UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Triennale in Ingegneria Gestionale

Tesi di Laurea di Primo Livello

SVILUPPO DI UN'INTERFACCIA GRAFICA PER UN SOFTWARE BIOMECCANICO IN AMBIENTE ANDROID

DEVELOPMENT OF A GRAPHICAL USER INTERFACE FOR A BIOMECHANICAL SOFTWARE IN ANDROID ENVIRONMENT

Relatore: Prof.ssa MONICA REGGIANI

Correlatore: Ing. CLAUDIO PIZZOLATO

Laureanda: FRANCESCA ROSSATO

Anno Accademico 2013-2014

RINGRAZIAMENTI

Desidero ringraziare tutti coloro che mi hanno aiutata nella stesura di questa tesi, ringrazio in particolare la prof.ssa Monica Reggiani e l'ing. Claudio Pizzolato per il loro aiuto e la loro disponibilità, ringrazio inoltre la mia famiglia, mio fratello Gianmarco, i miei compagni di studi e Marco, che mi hanno supportata fino alla fine.

INDICE

SOMMARIO

L'obiettivo del progetto è la realizzazione di un'interfaccia grafica che permette l'inserimento di alcuni parametri da parte dell'utente, i cui valori verranno poi salvati ed utilizzati da un'altra applicazione integrata ad essa. L'obiettivo è quello di realizzare un'applicazione compatibile con i diversi tipi di dispositivi che utilizzano il sistema operativo Android. Per ottenere questo obiettivo sono state utilizzate due diverse librerie di sviluppo: Android SDK e Kivy. Il risultato della tesi non è solo la creazione di due interfacce grafiche con le stesse funzionalità, ma anche la comparazione tra le due librerie per individuare la più efficace per i nostri bisogni. Nei primi due capitoli vengono presentati gli strumenti di progettazione utilizzati, descrivendone gli ambienti di sviluppo, gli oggetti forniti per la realizzazione dell'interfaccia utente, le regole di programmazione e la struttura dei progetti che vengono creati con essi. Nel terzo capitolo viene presentata la struttura dell'interfaccia che si vuole ottenere, con una descrizione di ciò che deve essere presente e delle possibili soluzioni da adottare. Gli ultimi due capitoli riguardano in modo specifico la progettazione dell'interfaccia, sia descrivendo il processo con Kivy che descrivendolo con Android SDK.

INTRODUZIONE

La diffusione di dispositivi elettronici portatili ha portato allo sviluppo di numerose applicazioni funzionanti su di essi, che permettono di semplificare le operazioni nel mondo del lavoro o nella vita quotidiana. Android è uno dei sistemi operativi più diffusi, funzionante sulla maggior parte di dispositivi quali smartphone e tablet, e molti software applicativi vengono progettati basandosi su di esso. Infatti, l'applicazione che si vuole ottenere verrà creata sulla base di due tool di progettazione che permettono di realizzare progetti compatibili con questo sistema operativo. L'obiettivo finale è quello di realizzare un'applicazione funzionante ed intuitiva, caratterizzata sostanzialmente da un'interfaccia grafica con la quale l'utente può interagire. L'interfaccia si compone di due pannelli dediti alla ricezione di input da tocco, grazie allo sfruttamento del touchscreen di cui dispongono i dispositivi moderni, e all'inserimento di valori da tastiera. Il progetto include soltanto una parte di un'applicazione più grande e completa a cui verrà integrato, e quindi si concentra soltanto sulla realizzazione di questi due pannelli, nonostante possa essere ampliato con l'aggiunta di altri. Il primo tool di progettazione è chiamato Kivy, che dispone di una libreria che permette lo sviluppo di applicazioni in linguaggio Python fornendo la possibilità di realizzare interfacce intuitive con un linguaggio semplificato, nonostante esso sia ancora un progetto in via di sviluppo che necessita di miglioramenti. Grazie a Kivy, lo sviluppatore alle prime armi riesce ad acquisire dimestichezza con gli elementi widget e con il concetto di layout, imparando a gestire il posizionamento degli oggetti nell'interfaccia, che possono essere sia decorativi (cioè statici) che funzionali, e le caratteristiche che possono offrire all'utente. Il tool Android SDK è molto più diffuso di Kivy e anch'esso permette di creare la stessa interfaccia, offrendo molte più funzionalità, ma è caratterizzato da un approccio meno intuitivo. Entrambi sono software open source, scaricabili da internet, ed entrambi verranno utilizzati per realizzare l'applicazione richiesta, che verrà supportata da una descrizione del processo di realizzazione compiuto sia con uno che con l'altro software. Però, come detto in precedenza l'applicazione ottenuta alla fine del processo non rappresenta il lavoro ultimato, e per poterne usufruire appieno bisogna integrarla con nuove funzionalità, aggiungendo parti di programma e inserendo nuovi pannelli. Per ora, infatti, il progetto si focalizza soltanto sulla parte grafica dell'applicazione, quella che stabilisce ciò che appare visivamente all'utente e quella che, sfruttando il touchscreen, gestisce le interazioni con esso. Dopo la progettazione ci sarà la fase di verifica, in cui, attraverso un emulatore o un dispositivo Android, si testa il funzionamento e l'appetibilità grafica dell'applicazione, oltre che la buona visibilità degli oggetti che la compongono, a prescindere dalle dimensioni dello schermo.

CAPITOLO 1

Introduzione a Kivy

Kivy è una libreria open source per lo sviluppo di software applicativo multi-touch in Python, compatibile in ambiente Android, iOS, Linux, OS X e Windows. L'obiettivo è la possibilità di realizzare in maniera intuitiva ed immediata applicazioni funzionanti su più piattaforme.
La struttura prevede la presenza di supporto per input da mouse, tastiera ed eventi multi-touch, una libreria grafica, un'ampia gamma di widget e un linguaggio semplificato per la loro progettazione e personalizzazione. Kivy è un progetto in via di sviluppo e quindi non è supportato da una ricca documentazione, ma, a suo favore, è caratterizzato da una NUI (Natural User Interface) che semplifica il controllo dell'applicazione da parte dell'utente, il quale ha la possibilità di sfruttare dei movimenti naturali per gestire e manipolare le applicazioni del computer.
Kivy è un progetto implementato da professionisti sviluppatori di software, con lo scopo di migliorarlo nel tempo e di integrarlo con nuove funzionalità; per promuoverne l'utilizzo, nel 2012 è stato proposto un contest in cui sviluppatori di app dovevano utilizzarlo per cimentarsi nella creazione di un gioco funzionante su vari dispositivi.

1.Struttura del framework

Kivy fornisce un linguaggio che permette di separare la parte grafica (design dell'interfaccia) dalla parte di programmazione in Python, scomponendo il progetto finale in due tipologie di file, uno con estensione .kv e l'altro, di programmazione, con estensione .py: nell'esempio riportato più in seguito i due file sono project1.kv (che definisce il design dell'interfaccia, ovvero ciò che viene visualizzato) e project1.py (che permette l'esecuzione del programma).
Nel file .kv si trovano le classi (insiemi di metodi e attributi), rappresentate tra parentesi < >, il layout, ovvero il modo in cui si vuole disporre gli oggetti graficamente, gli oggetti o widget con i relativi parametri che definiscono come devono essere visualizzati.
I layout sono definiti da widget che permettono di disporre gli oggetti nell'interfaccia nei seguenti modi:

- *BoxLayout*: i widget vengono ordinati verticalmente o orizzontalmente;
- *GridLayout*: i widget vengono disposti in una griglia in cui vengono definiti il tipo e il numero di righe (rows) e colonne (cols);
- *AnchorLayout*: i widget vengono ordinati agli estremi dello spazio disponibile, in alto, in basso, a destra o a sinistra;

- *RelativeLayout*: i widget sono disposti secondo certi parametri nello spazio disponibile, senza un ordine preciso;
- *FloatLayout*: i widget non hanno restrizioni di disposizione;
- *StackLayout*: i parametri sono “lr-tb”, se i widgets vengono disposti progressivamente da sinistra a destra e poi dall’alto verso il basso (left to right, then top to bottom), oppure “tb-lr”, “rl-tb”, “tb-rl”, “bt-lr”, “lr-bt”, “rl-bt” e “bt-rl”.

Gli eventi, in pratica richiamo di funzioni Python, gestiscono l’azione dell’utente sui widget, e in Kivy vengono in sostanza suddivisi in tre gruppi :

- *Class Events*: associati alla classe EventDispatcher, che registra i tipi di cambiamenti e li invia a determinate parti, come, per esempio, widget. Se un widget cambia le sue dimensioni vuol dire che è avvenuto l’evento che definisce il suo cambiamento. Gli eventi “on_press” o “on_release” riguardano il widget tasto, in inglese Button, il quale, se premuto, genera il cambiamento imposto dall’evento (come, per esempio, generare una finestra di popup);
- *Input Events*: riguardano, per esempio, i click del mouse, il tocco di un touchscreen, lo “scrolling” (un movimento che permette di scorrere delle pagine) e fanno parte della classe MotionEvent. Una volta che questi input sono stati letti dai dispositivi, vengono inviati a un modulo di post-elaborazione e dopo inviati all’applicazione. Si tramutano in eventi come “on_touch_down”, “on_touch_up”, “on_touch_move”;
- *Clock Events*: questi eventi permettono di programmare l’esecuzione futura di alcune funzioni, e possono essere o un evento unico, o un evento ripetitivo nel tempo, o un Trigger Event, cioè un evento che genera altri eventi come un susseguirsi di azioni.

Le proprietà dei widget sono collegate agli eventi e generano, a loro volta, un evento (quello di default ha la struttura “on_<nome della proprietà>”) che aggiorna tutte le proprietà associate allo stesso attributo di un oggetto, quando questo viene modificato. Le proprietà variano a seconda delle informazioni che devono rappresentare, una variabile String (insieme di caratteri) viene associata alla proprietà StringProperty, mentre un valore numerico viene associato alla proprietà NumericProperty, e lo stesso vale per gli altri tipi di valori. Se si vuole che, alla pressione di un tasto, un widget cambi una sua proprietà, definita inizialmente con valori di default, deve esistere una funzione Python che risponda all’evento, aggiornando questa con i nuovi valori nel punto in cui viene definito l’evento.

Tra gli elementi non widget troviamo :

- *animazioni*, che permettono di variare le proprietà di un widget in un certo intervallo di tempo, con diverse modalità di transizione;
- *orologio*, che viene utilizzato per stabilire il momento di esecuzione delle funzioni e gli intervalli di tempo;
- *classe Atlas*, per gestire gruppi di texture ed organizzarli in un'unica immagine, per velocizzare l'applicazione e ridurre il numero di immagini da caricare.

La parte di disegno, come la creazione di rette, poligoni, cerchi, ellissi o figure generiche che compongono i widget, è affidata al Canvas, ovvero ad una "tela" virtuale su cui disegnare. Il Canvas, che caratterizza ciascun widget, non è altro che un gruppo di istruzioni aggiunte nel linguaggio Python o nel file .kv, con il vantaggio che, nell'ultimo caso, le istruzioni vengono aggiornate automaticamente se avviene qualche cambiamento.

2.Introduzione ai widget

I widget sono gli elementi di base per la costruzione di un'interfaccia grafica, reagiscono alle interazioni dell'utente generando eventi, e possiedono un Canvas per il disegno. La struttura del widget è ad "albero", abbiamo il widget di base (root widget), i widget figli (children) contenuti all'interno di quello base, e questi ultimi, a loro volta, possono avere altri widget figli. Per aggiungere un widget figlio si utilizza il comando "add_widget()", per rimuoverlo si utilizza "remove_widget()", per rimuovere tutti i figli di un Widget si utilizza "clear_widget()". Per esempio, per aggiungere un widget Button (chiamato per convenzione button) ad un generico layout, generalmente il root widget, il comando è "layout.add_widget(button)". I widget possono essere personalizzati cambiando i valori dei parametri, per variare le proprietà e le caratteristiche grafiche. Nel pacchetto scaricabile da internet è incluso il file di un catalogo interattivo contenente alcuni widget, con la possibilità di variarne il codice e di visualizzarne gli effetti dei cambiamenti. Tra quelli presentati, oltre ai layout, troviamo i tipi:

- *Label:* è un widget per raffigurare il testo e supporta stringhe di tipo ascii e unicode;
- *Button:* è un widget che può essere considerato come una Label a cui vengono associate delle azioni, se questo viene premuto. Le proprietà di configurazione di un Button sono simili a quelle di una Label. Gli eventi associati ad esso sono, per esempio, "on_press" e "on_release";
- *CheckBox:* è un pulsante a due stati, lo stato "selezionato" (checked) e lo stato "non selezionato" (unchecked), e permette, quindi, le selezioni multiple da parte dell'utente.

Quando un CheckBox viene selezionato, compare un segno di spunta che identifica la selezione;

- *RadioButton:* è un CheckBox che appartiene ad un gruppo, in cui è possibile effettuare solo un'unica selezione tra le varie scelte;
- *ToggleButton:* è un pulsante rappresentato graficamente come un Button, ma che si comporta come un CheckBox o come un RadioButton;
- *Switch:* è un widget che permette all'utente di scorrere un pulsante per poterlo attivare (on) o disattivare (off);
- *Progress Bar:* è un widget che permette di visualizzare l'avanzamento di progresso di qualche operazione, come l'esecuzione di un video o di un brano musicale, ma non permette interazioni con esso;
- *TextInput:* è un widget che fornisce una casella in cui inserire un testo, che può essere unicode, multilinea, password o selezionabile;
- *Popup:* è un widget che genera una nuova schermata, in cui possono essere inseriti ulteriori widget. Si può creare dinamicamente un Popup cliccando su un apposito Button, al quale è associato l'evento che lo genera;
- *Spinner:* è un widget che permette di selezionare un elemento da un set di elementi. La selezione visibile sullo Spinner è quella di default; per cambiare selezione basta sceglierne una nuova nel menu a discesa dello Spinner;
- *ListView:* è un widget che permette il controllo e la selezione di file e cartelle disposti in una lista scorrevole. La visualizzazione dei file o cartelle può essere come testo o come icona .

I widget e le loro proprietà sono molto simili a quelle di Android SDK, che vedremo presentato in seguito.

CAPITOLO 2

Introduzione ad Android SDK

Dal sito Android Developers è possibile scaricare gli strumenti necessari per la realizzazione di applicazioni compatibili con il sistema operativo Android. L'Android SDK, dove SDK sta per Software Development Kit, è un kit open source per lo sviluppo software, che permette di scrivere del codice sorgente di qualsiasi versione di Android. Il codice sorgente, per la maggior parte delle applicazioni Android, è un testo scritto nel linguaggio di programmazione Java e nel linguaggio di markup XML (linguaggio che permette il controllo e la definizione degli elementi e parametri in formato testo). Come tool di progettazione risulta molto più sviluppato rispetto a Kivy, è supportato da una ricca documentazione e fornisce un emulatore per i dispositivi Android presenti sul mercato, permettendo di verificare il funzionamento delle applicazioni create.

1.Struttura del framework

L'ambiente di sviluppo incluso è Eclipse, dove troviamo strumenti per la programmazione in Java (mentre con Kivy si programmava in Python) e per il codice in XML, con tanto di visualizzazione grafica per verificare il design dell'interfaccia dell'applicazione, man mano che questa viene creata. Il progetto viene gestito da Eclipse, in un pacchetto/cartella che contiene, a sua volta, sottocartelle con elementi di tipo diverso, come file .java, .xml, .class, immagini .png o librerie esterne. La versione di Eclipse scaricabile dal sito Android Developers include l'ADT (Android Developer Tools), per facilitare la realizzazione di applicazioni Android e il loro debug. I progetti generati vengono archiviati in un file con estensione .apk, in modo da poter essere letti dai dispositivi Android.
Il plugin di Android SDK per Eclipse fornisce una gamma di interfacce di vari dispositivi Android (smartphone e tablet), per poter vedere come varia il layout dell'applicazione a seconda del dispositivo utilizzato, in modo da creare un progetto compatibile con gli standard voluti. Sono presenti quattro tipi di componenti fondamentali per la costruzione di un'applicazione:

- *Activity:* un'attività viene rappresentata come uno schermo con il quale l'utente può interagire, ovvero fornisce un'interfaccia utente (UI). La classe Activity permette la creazione di un pannello, in cui, con il metodo setContentView(View), è possibile posizionare gli elementi dell'interfaccia ;

- *Service:* un “service” è un componente che non prevede interazioni con l’utente, ma agisce “di nascosto”, in background, per eseguire lunghe operazioni o per processi remoti, e può essere avviato da una Activity ;
- *Content Provider:* i content provider gestiscono i dati dell’applicazione e, attraverso di essi, le altre applicazioni possono accedere a tali dati (che possono essere salvati in diverse location) oltre che, se permesso, modificarli. E’ utile per poter leggere informazioni non condivise con l’applicazione ;
- *Broadcast Receiver:* un broadcast receiver riceve e risponde agli avvisi di broadcast del sistema, dove per broadcast si intende una trasmissione di un’informazione a più dispositivi nella stessa rete. Un broadcast receiver non possiede un’interfaccia utente, ma può generare degli avvisi nella barra delle notifiche del dispositivo, come, per esempio, avvisi di batteria scarica o di un download avvenuto. In genere un broadcast receiver funge semplicemente da passaggio per gli elementi che vengono richiesti dall’applicazione: per esempio, potrebbe avviare un elemento service per eseguire alcune operazioni basate su un evento.

2.Struttura e componenti di un Android Project

Un Android Project è composto dai file del codice sorgente dell’applicazione, e da una serie di altri file che Android SDK, di default, genera nel momento in cui si crea il progetto stesso, di cui definiamo il nome e le caratteristiche. I file e cartelle presenti nell’Android Project sono:

- *src/:* è una cartella che contiene i file principali del codice sorgente dell’applicazione, e include una classe Activity, che viene eseguita all’avvio dell’applicazione;
- *res/:* contiene sottocartelle riguardanti le risorse di applicazione, in inglese app resources, ovvero risorse come immagini o stringhe, esternalizzate rispetto al codice dell’applicazione. L’esternalizzazione garantisce una migliore compatibilità dell’app con i vari dispositivi e più possibilità di configurazione, in quanto i file esternalizzati possono essere facilmente sostituiti con altri, a seconda delle esigenze;
- *layout/:* è una sottocartella di res/ e contiene i file .xml, che permettono di definire il layout dell’interfaccia utente;
- *values/:* è una sottocartella di res/ e contiene file .xml riguardanti insiemi di risorse relativamente semplici, come stringhe, colori, numeri;
- *drawable-hdpi/:* è una sottocartella di res/ che contiene oggetti “disegnabili virtualmente”, in inglese drawable objects, e progettati per schermi ad alta densità di pixel. Un esempio di questo tipo di oggetti è il bitmap, cioè un’immagine digitale;

- *AndroidManifest.xml:* è un file che contiene le informazioni necessarie al sistema Android per poter eseguire l'applicazione. Descrive le quattro componenti principali di cui è composta l'applicazione, attribuendo un nome alle classi associate ad esse e riportando le loro funzionalità, in modo che il sistema sappia se una componente può essere avviata. Dichiara quali permessi l'applicazione deve avere per accedere a parti protette dell'Interfaccia di Programmazione dell'Applicazione (API), e che permessi devono avere gli altri per poter interagire con le componenti dell'applicazione. Stabilisce quali processi possono essere ospitati dalle componenti e dichiara il livello minimo dell'Android API richiesto dall'applicazione.

3.Organizzazione e tipologie dei widget

Per realizzare un'interfaccia utente bisogna tenere conto di una struttura gerarchica, che riprende i concetti di root widget e widget figlio visti nell'introduzione a Kivy, composta da oggetti ViewGroup e da oggetti View. I primi sono oggetti non visibili e fungono da contenitori di altri widget (children), che vengono posizionati secondo un certo layout; i secondi sono le componenti di base della UI, widget interattivi come, per esempio, tasti, campi di testo, selezionatori. La classe ViewGroup è una sotto-classe della classe View e include i widget per i layout, simili a quelli di Kivy. Tra i layout disponibili abbiamo: LinearLayout, simile al BoxLayout in Kivy; RelativeLayout; la WebView, per visualizzare pagine web; la ListView, che permette di ottenenere un lista scorrevole di elementi in colonna, simile alla ScrollView in Kivy; la GridView, uguale al GridLayout in Kivy.
I widget, che riprendono le stesse funzionalità di quelli di Kivy, hanno parametri e disposizioni che vengono definite nel file .xml all'interno della cartella "layout", mentre gli eventi associati ad essi nel file .java dell'Activity principale. Alcuni dei widget utilizzati sono:

- *TextView:* è l'equivalente della Label in Kivy, ma con la possibilità di avere TextView predefiniti, per scegliere direttamente la grandezza del testo;
- *Button:* equivalente a quello in Kivy, ma l'evento con il metodo associato viene chiamato e definito nel file .java dell'Activity del pannello. In Kivy il metodo associato all'evento viene chiamato nel file .kv e definito nel file .py;
- *RadioButton/ToggleButton:* simili a quelli in Kivy, con le stesse caratteristiche di un Button. Per permettere una singola selezione in un set di opzioni bisogna definire un RadioGroup, che funge da layout root widget per i RadioButton/ToggleButton, mentre con Kivy basta inserire tra i parametri dei suddetti widget il gruppo di appartenenza, senza utilizzare un apposito root widget;

- *Spinner:* simile a quello in Kivy, ma gli elementi di selezione devono essere introdotti tramite un array, ovvero un insieme di elementi definito nel file strings.xml della cartella “values”. In Kivy gli elementi dello Spinner vengono introdotti direttamente nel file .kv, nel punto in cui viene definito il widget, senza ricorrere ad un array;
- *ImageView:* è un widget che permette di visualizzare una qualunque immagine, come un’icona, che può essere caricata da varie risorse o dai content provider. In Kivy viene chiamato Image;
- *EditText*: è l’equivalente del TextInput in Kivy, con la possibilità di avere degli EditText predefiniti che variano a seconda degli input (password, date, integer, decimal...);
- *Dialog:* è l’equivalente della finestra di Popup in Kivy. Generalmente una finestra di Dialog ha una grandezza che non occupa tutto lo schermo e permette all’utente di compiere delle azioni all’interno di essa;
- *CheckBox* e *ListView* sono equivalenti a quelli in Kivy;

Man mano che introduciamo nel linguaggio XML i widget nel progetto, le azioni effettuate possono essere visualizzate graficamente nella sezione Graphical Layout di Eclipse. Si tratta inoltre di una sezione interattiva, in cui si ha la possibilità di aggiungere, modificare o eliminare dinamicamente i widget, senza lavorare direttamente con il file .xml associato. A destra di questa sezione è presente un pannello, in cui, per ogni widget aggiunto, vengono riportate tutte le proprietà e le caratteristiche associate ad esso, con la possibilità di personalizzarlo, modificandone i valori di default. Le operazioni eseguite vengono comunque tradotte dinamicamente nel linguaggio XML, senza la necessità da parte dello sviluppatore di riportare tutte le modifiche effettuate nella sezione grafica. Nell’ambiente di sviluppo di Python non è possibile visualizzare al momento ciò che viene creato, ma per valutare una scelta si dovrà avviare l’applicazione Kivy, che eseguirà il progetto mostrandone la soluzione grafica. Se la struttura di Kivy è più semplice da comprendere, Android SDK permette di tenere sotto controllo con più facilità l’avanzamento del progetto, con una visualizzazione grafica immediata e interattiva, permettendo velocità di modifica e di personalizzazione.

CAPITOLO 3

Caratteristiche dell'interfaccia grafica

I due strumenti di sviluppo presentati precedentemente sono stati utilizzati per la realizzazione dell'interfaccia grafica del progetto. Il layout è simile in entrambi i casi, a cambiare è il design, dovuto anche alle configurazioni di default dei due strumenti, e la parte di programmazione. L'interfaccia creata si compone di due pannelli, il primo, per la maggior parte, si avvale di widget relativamente semplici, come Label e Checkbox, il secondo necessita di finestre di popup per problemi di spazio e per rendere più intuitive le azioni da parte dell'utente. Per rendere l'applicazione compatibile su più fronti, oltre che garantirne il funzionamento sul sistema operativo Android, è necessario che questa abbia una parte grafica che si adatti alla risoluzione dello schermo su cui viene visualizzata. Ciò significa che le dimensioni dei widget e dei pannelli devono poter cambiare dinamicamente, sebbene il design rimanga sempre lo stesso. Per non avere restrizioni di spazio verticale si utilizza un layout particolare, ScrollView, che consente di inserire i widget in uno spazio virtuale più grande dello screen, permettendo di poter scorrere il pannello. In questo modo non è importante quanto sia la dimensione verticale dello schermo, basta scorrere in su e giù il pannello, mediante l'utilizzo del touchscreen, per poterlo visualizzare tutto. Per quanto riguarda la larghezza, i widget devono essere configurati in modo tale che la loro dimensione orizzontale vari rispetto a quella dello schermo: in uno schermo "stretto" i widget risulteranno più corti, mentre in uno ampio risulteranno più larghi, rispettando i rapporti di spazio tra di essi. Senza la presenza di una ScrollView si dovrebbe effettuare lo stesso procedimento con le dimensioni verticali. Infatti, se le loro grandezze fossero statiche, ovvero non potessero variare dinamicamente, in uno schermo piccolo i widget tenderebbero a sovrapporsi, a causa dei limiti di spazio, peggiorandone la visualizzazione. I pannelli del progetto sono organizzati in tab, ovvero in schede raggruppabili e selezionabili, mediante un widget TabbedPanel predefinito (attualmente un widget ancora sperimentale) in Kivy, oppure realizzando la struttura nel programma in Android SDK. Ad ogni tab viene associato un nome, che lo identifica in un insieme di pannelli, in modo da facilitare la selezione da parte dell'utente: il primo pannello è stato chiamato "Execution", il secondo "Calibration".

1.Caratteristiche del pannello "Execution"

Partendo dall'alto verso il basso, il primo pannello si compone dei seguenti widget, scelti tenendo conto della struttura richiesta: una Label per il titolo principale; una serie di Spinner associati ad una Label per il titolo che li descrive; un TextInput associato anch'esso ad una

Label; una Label per il titolo di una sotto-sezione, che descrive un insieme di CheckBox per la selezione delle stringhe; tre pulsanti Button.

I valori e la struttura richiesta dal primo pannello sono come segue:

- titolo “NMSmodel Configuration” per la sezione principale;
- titolo “NMSmodel Type” e una selezione tra i parametri “Open Loop” e “Hybrid”;
- titolo “Tendon Type” e una selezione tra i parametri “Stiff”, “Elastic (Integration)” e “Elastic (RootSolving)”;
- titolo “Activation Type” e una selezione tra i parametri “Exponential” e “Piecewise”;
- titolo “Execution Mode” e una selezione tra i parametri “Online” e “Offline”;
- titolo “Sampling Frequency” e casella di testo per inserire un numero intero, con valore di default “200”;
- titolo “Degrees of Freedom” per una sotto-sezione;
- sotto-sezione composta da una scelta multipla di stringhe: “R_Knee_FE”, “R_HipAA”, “R_HipROT”, “R_HipFE”, “R_AnkleFE”, “R_AnkleSA”;
- pulsanti “Save”, “Load”, “Run”.

2.Caratteristiche del pannello “Calibration”

Per il secondo pannello, invece, i valori e la struttura richiesta è la seguente:

- titolo “Calibration algorithm” per una prima sezione;
- una scelta alternativa tra i parametri “Simulated Annealing” e “Particle Swarm”. Selezionando la prima alternativa si attiva la possibilità di editare i seguenti campi, con il relativo valore di default: “noEpsilon” (4), “rt” (0.3), “T” 20), “NS” (15), “NT” (5), “epsilon” (10*-3), “maxNoEval” (200000); selezionando la seconda si attiva la possibilità di editare i seguenti campi, con il relativo valore di default: “paramA” (1), “paramB” (2), “paramC” (3);
- titolo “NMS model Selection” per una seconda sezione;
- titolo “Tendon” e una selezione tra i parametri “Stiff”, “Elastic” e “Elastic_BiSec”;
- titolo “Activation” e una selezione tra i parametri “Exponential” e “Piecewise”;
- titolo “Calibration Steps” per una terza sezione;
- possibilità di aggiungere o rimuovere dinamicamente un numero variabile di oggetti “Step”, che aprono una nuova sezione con la seguente struttura:
- sezione “Degrees of Freedom” e scelta tra uno o più dei parametri “Knee_FE”, “Hip_FE”, “Hip_AA”, “Hip_ROT”, “Ankle_FE”, “Ankle_SA”;
- sezione “Parameter Set” e scelta tra uno o più dei parametri “Strength Coefficient”, “Shape Factor”, “C1”, “C2”, “Tendon Slack Length”, “Optimal Fibre Length”;

- sezione “Objective Function” e scelta tra uno solo dei parametri “minimize torque error”, “objFun2”, “objFun3”;

Il secondo pannello si compone, quindi, dei seguenti widget: una Label per il titolo della prima sezione; sempre nella prima sezione si utilizzano due RadioButton/ToggleButton per le due scelte alternative, che aprono finestre di popup contenenti una serie di TextInput e due Button per la chiusura e il salvataggio della finestra; una Label per il titolo della seconda sezione; due Spinner nella seconda sezione, associati a una Label per il titolo che li descrive; una Label per il titolo della terza e ultima sezione, costituita da tre Button che generano o rimuovono dei Button “Step”, che, a loro volta, generano finestre di popup contenenti una serie di CheckBox e RadioButton e due Button per la chiusura e il salvataggio.
Le soluzioni presenti per la realizzazione dell’interfaccia possono essere più di una, e più widget, secondo certe impostazioni, possono soddisfare una stessa funzione. Le scelte sono state prese tenendo conto del fattore estetico e funzionale. La scelta di introdurre le finestre di popup nel secondo pannello è stata presa per ottenere una visuale più ordinata e per ottimizzare lo spazio disponibile nella disposizione dei widget, oltre che per rendere più intuitivo l’approccio all’utente. Nei pannelli sono stati introdotti anche degli elementi “decorativi”, per un fattore puramente estetico, come, per esempio, suddividere visivamente le sezioni che compongono un pannello. Questi elementi di decorazione vengono creati o tramite canvas o tramite dei widget statici, che permettono di visualizzare un’immagine presa da una risorsa esterna. Se necessario, possono essere aggiunti ulteriori pannelli come tab, e i pulsanti denominati “Step” possono essere aggiunti o rimossi a seconda delle esigenze di configurazione.

CAPITOLO 4

Progettazione con Kivy

Il progetto realizzato con Kivy si compone di due file, uno con estensione .kv, per il design dell'interfaccia grafica, e uno con estensione .py, per la parte di programmazione. In questo capitolo verranno mostrate parti di codice utilizzate per realizzare l'applicazione, insieme a degli screenshot per facilitarne la comprensione del risultato finito.

1.Accenni iniziali

Per prima cosa nel file .py si devono importare i moduli necessari al programma (tra cui quelli riguardanti i widget), che sono delle librerie di codice predefinite. Quindi, nella parte iniziale del file si trova la riga di codice che serve ad importare il modulo per la classe App:

```
from kivy.app import App
```

La classe di base dell'applicazione che si vuole creare è di tipo App e, sempre nel linguaggio py, si definisce questa come segue:

```
class Project1App(App):
    def build(self):
        return Test()
```

Project1App è un nome arbitrario dato alla classe che rappresenta la propria applicazione, mentre tra parentesi la classe da cui Project1App è derivata, che è la classe App. Il metodo "def build(self)" insieme a "return Test()" permette di inizializzare e di dare come output il root widget (o una classe che rappresenta il root widget) dell'applicazione. In questo caso viene restituito come output la classe Test, che contiene tutta la struttura dell'interfaccia realizzata. Sempre nel file .py si trova la parte di codice che mette in esecuzione il programma, con il metodo "run()".

```
if __name__ == '__main__':
    Project1App().run()
```

Ogni volta che nel file .kv si vorrà creare una classe, questa deve essere inizializzata nel file .py come per la classe di base, indicando tra parentesi il nome della classe "madre" da

cui proviene. Una volta inizializzata, all'interno della sua struttura si possono definire i metodi associati ad essa, che vengono utilizzati nel programma. Per esempio, per il secondo pannello sono state create delle classi per delle finestre di popup personalizzate, caratterizzate nel file .kv e definite nel file .py, con i relativi metodi.
La classe Test, definita nel file .py, è derivante dalla classe TabbedPanel e permette di organizzare e raggruppare i pannelli "Execution" e "Calibration" in schede selezionabili. Nel file .kv i tab sono rappresentati come TabbedPanelItem e devono essere aggiunti uno ad uno alla classe Test, mentre, all'interno di essi, si definisce la struttura del pannello che rappresentano. Il primo TabbedPanelItem viene chiamato "Execution", il secondo "Calibration" . Nel linguaggio kv:

```
<Test>:
    pos_hint: {'center_x': .5, 'center_y': .5}
    do_default_tab: False
    TabbedPanelItem:
        text: 'Execution'
    TabbedPanelItem:
        text: 'Calibration'
```

Le proprietà "pos_hint" permettono di stabilire le posizioni dei widget, in questo caso il TabbedPanel, all'interno del layout in cui si trovano. Le proprietà "do_default_tab", se poste a False, impediscono che si generi un tab di default. Nel linguaggio py:

```
class Test(TabbedPanel):
       pass
```

Per ora non è stato definito alcun metodo all'interno di questa classe, quindi si aggiunge l'istruzione "pass", che di per sé non fa nulla, ma è richiesta dalla sintassi.

2.Struttura del pannello "Execution"

All'interno del TabbedPanelItem chiamato "Execution" si trova la struttura del primo pannello, con i relativi widget e layout. Affinché si possa scorrere il pannello verso il basso, si utilizza una ScrollView come layout principale, a cui si impone che lo scrolling avvenga soltanto lungo l'asse y. Tuttavia, all'interno di essa, può essere presente soltanto un widget figlio e la scelta è caduta sul widget GridLayout, che organizza gli oggetti del pannello in un'unica colonna. Con le proprietà "padding" e "spacing" vengono corrispondentemente stabilite la distanza dei widget figli dal bordo del layout che li contiene, e lo spazio che deve intercorrere

tra di essi. Nella struttura GridLayout si introduce come Label il titolo della sezione principale, allineato orizzontalmente a sinistra e con una grandezza del font di 25, in pixel. Nel linguaggio kv si traduce in:

```
TabbedPanelItem:
     text: 'Execution'
     ScrollView:
         do_scroll_x: False
         GridLayout:
             cols: 1
             spacing: 30
             padding: 40
             size_hint: 1 , None
             size: self.minimum_size
             Label:
                 text: 'NMSmodel Configuration'
                 text_size: self.size
                 halign: 'left'
                 font_size: 25
                 size_hint: 1, None
```

Le proprietà “size_hint” indicano quanto spazio può utilizzare il widget lungo l’asse x e y rispetto al widget “padre” che lo include, e si compongono di due valori percentuali decimali che vanno da 0 a 1. Le proprietà “size” definiscono in modo assoluto l’ampiezza e l’altezza del widget con due valori che di default sono posti a 100. Per fare in modo che le dimensioni dei widget varino automaticamente a seconda dello schermo, i valori delle proprietà “size_hint” vengono posti a 1, per x, e None, per annullare il vincolo di altezza in y. Inoltre, affinché lo scrolling verso il basso funzioni correttamente, i GridLayout che contengono un insieme di widget devono avere le proprietà “size” poste a “self.minimum_size”, ovvero la dimensione deve essere la minima necessaria per contenere i widget figli. Ad ogni modo, bisogna evitare di definire le dimensioni con valori assoluti, altrimenti la visualizzazione su schermi piccoli può peggiorare.

Dopo l’inserimento del titolo, si definisce sotto di esso un nuovo GridLayout, che mette in ordine, in due colonne e in cinque righe, le quattro coppie Label-Spinner e la coppia Label-TextInput. Per un puro fattore estetico, si è deciso di utilizzare uno specifico valore di default per l’altezza delle righe della griglia, inserendo tra i parametri del layout le istruzioni “row_default_height: 50” e “row_force_default: True”. Se si vuole posizionare gli oggetti secondo un certo ordine, bisogna tener conto che l’inserimento dei widget in griglia va da

sinistra verso destra e poi dall'alto verso il basso. Prendendo come unico esempio la prima coppia Label-Spinner, all'interno del secondo GridLayout si ha la struttura:

```
Label:
     text: 'NMSmodel type'
     text_size: self.size
     halign: 'left'
     valign: 'middle'
     size_hint: 1,None
     font_size: 18
Spinner:
     text: 'Open Loop'
     values:  'Open Loop', 'Hybrid'
     size_hint: 1,None
```

Le proprietà "text_size" permettono di stabilire le dimensioni del box a cui viene vincolato il testo. Ponendo il valore della proprietà a "self.size", si attiva la possibilità di ordinare il testo all'interno del box, attraverso i parametri halign e valign (in questo caso il testo viene allineato a sinistra, in centro), per una visualizzazione più ordinata. Lo Spinner permette una selezione tra un set di valori, che vengono introdotti mediante l'istruzione "values", e "Open Loop" e "Hybrid" sono i valori selezionabili. Nel linguaggio kv la coppia Label-TextInput si traduce in:

```
Label:
    text: 'Sampling Frequency'
    text_size: self.size
    halign: 'left'
    valign: 'middle'
    font_size: 18
    size_hint: 1,None
TextInput:
    text: '200'
    text_size: self.size
    font_size: 20
    padding_x: 13
    padding_y: 13
    multiline: False
    size_hint: 1,None
```

Le proprietà “multiline” dei TextInput permettono di stabilire se il testo da introdurre può svilupparsi su più righe o no. Con il valore posto a “False”, come in questo caso, al testo è permesso di svilupparsi solo su una riga.

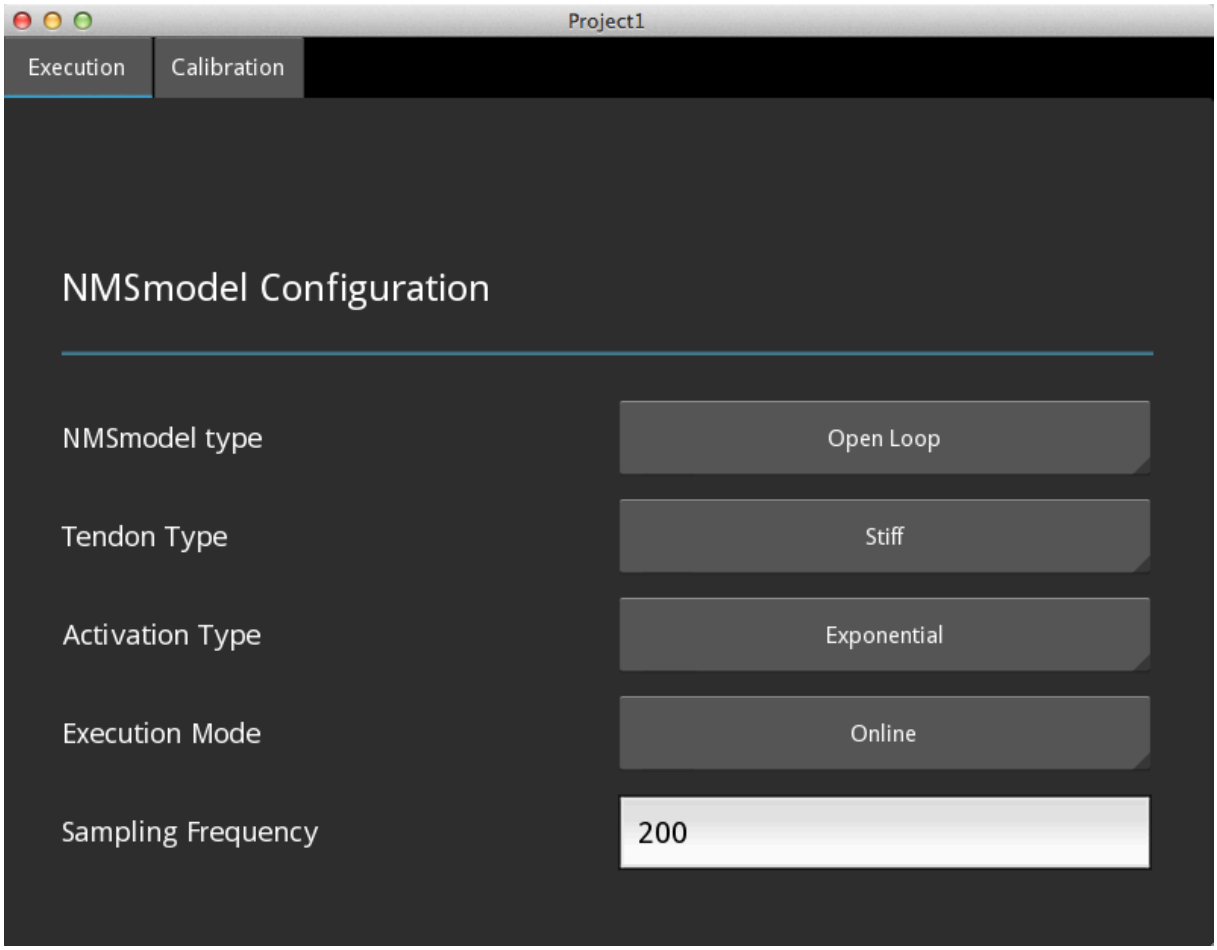

Figura 4.1.*Prima parte del pannello “Execution”, con titolo principale e griglia, contenente cinque Label poste a sinistra, quattro Spinner ed un Textinput, con valore di default “200”, posti a destra.*

Nella seconda parte del pannello, viene introdotta un’altra Label per il titolo di una sottosezione che contiene una serie di CheckBox. Ad ogni CheckBox viene associata una Label che lo descrive, e le coppie ottenute vengono quindi organizzare in una griglia, simile a quella contenente gli Spinner, ma con quattro colonne e tre righe. La Label per il titolo viene aggiunta nella struttura del primo GridLayout del pannello, insieme al GridLayout contenente i CheckBox, subito dopo la griglia contenente gli Spinner e il TextInput. Nel linguaggio kv la coppia Label-CheckBox all’interno del terzo GridLayout si traduce in:

```
Label:
    text: 'R.Knee.FE'
    text_size: self.size
    halign: 'left'
    valign: 'middle'
    size_hint: 1, None
CheckBox:
    size_hint: 1, None
```

Se l'utente vuole selezionare la stringa "R_Knee_FE", gli basterà cliccare sul quadratino del CheckBox, a destra della Label. Alla fine del pannello viene introdotto un BoxLayout, per disporre su una riga orizzontale tre pulsanti Button. Il BoxLayout va inserito all'interno del primo GridLayout, subito dopo la griglia di CheckBox. Nel linguaggio kv:

```
BoxLayout:
     orientation: 'horizontal'
     size_hint: 1, None
     spacing: 15
Button:
     text: 'Save'
     size_hint: 1, .5
Button:
      text: 'Load'
      size_hint: 1, .5
Button:
      text: 'Run'
      size_hint: 1, .5
```

In questo caso non viene definito alcun evento, per cui cliccando su uno dei tre pulsanti non accadrà nulla, in quanto non è presente alcuna funzione Python associato ad esso.
Le proprietà "orientation" dei BoxLayout stabiliscono se i widget figli devono essere disposti verticalmente o orizzontalmente, come in questo caso. I tre Button hanno la stessa dimensione in x rispetto al layout "padre" (size_hint_x: 1), quindi l'intero spazio disponibile del Boxlayout viene suddiviso in modo equo tra di essi.

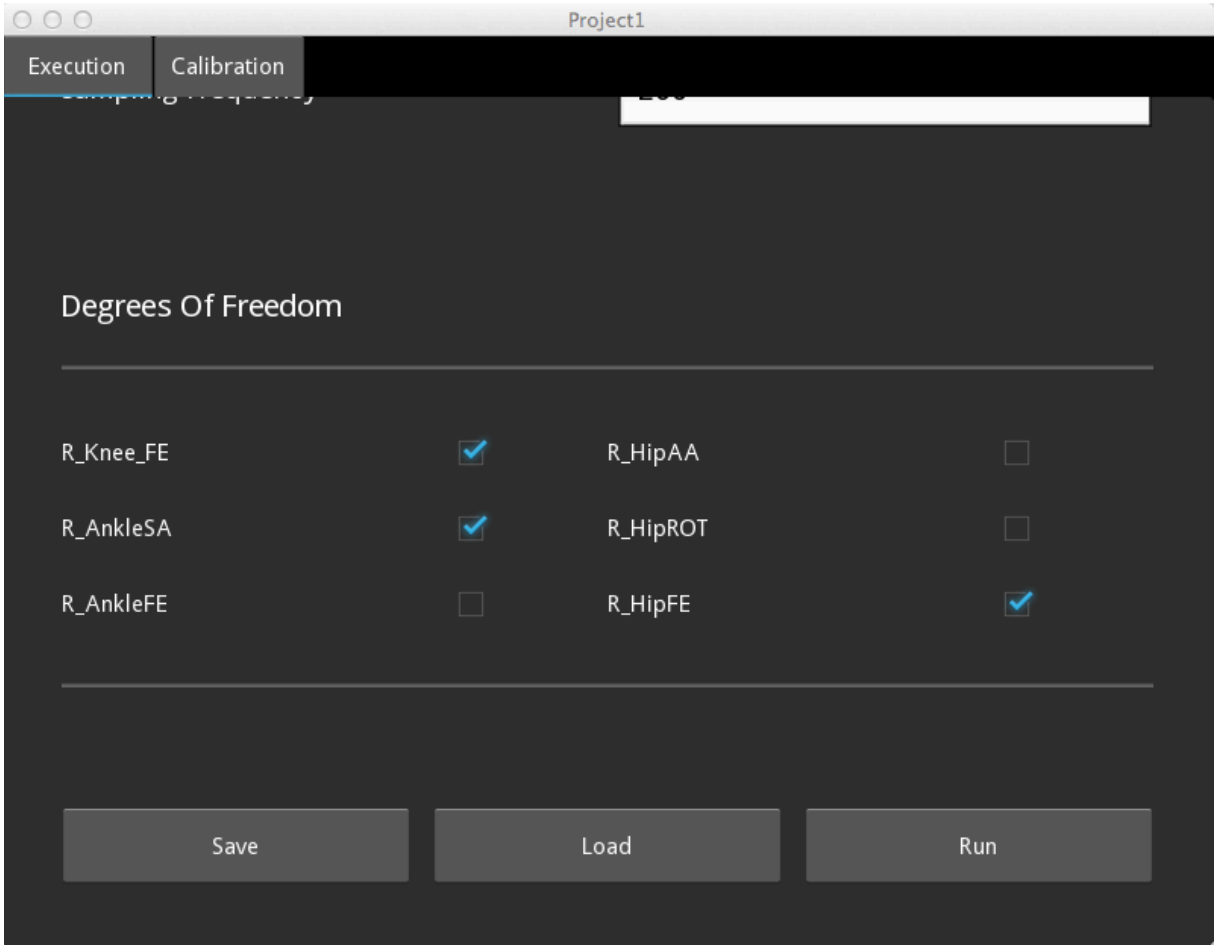

Figura 4.2.*Seconda parte del pannello "Execution", con titolo della sottosezione e griglia di Label-CheckBox selezionabili dall'utente. Alla fine del pannello si trovano i tre pulsanti "Save", "Load", "Run".*

Come si può notare dagli screenshot sono presenti degli elementi decorativi a forma di rette orizzontali colorate, situate una dopo il titolo principale, e le altre due poste a contornare la griglia di CheckBox. Questi elementi vengono creati mediante la realizzazione di un modello personalizzato, detto template. Nel linguaggio kv:

```
[HSeparator@Widget]:
    size_hint_y: .1
    canvas:
        Color:
            rgba: .3, .6, .7, .7
        Rectangle:
            size: self.width, 2.5
            pos: self.pos
```

Tra parentesi quadre troviamo il nome che si è voluto attribuire alla classe per questi separatori grafici, “HSeparator”, e la classe di base da cui si genera, cioè la classe Widget. Il Canvas permette di creare la retta separatrice, realizzando la forma di un rettangolo orizzontale, mediante la proprietà “Rectangle”, e attribuendole un determinato colore mediante l’istruzione “Color” (che necessita di quattro valori che vanno da 0 a 1). E’ stato creato anche un altro separatore grafico, “HSGrey”, simile al primo ma di colore diverso. Questi separatori vengono aggiunti nell’interfaccia grafica come dei normali widget, ossia inserendo nel file.kv, nel punto in cui si vuole che appaiano, la riga di codice “HSeparator:” o “HSGrey:”.

3.Struttura del pannello “Calibration”

Il secondo TabbedPanelItem, chiamato “Calibration”, include anch’esso una ScrollView e, all’interno di essa, un GridLayout ad una colonna come quello del pannello precedente, ma con un valore di spacing diverso. Riguardo le proprietà “size” e “size_hint”, sono state utilizzate le stesse tecniche del primo pannello per permettere ai widget di variare dinamicamente la loro dimensione a seconda dello schermo in cui vengono visualizzati. Come primi elementi, all’interno della griglia ad una colonna, troviamo il titolo della prima sezione del pannello e l’elemento HSeparator. La prima sezione è costituita da un BoxLayout con orientazione orizzontale, in cui sono organizzati due ToggleButton per la scelta univoca tra due alternative.

```
BoxLayout:
    orientation: 'horizontal'
    spacing: 20
    size_hint: 1, None
    ToggleButton:
      text: 'Simulated Annealing'
      size_hint: 1, 1
      group: "choose"
      on_press: root.show_popup1()
    ToggleButton:
      text: 'Particle Swarm'
      size_hint: 1, 1
      group: "choose"
      on_press: root.show_popup2()
```

Le proprietà “group” permettono di definire un gruppo di appartenenza per i ToggleButton, in modo che la scelta possibile per l’utente sia unica tra un insieme di opzioni.

In questo caso, alla pressione di uno dei due pulsanti ToggleButton, viene definito un evento che permette di generare una finestra di popup, mediante un metodo associato ad esso. Prendendo come esempio solo il primo pulsante, l'evento associato ad esso viene chiamato tramite la riga di codice "on_press: root.show_popup1()". Questo metodo, però, deve essere definito all'interno della classe in cui si genera, nel file .py. Quindi, all'interno della classe Test troviamo il metodo "show_popup1()", che indica al programma di aprire la finestra di popup assegnata all'elemento "p".

```
class Test(TabbedPanel):
    def show_popup1(self):
        p = CustomPopup1()
        p.open()
    def show_popup2(self):
        g = CustomPopup2()
        g.open()
```

Lo stesso avviene con il secondo ToggleButton, il quale, però, deve aprire la finestra di popup assegnata all'elemento "g", tramite il metodo "show_popup2()". Per realizzare il primo popup è stata creata nel file .kv una classe chiamata "CustomPopup1", che include una griglia di Label-TextInput per l'inserimento di valori da parte dell'utente. La classe deve poi essere definita nel file .py, e lo stesso procedimento si ripete per la classe del secondo popup, chiamata "CustomPopup2". Nel linguaggio kv:

```
<CustomPopup1>:
    size_hint:.6, .6
    auto_dismiss: False
    title: 'Simulated Annealing'
    BoxLayout:
        orientation: 'vertical'
        padding: 20
        GridLayout:
            cols: 2
            rows: 7
            row_default_height: 30
            row_force_default: True
            Label:
                text: 'noEpsilon'
                text_size: self.size
                halign: 'left'
```

```
            TextInput:
                text: '4'
                multiline: False
                .......
                .......
```

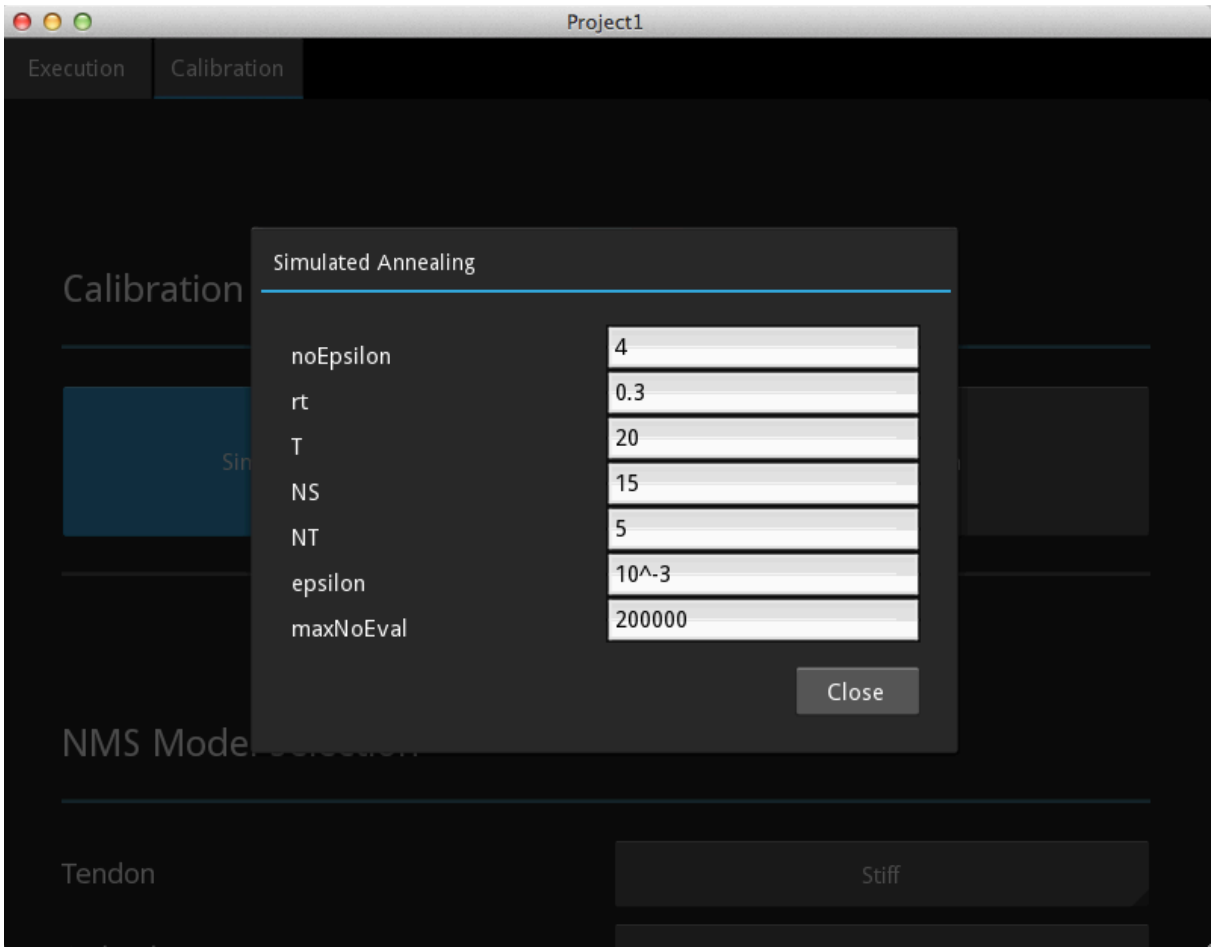

Figura 4.3.*Struttura della prima finestra di popup, costituita da sette coppie Label-TextInput, che si genera quando viene selezionata la scelta "Simulated Annealing".*

```
<CustomPopup2>:
    size_hint: .6, .6
    auto_dismiss: False
    title: 'Particle Swarm'
    BoxLayout:
        orientation: 'vertical'
        padding: 20
        GridLayout:
            cols: 2
```

```
rows: 3
row_default_height: 30
row_force_default: True
Label:
    text: 'paramA'
    text_size: self.size
    halign: 'left'
TextInput:
    text: '1'
    multiline: False
    .......
    .......
```

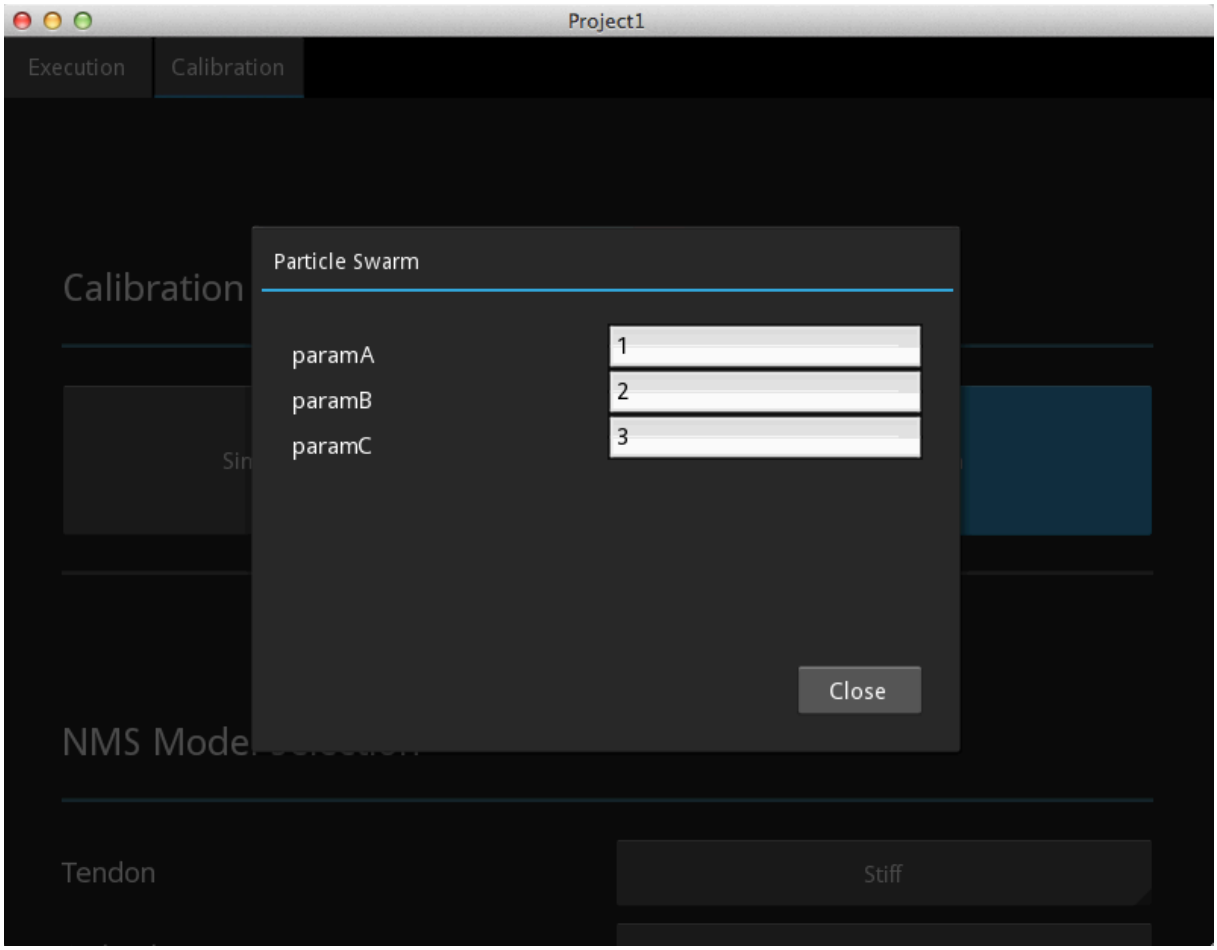

Figura 4.4.*Struttura della seconda finestra di popup, costituita da tre coppie Label-TextInput, che si genera quando viene selezionata la scelta "Particle Swarm".*

Per evitare che la finestra di popup si chiuda automaticamente se si preme in un punto qualunque al di fuori di essa, bisogna porre la sua proprietà "auto_dismiss" al valore "False".

La chiusura della finestra viene affidata ad un pulsante “Close”, posto all’interno di un AnchorLayout, subito dopo la griglia di Label-TextInput dei due popup.
Nel linguaggio kv:

```
AnchorLayout:
            anchor_x: 'right'
            anchor_y: 'bottom'
            Button:
                size_hint: .2, .25
                text: 'Close'
                on_press: root.dismiss()
```

Premendo sul tasto “Close”, il metodo “dismiss()” associato all’evento chiude la finestra di popup. Le classi “CustomPopup1” e “CustomPopup2” devono poi essere definite nel file .py:

```
class CustomPopup1(Popup):
    pass
class CustomPopup2(Popup):
    pass
```

Dopo aver introdotto il BoxLayout, contenente i ToggleButton, viene aggiunta una Label assieme ad un HSeparator per il titolo della seconda sezione del pannello, costituita semplicemente da una griglia contenente due coppie Label-Spinner.

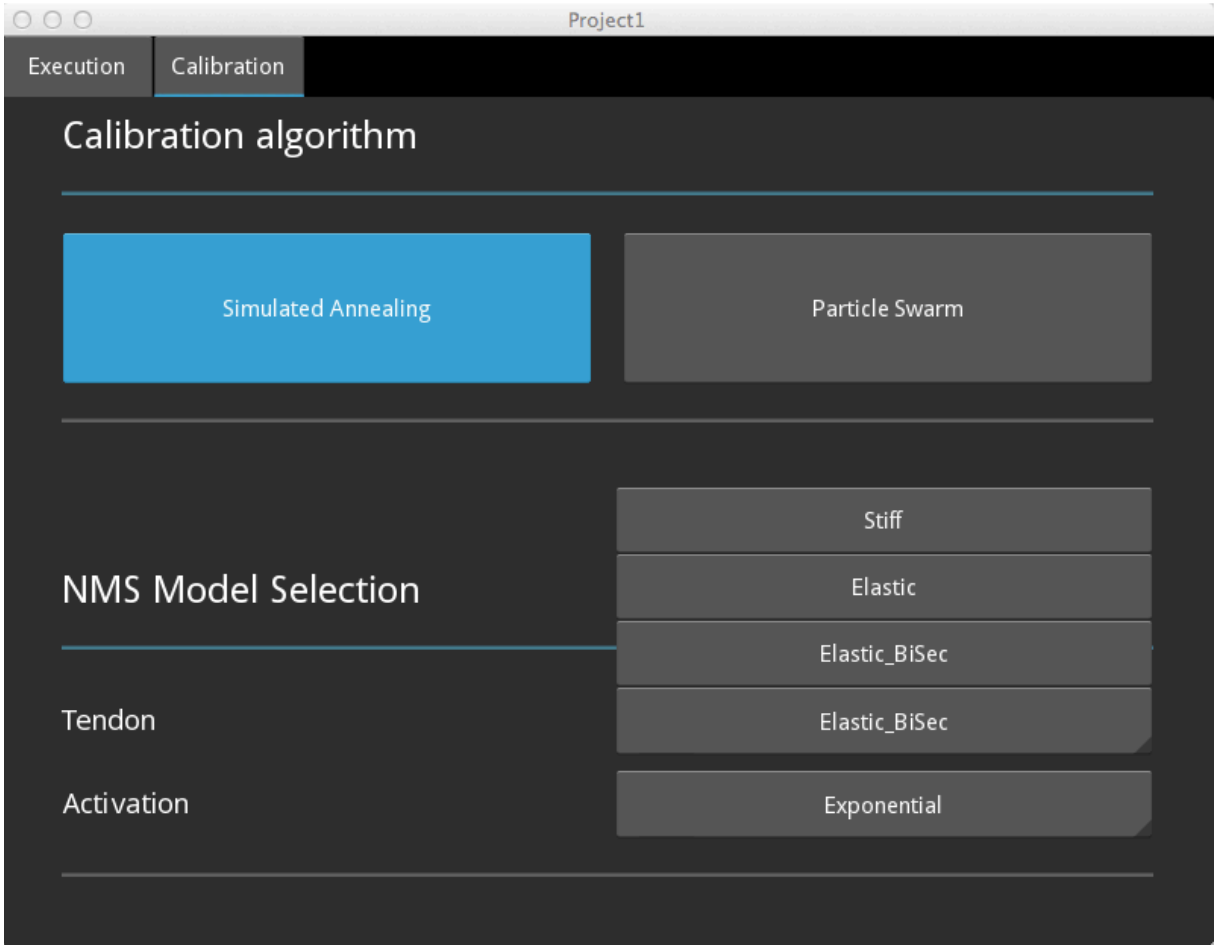

Figura 4.5.*Prima parte del pannello "Calibration", con due ToggleButton nella prima sezione e con due Spinner nella seconda. In questo caso è stata selezionata l'alternativa "Simulated Annealing" ed è stata aperta la finestra del primo Spinner, da cui è possibile visualizzare i tre valori selezionabili. Nel secondo Spinner è stato selezionato il valore "Exponential".*

Nella terza sezione del pannello troviamo una Label associata ad un HSeparator e, sotto di essa, sempre all'interno della struttura del primo GridLayout del pannello, è presente un BoxLayout che dispone in fila, orizzontalmente, tre Button. Sotto il BoxLayout, invece, è stato introdotto un GridLayout per disporre in due colonne i Button "Step" che vengono generati dinamicamente. Il primo Button disposto in fila è stato denominato "Add" e, se premuto, ha lo scopo di aggiungere un Button "Step" nel GridLayout. Il metodo associato ad esso è "load_content()", definito all'interno della classe Test. Nel linguaggio py:

```
def load_content(self):
    b=Btn()
    self.box.add_widget(b)
    lst.append(b)
```

In questo modo viene aggiunto un oggetto Btn, assegnato all'elemento "b", al GridLayout identificato con il nome "box", tramite la funzione "self.box.add_widget(b)". Inoltre, l'elemento "b" appena creato viene aggiunto ad una lista, dichiarata all'interno della classe Test mediante le righe di codice:

```
lst=[]
global lst
```

Inserire gli oggetti in lista permetterà di facilitare la rimozione di questi dal layout in cui vengono aggiunti, cioè si potrà rimuovere gli oggetti "Step" uno per uno, a partire dall'ultimo creato. Il programma, infatti, rimuoverà sempre l'oggetto che verrà identificato come l'ultimo inserito in lista, che è anche l'ultimo creato.
Il secondo Button, denominato "Remove", è quello che ha lo scopo di rimuovere uno per uno gli elementi "Step" che sono stati aggiunti al layout. Il metodo associato ad esso è "cancel_content()", definito all'interno della classe Test. Nel linguaggio py:

```
def cancel_content(self):
    if len(lst)>0 :
        self.box.remove_widget(lst[len(lst)-1])
        del lst[len(lst)-1]
```

L'oggetto "Step" viene rimosso solo se è presente nel GridLayout, cioè solo se la lunghezza della lista, dovuta al numero degli elementi che la compongono (in questo caso gli "Step" creati), è maggiore di zero. Se non c'è alcun elemento in lista significa che non è stato creato alcuno "Step" o che sono già stati rimossi tutti, quindi non c'è bisogno di rimuoverne nessun altro, quindi, se per caso l'utente preme il tasto "Remove", non accadrà nulla.
L'ultimo Button, denominato "Remove all", ha lo scopo di rimuovere in un solo colpo tutti gli oggetti "Step" che sono stati aggiunti al GridLayout "box". Il metodo associato ad esso è "cancel_all()", definito all'interno della classe Test. Nel linguaggio py:

```
def cancel_all(self):
    self.box.clear_widgets()
```

Nel linguaggio kv vengono chiamati i metodi di ogni Button e viene identificato come "box", mediante la proprietà "id", il GridLayout di interesse. L'elemento "box" deve essere dichiarato nel file .kv all'interno della classe Test, ma al di fuori dei TabbedPanelItem.

```
BoxLayout:
    orientation: 'horizontal'
    spacing: 15
    size_hint: 1, None
    size: 1, 50
    Button:
        text: 'Add'
        size_hint: 1, 1
        on_press: root.load_content()
    Button:
        text: 'Remove'
        size_hint: 1, 1
        on_press: root.cancel_content()
    Button:
        text: 'Remove all'
        size_hint: 1, 1
        on_press: root.cancel_all()
GridLayout:
    id: box
    cols:2
    size_hint: 1, None
    size: self.minimum_size
```

L'elemento "Step" viene definito mediante la creazione di una classe Btn, che assicurerà che l'oggetto, chiamato appunto "Step", sia un elemento Button in grado di aprire una finestra di popup. Nel linguaggio kv:

```
<Btn>:
    text: 'Step'
    size_hint: 1, None
    size: 1, 50
    on_press: self.show_popup3()
```

La classe Btn viene definita nel file .py, assieme al metodo "show_popup3()", che permette all'elemento "Step" di aprire la terza finestra di popup del pannello, ovvero la classe CustomPopup3. Tale classe, come per tutte le altre classi, deve anch'essa essere definita nel file .py.

```
class Btn(Button):
    def show_popup3(self):
        r = CustomPopup3()
```

```
        r.open()
class CustomPopup3(Popup):
    pass
```

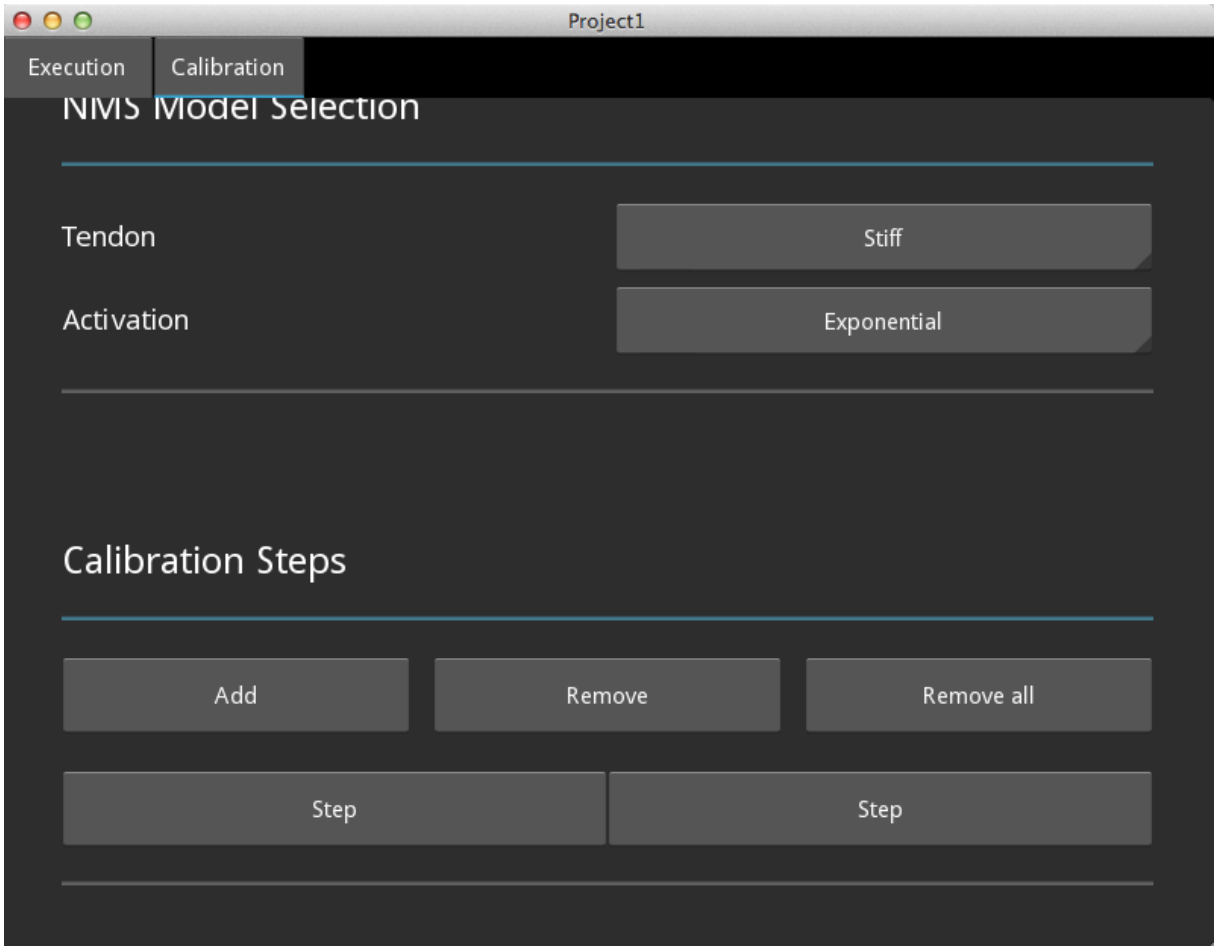

Figura 4.6.*Seconda parte del pannello "Calibration", in cui è presente la terza sezione "Calibration Steps" contenente tre Button in fila, per aggiungere e rimuovere dinamicamente gli elementi "Step" di tipo Button. In questo caso, attraverso il pulsante "Add", due di questi elementi sono stati aggiunti al layout.*

La terza finestra di popup, generata premendo il pulsante "Step", viene creata nel file .kv attraverso la classe personalizzata CustomPopup3. I widget contenuti in essa sono organizzati in un BoxLayout come quello dei popup precedenti. La finestra è divisa in tre sezioni, due costituite da una Label per il titolo e da una griglia per contenere sei coppie Label-CheckBox, e una costituita da una Label per il titolo e da una griglia per contenere tre coppie Label-RadioButton (che sono sempre dei CheckBox, ma appartenenti ad un gruppo). Anche in questo caso, sotto i tre layout a griglia, è presente un AnchorLayout, che posiziona nella finestra in basso a destra un pulsante "Close" per chiudere il popup.

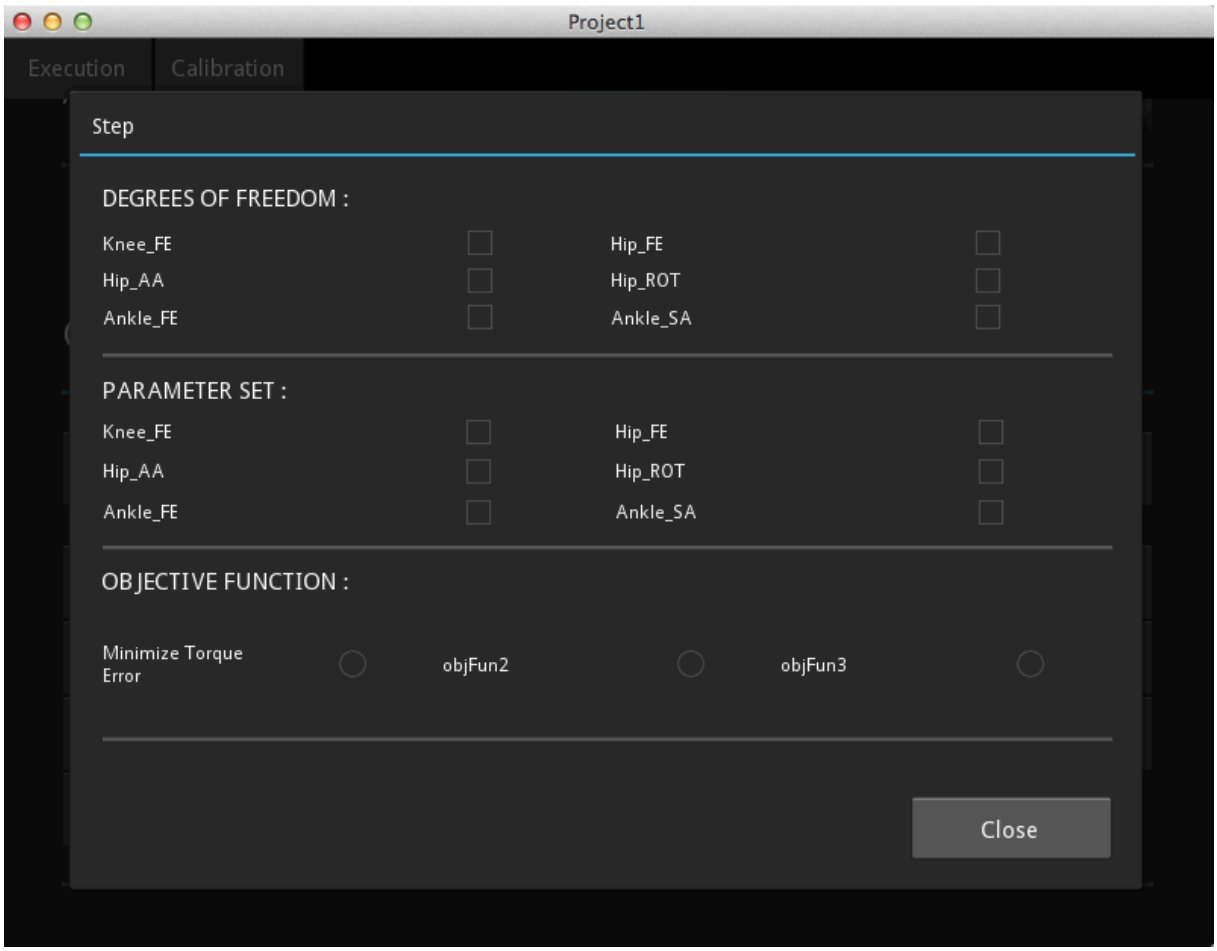

Figura 4.7.*Struttura della terza finestra di popup, denominata "Step", contenente tre sezioni per la selezione di parametri da parte dell'utente.*

Ogni oggetto "Step" apre la stessa struttura di popup, definita dalla classe CustomPopup3, ma in questo modo l'utente ha la possibilità di avere più "Step" contemporaneamente, che possono presentare delle selezioni diverse in ognuno.

CAPITOLO 5

Progettazione con Android SDK

Questo capitolo descrive la realizzazione dell'interfaccia grafica mediante l'utilizzo degli strumenti forniti da Android SDK. Come è stato descritto in precedenza, un Android Project si compone di vari file che riguardano, per esempio, la programmazione, i layout, le risorse utilizzate. Tutti questi file, organizzati in cartelle e sotto cartelle, competono al funzionamento dell'applicazione.

1.Accenni iniziali

Nel file AndroidManifest.xml, presente nell'Android Project creato, viene riportato l'API level minimo affinché l'applicazione possa funzionare, oltre che l'API level di target con cui si vuole che l'applicazione funzioni correttamente. L'API level è un valore intero che identifica la revisione della struttura API offerta da una versione della piattaforma Android, necessaria alla realizzazione dell'applicazione. Il sistema operativo Android eviterà di installare l'applicazione se l'API level del sistema su cui deve essere installata è inferiore a quello minimo definito nell'AndroidManifest. Come versione minima si è posto il valore pari a "14", per la versione target lo si è posto pari a "18", che corrisponde all'ultima versione disponibile attualmente. Comunque, l'applicazione potrà funzionare anche nelle versioni più vecchie, sempre che queste abbiano un API level maggiore o uguale a "14".

```
<uses-sdk android:minSdkVersion="14"
          android:targetSdkVersion="18" />
```

L'Activity dell'applicazione è costituita dal file MainActivity.java, in cui viene realizzato il widget (in Kivy chiamato "TabbedPanel"), contenente i tab "Execution" e "Calibration". All'interno di essa viene strutturato un TabHost che funge da "contenitore" per i tab desiderati, i quali vengono associati ad una Label su cui l'utente può cliccare per selezionarli, e ad un layout che permette di visualizzare i loro contenuti. Inoltre, per ogni tab, viene creata un classe che include un Activity, in cui si impone che il contenuto del tab venga definito da un file .xml presente nella cartella layout. Quindi, al tab "Execution" si associano un file activity_execution_tab.xml ed un file ExecutionTab.java, mentre al tab "Calibration" si associa un file activity_calibration_tab.xml ed un file CalibrationTab.java.
Considerando il file ExecutionTab.java, all'interno di esso si trovano le seguenti righe di codice:

```
public class ExecutionTab extends Activity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_execution_tab);
      }
```

Il metodo “onCreate()” è presente per ogni Activity e viene chiamato quando questa viene inizialmente creata. All’interno di esso si compiono varie azioni di base, come la creazione delle View o widget, utilizzate per creare componenti interattive dell’interfaccia utente, e la definizione dei metodi associati ad esse. L’Activity, inoltre, si preoccupa di creare una finestra in cui posizionare l’UI creata e, quindi, all’interno del metodo “onCreate()” troviamo anche l’istruzione “setContentView(View)”, che, come si può vedere dalla quinta riga, genera una finestra per la quale il contenuto è definito dal file activity_execution_tab.xml, presente all’interno della cartella layout. La stessa cosa si ottiene all’interno del file CalibrationTab.java, ma, in questo caso, nel metodo “onCreate()” si trova l’istruzione “setContentView(R.layout.activity_calibration_tab)”, che considera il contenuto del file activity_calibration_tab.xml, sempre posizionato nella cartella layout.

2.Struttura del pannello “Execution”

Il contenuto del pannello “Execution” viene inizialmente organizzato in un RelativeLayout, che viene collegato al file ExecutionTab.java mediante l’istruzione “tools:context=.CalibrationTab", il quale permette appunto di accedere a specifiche classi dell’applicazione. All’interno del RelativeLayout viene posizionata una ScrollView che permette di scorrere in su e in giù il pannello. All’interno di essa può essere presente un solo widget figlio e, quindi, si è optato per un LinearLayout che posiziona i restanti widget uno dopo l’altro, in verticale. Nel linguaggio xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ExecutionTab" >
    <ScrollView
```

```
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbarAlwaysDrawVerticalTrack="true" >
        <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical" >
```

Le proprietà “layout_width” e “layout_height” permettono di stabilire la grandezza del layout rispetto al widget “padre” che lo contiene. Con il valore “match_parent” si vuole che lo spazio necessario al layout copra tutta la grandezza del widget “padre”, con il vantaggio che la dimensione non risulti statica, come accadrebbe inserendo un valore assoluto, in pixel. Le proprietà “orientation” dei LinearLayout stabiliscono se i widget devono essere organizzati verticalmente od orizzontalmente. Le proprietà “padding” permettono di stabilire la posizione del widget in alto, in basso, a destra o a sinistra, rispetto al widget “padre”, con un valore in pixel. All’interno del LinearLayout viene posto una TextView per il titolo della sezione principale, equivalente ad una Label. La sezione principale contiene una serie di LinearLayout che organizzano in orizzontale i tre widget “ImageView”, “TextView” e “Spinner”, dove il primo permette di visualizzare una qualunque immagine caricata da varie risorse, mentre l’ultimo è l’equivalente dello Spinner in Kivy. Nel linguaggio xml:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="35dp"
    android:layout_marginBottom="15dp"
    android:layout_marginTop="15dp"
    android:orientation="horizontal" >
    <ImageView
        android:id="@+id/imageView1”
        android:layout_width="0dp"
        android:layout_weight=".12"
        android:adjustViewBounds="true"
        android:src="@android:drawable/alert_dark_frame"
        tools:ignore="ContentDescription" />
    <TextView
        android:id="@+id/textView1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_marginLeft="22dp"
```

```
            android:layout_weight="1.5"
            android:gravity="center_vertical"
            android:singleLine="true"
            android:text="@string/nmsmodel_type"
            android:textColorHint="#778899" />
        <Spinner
            android:id="@+id/spinner1"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1.2"
            android:entries="@array/choice1"
            android:paddingLeft="10dp"
            android:paddingTop="1dp" />
</LinearLayout>
```

Il widget ImageView visualizza l'immagine denominata "alert_dark_frame" dalla cartella "drawable" presente nelle risorse di sistema, e viene utilizzato per puro scopo decorativo. Attraverso le proprietà "id" è possibile identificare i vari widget in modo da poterli distinguere uno dall'altro quando è necessario operare con uno specifico. Nel widget TextView la proprietà "singleLine" , se posta a "true", stabilisce che il testo si sviluppi su una sola riga, mentre la proprietà "gravity" stabilisce come deve essere posizionato il testo all'interno del box che lo contiene. Con la proprietà "text" viene definito il testo che si vuole visualizzare e, per fare ciò, dobbiamo inserire una stringa creata nel file strings.xml all'interno della cartella values, con il relativo testo associato. Le opzioni selezionabili offerte dallo Spinner vengono inserite con la proprietà "entries", mediante un array definito nel file strings.xml, a cui vengono associati un nome e gli "item" che lo compongono. Considerando, per esempio, il primo Spinner, l'array denominato "choice1" viene quindi attribuito ad esso con l'istruzione "android:entries="@array/choice1"". Nel file strings.xml si trovano le righe di codice che definiscono l'array "choice1":

```
<string-array name="choice1">
    <item>Open Loop</item>
    <item>Hibrid</item>
</string-array>
```

Per permettere che i widget cambino la loro dimensione orizzontale rispetto allo schermo in cui vengono visualizzati, sono state poste tutte le loro proprietà "layout_width" pari a 0 dp (density-independent pixels), così che la loro dimensione vari a seconda dello spazio disponibile nel widget "padre" che li contiene.

In questo modo è necessario che per ogni widget venga introdotta la proprietà “layout_weight”, che esprime la quantità dello spazio disponibile, in percentuale, che deve essere allocato al widget a discapito degli altri. Al termine di questa serie di widget troviamo un LinearLayout che organizza in orizzontale una ImageView, una TextView ed un EditText, in cui gli input accettati sono di tipo numerico (istruzione “android:inputType="number"”) ed il valore di default presente viene inserito tramite una stringa salvata in strings.xml (istruzione “android:hint="@string/default_value"”). Nel linguaggio xml:

```
<EditText
    android:id="@+id/numInput”
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight=".7"
    android:hint="@string/default_value"
    android:inputType="number"
    android:paddingBottom="3dp"
    android:paddingLeft="11dp"
    android:singleLine="true"
    android:textSize="15sp" />
```

Sotto a ciò e sotto il titolo della sezione principale sono state poste delle ImageView che permettono di visualizzare un separatore grafico simile a quello in Kivy, ovvero una retta grigia orizzontale che ha lo scopo di facilitare la suddivisione visiva tra le varie sezioni di un pannello. L’immagine presa è stata introdotta mediante l’istruzione “android:src="@android:drawable/bottom_bar"”, situata all’interno della struttura di codice che definisce le ImageView.

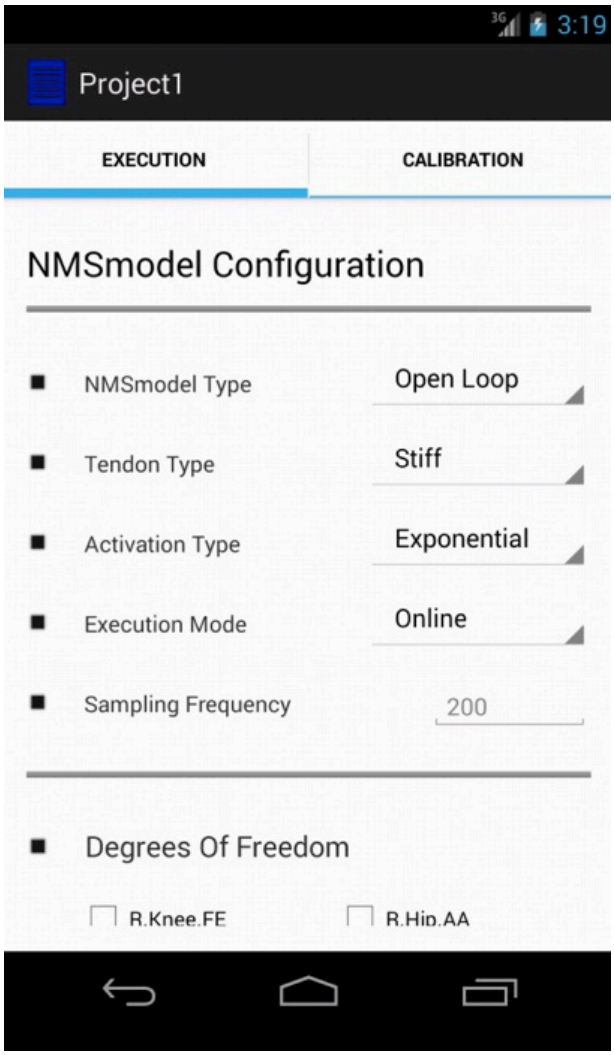

Figura 5.1. *Prima parte del pannello "Execution", intitolata "NMSmodel Configuration", con quattro serie composte dai widget ImageView, TextView e Spinner, ed una serie composta da un EditText al posto dello Spinner, con valore di default "200".*

La seconda parte del pannello si compone di una TextView per il titolo della sotto-sezione e da una serie di LinearLayout che organizzano in orizzontale due CheckBox ciascuno. All'interno del primo LinearLayout della serie si trovano le seguenti righe di codice:

```
<CheckBox
    android:id="@+id/checkBox1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
    android:layout_weight="1"
    android:button="?android:attr/listChoiceIndicatorMultiple"
    android:gravity="center|left"
    android:singleLine="true"
    android:text="@string/r_knee_fe"
    android:textSize="13sp" />
```

```
<CheckBox
    android:id="@+id/checkBox4"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
    android:layout_toEndOf="@id/title2"
    android:layout_weight="1"
    android:button="?android:attr/listChoiceIndicatorMultiple"
    android:singleLine="true"
    android:text="@string/r_hipaa"
    android:textSize="13sp" />
```

Il widget CheckBox ha uno spazio per il testo descrittivo introdotto con la proprietà “text”, mentre l’immagine del Button che indica la selezione si imposta con l’istruzione “android:button="?android:attr/listChoiceIndicatorMultiple"”, che può quindi essere personalizzata. Questo Button è costituito da una casellina che visualizza lo stato selezionato (checked) o non selezionato (unchecked), con la comparsa di un segno di spunta nel primo caso. Nella parte finale del pannello si trova un ulteriore LinearLayout, posto sempre all’interno della struttura della ScrollView, che organizza tre Button su una linea orizzontale. Per ora, questi Button non sono associati ad alcun evento, quindi cliccando su di essi non accadrà nulla. L’istruzione “style="?android:attr/buttonStyleSmall"” stabilisce l’apparenza grafica del Button ed è selezionabile tra varie impostazioni di default.

```
<Button
    android:id="@+id/buttonSave"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/save" />
<Button
    android:id="@+id/buttonLoad"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/load" />
<Button
    android:id="@+id/buttonRun"
    style="?android:attr/buttonStyleSmall"
```

```
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="@string/run" />
```

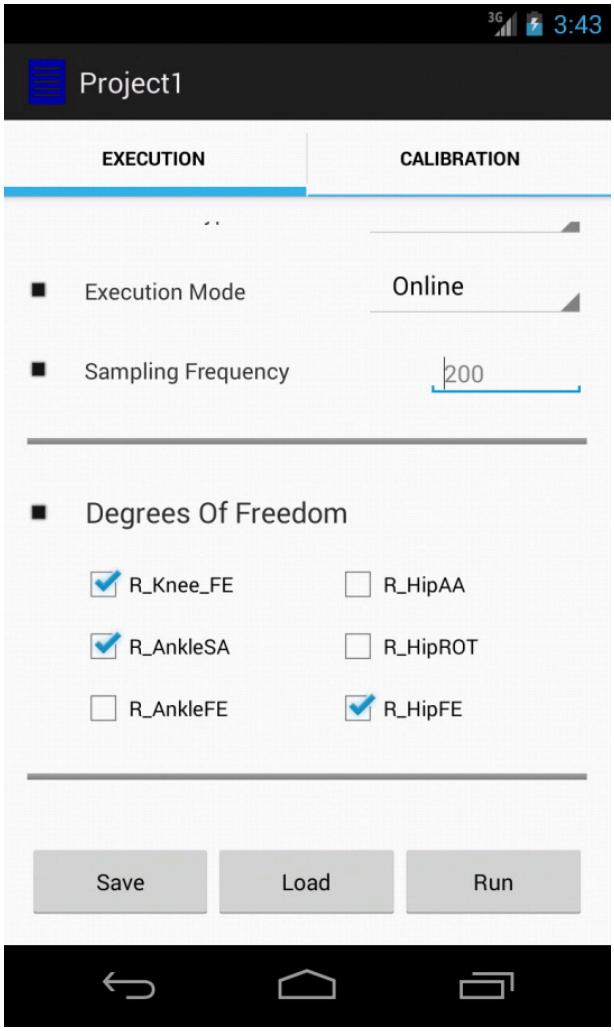

Figura 5.2.*Seconda parte del pannello "Execution", contenente la sotto-sezione chiamata "Degrees Of Freedom", composta da una serie di LinearLayout che contengono due CheckBox ciascuno. Nella parte finale del pannello si trovano i tre Button "Save", "Load", "Run".*

Per ogni LinearLayout la dimensione orizzontale è stata impostata con il valore "match_parent", ovvero deve occupare tutta l'ampiezza del widget "padre", mentre per la dimensione verticale si è impostato o un valore assoluto o il valore "wrap_content", cioè la giusta dimensione necessaria per contenere i widget figli.

3.Struttura del pannello “Calibration”

Anche il contenuto del secondo pannello “Calibration” viene organizzato in un RelativeLayout come quello del pannello precedente, e viene collegato al file CalibrationTab.java mediante l’istruzione “tools:context=".CalibrationTab"”. Il pannello si compone di tre sezioni e all’interno della sua struttura sono presenti una ScrollView ed un LinearLayout con orientazione verticale, uguali a quelli del pannello “Execution”. La prima sezione è costituita da una TextView per il titolo, da un separatore grafico e da due RadioButton. Questi ultimi sono organizzati in un RadioGroup con orientazione verticale e con una larghezza pari a quella del widget “padre”. Per ogni RadioButton viene associato un metodo che permette di visualizzare, al click, una finestra di popup derivante dalla classe Dialog. Nel linguaggio xml:

```
<RadioGroup
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginBottom="15dp"
      android:layout_marginTop="15dp"
      android:orientation="horizontal" >
      <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:singleLine="true"
            android:text="@string/simulated_annealing_"
            android:textOff="Simulated Annealing"
            android:textOn="Simulated Annealing"
            android:textSize="15sp" />
      <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:singleLine="true"
            android:text="@string/particle_swarm"
            android:textOff="Particle Swarm"
            android:textOn="Partcle Swarm"
            android:textSize="15sp" />
</RadioGroup>
```

Prendendo come esempio il primo RadioButton, nel file CalibrationTab.java si deve dichiarare un elemento “button” di tipo Button, che viene associato all’oggetto identificato tramite id come “radioButton1”. All’interno del metodo “onCreate()” si collega il metodo “onClick()” all’evento della pressione del Button, attraverso l’istruzione “setOnClickListener()”. La pressione del Button permette quindi di generare un elemento “dialog” di tipo Dialog, il cui contenuto viene associato al file popup_layout1.xml con il metodo “setContentView(View)”. All’interno della finestra di popup creata è presente un pulsante identificato con “close”, che viene attribuito all’elemento “dialogButton” di tipo Button, e il cui unico scopo è quello di chiudere il popup mediante un metodo “onClick()” contenente l’istruzione “dialog.dismiss()”, definito all’interno della struttura “onClick()” del “button”. Nel linguaggio java:

```
public class CalibrationTab extends Activity {
final Context context = this;
private Button button;
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_calibration_tab);
      button = (Button) findViewById(R.id.radioButton1);
      button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View arg0) {
            final Dialog dialog = new Dialog(context);
            dialog.setContentView(R.layout.popup_layout1);
            dialog.setTitle("Simulated Annealing");
            dialog.setCanceledOnTouchOutside(false);
            Button dialogButton = (Button) dialog.findViewById(R.id.close);
            dialogButton.setOnClickListener(new OnClickListener() {
                  @Override
                  public void onClick(View v) {
                        dialog.dismiss();
                  }
            });
            dialog.show();
        }
      });
}
}
```

L’elemento “button” è stato dichiarato all’interno della classe CalibrationTab, esternamente ai metodi, mentre all’interno del metodo “onCreate()” vengono definiti tutti i metodi associati ai

Button che compongono l'interfaccia utente. Dopo aver creato un nuovo elemento di tipo Dialog, si introduce l'istruzione "dialog.setCanceledOnTouchOutside(false)" per evitare che la finestra di popup si chiuda se si preme in un qualunque punto al di fuori di essa. L'istruzione "dialog.show()", presente nel metodo "onClick()" dell'elemento "button", permette di aprire la finestra di popup appena creata. Si effettua lo stesso procedimento per il secondo RadioButton, identificato come "radioButton2", che apre la finestra di popup associata al layout "popup_layout2". I layout dei popup vengono definiti con specifici file .xml creati all'interno della cartella layout. Per il primo popup, intitolato "Simulated Annealing", viene creato il file popup_layout1.xml, contenente una serie di LinearLayout che organizzano in orizzontale due widget ciascuno, cioè un "EditText" ed una "TextView". Per il secondo popup, intitolato "Particle Swarm", viene creato il file popup_layout2.xml, contenente tre LinearLayout che organizzano in orizzontale due widget ciascuno, che sono sempre un "EditText" ed una "TextView". Le strutture delle finestre di popup create contengono anche delle ScrollView per poterle scorrere, e dei pulsanti "Close", posti in basso, per la chiusura.

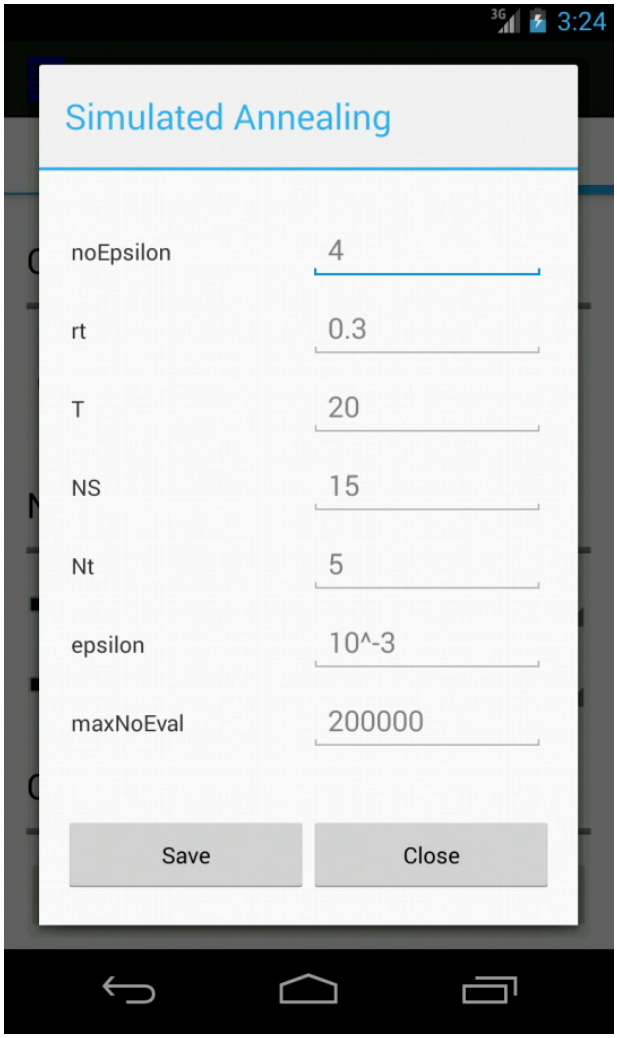

Figura 4.3.*Struttura della prima finestra di popup, costituita da sette coppie Label-TextInput, che si genera quando viene selezionata la scelta "Simulated Annealing".*

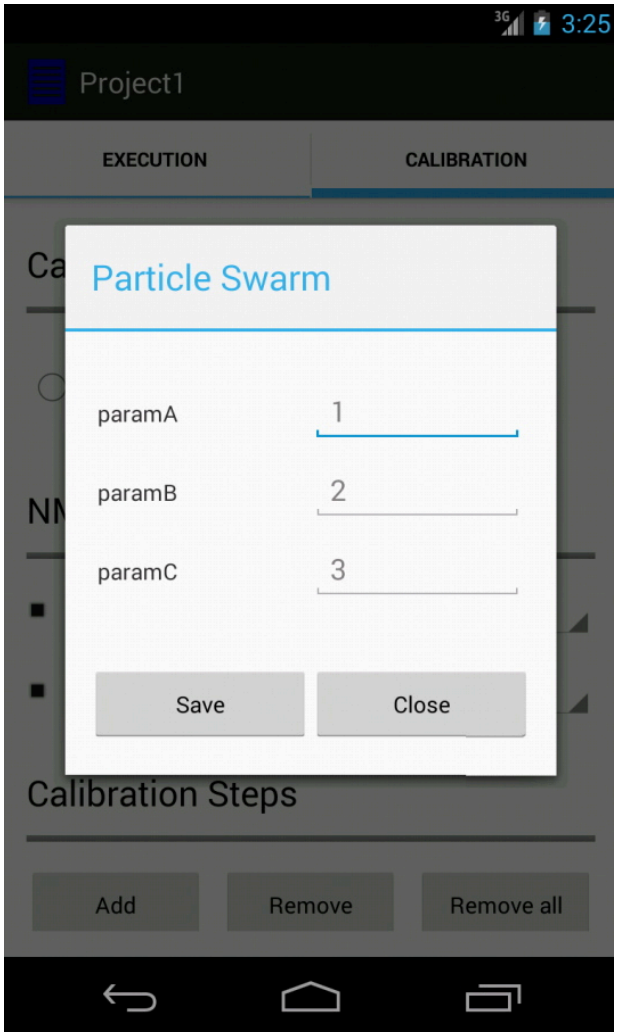

Figura 4.4.*Struttura della seconda finestra di popup, costituita da tre coppie Label-TextInput, che si genera quando viene selezionata la scelta "Particle Swarm".*

Nelle finestre di popup è stato introdotto anche un pulsante "Save", che ha lo scopo di salvare gli input introdotti dall'utente, definendo un metodo all'interno della struttura "onClick()" degli elementi Button che generano i popup.

La seconda sezione del pannello, invece, si compone di una TextView per il titolo, di un separatore grafico e di due LinearLayout che organizzano orizzontalmente, come nel pannello “Execution”, la terna di widget ImageView, TextView e Spinner.

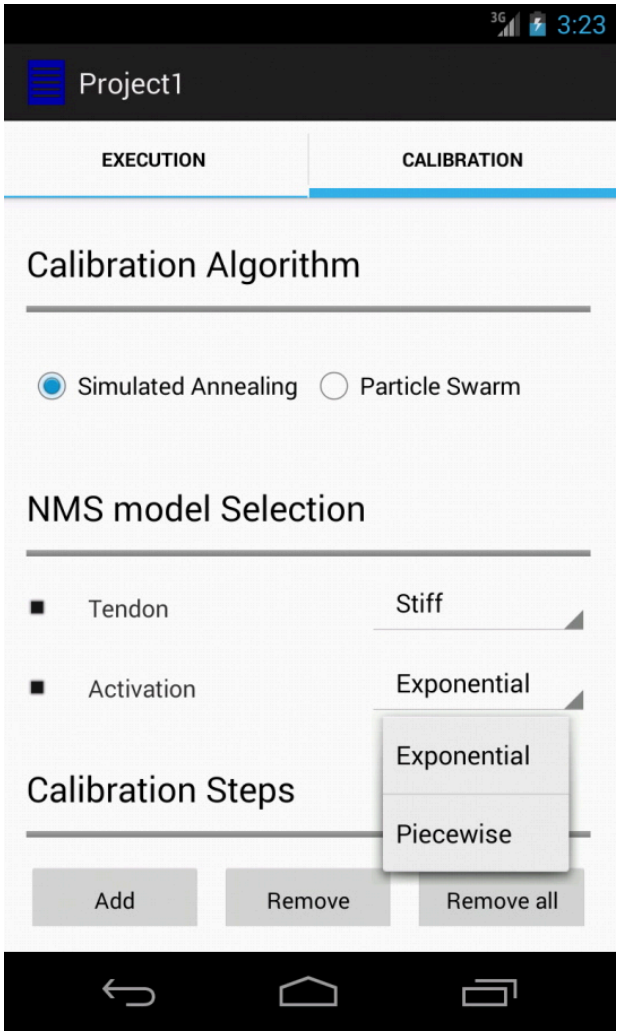

Figura 4.5.*Prima parte del pannello “Calibration”, con due RadioButton nella prima sezione, e con due Spinner nella seconda. E’ stata selezionata l’alternativa “Simulated Annealing” ed è stata aperta la finestra del secondo Spinner, da cui è possibile visualizzare i due valori selezionabili. Nel primo Spinner è stato selezionato il valore “Stiff”.*

La terza sezione del pannello si compone di una TextView per il titolo, di un separatore grafico e di un LinearLayout che organizza in orizzontale tre Button, i cui metodi vengono chiamati e definiti nel linguaggio java. Al di sotto di quest’ultimo è stato introdotto un ulteriore LinearLayout, identificato con l’id “lay”, per organizzare verticalmente gli elementi “Step” di tipo Button che vengono generati dinamicamente. Il Button “Add” viene definito nel file CalibrationTab.java allo stesso modo dei RadioButton, e viene associato ad esso un metodo che permette di generare un elemento “Step”. All’interno del metodo “onCreate()” sono presenti, infatti, le seguenti righe di codice:

```
button3.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            LinearLayout layout = (LinearLayout) findViewById(R.id.lay);
            layout.addView(button4());
            i= i+1;
      }
});
```

L'elemento "Step" viene associato ad un "id" pari alla variabile "i", a cui viene attribuito un numero intero con valore iniziale 0. Quando nel LinearLayout "lay" viene introdotto un elemento "Step" mediante l'istruzione "layout.addView(button4())", la variabile "i" viene incrementata di uno tramite l'istruzione "i= i+1". In questo modo si tiene traccia dei vari "Step" creati, in modo che, quando dovranno essere rimossi da "lay", si riesca sempre ad individuare ed a eliminare l'ultimo elemento "Step" aggiunto. Il secondo Button, a differenza del Button "Add", ha lo scopo di rimuovere dal layout "lay" gli elementi "Step". All'interno del metodo "onCreate()", quindi, si trovano le seguenti righe di codice:

```
button5.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            ViewGroup layout2 = (ViewGroup) findViewById(R.id.lay);
            if (i>0) {
                View btn = layout2.findViewById(i-1);
                layout2.removeView(btn);
                i= i-1;
            }
      }
});
```

L'istruzione if è necessaria per far sì che venga rimosso uno "Step" solo se questo è effettivamente presente nel layout "lay", altrimenti con l'id pari a 0 non viene rimosso nulla, in quanto significa che "lay" non contiene alcun elemento. Dopo aver rimosso uno "Step" mediante l'istruzione "layout2.removeView(btn)", la variabile "i" viene decrementata di uno e il processo si ferma quando essa sarà pari a 0. L'elemento "btn" non è altro che l'ultimo elemento "Step" aggiunto, che è ancora presente nel layout "lay". A differenza di Kivy gli elementi "Step" non vengono aggiunti ad una lista, ma, per rimuoverli correttamente, si è deciso di associare ad essi un id che viene modificato a seconda delle azioni compiute, ottenendo lo stesso risultato. L'ultimo Button, chiamato "Remove all", ha lo scopo di

rimuovere in un solo colpo tutti gli “Step” aggiunti al layout “lay”, mediante l’istruzione “layout2.removeAllViews()”, e pone quindi la variabile “i” pari a 0. Per fare ciò, all’interno del metodo “onCreate()” troviamo le seguenti righe di codice:

```
button6.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
            ViewGroup layout2 = (ViewGroup) findViewById(R.id.lay);
            layout2.removeAllViews();
            i=0;
      }
});
```

L’elemento “Step” viene definito mediante la dichiarazione della classe button4(), da cui è derivato. All’interno di essa, infatti, viene creato un elemento “item” di tipo Button chiamato appunto “Step”, per il quale vengono definiti i parametri di ampiezza ed altezza, e l’id pari al valore di “i”. Nel linguaggio java, all’interno della classe CalibrationTab abbiamo:

```
private Button button4() {
      final Button item = new Button(this);
      item.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
      LayoutParams.WRAP_CONTENT));
      item.setText("Step");
      item.setId(i);
}
```

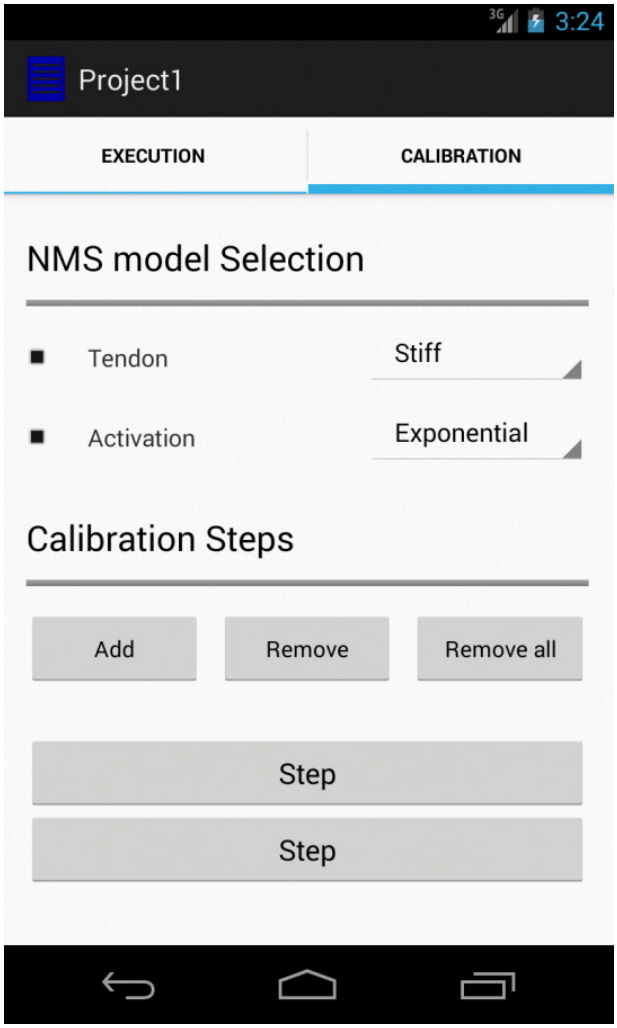

Figura 4.6.*Seconda parte del pannello "Calibration", in cui è presente la terza sezione "Calibration Steps" contenente tre Button in fila, per aggiungere e rimuovere dinamicamente gli elementi "Step" di tipo Button. In questo caso, attraverso il pulsante "Add", due di questi elementi sono stati aggiunti al layout.*

A questo elemento di tipo Button viene anche associato un metodo che permette di aprire la terza finestra di popup del pannello, il cui layout è definito dal file popup_layout3.xml. La modalità con cui lo "Step" può aprire il popup mediante il metodo "onClick()" è simile a quella descritta per i RadioButton, con la differenza che questo metodo viene chiamato e definito all'interno della classe button4(), mediante l'istruzione "item.setOnClickListener(new OnClickListener())". Dopo avere definito il metodo, il sistema restituisce l'elemento che è stato definito nella classe con l'istruzione "return item", in modo da poter essere utilizzato nell'interfaccia. La struttura della terza finestra di popup si compone di tre sezioni e viene organizzata anch'essa in una ScrollView. Le prime due sezioni contengono una TextView per il titolo e una serie di LinearLayout che organizzano in orizzontale due CheckBox ciascuno; l'ultima sezione si compone anch'essa di una TextView per il titolo, ma presenta invece un RadioGroup che organizza orizzontalmente tre RadioButton. Alla fine della finestra troviamo i Button "Save" e "Close", i cui metodi vengono definiti all'interno del metodo "onClick()" dell'elemento "Step".

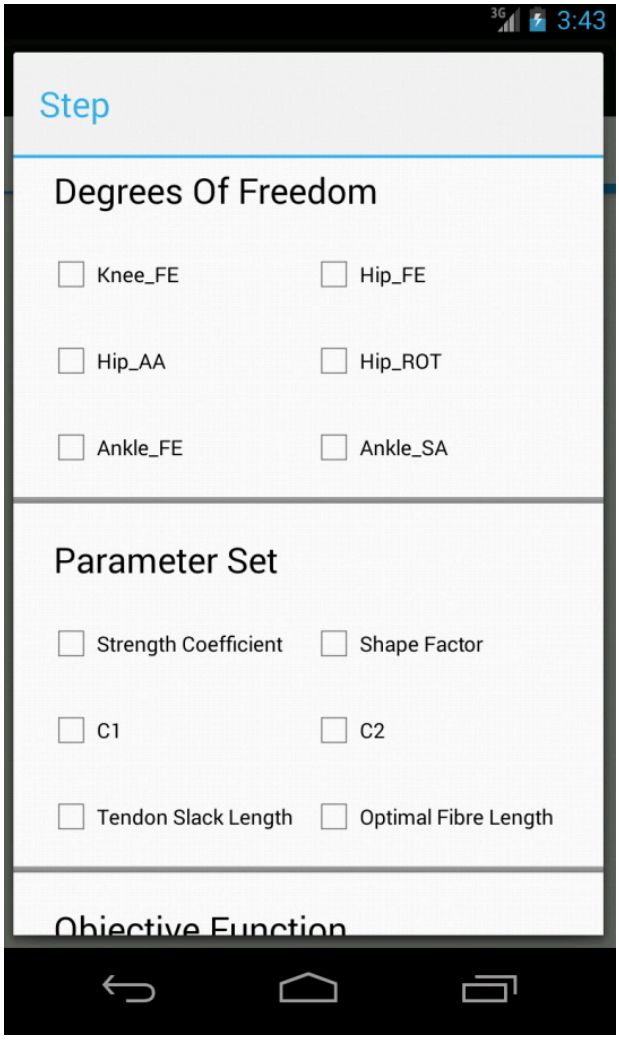

Figura 4.7. *Struttura della terza finestra di popup, denominata "Step", contenente tre sezioni per la selezione di parametri da parte dell'utente. In questo caso sono visibili solo le due sezioni contenenti i CheckBox, mentre la terza sezione è nascosta a causa della dimensione dello schermo (è quindi necessario scorrere il pannello verso il basso).*

Tutte le immagini inserite fino ad ora riguardano l'interfaccia grafica che viene visualizzata in uno schermo simile a quello del dispositivo Android Nexus 4. La visualizzazione, infatti, può variare a seconda delle dimensioni dello schermo e può capitare che, visualizzando il progetto su un tablet, non sia nemmeno necessario sfruttare la ScrollView per poter visualizzare interamente il pannello.

CONCLUSIONI

Entrambi gli strumenti utilizzati hanno portato alla realizzazione di un'interfaccia grafica sostanzialmente simile, che permette di avere le stesse funzionalità nonostante siano stati utilizzati mezzi diversi per la progettazione. In particolare, l'applicazione realizzata è funzionante su vari dispositivi con sistema operativo Android, permettendo di compiere le azioni richieste per l'inserimento di parametri da parte dell'utente, che andrà ad utilizzarla sfruttandone la sua interfaccia. Senza possedere necessariamente un dispositivo Android, la bontà della compatibilità dell'applicazione creata con Android SDK può essere testata con un emulatore o variando le configurazioni di schermo, grazie ad un plugin fornito all'ambiente di sviluppo Eclipse. In futuro, l'applicazione realizzata con Kivy potrà essere migliorata integrandola con nuove funzionalità (che sono già state introdotte in Android SDK), e l'intero progetto verrà ampliato con l'inserimento di nuovi pannelli, i quali saranno anch'essi progettati per la ricezione di input da parte dell'utente. Se necessario, gli stessi parametri di scelta nelle sezioni contenenti i CheckBox potranno aumentare, con l'inserimento di nuovi CheckBox. In sostanza, l'applicazione creata realizza le operazioni richieste e permette la personalizzazione e la modifica futura a seconda delle esigenze, oltre che la possibilità e necessità di essere integrata con le funzionalità di altre applicazioni, per ottenere determinati output.

BIBLIOGRAFIA

Siti web

The Python Tutorial (http://docs.python.org/2/tutorial/)

Kivy (http://kivy.org/)

Kivy Documentation (http://kivy.org/docs/pdf/Kivy-latest.pdf)

Android Developers (http://developer.android.com/index.html)

Android SDK (http://developer.android.com/sdk/index.html)

GitHub (https://github.com)

Stack Overflow (http://stackoverflow.com)