated API reference guide.* <http://www.juce.com/api/classes.html>

[5] *JUCE programming Tutorial*. <http://lubyk.org/en/software/mimas/document171.pdf>

[6] M. Puckette, *Max at seventeen*, Computer Music J., vol. 26, no. 4, pp. 31-43, 2002.