Anno accademico 2018-19
Progetto di Computer Music & Advance Coding Tools and Methodologies
Prof. Augusto Sarti, Francesco Bruschi
#MAEPolimi

HARMONEASY

Documentazione del progetto HTML/JavaScript

HTML/CSS - Vex Flow Library

Il DOM è composto da due div di contorno statiche e tre centrali, contenenti rispettivamente: interfaccia comandi, spartito, area di log.
I file play.css e switches.css contengono rispettivamente lo stile della pagina e il funzionamento degli switch.
L'applicazione si regge sulla libreria VexFlow (http://www.vexflow.com/), che offre delle funzioni per disegnare partiture all'interno di canvas.

Gestione del MIDI

Tramite l' API Web MIDI (http://webaudio.github.io/web-midi-api/) il browser in uso (https://caniuse.com/#feat=midi) è in grado di rilevare ed, insieme all'applicazione, di gestire i segnali MIDI in uscita dalla tastiera/controller MIDI connesso al computer.

Del segnale MIDI vengono processati solo i messaggi di *NoteOn* e *NoteOff* le quali funzioni, oltre al render della nota MIDI sul pentagramma dinamico (quello a sinistra), inseriscono/disinseriscono la nota MIDI nell'/dall' array *tempChord*, la struttura dati intelligente (v. a seguire) che verrà processata per la rilevazione dell'accordo.

L'inserimento in *tempChord* di ogni nota MIDI rilevata viene gestito tramite la funzione ricorsiva *recursiveMidiNoteDetection(midiNote)* ,

```
function recursiveMidiNoteDetection(midiNote){
  if(tempChord.length==0){
    return midiNote;
  } else if(midiNote < Math.min.apply(null,tempChord)+12 &&
            midiNote > Math.min.apply(null,tempChord)-12) {
    return midiNote;
  } else if (midiNote > Math.min.apply(null,tempChord)) {
    return recursiveMidiNoteDetection(midiNote - 12);
  } else {
    return recursiveMidiNoteDetection(midiNote + 12);
  }
}
```

che ha il compito di “stringere le parti” in modo tale di inserire la nota nell’ottava più vicina al basso; in seguito si verifica se la nota inserita non è un raddoppio di voce ed in quel caso si elimina intelligentemente il raddoppio:

```
for (var i = 0; i<tempChord.length; i++){
  for(var j = 0; j<tempChord.length; j++){
    if(i!=j && (tempChord[i]==tempChord[j] || Math.abs(tempChord[i] - tempChord[j])%12==0)){
      console.log("Doubled voice!");
      if(tempChord[i]>tempChord[j]) {
        doubledVoice.push(tempChord[i]);
        tempChord.splice(i, 1); //remove the doubled voice from tempChord (higher one)
      } else if (tempChord[i]<tempChord[j]) {
        doubledVoice.push(tempChord[j]);
        tempChord.splice(j, 1);
      } else { //tempChord[i]==tempChord[j]
        doubledVoice.push(tempChord[i]);
        tempChord.splice(i, 1);
      }
    }
  }
}
```

In conclusione, con lo scopo di aiutare al massimo la successiva individuazione dell’accordo, *tempChord* risulterà la forma a parti strette e senza raddoppi dell’accordo suonato.

Individuazione dell’accordo

Il passaggio successivo è quello di definire l’array (*tempChordInterval*) contenente gli intervalli in semitoni fra le note di *tempChord:*

```
//[60, 64, 63] -> [4, 3]
```

L’array risultante viene poi confrontato con una serie di array rappresentanti le sequenze di intervalli delle triadi e quadriadi scelte da noi come set degli accordi possibili che l’applicazione può rilevare.

```
// BICHORD
var fifth = [7];
var majorThird = [4];
var minorThird = [3];
var fourth = [5];
```

```
//TRIAD
var majorTriadRoot = [4, 3];
var majorTriadFirst = [3, 5];
var majorTriadSecond = [5, 4];
```

(ecc)

```
//TETRACHORD (QUADRIADI)
var minor7Root = [3,4,3];   //3,4,3,2
var minor7First = [4,3,2];
var minor7Second = [3,2,3];
var minor7Third = [2,3,4];
var major7majRoot = [4,3,4];      //4,3,4,1
```

(ecc)

L’applicazione gestisce il rilevamento dei rivolti degli accordi ne consegue, quindi, anche la gestione di eventuali ambiguità (presenti solo nelle triadi).

```
var stringTemp = JSON.stringify(tempChordInterval)
switch (stringTemp) {
  //TRIAD
  case JSON.stringify(minorTriadRoot):
    nameChord = ( notes[(tempChord[0]%12)] + " minor triad root /" +
                notes[(tempChord[1]%12)] + " major 6 no 5 second riv");
    break;
```

Rilevamento della tonalità

Individuare la tonalità in itinere con la sequenza di accordi è fondamentale ai fini di effettuare una localizzazione corretta degli errori armonici; così importante quanto ostica, in particolare la realizzazione della gestione di tutti i casi possibili e la messa a punto di algoritmi che traducono un aspetto ed un linguaggio musicale non sempre definito e spesso leggibile correttamente in più “chiavi” diverse.

In primis, la struttura dati protagonista di cui ci siamo avvalsi è stato il circolo delle quinte: tanto ovvia come scelta quanto sorprendente la sua capacità di adattarsi “matematicamente” all’interno dei nostri algoritmi e in generale al contesto informatico.

```
tonalMat = [
//[C,#,D,#,E,F,#,G,#,A,#,B] C
  [1,0,1,0,1,1,0,1,0,1,0,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] G
  [1,0,1,0,1,0,1,1,0,1,0,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] D
  [0,1,1,0,1,0,1,1,0,1,0,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] A
  [0,1,1,0,1,0,1,0,1,1,0,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] E
  [0,1,0,1,1,0,1,0,1,1,0,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] B
  [0,1,0,1,1,0,1,0,1,0,1,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] F#
  [0,1,0,1,0,1,1,0,1,0,1,1],
//[C,#,D,#,E,F,#,G,#,A,#,B] C#
  [1,1,0,1,0,1,1,0,1,0,1,0],
//[C,#,D,#,E,F,#,G,#,A,#,B] Ab
  [1,1,0,1,0,1,0,1,1,0,1,0],
//[C,#,D,#,E,F,#,G,#,A,#,B] Eb
  [1,0,1,1,0,1,0,1,1,0,1,0],
//[C,#,D,#,E,F,#,G,#,A,#,B] Bb
  [1,0,1,1,0,1,0,1,0,1,1,0],
//[C,#,D,#,E,F,#,G,#,A,#,B] F
  [1,0,1,0,1,1,0,1,0,1,1,0]
];
```

L’algoritmo si spezza in due rami: il primo assegna la tonalità al primo accordo che inizia la sequenza, il secondo ramo rileva la tonalità dagli accordi successivi.

Per quanto riguarda l'assegnazione della tonalità al primo accordo, pensando l'applicazione come uno strumento di risoluzione di esercizi armonici, abbiamo escluso i casi estremi mai visibili nei classici esercizi di armonia classica; i casi gestiti ai quali viene sempre assegnata una tonalità sono:

- accordo maggiore e minore
- accordo di settima di dominante (sebbene sia raro che possa risultare come primo accordo di un esercizio armonico)

In seguito, ogni accordo successivo viene "parsato" all'interno della funzione *isSameTonality(tonalityIndex)*, alla quale viene passato l'indice della riga della matrice *tonalMat* corrispondente alla tonalità attuale, e quindi verificando se l'accordo rientra all'interno della tonalità o meno.

```
function isSameTonality(tonalMatIndex){
  alteredNotes = [];
  var isAltered = false;
  for (var i=0; i<tempChord.length; i++ ){
    var note = midiToNote(tempChord[i]);
    note = note.toString();
    note=note.slice(0, note.indexOf('/')); // remove reference to the octave 'c/5' -> 'c'
    if (tonalMat[tonalMatIndex][names.indexOf(note)]!=1){ //the tempChord note isn't in the tonality
      alteredNotes.push(names.indexOf(note));
      isAltered = true;
      break;
    }
  }

  if (isAltered)
    return false;
  else
    return true;
}
```

Se l'accordo possiede una o più note fuori dal set di note della tonalità attuale (corrispondenti agli 1 della relativa riga di *tonalMat*) la funzione ritorna falsa e dunque sta avvenendo una modulazione.

Il passaggio successivo evidenzia la versatilità e potenza matematica del circolo delle quinte.
L'accordo modulante viene iterativamente "riparsato" all'interno della funzione *isSameTonality(tonalityIndex)*, alla quale viene passata l'indice delle tonalità vicine (senso orario e senso antiorario alternatamente). La prima tonalità che matcha con l'accordo viene scelta quindi come tonalità di arrivo.

```
for (var i=0; i<6; i++){

  if((tonalityIndex-flat)<0)
    temp = 12 + (tonalityIndex-flat);
  else
    temp = tonalityIndex-flat;

  if(isSameTonality((tonalityIndex + sharp)%12)){ //clockwise on the circle of the fifths
    tonalityIndex = (tonalityIndex + sharp)%12;
    tonality = tonalName[tonalityIndex];
    break;
  } else if(isSameTonality(temp)){ //counterclockwise on the circle of the fifths
    tonalityIndex = temp;
    tonality = tonalName[tonalityIndex];
    break;
  }
  sharp++;
  flat++;
}
```

L'algoritmo gestisce (non ci addentreremo nella spiegazione in questa documentazione) inoltre tutti i casi anomali ed ambiguità determinate dalla natura dell'accordo modulante e dalle numerose chiavi di lettura che ne derivano da questo argomento.

Analisi del moto delle voci e localizzazione degli errori armonici

L' analisi del moto delle parti prende in esame gli ultimi due accordi rilevati e ne ricava un vettore dei moti, più una variabile specifica per il canto.

```
// MOTION
var motion = new Array(newlen);
if( oldlen == newlen)
  for(var i=0; i<newlen; i++){
    motion[i] = (newChord[i]-oldChord[i]);
    //console.log("---> motion "+i+": "+newChord[i]+" - "+oldChord[i]);
  }

var cantusMotion = newChord[newChord.length-1] - oldChord[oldChord.length-1];
```

A questo punto vengono controllate separatamente le condizioni relative alle funzioni attivabili nell'interfaccia, ovvero:

- Intervallo di tritono
- Moti retti paralleli
- Sensibile non risolta
- Settima di dominante non risolta
- Infrazione norme di Cantus Firmus

Tritoni:
Vengono rilevati moti di tritono rilevando se la n-esima voce dell'accordo precedente e la n-esima dell'attuale distano 6 semitoni.
Non è eseguito alcun controllo circa l'ammissibilità del tritono.

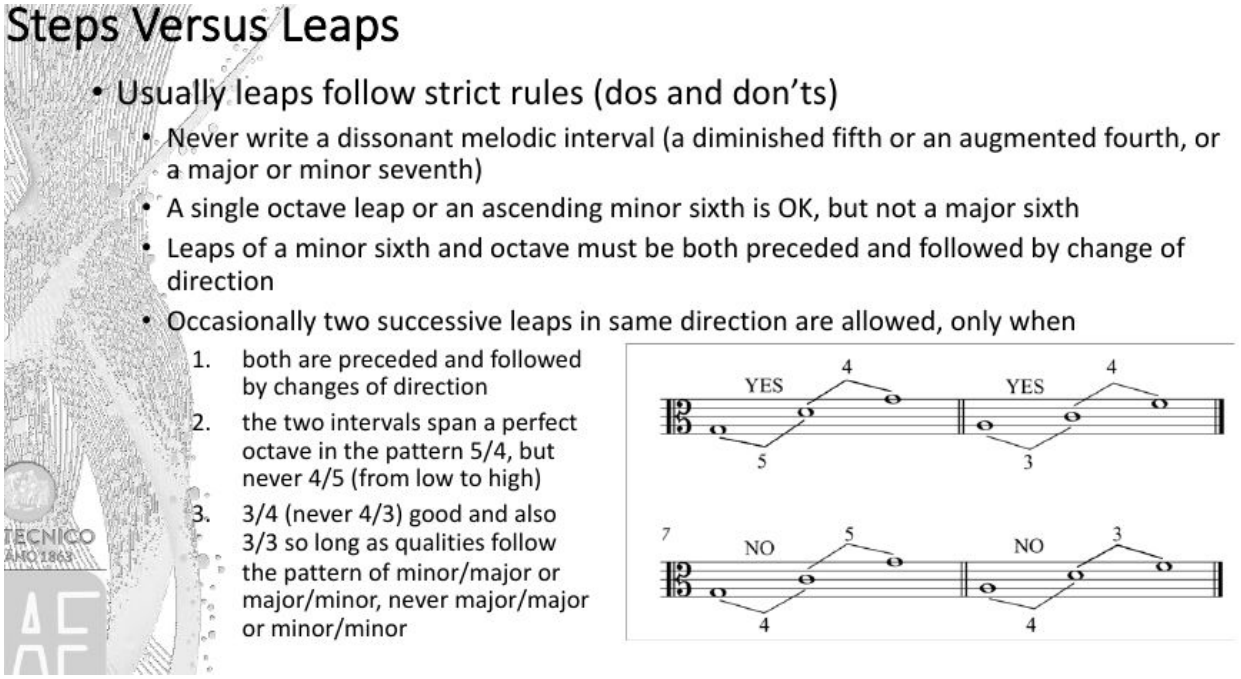
Quinte/ottave parallele/nascoste:
Per ogni coppia di voci dell'accordo corrente, se esse formano una quinta o un'ottava, si segnala l'eventuale approccio per moto retto (ottava/quinta nascosta). Non si specifica se si tratta di moto parallelo (ovvero raggiunto da un intervallo identico).

Sensibile:
Per il rilevamento della sensibile si fa fede alla tonalità rilevata; se viene rilevata una sensibile nell'accordo precedente e la voce analoga nell'accordo attuale non risulta essere la tonica viene effettuata la segnalazione.

Settima di dominante
Per questo controllo si fa riferimento al parametro *oldNameChord*, se esso contiene il termine "*dominant*" significa che la voce corrispondente deve scendere di semitono, altrimenti viene effettuata la segnalazione.

Cantus Firmus
Per i controlli relativi al cantus firmus si fa riferimento principalmente alle indicazioni presenti sulle slide del corso Computer Music - Primo Modulo.
In particolare NON vengono implementati controlli relativi a climax, ne a cantabilità e altri parametri soggettivi, inoltre NON si controlla che la melodia parta e finisca in tonica.

Al di fuori di queste eccezioni viene considerata per intero l'indicazione circa i salti



e le seguenti regole:

- Diatonicità (no cromatismi fuori scala)
- Non mantenere la stessa nota
- Non eseguire intervalli dissonanti
- Non superare l'intervallo di 10a (nella finestra considerata)

Le segnalazioni qui rilevate vengono scritte su una struttura dati chiamata *errarr*, una matrice che rappresenta, colonna per colonna, gli accordi presenti sul canvas. Ogni elemento della matrice corrisponde al colore che deve essere mostrato per la relativa nota (nero per default, colorato se attivato il corrispondente pulsante).