UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze e Tecnologie

Corso di Laurea Triennale in Scienze e Tecnologie della comunicazione musicale

UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Sviluppo di un software didattico per l'apprendimento delle scale modali

Relatore:

Chiar.mo Prof. Luca Andrea Ludovico

Correlatore:

Chiar.mo Prof. Adriano Baratè

Elaborato Finale di:

Carlo Pulvirenti

Matricola n. 710352

Anno Accademico 2012/2013

“Quando non sai cos'è ... allora è Jazz”
La leggenda del pianista sull'oceano

Indice generale

Introduzione....5

Capitolo 1: L'utilizzo delle scale modali nell'improvvisazione....6

1.1 Cosa sono le scale modali e come si ricavano....8

1.1.1 Modi della scala Maggiore....8

1.1.2 Modi della scala minore melodica....10

1.2 Come e quando usare le scale modali....12

Capitolo 2: Tecnologie coinvolte....13

2.1 Caratteristiche di C# 13

2.1.1 Differenze con Java e precursori 13

2.1.2 JIT e JITter 14

2.2 Integrated Development Environment (Visual studio 2010 Express)....15

2.2.1 Agevolazione durante la compilazione....15

2.2.2 Rilevazione di errori....15

2.3 Interfaccia grafica....16

Capitolo 3: Costituzione del software....18

3.1 Fase di compilazione....18

3.1.1 Preparazione....20

3.1.2 Gestione della memoria....21

3.1.3 Navigazione....22

3.1.4 Creazione di metodi ricorrenti....22

3.1.5 Pulsante “>>”....24

3.1.6 Pulsante “<<”....27

3.1.7 Pulsanti “|<” e “>|”....28

3.1.8 Salvataggio del file....28

3.1.9 Caricamento dei file....30

3.1.10 Pulsante “Play”....33

3.2 Fase di esecuzione real time....36

3.2.1 Pulsante “Start”.....38
3.2.2 Scadenza del timer 39
3.2.4 Pulsanti di navigazione: “|<”, “<<”, “>>” e “>|”.....41
3.2.5 Evento “textchange”: caricamento misure, gestione delle label e modifica delle immagini.....43

Capitolo 4: Conclusioni.....46
4.1 Sviluppi futuri.....46

Bibliografia e Sitografia.....48
Appendici: Manuale d'istruzione.....49

Introduzione

I generi musicali di stampo jazzistico sono molto apprezzati per la loro intensità e capacità di trasmettere emozioni e sensazioni: il musicista solista, infatti, durante l'improvvisazione raggiunge il massimo grado di espressione artistica, avendo la possibilità di esternare completamente, grazie ad un ricco vocabolario sonoro, sentimenti come gioia, tristezza, passione, tensione e instabilità. La ricchezza comunicativa di questo linguaggio è dovuta ad un vasto assortimento di scale e sonorità, in grado di richiamare efficacemente stati d'animo ben precisi. Si pensi ad esempio ad artisti del calibro di *Miles Davis, Sonny Rollins,* e *Charlie Parker*, le cui sonorità sono facilmente riconoscibili dopo pochi secondi di ascolto. Tuttavia, imparare a esprimere ciò che si vuole comunicare (cioè a padroneggiare tutte le scale esistenti), è un'ardua impresa e, questa difficoltà, spesso scoraggia lo studente che, dopo qualche tentativo, rinuncia.

La tecnologia attuale non dispone di mezzi adeguati per alleggerire o semplificare un tale carico di lavoro. Per questo motivo, si è deciso di creare un software didattico per introdurre più agevolmente gli studenti al linguaggio jazzistico, dando loro la possibilità di apprendere gradualmente, e più facilmente, questo difficile stile. Per la creazione del software, a cui è stato dato il nome di "*Suggeritore modale*", si è utilizzato il linguaggio di programmazione a oggetti "*C#*" e il compilatore "*Visual Studio*".

L'interfaccia del *Suggeritore modale,* presenta dei campi di compilazione in cui l'utente dovrà inserire gli accordi di uno spartito, rispettando il loro posizionamento all'interno di ogni misura. Eseguito questo passaggio, l'intero spartito sarà visualizzato una misura alla volta, indicando, accordo per accordo, le relative scale consonanti. Grazie all'implementazione di un metronomo, è inoltre possibile far scorrere le misure sullo schermo alla velocità desiderata. Ciò garantisce la continuità della visualizzazione dello spartito senza ulteriori comandi; in questo modo, l'esecutore può improvvisare sul brano senza interruzioni. Il supporto fornito dal *Suggeritore modale* è esclusivamente grafico, sarà compito del docente eseguire l'accompagnamento armonico, ed eventualmente ritmico, durante l'improvvisazione. Il compito dello studente è quello di leggere e memorizzare un tipo di scala (per accordo) alla volta; in questo modo, con l'ausilio del *Suggeritore modale* e il costante studio dei brani, sarà garantito un veloce apprendimento di tutte le scale modali. Fine ultimo del software è quello di rendere l'allievo indipendentemente dalla lettura.

Capitolo 1: L'utilizzo delle scale modali nell'improvvisazione

Ascoltando un brano Jazz o Blues[1], non si può fare a meno di notare che la componente principale è l'improvvisazione: la macrostruttura più frequente in questi generi, infatti, è la sequenza "Tema-Soli-Tema", che lascia ampio spazio al momento solistico distribuito tra uno o più musicisti. In uno spartito tipico il tema viene descritto solamente da una spoglia linea armonico-melodica: in questo modo, le esecuzioni sono altamente personalizzabili, poiché lasciano agli interpreti la possibilità di inserire variazioni nello scheletro iniziale.

Le sezioni di uno spartito sono indicate come di seguito:

- *ritmica*: non viene mai definita, se non con cenni di stile come "shuffle", "straight", "swing" etc.;
- *basso*: è segnato solo in caso di melodie particolarmente complesse;
- *armonia*: è sempre riportata con la notazione anglosassone[2] senza indicare le altezze delle note che la compongono[3];
- *melodia*: nonostante sia riportata con molta più precisione, è tuttavia scritta con il chiaro intento di essere variata durante l'esecuzione al punto che neanche gli autori, suonando i propri brani, seguono la linea melodica esattamente come scritta nella partitura.

Di seguito sono riportati due spartiti Jazz molto differenti tra loro: *A night in Tunisia* di Dizzy Gillespie e *My favorite things* di John Coltrane. Lo spartito di *A night in Tunisia* è molto più dettagliato rispetto a quello di Coltrane: il primo, infatti, presenta un'indicazione ritmico/stilistica per la batteria ("MedAfro") e la linea di basso è riportata integralmente per le prime misure; in *My favorite things*, invece, vengono riportate solo l'armonia e la melodia, mentre sono totalmente assenti le notazioni ritmiche, quelle stilistiche, la velocità d'esecuzione (tipo "Fast", "Allegro", "Lento" etc.) e la linea di basso, che possono essere reperite dall'ascolto del brano originale.

1 Il discorso si può però estendere anche ai generi Funk e Fusion.
2 Nella notazione anglosassone il "La" corrisponde alla lettera "A", il "Si" alla "B" fino ad arrivare al "Sol" cui corrisponde la "G". I segni che seguono la lettera servono a identificare il tipo di accordo.
3 Sarà l'esecutore a decidere come distribuire le note dell'accordo sull'estensione del proprio strumento in base al gusto e l'esperienza personale.

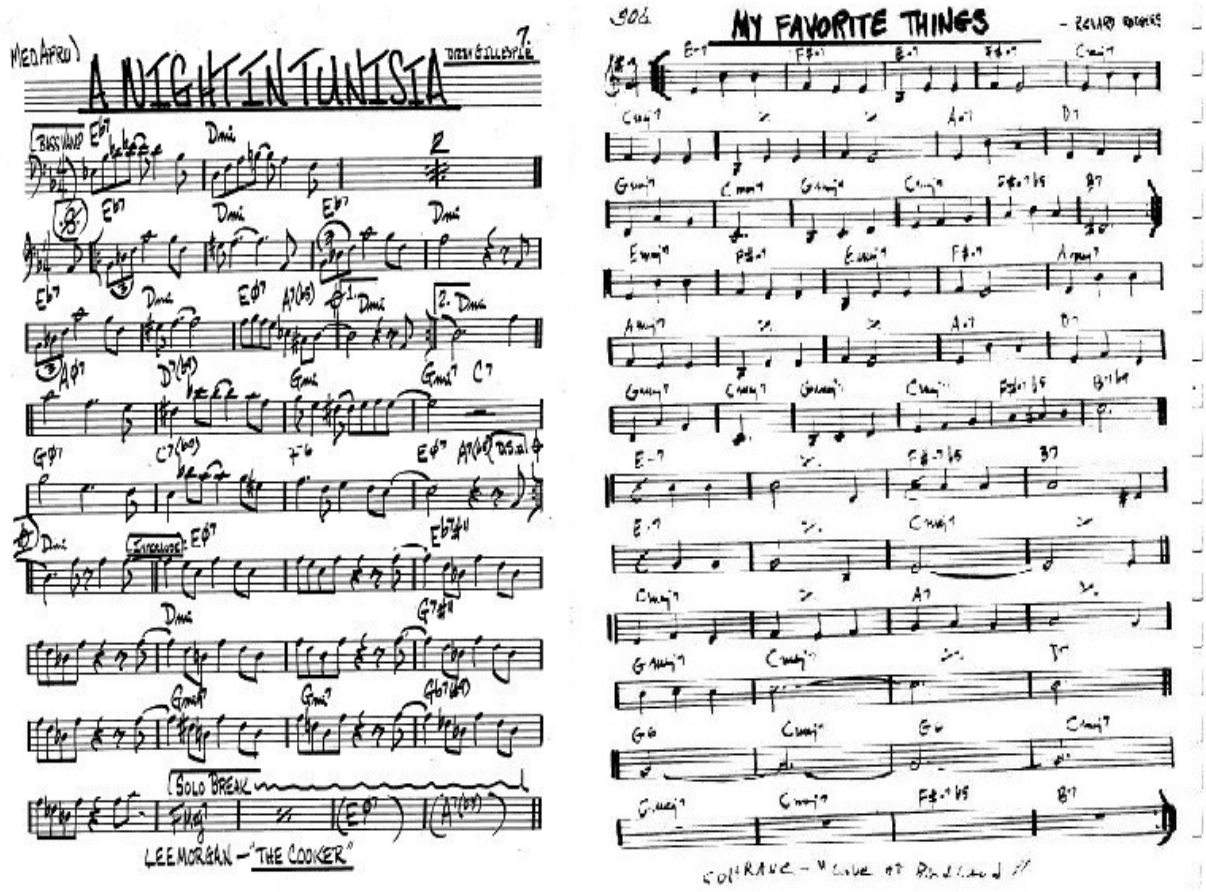


Figura 1: "A night in Tunisia" e "My favorite things".

Per quanto riguarda la fase solistica, l'esecutore deve mostrare le proprie capacità di coinvolgimento inventando sul momento una melodia interessante per tutti i tipi di ascoltatori, dal maestro all'appassionato. Tuttavia, improvvisare non è semplice come leggere uno spartito: si deve avere una certa intelligenza musicale, una confidenza approfondita col proprio strumento e conoscenze teoriche non indifferenti. Una buona improvvisazione è infatti costituita da variazioni ritmiche (chiamate "fraseggi") e variazioni melodiche basate sull'uso di "scale modali"; l'orecchio jazzistico preferirà una linea eseguita con frequenti, ma sensate, alternanze di fraseggi e melodie, piuttosto che una più stabile, ma meno varia, tipica del Pop e della musica leggera[4]. Mentre i fraseggi si imparano ascoltando i grandi artisti, le variazioni melodiche si devono apprendere attraverso esercizi mirati.

4 Nella "musica commerciale", infatti, la libertà del solista è fortemente limitata alla tonalità d'impianto del brano. L'esecutore può spaziare tra le note di una sola scala, che spesso viene ulteriormente limata diventando "pentatonica", al fine di evitare ogni dissonanza.

1.1 Cosa sono le scale modali e come si ricavano

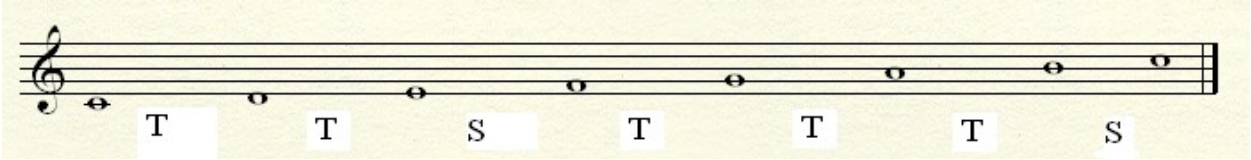
Una scala formata da sette note (otto, considerando anche la note di arrivo) si definisce “Scala eptafonica” e viene classificata in base alla distanza, espressa in “semitoni”, tra le note che la compongono. Infatti, applicando una sequenza di intervalli codificata a una qualsiasi nota[5], si otterrà la scala desiderata. In questo studio verranno prese in esame solo scale eptafoniche diatoniche basate sul “Temperamento equabile”, che prevede la suddivisione di un'ottava in dodici semitoni equidistanti. Le note di una scala vengono definite “gradi” e sono ordinate (e rinominate) in base alla distanza dalla tonica.

Una scala ricavata partendo da un qualsiasi grado di un'altra scala si definisce “scala modale” . I Modi presi in esame per la scrittura del software sono costruiti sulle scale eptafoniche Maggiore, minore melodica e minore armonica[6].

1.1.1 Modi della scala Maggiore

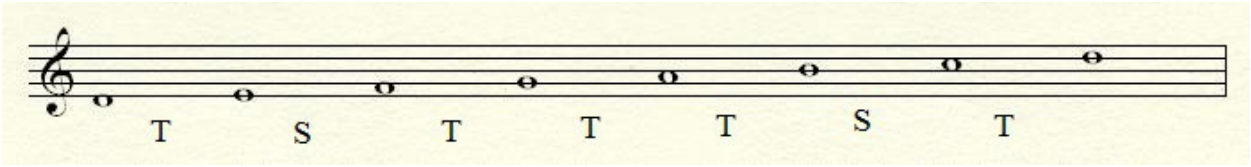
La scala Maggiore è codificata dagli intervalli “Tono-Tono-Semitono-Tono-Tono-Tono(-Semitono)” come segue:

Figura 2: scala di Do Maggiore.



La stessa scala, suonata con le stesse note e le stesse alterazioni ma partendo dal secondo grado (in questo caso “Re”), genererebbe una nuova sequenza di intervalli: “T-S-T-T-T-S(-T)”, che ovviamente sarà diversa da quella maggiore.

Figura 3: scala di Re Dorico



Questo tipo di scala, detta “Dorica”, è situata sul secondo grado della scala maggiore cui fa riferimento. Ripetendo questo procedimento su tutti i gradi della scala si avranno i sette modi di

5 La prima nota di una scala viene definita “Tonica”.
6 In queste tre scale, i gradi oscillano tra una distanza minima di un semitono ad una massima di un tono e mezzo.

Do maggiore, rispettivamente chiamati: “Do Ionico”, “Re Dorico”, “Mi Frigio”, “Fa Lidio”, “Sol Misolidio”, “La Eolico” e “Si Locrio”.

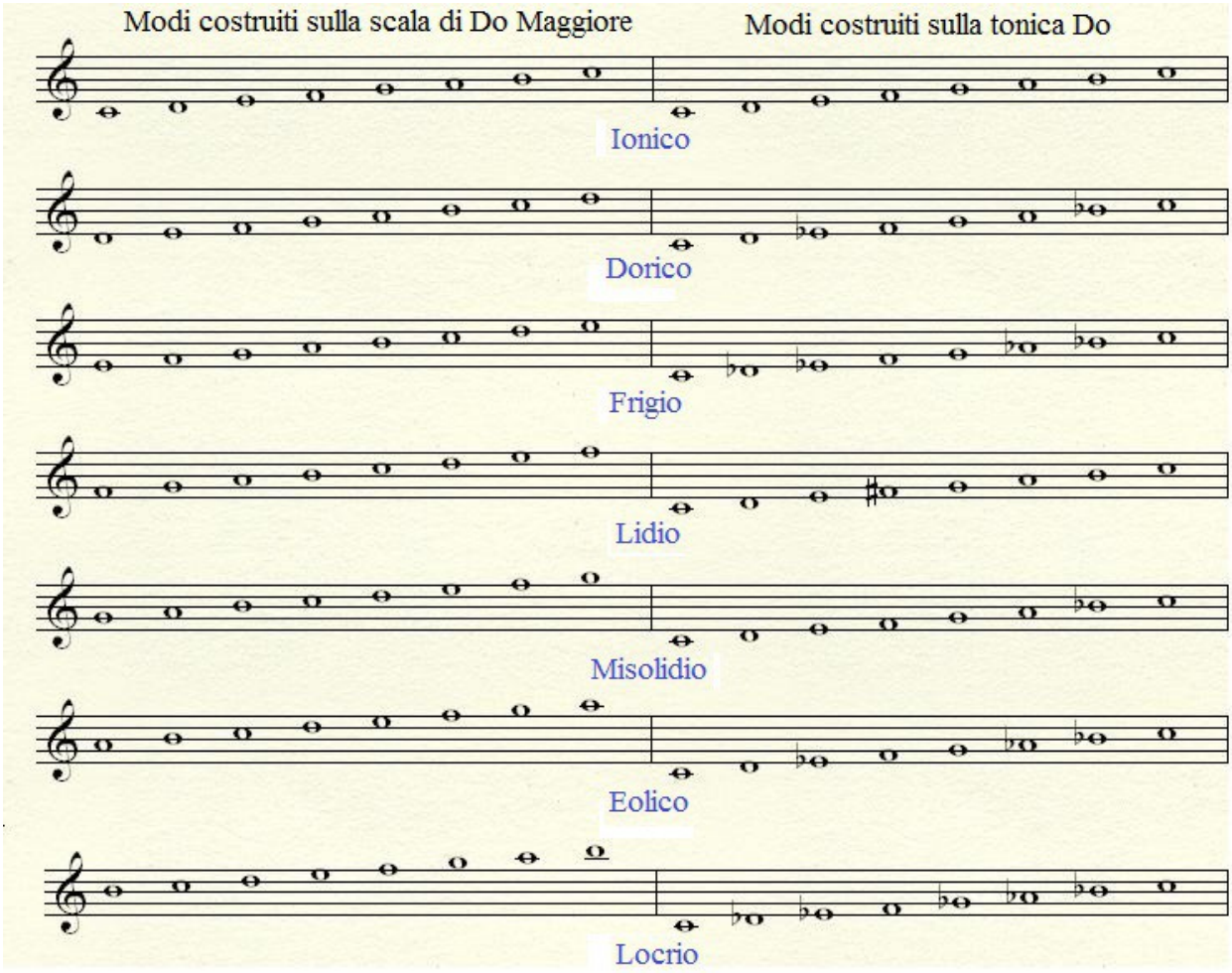


Figura 4: Sulla sinistra dell'immagine sono riportati i modi relativi alla scala di Do maggiore mentre sulla destra sono mostrati, per completezze d'informazione, i modi maggiori basati sulla tonica Do.

È bene ricordare che ogni Modo presenti sonorità proprie, nonostante sia composto dalle stesse note della scala da cui è ricavato. La singolarità di ogni scala modale è percepibile solo se suonata sull'armonia di una accordo di quattro note (detto quadriade)[7].

7 Vedi capitolo 1.2 a pag 12

1.1.2 Modi della scala minore melodica

La scala minore melodica ha la caratteristica di proporsi in due modi diversi in base alla modalità di esecuzione: essa infatti prevede la sequenza "T-S-T-T-T-T(-S)" in fase ascendente e di "T-T-S-T-T-S-T" in fase discendente.

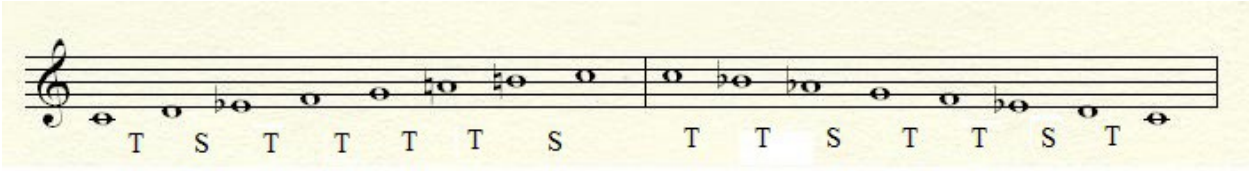


Figura 5: scala di Do minore melodica

Nonostante questa differenziazione, durante l'improvvisazione è consigliabile utilizzare le sole note della fase ascendente per dar maggior risalto alla singolarità della scala:; infatti, l'orecchio non percepirebbe una scala minore melodica in fase ascendente e discendente, bensì l'uso di due scale diverse: minore melodica e minore naturale. Così come si costruiscono i sette modi della scala maggiore, allo stesso modo si possono ricavare quelli della scala minore melodica. La codifica degli intervalli ottenuti dà vita ai seguenti modi: "Ipoionico", "Dorico b2", "Lidio aumentato", "Lidio b7", "Misolidio b13", "Locrio #2", "Super locrio".

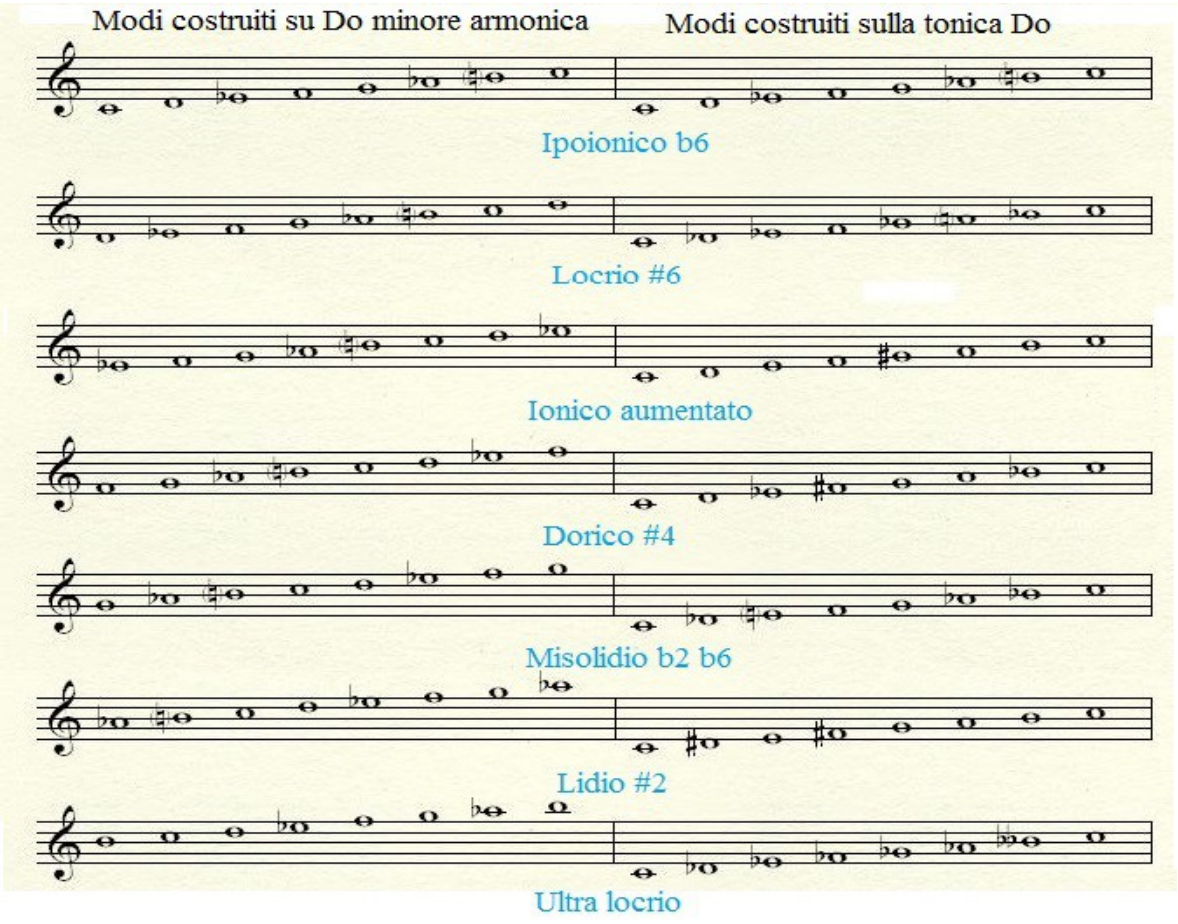


Figura 6: modi della scala minore melodica

1.1.3 Modi della scala minore armonica

L'ultima scala generatrice da considerare è la *minore armonica*. Essa è simile a quella melodica ma, a differenza delle altre scale, prevede un caratteristico intervallo di un tono e mezzo tra il sesto e il settimo grado che le attribuisce una sonorità molto caratteristica[8]. I gradi della scala minore armonica si presentano a questa distanza: "T-S-T-T-S-TS-S"

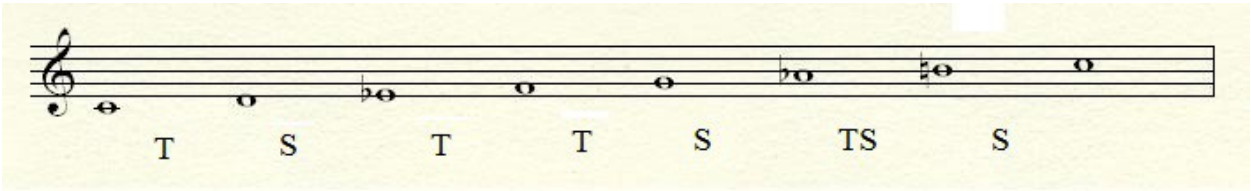


Figura 7: scala minore armonica

Applicando ancora una volta lo stesso algoritmo, otterremo i Modi della scala minore armonica: "Ipoionico b6", "Locrio #6", "Ionico aumentato", "Dorico #4", "Misolidio b2 b6", "Lidio #2", "Ultra locrio".

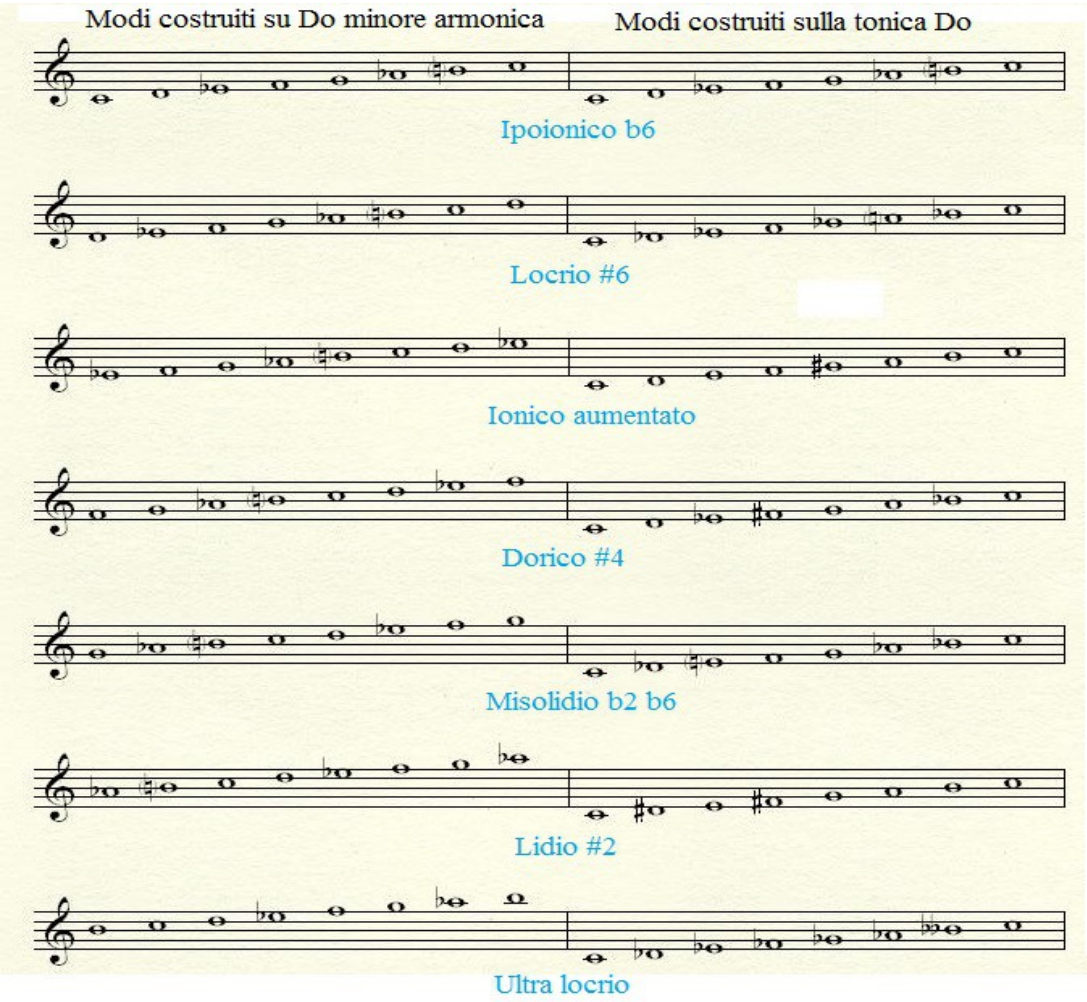


Figura 8: modi della scala minore armonica

8 Tale sonorità viene spesso abbinata alle melodie tipiche del Nord Africa.

1.2 Come e quando usare le scale modali

Ora che si conoscono le ventuno scale modali si può capire la loro relazione con gli accordi e quindi il loro utilizzo in fase improvvisativa. Ogni accordo, in base alla propria entità[9], fornisce una o più scale compatibili da utilizzare senza stonare durante un solo. Per sapere quali esse siano, è sufficiente cercare quali scale modali contengano tutte le note della quadriade in analisi (alterazioni incluse).

Ad esempio, volendo conoscere quali scale siano compatibili con l'accordo "Fa Maggiore 7"[10] si dovrebbe cercare, tra le ventuno scale modali di Fa[11], quelle contenenti le note "Fa-La-Do-Mi"[12] . Dopo aver confrontato tutte le scale si otterrà la risposta: modo "Ionico", "Lidio" e "Lidio #2" di Fa.

Estendendo questo procedimento ai sette tipi di accordo presenti nelle scale Maggiore, minore armonica e minore melodica, si nota che indipendentemente dalla nota di partenza (tonica), dato un tipo di accordo si possono sfruttare sempre le stesse determinate scale, come riassunto nel seguente specchietto:

Tipo di accordo	**Scale compatibili**
Maggiore sette "Maj 7"	Ionico, Lidio, Lidio #2
Maggiore con quinta eccedente "Maj 7 (5#)"	Lidio aumentato, Ionico aumentato
Minore sette "min 7"	Dorico, Frigio, Eolico, Dorico b2, Dorico #4
Minore con settima maggiore "min (7M)"	Ipoionico, Ipoionico b6
Settima di dominante "7"	Misolidio, Lidio b7, Misolidio b13, Frigio di dominante
Semidiminuito "ø"	Locrio, Locrio #2, Super locrio, Locrio #6
Diminuito "°7"	Ultra locrio

9 L'entità di un accordo è data dal tipo di intervalli che intercorrono tra le note della quadriade. In base a queste ultime, esso potrà essere classificato come "Maggiore", "minore", "diminuito", "eccedente" etc.

10 Detto anche "F Maj 7".

11 Vale a dire "le scale che hanno come tonica la nota Fa".

12 Rispettivamente primo, terzo, quinto e settimo grado dell'accordo di "F Maj 7".

Capitolo 2: Tecnologie coinvolte

Il linguaggio di programmazione utilizzato per la scrittura del *Suggeritore modale* è "C#"[13]. Creato da *Anders Hejlsberg*[14], viene lanciato sul mercato nell'anno 2000 insieme alla piattaforma di sviluppo ".Net", che ne rappresenta l'ambiente di sviluppo.

2.1 Caratteristiche di C#

Così come *C*, *C++* e *Java*, anche *C sharp* è un linguaggio orientato agli oggetti ("Object Oriented Programming"[15]). La sintassi semplice ereditata dai suoi precursori e gli strumenti di programmazione visiva di *Visual Basic*, ne hanno favorito lo sviluppo e la diffusione in campo informatico. L'ultima versione rilasciata è la *5.0*.

Pur essendo un linguaggio sensibile alla differenziazione tra lettere maiuscole e minuscole[16], *C#* si presenta con una sintassi semplice, intuitiva e di facile comprensione; inoltre, garantisce un certo livello di sicurezza grazie ai comandi di gestione delle eccezioni.

2.1.1 Differenze con *Java* e precursori

C# si differenzia da *C*, *C++* e *Java* per le seguenti caratteristiche:

- non sono ammessi i puntatori, con l'unica eccezione dei blocchi di codice dichiarati "unsafe",
- si introduce il garbage-collector[17],
- non viene supportata l'ereditarietà multipla fra classi, ma è consentita l'implementazione di un qualsiasi numero di interfacce,
- i "*template*" di *C++* sono sostituiti dai "tipi generici",
- a differenza di *Java*, è utile definire le proprietà di una classe grazie al "*getter*" e "*setter*", e non è obbligatorio gestire le eccezioni,
- sono supportati i "*delegate*".

13 Altresì detto *C sharp*.
14 Inventore del linguaggio *Delphi*.
15 Abbreviato in *OOP*.
16 In inglese *case sensitive*.
17 Maggiori approfondimenti nella prossima pagina.

2.1.2 JIT e JITter

Per migliorare le performance durante l'esecuzione di un'applicazione, il linguaggio C# sfrutta un compilatore di tipo "*Just in time*"[18].

«L'obiettivo finale dei sistemi JIT è di combinare i vantaggi della compilazione del bytecode a quelli della compilazione nativa, aumentando le prestazioni quasi al pari di una compilazione direttamente in linguaggio macchina. »[19]

In fase di esecuzione, il JIT si avvale di un compilatore "*JITter*" per convertire le istruzioni in linguaggio binario, traendo i seguenti vantaggi:

- durante la prima invocazione di un metodo, il codice binario risultante dalla compilazione viene salvato in una memoria cache; così facendo, la successiva invocazione sarà più veloce,
- le parti di codice non necessarie durante la sessione non vengono caricate, ma sono eseguite esclusivamente le istruzioni richieste al momento della loro invocazione; in questo modo si evita di sovraccaricare inutilmente il processore.

2.1.3 Garbage collector

A differenza dei precedenti *C* e *C++*, dove lo spazio di memoria era gestito dal programmatore, in *C#* è stata creata un'entità che si occupa di deallocare implicitamente la memoria occupata da oggetti che non vengono più usati; in questo modo, si possono evitare spiacevoli conseguenze dovute a banali errori di distrazione.

Il "*Common Language Runtime*"[20], anima del .Net framework, gestisce il garbage-collector tenendo i riferimenti degli oggetti istanziati (che quindi occupano memoria): quando uno di questi oggetti non ha più ragione di esistere, ovvero quando non ha più un riferimento, il CLR ripulisce la zona di memoria in cui l'oggetto risiedeva[21].

18 Abbreviato in "*JIT*".
19 Cit. [6]
20 Abbreviato in "*CLR*".
21 Cit. [1]

2.2 Integrated Development Environment (Visual studio 2010 Express)

Per facilitare la creazione di un programma è necessario l'uso di un "Integrated Development Environment", cioè un ambiente di sviluppo creato per facilitare il programmatore durante la scrittura del codice sorgente. Per la scrittura del *Suggeritore modale* è stato usato l'IDE open source messo a disposizione da Microsoft: "*Visual Studio 2010 Express*". Di seguito vengono riportati alcuni dei vantaggi derivanti dall'utilizzo di un IDE.

2.2.1 Agevolazione durante la compilazione

In fase di scrittura, *Visual Studio* intuisce il frammento di codice che l'utente vuole inserire e, tramite un menù a tendina, mostra le istruzioni utilizzabili in tali circostanze. Grazie a questa funzione, è possibile capire se il codice che si vuole inserire sia riportato nella giusta posizione: se *Visual Studio* non ne suggerisce l'utilizzo, significa che non si può utilizzare in quel punto.

2.2.2 Rilevazione di errori

Pur inserendo i giusti frammenti di codice nelle giuste posizioni, sarà sempre possibile incorrere in errori dovuti, ad esempio, a situazioni di conflitto o di incompatibilità; questo tipo di situazioni sono previste e tempestivamente segnalate da *Visual Studio* per mezzo di una sottolineatura zigzagata sotto il frammento errato. Questa funzione si rivela particolarmente utile in fase di scrittura perché consente di identificare istantaneamente gli errori, facendo risparmiare tempo al programmatore.

2.2.3 Modalità "debug" e "breakpoint"

Per verificare l'efficacia del codice in fase di esecuzione e il corretto funzionamento del software, *Visual Studio* ha implementato la modalità "debug", che permette di simulare il codice scritto senza aprire il file eseguibile.

Le impostazioni avanzate consentono l'uso di una funzione chiamata "breakpoint"[22]: inserendo un'interruzione in un determinato punto del codice, sarà possibile verificare, istruzione per istruzione, il corretto svolgimento del codice stesso. Questo stratagemma viene frequentemente utilizzato per rilevare errori che sarebbero altrimenti difficili da individuare.

22 "*Punto di interruzione*".

2.3 Interfaccia grafica

Visual Studio, oltre a supportare l'utente nella scrittura del codice sorgente, dà la possibilità di gestire l'interfaccia grafica in modo semplice e intuitivo: con dei semplici comandi di drag&drop è possibile disegnare l'anteprima del software in base alle proprie esigenze, modificare i valori dei campi inseriti e associare elementi di interazione.

2.3.1 Elementi ricorrenti

Per la compilazione del *Suggeritore modale* sono stati utilizzati i seguenti elementi:

- **Textbox**: campo di inserimento testo. Per comodità, in fase di compilazione viene usato il suffisso "*Txt*" come "*Nome del brano*", "*misuraNumero*" (form1) e "*misuraAttuale*" (form2).
- **Combobox**: campo di selezione multipla tra frammenti di testo precedentemente inseriti. È stata utilizzata per registrare i campi di "*tonica*" e "*accordo*" di tutti i movimenti della form1. Per identificarle si è deciso di utilizzare il suffisso "*Cbb*".
- **Radio button**: indicano la possibilità di effettuare una scelta multipla tra le suddivisioni. Salvati con il suffisso "*Rbt*", la loro selezione influenza l'aspetto della form abilitando o meno l'interazione dei movimenti in fase di scrittura.
- **Pulsanti**: detti anche bottoni, sono elementi di interazione che, tramite un click, eseguono le istruzioni che sono state assegnate loro. Indicati dal suffisso "*Btn*", vengono usati frequentemente in entrambe le form: pulsanti di navigazione ("|<","<<", ">>" e ">|"), pulsanti di load&save ("*Apri*" e "*Salva*") e pulsanti di gestione della form ("*Continua*", "*Start*" e "*Play*").
- **Label**: nominate anche "*etichette*", riportano frammenti di testo, solitamente statici, inseriti nelle form. Sono tipicamente usate come intestazione per indicare qualcosa, ad esempio il tipo di campo da inserire nella combobox o textbox adiacente. Per segnalarle è stato scelto il suffisso "*Lbl*". In fase di esecuzione l'eleganza e la versatilità delle Label viene sfruttata rendendole dinamiche grazie a dei comandi di modifica del testo.
- **Picturebox**: area creata per ospitare immagini. Le picturebox, salvate con il suffisso "*Img*", sono utilizzate in maniera altamente dinamica: il loro contenuto è in costante mutamento in base al testo riportato nelle label. Il *Software modale* utilizza immagini in formato "*.JPG*".

- **Numeric up-down**: campo di selezione multipla di valori numerici. Questo elemento dà la possibilità di scrivere la cifra desiderata o modificare il valore corrente con dei piccoli tasti di aumento e decremento. Il *Suggeritore modale* sfrutta un numeric up-down nella form2 per settare i bpm[23] del brano da eseguire (i valori accettati rientrano tra "20" e "250").
- **Timer:** cronometro. Con questa funzione viene avviato un conteggio, di default impostato su mille millisecondi (un secondo), al termine del quale saranno eseguite le istruzioni assegnate in fase di programmazione. Il conteggio verrà ripetuto ciclicamente fino alla ricezione di un comando di arresto.

Il *Suggeritore modale* fa largo uso del timer poiché esso, con l'opportuna implementazione, si rivela un comodo metronomo: la durata delle pulsazioni è legata al valore riportato nella casella *numeric up-down*.

23 Battiti Per Minuto.

Capitolo 3: Costituzione del software

Al fine di garantire l'utilizzo del programma a tutti, si è deciso di renderne il funzionamento intuitivo anche per gli utenti privi di conoscenze informatiche. Per questo motivo il lavoro è stato diviso in due fasi: una di compilazione, nella quale inserire gli accordi del brano da studiare, ed una di esecuzione, finalizzata all'esercitazione in real time.

3.1 Fase di compilazione

In fase preliminare, il *Suggeritore modale* dà la possibilità di interagire con pochi campi, disabilitando quelli che potrebbero creare confusione e lasciando i seguenti abilitati :

- una *textbox* nella quale inserire il nome del brano,
- quattro *radio button* rappresentanti il numero di suddivisioni del brano[24],
- un *pulsante* "*Continua*",
- Due *pulsanti* dedicati alle funzioni di "*load&save"* del contenuto ("*Apri*" e "*Salva*").

Figura 9: finestra di apertura.

24 Considerata la grande diffusione di musica scritta in "4/4", si è deciso di impostare il "4" come valore di default impostando su "*true*" il campo "*checked*" in fase di compilazione.

Di seguito vengono riportate le prime righe del codice.

```
// Intestazione iniziale
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Assegnazione di un nome al namespace
namespace WindowsFormsApplication1
{

// Creazione e descrizione della prima form
    public partial class Progettazione : Form
    {

// Dichiarazione di una seconda form
        public Esecuzione form2;

// Inizializzazione della form di progettazione
        public Progettazione()
        {
            InitializeComponent();

// Disabilitazione delle combobox relative alle toniche
            mov1TonicaCbb.Enabled = false;
            mov2TonicaCbb.Enabled = false;
          [...]

// Disabilitazione delle combobox relative agli accordi
            mov1AccordoCbb.Enabled = false;
            mov2AccordoCbb.Enabled = false;
          [...]

// Disabilitazione dei pulsanti di navigazione
            successivaBtn.Enabled = false;
            precedenteBtn.Enabled = false;
            playBtn.Enabled = false;
            inizioBtn.Enabled = false;
            fineBtn.Enabled = false;

        }

// Dichiarazioni della variabile numeroSuddivisioni[25]
        public int numeroSuddivisioni = 0;
```

25 Vedi cap 3.1.3 a pag. 22

3.1.1 Preparazione

Dopo aver compilato la textbox "*nomeBranoTxt*"[26] e selezionato il numero di suddivisioni, la pressione del tasto "*Continua*" apporterà le seguenti modifiche:

- disabilitazione dei campi agibili in apertura e modifica del colore, fatta eccezione per i pulsanti "*Salva*" e "*Apri*",
- abilitazione dei campi relativi al numero di suddivisioni [27] e dei pulsanti di navigazione[28],
- aggiornamento della variabile globale "*numeroSuddivisioni*",
- aggiunta del nome del brano nell'intestazione,
- salvataggio della prima misura (vuota),
- aggiornamento di "*numeroMisuraTxt*" e "*totMisureLbl*".

```
//------Pulsante "CONTINUA"
        private void continua_Click(object sender, EventArgs e)
        {

//Messagebox di verifica preliminare
            if (nomeBranoTxt.Text == "") MessageBox.Show("Inserire il nome
del brano");
            else
            {

//Disabilitazione dei campi precedentemente agibili
                continuaBtn.Enabled = false;
                nomeBranoTxt.Enabled = false;

                dueSuddivisioniRbt.Enabled = false;
                treSuddivisioniRbt.Enabled = false;
                quattroSuddivisioniRbt.Enabled = false;
                cinqueSuddivisioniRbt.Enabled = false;

// Cambio di colore per i radio button ed etichette prive di valore
                selezionaSuddivisioniLbl.ForeColor = Color.FromArgb(128, 128,
128);
                nomeBranoLbl.ForeColor = Color.FromArgb(128, 128, 128);

                dueSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128, 128);
                treSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128, 128);
                quattroSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128,
128);
                cinqueSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128,
128);

                numeroMisuraLbl.ForeColor = Color.FromArgb(0, 0, 0);
                totMisureLbl.ForeColor = Color.FromArgb(0, 0, 0);

// Istruzioni da svolgere in base al radio button selezionato
```

26 In caso di mancata compilazione si aprirà una messagebox per segnalare l'errore ed arrestare l'avanzamento.
27 Le suddivisioni eccedenti resteranno colorate di grigio e le relative combobox rimarranno disabilitate.
28 "|<", "<<", ">>" e ">|"

```
                if (dueSuddivisioniRbt.Checked)
                {
                    mov1TonicaCbb.Enabled = true;
                    mov1AccordoCbb.Enabled = true;
                    mov1Lbl.ForeColor = Color.FromArgb(0, 0, 0);
                    mov1AccLbl.ForeColor = Color.FromArgb(0, 0, 0);
                    mov1TonLbl.ForeColor = Color.FromArgb(0, 0, 0);

                    mov2TonicaCbb.Enabled = true;
                    mov2AccordoCbb.Enabled = true;
                    mov2Lbl.ForeColor = Color.FromArgb(0, 0, 0);
                    mov2AccLbl.ForeColor = Color.FromArgb(0, 0, 0);
                    mov2TonLbl.ForeColor = Color.FromArgb(0, 0, 0);

                    numeroSuddivisioni = 2;
                }
// [...]

// Abilitazione dei pulsanti utili
                successivaBtn.Enabled = true;
                precedenteBtn.Enabled = true;
                playBtn.Enabled = true;
                inizioBtn.Enabled = true;
                fineBtn.Enabled = true;


// Inserimento del nome del brano nell'intestazione
                this.Text = "Suggeritore modale - " + nomeBranoTxt.Text;

// Auto-salvataggio della prima misura (vuota)
                misura = new List<Movimento>();
                SalvaMovimenti();
                misure.Add(misura);

//Aggiornamento di "misuraAttualeTxt"
                int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text);
                misuraAttuale++;
                numeroMisuraTxt.Text = Convert.ToString(misuraAttuale);

// Aggiornamento di "totMisureLbl" in base al numero di misure salvate
                totMisureLbl.Text = "/ " + Convert.ToString(misure.Count);
            }
        }
```

3.1.2 Gestione della memoria

Al fine di rendere la scrittura dello spartito semplice ed ordinata, si è deciso di creare una memoria annidata di questo tipo:

- una *List* "*misure*" composta da una lista descrivente ogni misura,
- una *List* "*misura*" formata da tanti movimenti quanti richiesti dal brano,
- una *Struct* "*Movimento*" composta da due campi string relativi a "*Tonica*" e "*Accordo*".

Tutti i campi sono impostati su *public* per poter garantire l'usabilità nella seconda form (relativa all'esecuzione in real time). Nella finestra di codice seguente si può individuare la creazione di

due metodi finalizzati a creare, tramite il prefisso *new*, una nuova istanza all'interno della lista. Tali metodi vengono richiamati durante la navigazione tra le misure e il loro utilizzo è necessario per modificare battute precedentemente scritte o ancora da scrivere[29].

```
// Creazione della struct "Movimento"
        public struct Movimento
        {

// Campi della struct
            public string Tonica;
            public string Accordo;
        }

//Creazione della lista "misure"
         public List<List<Movimento>> misure = new List<List<Movimento>>();

//Creazione della lista "misura"
         public List<Movimento> misura = new List<Movimento>();
```

3.1.3 Navigazione

Come si avrà modo di vedere nei prossimi paragrafi, analizzando il codice si può notare la presenza di due variabili ricorrenti e il costante aggiornamento degli elementi grafici che le contengono: tali variabili sono "*misuraAttuale*" e "*misure.count*". Mentre la prima, mostrata in "*numeroMisuraTxt*", indica all'utente quale misura stia consultando, la seconda, visualizzata in "*totMisureLbl*", riporta il numero complessivo di misure inserite al momento della lettura. Il loro valore si eguaglia quando l'utente si posiziona nell'ultima misura creata. Il confronto avverrà molto frequentemente poiché il risultato di tale comparazione modificherà le istruzioni da svolgere in più punti del codice.

3.1.4 Creazione di metodi ricorrenti

Il frammento di codice della prossima finestra è estratto dal metodo "*SalvaMovimenti*( )" e illustra i passaggi della struct "*Movimento*" in fase di salvataggio:

- la condizione del primo "*if*" serve ad impedire il salvataggio di un movimento estraneo allo spartito (se una combobox è disabilitata, il movimento a cui fa riferimento è inesistente),
- una volta soddisfatta tale condizione, prima di salvare i movimenti presenti nella misura, il software dovrà accertare che l'utente abbia correttamente compilato tutti i campi: la seconda e terza condizione, verificano che tutti i movimenti abbiano la propria tonica ed il proprio accordo,

29 Approfondimenti a seguire, nel paragrafo 3.1.4

- ultimata la verifica delle condizioni preliminari, il metodo "*salvaMovimenti*( )" svolgerà la propria funzione creando una nuova istanza di nome "*mov1*" all'interno della struct "*Movimento*" e inserendo "*mov1*" nella lista "*misura*"; l'istruzione viene ripetuta un numero di volte pari al numero di suddivisioni selezionate[30].

La seconda e la terza condizione accettano movimenti vuoti (senza "*tonica*" e "*accordo*"), ma respingono movimenti con un solo campo selezionato: nel caso di dimenticanza da parte dell'utente, l'istruzione sarà immediatamente interrotta e verrà visualizzata una messagebox d'errore riportante l'entità dello stesso: "*Selezionare un tipo di accordo per il primo movimento*" o "*Selezionare una tonica per il primo movimento*" in base al tipo di errore e movimento.

```
// Creazione del metodo "SalvaMovimenti()"
         public void SalvaMovimenti()
         {
// Verifica della prima condizione
                if (mov1TonicaCbb.Enabled == true)
                {

// Verifica della seconda condizione
                    if (mov1TonicaCbb.SelectedIndex != 0 &&
mov1AccordoCbb.SelectedIndex == 0)
                    {

// Eventuale messagebox di segnalazione
                        MessageBox.Show("Selezionare un tipo di accordo per
il primo movimento.");
                        return;
                    }
                    else if (mov1AccordoCbb.SelectedIndex != 0 &&
mov1TonicaCbb.SelectedIndex == 0)
                    {

// Eventuale messagebox di segnalazione
                        MessageBox.Show("Selezionare una tonica per il primo
movimento.");
                        return;
                    }
// Salvataggio di "tonica" e "accordo" nella nuova istanza "mov1"
// di "Movimento"
                    Movimento mov1 = new Movimento();
                    mov1.Tonica = mov1TonicaCbb.Text;
                    mov1.Accordo = mov1AccordoCbb.Text;

                    misura.Add(mov1);
                }
// [...]
         }
```

30 Un brano a quattro suddivisioni, infatti, non soddisferà la condizione ("*mov5tonicaCbb == true*") e non eseguirà l'istruzione.

Il metodo proposto si limita a creare una misura senza aggiungerla alla list "*misure*"; questa scelta, apparentemente deficitaria, permette in realtà di gestire elasticamente l'aggiornamento di misure già scritte e l'aggiunta di nuove[31].

Il solo metodo di salvataggio movimenti non è sufficiente per adempire alle esigenze del *Suggeritore modale* poiché, in fase di compilazione, sarà spesso necessario visualizzare ciò che si è scritto nelle misure precedenti; diventa quindi indispensabile la creazione di un metodo finalizzato al caricamento delle misure già scritte. Il metodo in questione, nominato "*CaricaMovimenti* ( )", ha quindi il compito di verificare il numero di movimenti utili e riportare, nelle combobox relative, le toniche e gli accordi della misura da visualizzare.

Di seguito il codice del metodo "*CaricaMovimenti( )*":

```
// Creazione del metodo "CaricaMovimenti()"
         public void CaricaMovimenti()
         {

// Verifica della misura da richiamare
             int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text)-1;

// Verifica del numero di movimenti disponibili
             if (mov1TonicaCbb.Enabled == true)
             {

// Richiamo di "tonica" e "accordo" della misura "misuraAttuale"
// (visualizzata nella form) e aggiornamento delle combobox
                 mov1TonicaCbb.Text = misure[misuraAttuale][0].Tonica;
                 mov1AccordoCbb.Text = misure[misuraAttuale][0].Accordo;
             }
// […]
         }
```

3.1.5 Pulsante ">>"

Dopo aver premuto il tasto "*Continua*", la form si presenta in questo modo: le combobox "*Tonica*" e "*Accordo*" relative ai movimenti selezionati e i tasti di navigazione sono adesso abilitati; da questo momento l'utente ha la possibilità di inserire gli accordi dello spartito selezionandoli dalla lista disponibile. Restano attivi i pulsanti di load&save.

31 Maggiori spiegazioni sono fornite nel codice relativo ai pulsanti "<<" e ">>" nei paragrafi seguenti.

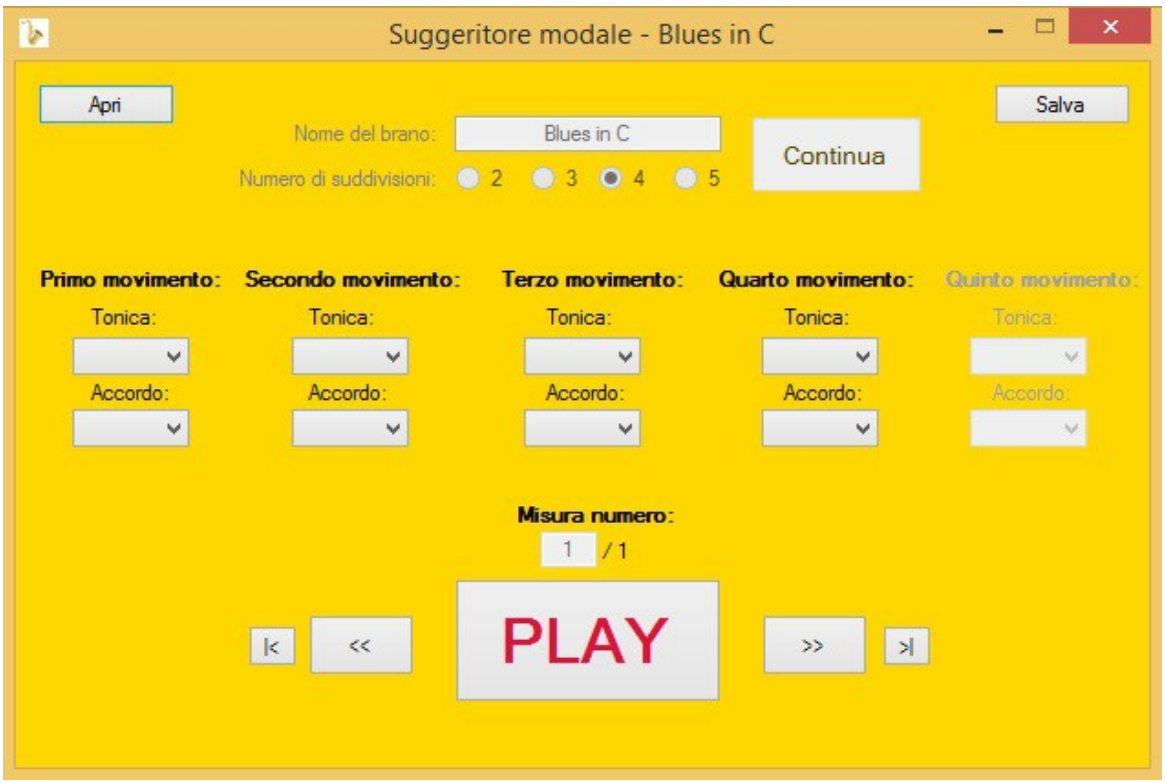


Figura 10: pressione del tasto "Continua"

La pressione del *tasto* ">>", grazie all'invocazione dei metodi "*misura = new List<Movimento>( )*" e "*SalvaMovimenti( )*", crea una misura contenente i movimenti appena inseriti nelle combobox e, tramite l'assegnazione "*misure[misuraAttuale] = misura",* sovrascrive il contenuto della misura visualizzata.

A questo punto, confrontando i valori delle variabili "*misuraAttuale*" e "*misure.count*", si può decretare quali istruzioni saranno eseguite dal software in risposta alla condizione del primo "if": se la misura visualizzata è l'ultima inserita dall'utente[32] (uguaglianza del valore delle variabili, risposta "true") saranno eseguite le seguenti istruzioni:

- cancellazione di tutti i campi "*Tonica*" e "*Accordo*" (impostando l'indice di tutte le combobox a "*0*"), consentendo la scrittura della misura successiva,
- creazione di una nuova misura,
- inserimento nell'ultima posizione della list "*misure*", tramite il comando "*misure.add(misura)*", della *misura* appena creata,
- incremento della variabile "*misuraAttuale*", modifica della relativa textbox "*numeroMisuraTxt*" e aggiornamento di "*totMisureLbl*"[33].

32 La verifica si può ugualmente effettuare confrontando il valore riportato in "*numeroMisuraTxt*" e "*totMisureLbl*".

33 Aggiornamento dovuto all'incremento del numero di misure e quindi del valore di "*misure.count*"

Al contrario, se il valore di "*misuraAttuale*" non fosse uguale a quello di "*totMisureLbl*", la condizione di "if" risulterebbe "false", facendo così entrare il software all'interno delle istruzioni contenute in "*else*".

Dato che la condizione iniziale richiedeva che "*misuraAttuale*" fosse inferiore di "*misure.count*", la pressione del *tasto* ">>" porterà sicuramente l'utente a visualizzare una misura già creata; per questo motivo, dopo aver sovrascritto la *misura* appena visualizzata, sarà sufficiente incrementare il valore di "*numeroMisuraTxt*" e invocare il metodo "*CaricaMovimenti( )*", il quale consente all'utente di visualizzare quanto scritto in precedenza nella misura di destinazione. Questa differenza di istruzioni spiega come mai il metodo "*SalvaMovimenti( )*" sia stato scritto in maniera incompleta.

```
// Descrizione delle azioni derivate dall'evento "button.click" del
// pulsante ">>"
        private void prossima_Click(object sender, EventArgs e)
        {
            int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text)-1;

// Creazione di una list di movimenti nominata "misura"
            misura = new List<Movimento>();

// Salvataggio dei movimenti all'interno della misura appena creata
            SalvaMovimenti();

// Sovrascrittura della misura visualizzata
            misure[misuraAttuale] = misura;

// Azioni da compiere se l'utente sta compilando l'ultima misura
// dello spartito

            if (misuraAttuale+1 == misure.Count)
            {

//Cancellazione dei campi delle toniche
                mov1TonicaCbb.SelectedIndex = 0;
                mov2TonicaCbb.SelectedIndex = 0;
// [...]

// Cancellazione dei campi degli accordi
                mov1AccordoCbb.SelectedIndex = 0;
                mov2AccordoCbb.SelectedIndex = 0;
// [...]

// Creazione e salvataggio di una nuova misura (vuota)
                misura = new List<Movimento>();
                SalvaMovimenti();
                misure.Add(misura);

// Aggiornamento delle variabili di navigazione e relativi elementi grafici
                misuraAttuale++;
                numeroMisuraTxt.Text = Convert.ToString(misuraAttuale+1);
                totMisureLbl.Text = "/ " + Convert.ToString(misure.Count);
            }
```

```
// Azioni da compiere se l'utente sta visualizzando o aggiornando una
// misura già scritta
            else
            {

// Incremento della variabile "misuraAttuale" e relativa textbox
// "numeroMisuraTxt"
                misuraAttuale++;
                numeroMisuraTxt.Text = Convert.ToString(misuraAttuale+1);

// Invocazione del metodo "CaricaMovimenti" per visualizzare i campi
// della misura successiva
                CaricaMovimenti();
            }
        }
```

3.1.6 Pulsante "<<"

Il pulsante "<<", al contrario di ">>", è formato da poche istruzioni, che vengono eseguite alla stessa maniera indipendentemente dalla misura di posizionamento. Questo pulsante, quando viene premuto, si occupa di sovrascrivere la *misura* correntemente visualizzata e di aggiornare "*numeroMisuraTxt*"; in questo modo, l'esecuzione del metodo "*CaricaMovimenti( )*" avviene correttamente. Per impedire di visualizzare misure inesistenti è stata inserita una semplice condizione di "if": in questo modo il valore di "*numeroMisura"* non scenderà mai sotto uno.

```
// Descrizione delle azioni derivate dall'evento "button.click" del
// pulsante "<<"
        private void precedenteBtn_Click(object sender, EventArgs e)
        {
            int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text)-1;

// Creazione di una list di movimenti nominata "misura"
            misura = new List<Movimento>();

// Salvataggio dei movimenti all'interno della misura appena creata
            SalvaMovimenti();

// Sovrascrittura della misura visualizzata
            misure[misuraAttuale] = misura;

// Condizione di decremento della variabile "misuraAttuale" e
// aggiornamento di "numeroMisuraTxt"
            if (misuraAttuale > 0) misuraAttuale--; else return;
            numeroMisuraTxt.Text = Convert.ToString(misuraAttuale+1);

// Invocazione del metodo "CaricaMovimenti" per visualizzare i campi
// della misura successiva
            CaricaMovimenti();
        }
```

3.1.7 Pulsanti "|<" e ">|"

I due pulsanti "|<" e ">|" sono stati aggiunti per velocizzare la compilazione e la rilettura delle misure scritte. Il loro codice è estremamente semplice: consiste infatti nel modificare il valore di "*numeroMisuraTxt*" e nel richiamare il metodo "*CaricaMovimenti( )*". In base al pulsante selezionato, il valore di "*numeroMisuraTxt*" sarà impostato a "1" o associato al valore "*misure.count*".

```
// Pulsante "|<"
        private void inizioBtn_Click(object sender, EventArgs e)
        {

// Impostazione del nuovo valore di "numeroMisuraTxt"
            numeroMisuraTxt.Text = "1";

// Invocazione del metodo "CaricaMovimenti( )" per visualizzare la
// misura richiesta
            CaricaMovimenti();
        }
```

Codice di ">|":

```
// Pulsante ">|"
        private void fineBtn_Click(object sender, EventArgs e)
        {

// Impostazione del nuovo valore di "numeroMisuraTxt"
            numeroMisuraTxt.Text = Convert.ToString(misure.Count);

// Invocazione del metodo "CaricaMovimenti( )" per visualizzare la
// misura richiesta
            CaricaMovimenti();
        }
```

3.1.8 Salvataggio del file

Per consentire all'utente di esercitarsi nell'esecuzione di un brano, senza bisogno di scriverlo interamente ad ogni avvio di programma, è stata implementata la funzione di salvataggio e di caricamento[34]: la prima è stata inserita nel pulsante "*Salva*" mentre la seconda si avvia premendo il tasto "*Apri*".

Le istruzioni trascritte in "*Salva*" sono finalizzate alla creazione di un file di testo avente tutte le informazioni inserite durante la compilazione: il nome del brano, il numero di suddivisioni, il quantitativo di misure memorizzate e l'esatto contenuto di ognuna di esse[35].

Di seguito sono spiegate le funzioni e il relativo codice sorgente:

34 Chiamate anche "*load & save*" .
35 Un campo "*Tonica*" e uno "*Accordo*".

- salvataggio in memoria della misura visualizzata: questo comando è stato inserito per evitare che l'utente possa perdere le modifiche apportate alla misura,
- apertura di una finestra di dialogo e creazione di un flusso di dati chiamato "*stream*",
- inserimento del nome del file nella finestra di dialogo e avvio della scrittura,
- scrittura del nome del brano della prima riga del file,
- trascrizione del valore di "numeroSuddivisioni" nella seconda riga,
- copia del valore di "misure.Count"
- salvataggio, tramite un doppio ciclo "for", di tutti i movimenti del brano (ordinati per misure),
- chiusura del flusso di dati.

Una volta chiuso il flusso, nel caso di un corretto salvataggio sarà visualizzata una *messagebox* di avvenuta scrittura, altrimenti ne verrà aperta una riportante l'errore incorso.

```
// Pulsante "Salva" e funzione di salvataggio
       private void salva_Click(object sender, EventArgs e)
        {
            int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text) - 1;

// Sovrascrittura della misura visualizzata
            misura = new List<Movimento>();
            SalvaMovimenti();
            misure[misuraAttuale] = misura;

// Creazione di una finestra di salvataggio
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();

// Creazione di un flusso di dati chiamato "stream"
            Stream stream;

            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {

// Descrizione di una funzione "try & catch" per gestire eventuali problemi
                try
                {
                    if ((stream = saveFileDialog1.OpenFile()) != null)
                    {

// Scrittura del flusso di dati
                        StreamWriter writer = new StreamWriter(stream);

// Salvataggio del nome del brano nella prima riga di flusso
                        writer.WriteLine(nomeBranoTxt.Text);

// Inserzione del numero di suddivisioni nella seconda riga di flusso
                        writer.WriteLine(numeroSuddivisioni);
```

```
// Scrittura del numero di misure salvate riportando il valore
// di "misure.count"
                        writer.WriteLine(misure.Count);

// Il doppio ciclo "for" seguente è finalizzato al salvataggio delle toniche
// e accordi di ogni movimento di ogni misura
                        for (int i = 0; i < misure.Count; i++)
                        {
                            for (int j = 0; j < numeroSuddivisioni; j++)
                            {
                                writer.WriteLine(misure[i][j].Tonica);
                                writer.WriteLine(misure[i][j].Accordo);
                            }
                        }

// Dopo aver salvato tutte le informazioni utili, la fase di salvataggio
// sarà terminata
                        writer.Close();
                        stream.Close();
                    }

// Creazione di una messagebox informativa (in caso di errore)
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Errore in fase di scrittura, errore
originale: " + ex.Message);
                }

// Comparsa di una messagebox di avvenuta scrittura
            MessageBox.Show("Scrittura eseguita.");
            }
        }
```

3.1.9 Caricamento dei file

La funzione di caricamento, inserita nel *tasto* "*Apri*", è composta da molte più istruzioni rispetto a quella di salvataggio: oltre a dover caricare i dati già salvati, dovrà preparare il layout della form, ricreare ordinatamente la list "*misure*" e quindi assegnare ai movimenti caricati la propria misura di appartenenza. Analogamente a quanto fatto durante il salvataggio, in prima istanza si dovrà creare un oggetto finalizzato alla lettura del flusso di dati da caricare: ad ogni riga è associato un campo o un movimento specifico.

Per primo viene modificato il titolo della form con un'istruzione di tipo "*nomeBranoTxt.Text = stream.ReadLine( )*". A questo punto, il valore riportato in seconda riga (*numeroSuddivisioni)* viene letto e subito usato per modificare l'aspetto della form ed abilitare (o disabilitare) le combobox necessarie. Successivamente, verrà svolta la funzione principale: la ricostruzione della list "*misure*".

Per ottimizzare l'efficienza e la lunghezza del codice, è stato sfruttato un doppio ciclo "*for*":

quello esterno crea una nuova misura ad ogni ripetizione del ciclo[36], mentre quello interno, inserisce i movimenti (comprensivi di "*Tonica*" e "*Accordo*") nella misura appena creata. Una volta ricostruita la misura, questa sarà aggiunta alla list "*misure*".

Le rimanenti istruzioni del pulsante "*Apri*", dopo aver abilitato (o disabilitato) le combobox e le label utili, si occuperanno della gestione di eventuali errori e termineranno il proprio svolgimento mostrando la prima misura del brano caricato.

```
// Pulsante "Apri" e funzione di apertura
private void apri_Click(object sender, EventArgs e)
        {

// Apertura di una finestra di dialogo per il caricamento del file
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            StreamReader stream = null;

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                try
                {

// Creazione di una funzione di lettura del file
                    stream = new StreamReader(openFileDialog1.FileName);

// Assegnazione dei campi in base alla loro posizione nel flusso: prima
// riga = nomeBranoTxt
                    nomeBranoTxt.Text = stream.ReadLine();

// Seconda riga del flusso: numeroSuddivisioni
                    numeroSuddivisioni = Convert.ToInt32(stream.ReadLine());

// In base al valore di numeroSuddivisioni, imposta i campi abilitati
// nella form
                    if (numeroSuddivisioni == 2)
                    {
                        dueSuddivisioniRbt.Checked = true;

                        mov1TonicaCbb.Enabled = true;
                        mov1AccordoCbb.Enabled = true;
                        mov1Lbl.ForeColor = Color.FromArgb(0, 0, 0);
                        mov1AccLbl.ForeColor = Color.FromArgb(0, 0, 0);
                        mov1TonLbl.ForeColor = Color.FromArgb(0, 0, 0);

                  // [...]

                        mov3TonicaCbb.Enabled = false;
                        mov3AccordoCbb.Enabled = false;
                        mov3Lbl.ForeColor = Color.FromArgb(128, 128, 128);
                        mov3AccLbl.ForeColor = Color.FromArgb(128, 128, 128);
                        mov3TonLbl.ForeColor = Color.FromArgb(128, 128, 128);

                  // [...]
                    }
                    else if (numeroSuddivisioni == 3)
                    {   // [...]    }
```

36 L'istruzione viene ripetuta un numero di volte pari al numero delle misure salvate.

```
                    else if (numeroSuddivisioni == 4)
                    {   // [...]    }

                    else if (numeroSuddivisioni == 5)
                    {   // [...]    }

                    int totaleMisure = Convert.ToInt32(stream.ReadLine());

// Creazione di una nuova lista di liste chiamata "misure"
                    misure = new List<List<Movimento>>();

// Doppio ciclo "for" per inserire le misure e i movimenti nella
// lista "misure"
                    for (int i = 0; i < totaleMisure; i++)
                    {

// Creazione di una nuova misura
                        misura = new List<Movimento>();
                        for (int j = 0; j < numeroSuddivisioni; j++)
                        {
// Salvataggio dei movimenti caricati all'interno della misura
                            Movimento mov = new Movimento();
                            mov.Tonica = stream.ReadLine();
                            mov.Accordo = stream.ReadLine();
                            misura.Add(mov);
                        }

// Aggiunta della lista "misura" alla list di list "movimenti"
                        misure.Add(misura);
                    }

// Disabilitazione dei campi superflui
                    continuaBtn.Enabled = false;
                    nomeBranoTxt.Enabled = false;

                    dueSuddivisioniRbt.Enabled = false;
                    treSuddivisioniRbt.Enabled = false;
                    quattroSuddivisioniRbt.Enabled = false;
                    cinqueSuddivisioniRbt.Enabled = false;

// Modifica del colore dei campi superflui
                    nomeBranoLbl.ForeColor = Color.FromArgb(128, 128, 128);
                    selezionaSuddivisioniLbl.ForeColor = Color.FromArgb(128,
128, 128);

                    dueSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128,
128);
                    treSuddivisioniRbt.ForeColor = Color.FromArgb(128, 128,
128);
                    quattroSuddivisioniRbt.ForeColor = Color.FromArgb(128,
128, 128);
                    cinqueSuddivisioniRbt.ForeColor = Color.FromArgb(128,
128, 128);

// Abilitazione dei pulsanti utili
                    successivaBtn.Enabled = true;
                    precedenteBtn.Enabled = true;
                    playBtn.Enabled = true;
                    inizioBtn.Enabled = true;
                    fineBtn.Enabled = true;
```

```
// Modifica del colore delle label di navigazione
                    numeroMisuraLbl.ForeColor = Color.FromArgb(0, 0, 0);
                    totMisureLbl.ForeColor = Color.FromArgb(0, 0, 0);

// Riassegnazione del testo di "totMisureLbl"
                    totMisureLbl.Text = "/ " + Convert.ToString
(totaleMisure);
                }

// Gestione di eventuali errori
                catch (Exception ex)
                {
                    MessageBox.Show("Errore, il file non può essere letto,
errore originale: " + ex.Message);
                }

// Caricamento della prima misura tramite il metodo "CaricaMovimenti( )"
                numeroMisuraTxt.Text = "1";
                CaricaMovimenti();
            }
        }
```

3.1.10 Pulsante "Play"

La fase di compilazione si può considerare terminata dopo aver inserito e salvato tutte le misure dello spartito; a quel punto, tramite la pressione del pulsante "*Play*", il software sarà pronto e l'utente potrà passare alla fase di esecuzione real time.

Le istruzioni contenute nel bottone "*Play*" sono riepilogabili come segue:

- sovrascrittura dell'ultima misura,
- creazione, con il comando "*new*", di una nuova form chiamata "*Esecuzione*" (inizializzata durante la creazione della prima form),
- esportazione, con il frammento "*form2(misure)*", delle misure salvate in memoria,
- copia delle informazioni relative alle combobox abilitate ed esportazione delle variabili "*numeroSuddivisioni*" e "*misure.count*",
- invocazione del metodo "*form2.show( )*" e relativa apertura di form2,
- modifica del nome del titolo di form2 grazie all'istruzione "*form2.Text = ...*".

Per rendere funzionante la seconda metà del software, risulta necessario esportare nella form relativa all'esecuzione real time di alcuni parametri dichiarati ed usati nella prima. Tali parametri sono copiati durante l'inizializzazione della seconda form e sono:

- i movimenti abilitati in fase di scrittura,
- il numero di suddivisioni,
- una copia della struct “Misure”,
- il valore di “*misure.count*”.

```
// Descrizione delle azioni derivate dall'evento “button.click” del
// pulsante “Play”
        private void play_Click(object sender, EventArgs e)
        {
            int misuraAttuale = Convert.ToInt32(numeroMisuraTxt.Text) - 1;

// Sovrascrittura della misura visualizzata
            misura = new List<Movimento>();
            SalvaMovimenti();
            misure[misuraAttuale] = misura;

// Esportazione delle misure nella seconda form
            form2 = new Esecuzione(misure);

// Esportazione delle combobox abilitate ed esportazione di tale valore
// bool nella form2
            form2.mov1Enabled = mov1TonicaCbb.Enabled;
            form2.mov2Enabled = mov2TonicaCbb.Enabled;
//        [...]

// Oscuramento delle suddivisioni extra
            if (numeroSuddivisioni==2)
            {
                form2.mov3TonAccLbl.Hide();
                form2.mov3titleLbl.Hide();

                form2.mov4TonAccLbl.Hide();
                form2.mov4titleLbl.Hide();

                form2.mov5TonAccLbl.Hide();
                form2.mov5titleLbl.Hide();
            }

            if (numeroSuddivisioni == 3)
            {
                form2.mov4TonAccLbl.Hide();
                form2.mov4titleLbl.Hide();

                form2.mov5TonAccLbl.Hide();
                form2.mov5titleLbl.Hide();
            }

            if (numeroSuddivisioni == 4)
            {
                form2.mov5TonAccLbl.Hide();
                form2.mov5titleLbl.Hide();
            }

// Esportazione del valore della variabile “numeroSuddivisioni” e
// “misure.count” nella seconda form
            form2.totaleMisure = misure.Count;
```

```
            form2.suddivisioni = numeroSuddivisioni;

// Apertura della form2, intitolata con il nome presente nella
// textbox "nomeBranoTxt"
            form2.Show();
            form2.Text = "Suggeritore modale - " + nomeBranoTxt.Text;
        }
```

3.2 *Fase di esecuzione real time*

Una volta terminata la fase di compilazione, il software è in condizione di funzionare. La seconda form si presenta con i seguenti campi (vedi figura 11):

- una *textbox* (non modificabile) che indica il contenuto della misura visualizzata,
- un *numericUpDown* per impostare i bpm[37] del brano,
- una coppia di *label* (di cui una dinamica) per ogni movimento[38],
- due *pulsanti* di navigazione *"<<"* e *">>"*,
- un *pulsante* "*Start*".

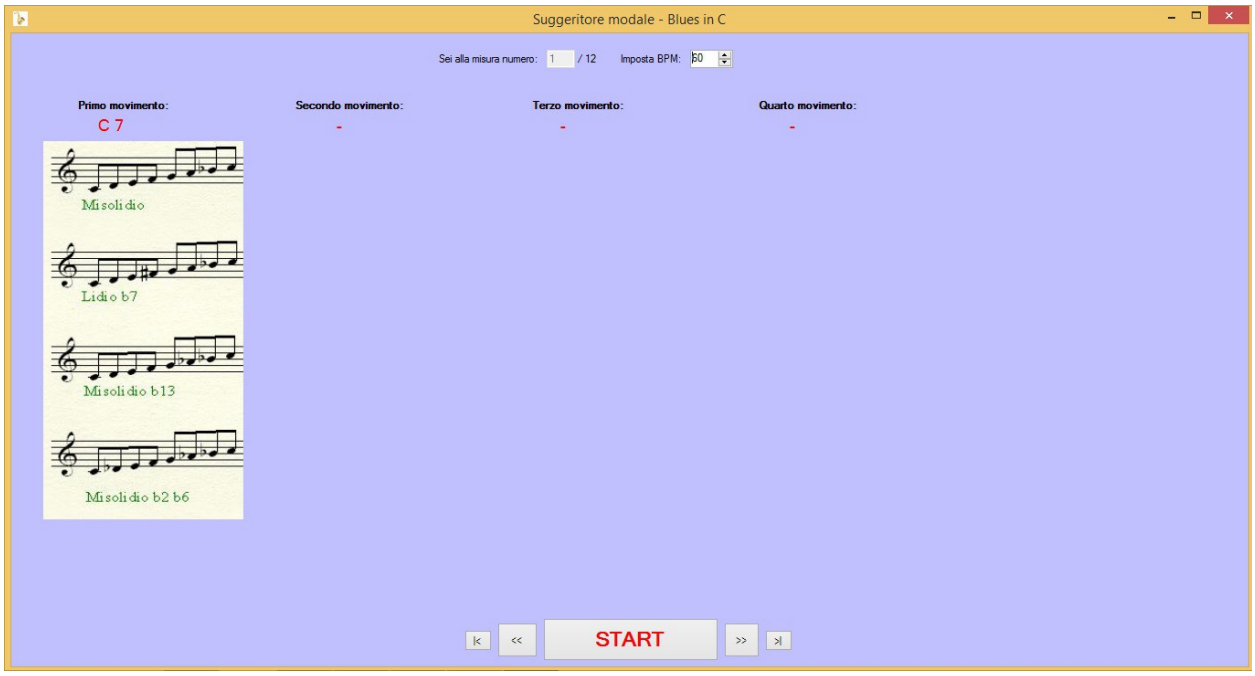


Figura 11: anteprima della seconda form in apertura

L'aspetto della seconda form è fortemente legato a quanto scritto in fase di progettazione: analizzando le righe di codice del pulsante "Play" della prima form, si può infatti notare la cancellazione delle etichette relative ai movimenti non utilizzati nel brano.

Il codice della seconda form si apre con la copia dei valori esportati in fase di progettazione[39] e la creazione di una nuova variabile globale: "*pulsazioniTrascorse*"[40]. Essa sarà fondamentale durante l'esecuzione per garantire il corretto scorrimento delle misure sullo schermo.

37 Battiti Per Minuto. Il valore di default è 60.
38 Le label relative al quinto movimento non sono visibili perché oscurate: la seconda form mostra solo il numero di suddivisioni stabilite nella prima.
39 Movimenti abilitati, *list* "*Misure*" e le variabili "*Suddivisioni", "misuraAttuale"* e "*totaleMisure"*.
40 Inizializzata con valore iniziale uguale a zero.

Infine, per visualizzare il contenuto della prima misura, durante il caricamento della seconda form viene invocata la funzione di caricamento e viene impostato il testo di "*misureTotLbl*" (uguale al valore di "*totaleMisure*")[41].

Di seguito sono riportate le prime righe di codice della seconda form:

```
// Intestazione
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Media;[42]

// Inizializzazione di namespace e  Form
namespace WindowsFormsApplication1
{
    public partial class Esecuzione : Form
    {
// Importazione dei parametri provenienti dalla prima form
        public List<List<Progettazione.Movimento>> misure;

        public bool mov1Enabled;
        public bool mov2Enabled;
        public bool mov3Enabled;
        public bool mov4Enabled;
        public bool mov5Enabled;

// Creazione di variabili globali
        public int suddivisioni;
        public int misuraAttuale;
        public int totaleMisure;
        int pulsazioniTrascorse = 0;

// Importazione della lista "Movimento" e "misure"
        public Esecuzione(List<List<Progettazione.Movimento>> _misure)
        {
           InitializeComponent();
           misure = _misure;
           misuraAttuale = Convert.ToInt32(misuraNumeroTxt.Text);
        }
// Caricamento della form
        private void Form2_Load(object sender, EventArgs e)
        {
            textBox1_TextChanged(null, null);
            misureTotLbl.Text = "/ " + Convert.ToString(totaleMisure);
        }
```

41 A differenza della prima form, il testo della label contenente il numero di misure risulta statico poiché non viene mai modificato.

42 La libreria "System.Media" è stata aggiunta a quelle presenti per garantire l'utilizzo di file audio.

3.2.1 Pulsante “Start”

Il pulsante “*Start*” regola la gestione del *timer*, e di conseguenza il metronomo: premendo questo tasto si può avviare o interrompere la fase di esercitazione real time. Per rendere più intuitivo l'utilizzo del *Suggeritore modale* e liberare lo schermo da pulsanti superflui, è stato deciso di utilizzare un solo *bottone* per la gestione di entrambe le funzioni; le istruzioni da eseguire variano quindi in base al testo del pulsante: quando il timer è fermo, il testo del bottone sarà “*Start*”, viceversa “*Stop*”.

Premendo il pulsante in fase di fermo, sarà subito necessario verificare che i battiti per minuto siano stati inseriti correttamente; in caso contrario sarà visualizzata una messagebox di richiamo. In seguito a questa verifica preliminare, sarà assegnato il valore “zero” alla variabile “*pulsazioniTrascorse*” (in modo da iniziare sempre l'esecuzione di una battuta dal primo movimento), avviato il timer e modificato il testo del pulsante in *“Stop”*.

L'ultimo importantissimo elemento da prendere in considerazione è il decremento del valore di “*misuraAttuale*”. Questa operazione non ha alcuna conseguenza grafica ma, eseguendo questa istruzione, la misura di partenza (quella visualizzata durante la pressione del tasto “*Start*”) sarà letta e suonata due volte dal metronomo: in questo modo, l'esecuzione inizierà con la “*misura in levare*” tipica della musica d'insieme[43].

Qualora il tasto “*Start*” fosse premuto durante l'esecuzione, si verificherà il solo arresto del timer e la modifica del testo del pulsante da “*Start*” a“*Stop*”.

```
// Gestione dello START & STOP
        private void start_Click(object sender, EventArgs e)
        {

// Azioni da compiere se il timer è IN FUNZIONE
            if (timerBtn.Text == "STOP")
            {

// Stop del timer e modifica del testo del pulsante
                timer1.Stop();
                timerBtn.Text = "START";
            }

// Azioni da compiere se il timer è FERMO
            else
            {

// Messagebox d'errore in caso di omissione del valore “BPM”
              if (bpmNup.Text == "")
                {
                    MessageBox.Show("Impostare i BPM");
```

43 La “misura in levare” fornisce all'utente il tempo necessario per capire la velocità d'esecuzione e posizionare le mani sullo strumento.

```
                    return;
                }
                else
                {
// Misura in levare
                    misuraAttuale = misuraAttuale-1;
                    pulsazioniTrascorse = 0;

// Avvio del timer e modifica del testo del pulsante
                    timer1.Start();
                    timerBtn.Text = "STOP";
                }
            }
        }
```

3.2.2 Scadenza del timer

L'evento "*timer.Tick*", contiene le istruzioni da compiere ad ogni scadenza di intervallo; questo oggetto costituisce il primo di una catena di eventi necessaria per il corretto funzionamento della fase improvvisativa.

La principale funzione del timer, essendo la rappresentazione del metronomo, consiste nel suonare il tipico "*click*" di pulsazione e, nel caso del *Suggeritore modale*, nello "sfogliare" lo spartito elettronico mostrando all'utente la misura in esecuzione. Le istruzioni da eseguire, quindi, varieranno in base al punto di riproduzione dello spartito.

Per gestire efficacemente l'algoritmo, il metronomo avvia la propria esecuzione accertando che la pulsazione da eseguire non sia estranea alla misura: tale verifica è compiuta dall'istruzione "*if (pulsazioniTrascorse < suddivisioni)*". In caso di risposta "*true*", il metronomo si limiterebbe a suonare la pulsazione, richiamando il metodo "*playClick( )*"[44], ed incrementare il valore di "*pulsazioniTrascorse*" con il comando "*pulsazioniTrascorse++*". Dopo aver eseguito tutta la misura, la condizione iniziale di "*if*" risulterà ovviamente "*false*", rendendo necessario verificare se la misura appena riprodotta sia o meno l'ultima. La risposta "*true*" a questa seconda interrogazione decreterebbe l'arresto del timer e la modifica del testo di "*startBtn"* da "*Stop*" a "*Start*". Viceversa, non trattandosi dell'ultima misura, saranno ricreate le condizioni necessarie per eseguire dall'inizio la nuova battuta: incremento del valore di "*misuraAttualeTxt*" (con conseguente caricamento dei movimenti successivi), modifica del valore di "*pulsazioniTrascorse = 1*" e riproduzione di un click tramite l'invocazione di "*playClick( )*".

44 Maggiori approfondimenti nel paragrafo seguente

Codice di “timer.tick”:

```
// Imposto le azioni da eseguire ogni volta che scade il timer
        private void timer_Tick(object sender, EventArgs e)
        {

// Azione da compiere se il metronomo non ha terminato di suonare la misura
            if (pulsazioniTrascorse < suddivisioni)
            {

// Incremento delle pulsazioni trascorse e invocazione del metodo
// “playClick( )” per fare suonare il click.
                pulsazioniTrascorse++;
                playClick();
            }

// Azione da compiere se il metronomo si trova all'ultima misura
            else if (misuraAttuale == totaleMisure)
                {

// Stop del timer, modifica del testo del pulsante “Play”,
// interruzione dell'esecuzione
                    timer1.Stop();
                    startBtn.Text = "START";
                    return;
                }

// Azione da compiere se il metronomo ha terminato di eseguire una misura
// e deve procedere con quella successiva
                else
                {
// Incremento della variabile “misuraAttuale”, aggiornamento della
// textbox “misuraNumero”, modifica del valore della variabile
// “pulsazioniTrascorse” ed invocazione del metodo “playClick ( )”
                    misuraAttuale++;
                    misuraNumeroTxt.Text = Convert.ToString(misuraAttuale);
                    pulsazioniTrascorse = 1;
                    playClick();
                }
            }
```

3.2.3 Metodo “*playClick( )*”

Il metodo “*playClick( )*”, richiamato allo scadere del timer, è stato inserito nel codice per riprodurre il tipico “click” del metronomo e dare all'utente la percezione della velocità d'esecuzione. La durata di ogni pulsazione viene stabilita durante l'invocazione del metodo ed è basata sul valore riportato nella casella numeric up-down “*bpmNup*”.

Dovendo dialogare con un software, per esprimere la durata dell'intervallo si rende necessario indicare la distanza temporale, espressa in millisecondi, tra una pulsazione e l'altra anziché riportare direttamente il numero di pulsazioni per minuto. L'algoritmo risultante è quindi riassumibile con “*(60 / bpm) * 1000*”. L'algoritmo di calcolo viene eseguita ad ogni invocazione

del metodo "*playClick( )*", in questo modo sarà possibile modificare la velocità di riproduzione anche in fase di esecuzione.

```
//Inizializzazione dello stacchetto e dell'intervallo
        public void playClick()
        {

// Impostazione dell'intervallo del metronomo
            decimal intervallo = (60 / Convert.ToDecimal(bpmNup.Text)) *
1000;
            timer1.Interval = Convert.ToInt32(intervallo);

// Creazione di un elemento chiamato "click" ed assegnazione di un file wave
            SoundPlayer click = new SoundPlayer("click.wav");

// Invocazione di un metodo per l'esecuzione di "click"
            click.Play();
        }
```

3.2.4 Pulsanti di navigazione: "|<", "<<", ">>" e ">|"

I pulsanti di navigazione della seconda form, contengono un codice decisamente più semplice rispetto all'implementazione fatta in fase di programmazione poiché il loro compito è limitato a una funzione di richiamo: dopo aver controllato l'esistenza della misura richiesta (per i tasti "<<" e ">>"), verrà modificato il valore di "*misuraAttuale*" e aggiornato il testo di "*misuraNumeroTxt*"; in questo modo si innesterà un evento di "*textchange*" e sarà visualizzata la misura corrispondente. Questi pulsanti sono stati inseriti anche in fase improvvisativa per consentire all'utente di scegliere il punto di partenza dell'esercitazione; così facendo, l'esecutore potrebbe ripetere (o saltare) i passaggi di particolare interesse.

Nel caso del pulsante ">>", l'istruzione legata all'evento "*button.click*" sarà eseguita nel solo caso in cui il valore di "*misuraAttuale*" non fosse uguale a quello di "*totaleMisure*" (cioè "*misure.count*"); in caso contrario, trovandosi all'ultima misura creata, l'istruzione non verrebbe svolta.

```
//------Pulsante ">>"
        private void button4_Click(object sender, EventArgs e)
        {

// Creazione della variabile "misuraAttuale"
            int misuraAttuale = Convert.ToInt32(misuraNumeroTxt.Text);

/* Se la misura visualizzata corrisponde all'ultima, l'istruzione non verrà
eseguita, altrimenti verrà aumentato il valore di misura attuale ed
aggiornata la textbox "misuraNumeroTxt"
*/
            if (misuraAttuale == totaleMisure) return;
```

```
            else
            {
                misuraAttuale++;
                misuraNumeroTxt.Text = Convert.ToString(misuraAttuale);
            }
        }
```

Analogamente al tasto “>>”, il pulsante “<<” non svolgerà la funzione di decremento e aggiornamento nel caso in cui la misura visualizzata fosse la prima.

```
//------Pulsante "<<"
        private void button1_Click(object sender, EventArgs e)
        {

// Creazione della variabile “misuraAttuale”
            int misuraAttuale = Convert.ToInt32(misuraNumeroTxt.Text);

// Se la misura visualizzata corrisponde alla prima, l'istruzione non
// verrà eseguita
            if (misuraAttuale == 1) return;
            else
            {

// Altrimenti la variabile “misuraAttuale” verrà decrementata  e
// sarà aggiornata la textbox “numeroMisura”
                misuraAttuale--;
                misuraNumeroTxt.Text = Convert.ToString(misuraAttuale);
            }
        }
```

Una volta terminata la prima esecuzione, per poter ripetere velocemente il brano dall'inizio, è stato inserito un pulsante di riavvio: la pressione di tale bottone riporta l'utente alla prima misura dello spartito, consentendogli di cominciare da capo l'esecuzione. L'istruzione inserita è una semplice modifica del testo di “*misuraNumeroTxt*”, la quale innescherà l'evento di “*textchange*” e visualizzerà nuovamente la prima battuta sullo schermo.

```
// Pulsante “|<”
private void primaBtn_Click(object sender, EventArgs e)

        {
// Impostazione del nuovo valore di “misuraNumeroTxt”
            misuraNumeroTxt.Text = "1";
        }
```

L'ultimo tasto di navigazione inserito ha lo scopo di mostrare l'ultima battuta del brano. Analogamente a “|<”, la pressione del pulsante modifica il testo di “*misuraNumeroTxt*” associandole il valore di “*totaleMisure*” (“*misure.count”*).

```
// Pulsante ">|"
private void ultimaBtn_Click(object sender, EventArgs e)
        {

// Impostazione del nuovo valore di "misuraNumeroTxt"
            misuraNumeroTxt.Text = Convert.ToString(totaleMisure);
        }
```

3.2.5 Evento "*textchange*": caricamento misure, gestione delle *label* e modifica delle *immagini*

L'evento "*textchange*" associato a "*misuraNumeroTxt*", viene attivato ad ogni modifica al testo di quest'ultima ma, essendo la *textbox* disabilitata alle modifiche esterne, il suo valore può mutare in due sole occasioni: interagendo, a timer fermo, con i pulsanti di navigazione, o attendendo che, in fase di esecuzione, la funzione "metronomo" incrementi il valore di "*misuraAttuale"*.

Le istruzioni associate al textchange sono riepilogabili in tre passaggi:

- caricamento dei movimenti,
- modifica del testo della label,
- aggiornamento della picturebox.

Queste istruzioni, eseguite in sequenza, sono gli ultimi anelli della catena avviata in fase di programmazione; la loro conclusione corrisponde al caricamento temporizzato sullo schermo dell'immagine relativa all'accordo in esecuzione.

Per evitare di caricare accordi appartenenti ad altre misure, si è posta la condizione iniziale "*if (mov1enabled == true)"*; nel caso di risposta "*false*", le istruzioni non saranno eseguite. La variabile "*mov1Enabled*", importata dalla prima form, risulta "*true*" qualora la combobox "*mov1TonicaCbb*" sia agibile, indicando che il brano prevede la presenza di quel movimento. Una volta accertata la sua esistenza nella partitura (condizione "true"), la prima istruzione da svolgere consiste nel richiamare i campi "*tonica*" e "*accordo*" dei movimenti abilitati: vengono create due variabili locali di tipo "*string*" nominate "*primaTonica*" e "*primoTipo*"[45]. Il testo di tali variabili è prelevato direttamente dalla memoria del programma invocando con precisione la tonica e l'accordo desiderato grazie a una funzione di richiamo di tipo "*misure[misuraAttuale -1] [0].Tonica*" e "*misure[misuraAttuale -1][0].Accordo*". Nel caso di un movimento vuoto, la lettura di "*primaTonica + primoTipo*" restituirà una stringa vuota; al contrario, il caricamento di

45 Nel caso del primo movimento.

un accordo avente una tonica ed un tipo, darà in uscita il nome della quadriade per intero[46].

La seconda fase dell'evento textchange, consiste nella modifica del testo della label dinamica riportata sopra ad ogni movimento. L'assegnazione di una string piuttosto che un'altra è regolata dalla condizione "*if (primaTonica + primoTipo == "")*": nel caso di una stringa vuota, dovuta all'assenza di un accordo in quel movimento, il testo sarà "-"; viceversa, l'esecuzione di "*else*" assegnerà alla label il nome della quadriade ricavata.

L'ultimo anello della catena di istruzioni, consiste nel caricamento temporizzato di un'immagine contenente le scale utilizzabili durante l'improvvisazione: esso si basa sul testo della label appena modificata e, eseguendo l'istruzione "*mov1Img.Image = Image.FromFile(mov1TonAccLbl.Text + ".jpg")*", preleva dal computer l'immagine .jpg dal nome indicato. Nella cartella contenente i file del programma sono state inserite tutte le immagini relative agli accordi più un'immagine neutra, dello stesso colore dello sfondo, di nome "-"; in questo modo, incorrendo in un movimento vuoto, l'istruzione non genererà mai errore e, al cambio di misura, non sarà visualizzato un accordo sbagliato o antecedente.

```
// Eventi relativi al textchange di "misuraNumeroTxt"
       private void misuraNumeroTxt_TextChanged(object sender, EventArgs e)
        {
             misuraAttuale = Convert.ToInt32(misuraNumeroTxt.Text);

// Verifica dell'esistenza del movimento
            if (mov1Enabled == true)
            {

// Funzione di richiamo dalla memoria
                string primaTonica = misure[misuraAttuale - 1][0].Tonica;
                string primoTipo = misure[misuraAttuale - 1][0].Accordo;

// Assegnamento del testo della label
                if (primaTonica + primoTipo == "") mov1TonAccLbl.Text = "-";
                else mov1TonAccLbl.Text = primaTonica + " " + primoTipo;

// Caricamento dell'immagine relativa
                mov1Img.Image = Image.FromFile(mov1TonAccLbl.Text + ".jpg");
            }
// [...]
        }
```

46 Ad esempio "E min7".

Screenshot del *Suggeritore modale* durante l'esecuzione di un Blues in C.

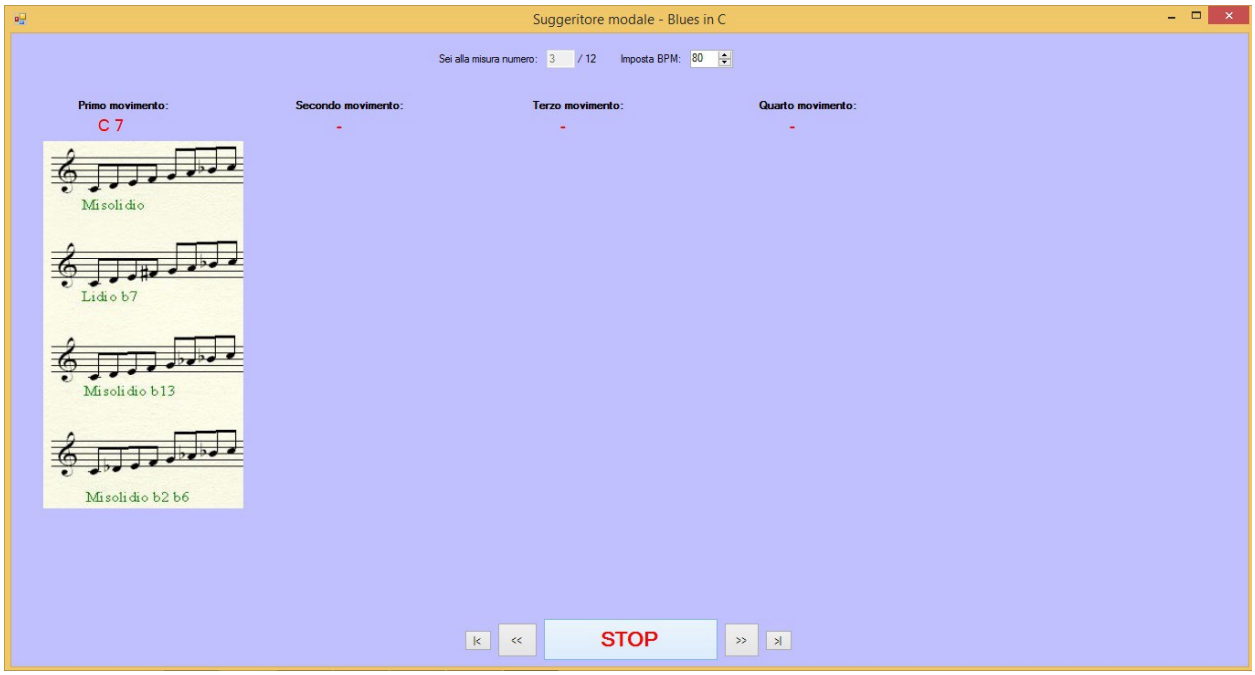


Figura 12: Suggeritore modale in fase improvvisativa.

Capitolo 4: Conclusioni

Dopo diverse modifiche e rielaborazioni al codice e all'interfaccia grafica, il risultato finale del *Suggeritore modale* è da considerarsi pienamente soddisfacente. Sono state rispettate le aspettative e, dopo qualche sessione di prova con studenti volontari, sono emerse le seguenti valutazioni:

- la grafica è semplice ma appagante,
- l'utilizzo del software è estremamente intuitivo sia in fase di compilazione che di esecuzione,
- l'elemento visivo permette all'allievo di ricordarsi più facilmente quali scale siano state già usate e quali no, evitando di ripetersi,
- anche i meno esperti del genere, dopo poche esecuzioni, sono riusciti ad inserire qualche sonorità jazzistica nei soli.

4.1 Sviluppi futuri

Trattandosi di un prototipo, il *Suggeritore modale* presenta margini di miglioramento. Poichè questa versione è stata ideata per i neofiti, si è preferito non inserire troppi elementi per evitare di creare confusione nello studente.

Tuttavia, al fine di portare il software nelle mani di musicisti più esigenti, è bene raccogliere i punti deboli rilevati spiegando il perché di tali scelte:

- il *Suggeritore modale* richiede uno schermo grande, costringendo all'esecuzione su un computer o un tablet anziché un telefono cellulare.
    - Lo schermo grande è necessario per visualizzare le scale in maniera leggibile: un'ipotetica implementazione su cellulare dovrebbe necessariamente fare a meno dei frammenti visivi e riportare il solo nome delle scale compatibili.
- impossibilità di creare loop per esercitarsi ad libitum su determinati frammenti.
    - Questa possibilità è stata scartata per evitare di lasciare troppa libertà allo studente, il quale dovrebbe esercitarsi sotto la tutela del docente. Tuttavia, rappresenta un ottimo

spunto di miglioramento.

- mancata implementazione di scale famose come la *Blues,* la *Pentatonica,* l'*Esatonale* e l'*Ottofonica,*
    - La loro assenza è volontaria: il *Suggeritore modale* arriva a fornire fino a cinque scale per accordo; inserirne di altre avrebbe creato sicuramente confusione nello studente.
- assenza di un player di sottofondo per l'esecuzione degli accordi,
    - L'implementazione di un player è raramente soddisfacente se non ad altissimi livelli; si è quindi preferito evitare, lasciando questa responsabilità al docente.
- non sono previste tensioni oltre la settima,
    - i gradi di nona, undicesima e tredicesima, così come le scale *Blues* e *Ottofonica*, sono stati volutamente omessi per non introdurre troppo materiale e creare confusione nello studente neofita.

Bibliografia e Sitografia

[1] Antonio Agliata - Alessandro Forte - Paolo Gambardella, "*Programmare in C# partendo da zero*", edizionefutura, Castelfidardo 2012

[2] Bochicchio Daniele - Civera Cristian - De Sanctis Marco - Golia Riccardo "C#5, guida completa per il programmatore", Hoepli, Milano 2013

[3] Mattia G. Bergomi – Simone Geravini, "*I modi delle scale*", casa musicale eco, Monza 2012

[4] Walter Piston, "*Armonia*", Edt, New York 1987

[5] Sergio Bianchi, "*Manuale teorico-pratico di armonia*", pro-manuscripto, Lecco 2012

[6] http://it.wikipedia.org/wiki/Jit

[7] http://msdn.microsoft.com/

Appendici: Manuale d'istruzione

L'interfaccia grafica del *Suggeritore modale* è stata creata per rendere il programma di facile utilizzo anche agli utenti con poca dimestichezza nel campo informatico. Tuttavia, per garantire lo sfruttamento delle piene potenzialità, viene di seguito riportato un veloce riepilogo delle fasi salienti in cinque punti.

Fase 1: Una volta aperto il *Suggeritore modale*[47], prima di iniziare a inserire gli accordi dello spartito, è necessario compilare il campo "*Nome del brano*" e selezionare il numero di suddivisioni[48]; una volta fatto, premendo il tasto "*Continua*" si possono iniziare a inserire gli accordi.

Fase 2: uno spartito come "*My favorite things*"[49] prevede la presenza di un accordo nel primo movimento di ogni misura. Volendo trascrivere questo brano, bisognerà iniziare cercando sotto la voce "*Primo movimento*" il campo di selezione multipla "*Tonica*", aprirlo, selezionare "*E*"[50] dal menù a tendina e ripetere il procedimento per "*Accordo*" (inserendo "*min7*"). Una volta inseriti analogamente tutti i movimenti della misura, sarà possibile procedere alla compilazione di quella successiva; per salvare la misura scritta e prepararsi alla compilazione di quella seguente, sarà invece necessario premere il tasto ">>". La misura può essere ugualmente salvata schiacciando il pulsante *"<<",* "*Play*" e "*Salva*". Dopo aver inserito tutte le misure, premendo il tasto "*Play*" sarà possibile iniziare l'esercitazione, tuttavia, è consigliabile salvare lo spartito trascritto premendo il tasto "*Salva*" prima di passare alla Fase3.

Nel caso in cui si volesse interpretare un brano già salvato, premendo "Apri" sarebbe possibile selezionare la composizione scelta e saltare le due fasi appena scritte.

Fase 3: la seconda finestra mostrerà gli accordi delle misure inserite con le relative scale consonanti; premendo i tasti di navigazione è possibile visionare preventivamente le scale e le

47 Vedi figura 9.
48 I valori inseriti sono adatti per più tipi di suddivisione, ad esempio, il "2" è adatto per brani in "6/8 – 2/2 – 2/4 - C tagliato e 6/4".
49 Vedi figura 1.
50 Nella nomenclatura anglosassone la lettera "E" corrisponde alla nota "Mi".

misure scritte[51]. Prima di avviare la fase di esecuzione con il tasto "Start", è consigliabile impostare la velocità di riproduzione selezionando dal casellino numerico i BPM relativi.

Fase 4: posizionando il lettore sulla misura di maggior utilità, è possibile iniziare a suonare da quel punto, evitando così di ripetere passaggi già memorizzati o di minore interesse. È inoltre utile notare che la pressione del tasto "Start" avvii la riproduzione partendo "una misura in levare" rispetto a quella visualizzata. In questo modo, il musicista avrà il tempo di posizionare le mani sullo strumento e prepararsi ad attaccare.

Fase 5: esercitarsi fino al pieno apprendimento delle scale.

51 Vedi figura 11.