UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA TRIENNALE IN

INFORMATICA MUSICALE

PROTOTIPO IN MAX 7 DI UN GENERATORE SIMBOLICO PILOTATO DA UNA ESECUZIONE MUSICALE

Relatore: Prof. Luca Andrea Ludovico

Correlatore: Prof. Giorgio Presti

Tesi di Laurea di:

Conconi Dario

Matr. Nr. 823479

ANNO ACCADEMICO 2015-2016

INDICE

INTRODUZIONE

Questo prototipo nasce con un obiettivo principale: estendere le possibilità dell'esecuzione musicale. Oggigiorno la musica dal vivo si realizza con due modalità che simboleggiano anche due mondi distinti: con clock, ossia con un tempo musicale fissato metronomicamente (che può essere costante o variabile, ma in ogni caso si intende generato digitalmente), oppure senza clock.

La prima è abbastanza giovane, esiste da meno di cinquant'anni; è nata nell'ambito della musica popolare elettronica, ed è il fondamento di tutta la musica dancefloor-oriented, e anche di tutta la musica elettronica più alternativa che nasce come espressione artistica degna di un ascolto più intellettuale, e non solo "corporeo". Questa modalità ha invaso anche alcuni generi di musica popolare contemporanea che prima ne erano estranei (rock, pop, jazz), per due motivi: uno è artistico, ossia la contaminazione stilistica da parte della musica elettronica (con questo termine intendiamo ancora il genere di musica popolare elettronica citato poc'anzi); l'altro è pratico, infatti avere a disposizione un tempo metronomico fissato permette, anche in esecuzioni dal vivo, di suonare e al contempo riprodurre sequenze di suoni pre-registrati, che integrano l'esecuzione in tempo reale. Infatti, se il musicista suona seguendo il clock relativo alle sequenze, si assicura la completa sincronizzazione di ciò che sta suonando con le suddette sequenze, che conterranno suoni provenienti da strumenti e musicisti non presenti sul palco.

La seconda modalità esiste da quando esiste la musica. Il musicista libero da metronomo può scegliere in tempo presente di rallentare, accelerare, fermare la sua esecuzione, sia da solo sia in gruppo (in questo caso la qualità del risultato dipende dall'interplay che si riesce a creare, ma è comunque possibile). Nel caso in cui invece il musicista/i musicisti scelgano di tenere lo stesso tempo per tutta la durata dell'esecuzione, questo sarà soggetto a minuscole variazioni (in misura ovviamente variabile in funzione della bravura ritmica dei musicisti, o di chi è delegato alla parte ritmica). Queste piccole variazioni sono il frutto dell'ingrediente umano, dell'imperfezione che governa il mondo animale in contrasto con la precisione meccanica dei calcolatori.

Fatta questa premessa, delineiamo le finalità e le caratteristiche di questo prototipo dichiarando fin da subito che vuole essere un supporto informatico al secondo tipo di modalità esecutiva, appena descritto. Permette quindi di arricchire l'esecuzione con elementi (note e controlli) diversi da quelli suonati direttamente dal musicista, ma contrariamente a quanto avviene con le sequenze audio pre-registrate, che costringono il musicista a seguire un clock (o anche in assenza di clock lo costringono a seguire un tempo musicale fornito dall'esterno), gli elementi che vengono aggiunti da questo prototipo seguono l'esecuzione in tempo reale. Esso è

pilotato da un'esecuzione musicale, ovvero riceve in input un'esecuzione e *genera* note e controlli *di pari passo con essa,* in termini di tempo musicale ma anche di dinamica. Per questo motivo gli è stato dato il nome di '**Pilnote**' (da pronunciarsi *pailnot*), una composizione delle parole inglesi 'pilot' (pilota) e 'note' (nota).

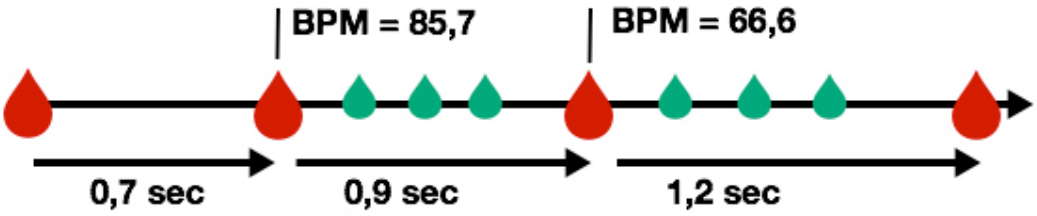
Specifichiamo, per evitare fraintendimenti sul termine *generare*, che tutti gli elementi aggiuntivi sono scelti dall'utente. Al sistema non è lasciato alcun margine "decisionale"; tutti i calcoli effettuati non riguardano assolutamente il contenuto dei dati in uscita, ma ottemperano solamente alla loro sincronizzazione con l'esecuzione. Esistono casi di studio che offrono un sistema che calcola quali note produrre in funzione dei dati storici e di margini di probabilità regolabili, ma ciò sarebbe oltremodo distante dagli obiettivi di questo lavoro.

La situazione esecutiva, allo stato attuale, consiste in un dispositivo o una catena di dispositivi collegati in cascata al computer e in grado scambiare con esso messaggi usando il protocollo di trasferimento delle informazioni musicali, il MIDI (Musical Instrument Digital Interface). Per una spiegazione sui fondamenti del MIDI si rimanda al capitolo sulle tecnologie utilizzate. I già citati controlli corrispondono ai messaggi di *control change* previsti dal protocollo. Il prototipo non si occupa della generazione del suono ma soltanto di ricezione/invio delle informazioni musicali (note, dinamica, controlli vari) conformi al protocollo MIDI, secondo questo schema:

1. Ricezione delle note in input e eventuale routing delle stesse verso un modulo di sintesi
2. Elaborazione
3. Invio a un modulo di sintesi sonora delle note generate.

Il fatto di avere un output sincronizzato con l'input delimita anche le caratteristiche dell'output stesso. Poniamo il caso in cui l'esecuzione in input consista in una melodia in quattro quarti con un ritmo sui quarti; l'output dovrà seguire lo stesso andamento ritmico, infatti se si volesse in uscita una sequenza di note con un ritmo sui sedicesimi, il sistema non saprebbe quando far suonare le note sulle suddivisioni ritmiche, quelle cioè comprese tra un input e il successivo, non avendo a disposizione un'indicazione di tempo. Una possibile soluzione a questo problema potrebbe essere quella di desumere il tempo musicale dell'esecuzione misurando il tempo che intercorre tra un input e l'altro, e far suonare l'output di conseguenza. Questo però metterebbe un freno alla libertà esecutiva di cui sopra; nel caso di un rallentando, ad esempio, le note in uscita avranno un certo tempo, e quindi non potranno partecipare al rallentamento attuato dell'esecutore (fig.1). Anche nel caso in cui si ottenga una qualsiasi ottimizzazione di questo modello, il problema concettuale rimane: laddove l'indicazione di tempo sia il frutto di un

figura 1



calcolo, la libertà ritmica dell'esecuzione ne risulta compromessa (e anche la bontà del risultato sarebbe artisticamente discutibile).

A fronte di quest'unico svantaggio, consideriamo invece i numerosi punti di forza di questo prototipo:

1. La generazione sincronizzata degli eventi musicali (sia note che controlli), che come si è già visto ne è l'idea fondante;
2. La generazione simultanea di un numero illimitato di note per ogni nota in input, fatto salvo il limite di note esistenti nel protocollo MIDI e un eventuale limite di polifonia presente nel dispositivo di sintesi sonora;
3. La generazione, oltre alle note, anche di controlli assegnabili, con i quali è possibile modificare il suono automaticamente in sincronia con l'esecuzione;
4. La generazione di note che hanno la stessa dinamica delle note corrispondenti in input;
5. La possibilità di aprire più sistemi di calcolo in parallelo, il cui output sia indirizzato a canali diversi dello stesso modulo sonoro, o a moduli differenti;
6. La possibilità di navigare tra gli elementi dell'input per posizionarsi in qualunque momento in un punto a piacere del brano;
7. La possibilità di scegliere tra due modalità di durata delle note in output:
    - *free*: tutte le note generate hanno un valore di durata fissato, scelto dall'utente;
    - *follow*: le note generate in corrispondenza in una nota X in input, vengono rilasciate nell'istante esatto in cui viene rilasciata X;
8. La possibilità di bloccare una nota della melodia pilota e le note generate corrispondenti per un tempo indefinito, e nel frattempo andare avanti a suonare il resto della successione (nella modalità *follow*).

STATO DELL'ARTE

Un precursore di questo prototipo è sicuramente il Metapiano di Jean Haury. Pur avendo obiettivi molto diversi da Pilnote, condivide con esso alcuni aspetti importanti.
Il Metapiano è stato concepito per separare la parte interpretativa di un'esecuzione dal suo contenuto. L'interpretazione ha a che fare con il *come* suonare un certo brano, quindi la dinamica delle note, il tempo, tutti quegli aspetti che rendono un'esecuzione unica, e strettamente legata alla soggettività dell'artista che, appunto, interpreta l'opera. Il contenuto riguarda invece *cosa* suonare, la parte scritta nero su bianco dal compositore. Jean Haury, medico, pianista e appassionato di informatica, proprio con l'aiuto di quest'ultima si pone l'obiettivo di scindere queste due componenti dell'esecuzione per creare un nuovo tipo di esecuzione, che verta esclusivamente sull'interpretazione e deleghi la produzione del contenuto al calcolatore.

Delle due informazioni musicali principali ricevute dall'utente, cioè altezza della nota e intensità, solo la seconda viene utilizzata, mentre la prima viene di fatto ignorata. L'azione delle dita del pianista decidono il ritmo con cui far suonare le note memorizzate nel software e l'intensità da assegnare a tali note*, indipendentemente da quale tasto si sta premendo*. Come dichiara anche Haury, tecnicamente è possibile far suonare l'intero brano usando soltanto un tasto, anche se praticamente è più comodo avere a disposizione più tasti per poter sfruttare l'agilità di tutte le dita, soprattutto quando si suona a velocità elevate. Il ricercatore francese ha creato una piccola tastiera di 9 tasti, come quella che si otterrebbe isolando i tasti del pianoforte dal MI al DO successivo, aventi una pesatura che emula quella del pianoforte.

Per poter suonare un Metapiano occorre trascrivere il brano musicale in una particolare codifica creata appositamente dal dott. Haury, chiamata Pianotechnie.
Ciò avviene principalmente in due fasi:

1. Suddivisione dello spartito in voci;
2. Memorizzazione per ogni nota di un valore di durata. Dato che, anche nel Metapiano, il ritmo e quindi il tempo è scandito dall'esecuzione, il valore di durata consiste nello specificare dopo quanti eventi ritmici successivi programmare il rilascio della nota.

La notazione ha come risultato una matrice a due entrate, una sorta di tabella. Questa matrice memorizza l'organizzazione relativa alle intonazioni del brano, cioè l'ordine di apparizione e di rilascio delle note, la loro sincronicità, la loro ripartizione in voci polifoniche. Una riga della matrice riunisce le intonazioni appartenenti a una voce musicale, mentre una colonna sovrappone le diverse intonazioni sincronizzate in un accordo prodotto dalla somma di più voci.

Le intonazioni sono dunque memorizzate in modo melodico e armonico, quindi il primo compito della Pianotechnie è analizzare il brano musicale secondo i rapporti melodici e armonici e di contrappunto tra le note[1]. Come avviene nel MIDI, le intonazioni sono rappresentate numericamente con riferimento assoluto (*continuous pitch code*).
Lo statuto designa il comportamento di una nota in relazione al il movimento dei tasti. Quattro diversi statuti sono sufficienti per coprire tutte le possibilità del comportamento di una nota. Questi corrispondono a quattro simboli diversi, e sono:

- [] Nota della durata di un ciclo. Haury chiama 'ciclo' la pressione di un tasto seguita dal suo rilascio.
- [<> Nota con staccato obbligato, indipendentemente dalla durata del ciclo.
- [< Nota tenuta su più cicli. Viene indicato l'inizio della nota ma viene posticipata la fine a un ciclo successivo.
- [> o]> Fine della nota dopo più cicli, in corrispondenza della pressione del tasto / del rilascio del tasto.

Si è detto che l'informazione di dinamica ricevuta in input viene utilizzata per produrre l'output corrispondente. Tuttavia il valore numerico che codifica l'intensità non viene distribuito a tutte le voci in maniera uguale. I rapporti di intensità che si desidera stabilire tra le diverse voci si realizza integrando dei dati supplementari di 'pesatura' al fine di produrre delle intensità differenti per ognuna delle voci dell'armonia.

Questo prototipo è stato applicato fino ad ora soltanto alle esecuzioni di musica pianistica, offrendo uno strumento a scuole e insegnanti di musica per consentire agli allievi di allenarsi e prendere consapevolezza soltanto del lato interpretativo di un'esecuzione [2].

TECNOLOGIE UTILIZZATE

- MIDI -

Come già scritto nell'introduzione, il software presentato è un processore di messaggi MIDI. L'acronimo sta per Musical Instrument Digital Interface. È stato creato negli anni ottanta per permettere la comunicazione tra i sintetizzatori di diverse case produttrici, ma anche tra sintetizzatori e personal computer.

Nel MIDI vengono codificate le informazioni legate a un'esecuzione musicale: intonazione (altezza) delle note, inizio e fine delle note, intensità delle note, ma anche controlli che permettono di simulare delle variazioni di suono (come il pedale del pianoforte, il vibrato dei violini) oppure di inventarne di nuove.

La sintassi di base di un messaggio MIDI consiste di 1 byte di stato seguito da n bytes di dati. Il bit più significativo di ogni byte è stato dedicato alla differenziazione rapida tra i due tipi di byte: se è uguale a 1 siamo di fronte a un byte di stato, se è uguale a 0 a un byte di dati. Di conseguenza ogni byte consta di 7 bit.

I primi 3 bit del byte di stato servono per indicare la funzione del messaggio: NOTE ON (001) e NOTE OFF (000) per l'inizio e la fine delle note, CONTROL CHANGE (110) per i controlli, e altre funzioni.

Gli altri 4 bit del byte di stato servono per identificare uno dei 16 canali MIDI. I canali midi li possiamo immaginare come le varie frequenze su cui si sintonizza una radio, cambiando canale cambia la musica; analogamente possiamo associare, sul modulo di sintesi sonora, un suono diverso per ogni canale. I messaggi MIDI, anche se provenienti dallo stesso sequencer o generatore di messaggi, vengono direzionati ai diversi canali indicati nel byte di stato.

A ogni byte di stato seguono i bytes di dati, in numero variabile in base alla funzione. In essi è contenuto il valore numerico del messaggio. Per i messaggi di NOTE ON e NOTE OFF sono richiesti due bytes di dati, uno per specificare il *pitch* della nota (termine inglese per 'intonazione'), uno per la *velocity*, ossia il valore di dinamica. Alcuni sistemi interpretano come NOTE OFF un messaggio di NOTE ON con valore di velocity uguale a 0, ed è il caso di questo prototipo. Anche per i CONTROL CHANGE sono necessari due byte di dati, uno per specificare il numero del controllo, uno per il valore. Nonostante i alcuni numeri di control change abbiano degli usi standardizzati (modulation wheel, damper pedal, pan per citarne alcuni) tutti gli altri sono assegnabili a piacimento (tabella 1).

tabella 1: control cahnge messages (fonte www.midi.org)

Control Number (2nd Byte Value)			Control Function	3rd Byte Value	
Decimal	**Binary**	**Hex**		**Value**	**Used As**
0	00000000	00	Bank Select	0-127	MSB
1	00000001	01	Modulation Wheel or Lever	0-127	MSB
2	00000010	02	Breath Controller	0-127	MSB
3	00000011	03	Undefined	0-127	MSB
4	00000100	04	Foot Controller	0-127	MSB
5	00000101	05	Portamento Time	0-127	MSB
6	00000110	06	Data Entry MSB	0-127	MSB
7	00000111	07	Channel Volume (formerly Main Volume)	0-127	MSB
8	00001000	08	Balance	0-127	MSB
9	00001001	09	Undefined	0-127	MSB
10	00001010	0A	Pan	0-127	MSB
11	00001011	0B	Expression Controller	0-127	MSB
12	00001100	0C	Effect Control 1	0-127	MSB
13	00001101	0D	Effect Control 2	0-127	MSB
14	00001110	0E	Undefined	0-127	MSB
15	00001111	0F	Undefined	0-127	MSB
16	00010000	10	General Purpose Controller 1	0-127	MSB
17	00010001	11	General Purpose Controller 2	0-127	MSB
18	00010010	12	General Purpose Controller 3	0-127	MSB
19	00010011	13	General Purpose Controller 4	0-127	MSB
20	00010100	14	Undefined	0-127	MSB
21	00010101	15	Undefined	0-127	MSB
22	00010110	16	Undefined	0-127	MSB
23	00010111	17	Undefined	0-127	MSB

- MAX 7 -

Max è un ambiente di sviluppo grafico per la musica e la multimedialità ideato dall'azienda di software Cycling '74.
Consiste in un pannello su cui si inseriscono svariati oggetti dotati di uno o più inputs e outputs (rappresentati graficamente da mattoncini con dei fori sulla parte superiore e inferiore) chiamati *inlets* e *outlets* che si interconnettono tramite cavi virtuali. Ogni tipo di oggetto ha un numero di *inlet* e *outlet* adeguato a seconda della loro funzione [3].
Una patch può presentarsi in due modalità:

- *Unlocked*: è la modalità di lavoro, di sviluppo della patch stessa, con la quale è possibile creare, modificare e interconnettere gli oggetti;
- *Locked*: in questa modalità di utilizzo, non è possibile modificare e creare alcunchè, bensì si può interagire con bottoni e pulsanti che sono oggetti particolari che inviano messaggi di bang i quali azionare i vari oggetti.

È possibile creare delle sottopatch per mantenere il progetto il più possibile in ordine. Una sottopatch è una patch racchiusa dentro a un oggetto di tipo **p** a cui possono essere aggiunti liberamente inlet e outlets, e quindi possono diventare degli oggetti collegati ad altri oggetti all'interno di una patch. Per aprire il contenuto di una sottopatch in una finestra dedicata basta fare doppio clic sull'oggetto **p** in modalità locked.
È possibile aprire più istanze di una stessa patch. Ciò, come verrà spiegato nella conclusione, moltiplica ulteriormente le possibilità di generazione al momento dell'esecuzione.
I bottoni (**button**) sono oggetti che, se cliccati in modalità locked, inviano un bang. Inoltre trasformano qualsiasi messaggio ricevuto nell'inlet in un bang, che viene inviato. Sono utili anche per monitorare il passaggio dei bang su un collegamento, dato che ogni volta che inviano un bang si accende per un breve istante un piccolo cerchiolino posto sulla loro icona.
I **messaggi** sono particolari oggetti che memorizzano, ricevono o inviano parole chiave utili agli oggetti per assolvere le proprie funzioni. Ciò che viene ricevuto dall'inlet di destra sovrascrive il conenuto del messaggio senza essere inviato. Il messaggio di tipo *'bang'* serve per triggerare gli oggetti a svolgere un'attività o a produrre un output, a seconda dell'oggetto. Se un messaggio riceve un bang dall'outlet di sinistra, il messaggio viene inviato.

Pilnote è una patch di Max. Per questo motivo può essere aperta solamente all'interno di Max, non può esistere indipendentemente (*stand alone*).
Il vantaggio nell'uso di Max è di poter modificare e vedere istantaneamente il risultato della modifica, non essendoci nessun tipo di compilazione. Lo svantaggio, nel caso particolare di

questo prototipo, sta nel fatto che creare software con un'interfaccia utente che tenga nascosti l'implementazione e i sotterfugi risulta molto scomodo, dato che nella modalità di lavoro (unlocked) bisogna comunque trovare *fisicamente* posto per tutto.

Quello che segue è un elenco degli oggetti principali che sono stati usati nell'implementazione:

- **Send e Receive** (anche chiamati **s** e **r**): questi oggetti sono indispensabili quando si vuole collegare due oggetti che stanno su finestre diverse. Si crea una coppia di oggetti **send** e **receive** con la stessa denominazione (è sufficiente scrivere send + [nome] e receive + [nome] al momento della creazione degli oggetti), dopodichè di collega l'output del primo oggetto al send, e ril receive all'input del secondo.
- **Number**: è un oggetto utile per l'interfaccia grafica in ogni punto della patch in cui all'utente è richiesto di inserire un valore numerico. Clccandoci sopra e trascinando verso l'alto o verso il basso (o utilizzando le freccette della tastiera QWERTY), il numero rappresentato sull'icona si incrementa o decrementa e viene inviato istantaneamente, a condizione che l'oggetto sia modificabile.
- **Radiogroup**: anche questo è un oggetto usato per l'interfacciamento con l'utente. È un selettore di tipo radio, vale a dire che offre due opzioni cliccabili ma è possibile selezionarne solo una. Invia 0 o un 1 a seconda che l'opzione selezionata sia quella in alto o quella in basso.
- **Umenu**: oggetto che riceve una lista di elementi e crea con essi un menu a tendina. L'opzione scelta dall'utente viene inviata.
- **Textbutton**: altro oggetto fondamentale per l'interfaccia utente, è esattamente come un **button** ma con un testo al posto del bollino lampeggiante.
- **Sel**: è un selettore. Se l'input in arrivo dall'inlet di sinistra è uguale all'argomento di sel, verrà inviato un bang sull'outlet di sinistra, altrimenti verrà inoltrato l'input sull'outlet di destra. Con l'inlet di destra è possibile configurare l'argomento di sel. L'argomento di default è 0.
- **Gswitch2**: molto usato in questo prototipo, è un switch che permette di selezionare uno tra due o più output per un unico input. Talvolta è stato usato anche per disabilitare un collegamento su comando, lasciando vuota una delle due uscite.
- **Coll**: è la struttura dati su cui si basa il prototipo. Consiste di una collezione di liste. Il contenuto di questa struttura è posto in un file .txt che può essere letto, modificato, salvato. Gli indici, a inizio riga, sono separati con una virgola da tutti gli altri elementi. A fine riga, dopo l'ultimo elemento, troviamo sempre un punto e virgola. Un puntatore

scorre la lista a seconda dei comandi che vengono ricevuti dall'oggetto tramite messaggi.

Accetta i seguenti messaggi (sono elencati solo quelli utilizzati nel prototipo):

- *'start'*: posiziona il puntatore all'inizio della collezione, senza inviare;
- *'next'*: invia gli elementi attualmente puntati e sposta il puntatore in avanti di una posizione;
- *'prev'*: invia gli elementi attualmente puntati e sposta il puntatore in avanti di una posizione;
- [numero]: invia gli elementi puntati dal numero ricevuto, senza spostare il puntatore;
- *'goto* [numero]' : sposta il puntatore all'indice indicato dal numero, senza inviare;
- *'write'*: apre una finestra di sistema per scrivere su file la collezione;
- *'open'*: apre una finestra con il contenuto della collezione;
- *'clear'*: elimina il contenuto dell'intera collezione;
- *'store* [numero] [lista]': memorizza la lista alla posizione indicata dal numero
- *'insert* [numero]': inserisce un nuovo indice alla posizione indicata dal numero, facendo incrementare di uno tutte le posizioni esistenti maggiori o uguali di quel numero;
- *'delete* [numero]': elimina la posizione indicata dal numero, compresi tutti i suoi elementi;
- *'merge* [numero] [lista]': unisce la lista agli elementi già presenti alla posizione indicata dal numero;

- **Int** : memorizza un numero intero proveniente dall'inlet di destra senza inviarlo. Non appena riceve un bang dall'inlet di sinistra, invia il numero memorizzato.
- **Bag** : memorizza una lista di numeri. Quando un numero viene ricevuto nell'inlet di sinistra, viene memorizzato o cancellato a seconda del numero ricevuto nell'inlet di destra (0: cancellato, >0: memorizzato). Non appena riceve un bang dall'inlet di sinistra, invia la lista di elementi. Alla ricezione del messaggio *'clear'*, l'intera lista viene cancellata;
- **Route**: è un router. Possiede tanti outlet quanti sono gli argomenti. Riceve nell'inlet di sinistra coppie di messaggi. Se il primo elemento della coppia coincide con uno degli argomenti, il secondo elemento viene inviato dall'outlet corrispondente, altrimenti viene

inviato dall'outlet più a destra. Se invece di ricevere coppie riceve un solo elemento, viene inviato un bang dall'outlet corrispondente.

- **Pack**: impacchetta più elementi provenienti onguno da un inlet in un'unica lista. Non appena riceve un elemento dall'outlet di sinistra (se ci sono più di due inlet si intende il più a sinistra), invia tutta la lista, compreso l'elemento appena ricevuto
- **Iter**: riceve una lista e inoltra gli elementi della lista uno per volta;
- **Kslider**: è la rappresentazione di una tastiera musicale. Dall'inlet di sinistra riceve i valori di pitch, da quello di destra i valori di velocity. Le note ricevute (con o senza velocity) vengono evidenziate sulla tastiera e inoltrate sui due outlet (sinistra pitch, destra velocity). Cliccando sui tasti, questi vengono evidenziati e le note corrispondenti inviate sui due outlet (sinistra pitch, destra velocity).
- **Stripnote:** riceve dai due inlet pitch (sinistra) e velocity (destra) e inoltra i dati sui due outlet (sinistra pitch, destra velocity) filtrando (quindi NON inviando) tutti i messaggi di NOTE OFF;
- **Notein**: riceve i messaggi di NOTE ON e NOTE OFF da qualunque device MIDI e li invia sui tre outlet (sinistra pitch, centro velocity, destra canale).
- **Noteout**: l'inverso di notein, trasmette i messaggi di NOTE ON e NOTE OFF a un device MIDI. Si può scegliere il device cliccando sull'oggetto in modalità *locked*, oppure cliccando tenendo premuto cmd in modalità *unlocked.* Riceve il pitch dall'inlet di sinistra, la velocity dall'inlet al centro, il canale dall'inlet di destra.
- **Midiin**: resta in alscolto di una specifica porta MIDI e invia i dati MIDI ricevuti (i dati grazzi, la successione di byte tale quale a come viene ricevuta).
- **Makenote**: una volta ricevuti in input pitch, velocity e durata delle note (rispettivamente inlet di sinistra, centro e di destra), per ognuna crea e invia una coppia di messaggi NOTE ON e NOTE OFF. Invia pitch e velocity su due outlet separati (sinistra pitch, destra velocity).
- **Midiparse**: invia il contenuto di un messaggio MIDI su un outlet diverso a seconda della funzione indicata nel byte di stato.
- **Midiinfo**: connesso con un oggetto umenu, invia ad esso la lista dei dispositivi MIDI disponibili, in input o in output a seconda che si riceva uno 0 dall'inlet di sinistra o di destra.
- **Counter**: questo oggetto è un contatore. Quando riceve un bang invia il valore corrente e si incrementa di 1. Tra gli argomenti si possono specificare il valore minimo e il valore massimo. Si può impostare per funzionare in diverse modalità:

- loop ascendente: incrementa di 1 partendo dal valore minimo, al raggiungimento del valore massimo ricomincia il conteggio dal valore minimo;
- loop discendente: decrementa di 1 partendo dal valore massimo, al raggiungimento del valore minimo ricomincia il conteggio dal valore massimo;
- up&down.

Senza argomenti parte da 0 e si incrementa di 1 senza limiti. Con due argomenti, il primo argomento rappresenta il valore minimo e il secondo il valore massimo. Con tre argomenti, il primo serve per impostare la modalità (0: loop ascendente, 1: loop discendente, 2: up&down), il secondo per il minimo, il terzo per il massimo. Se non viene specificato diversamente, la modalità di default è il loop ascendente.

- **Pipe**: permette di ritardare qualsasi messaggio o lista. Occorre digitare come argomento il valore numerico (espresso in millisecondi) del ritardo. Senza argomenti non ha effetto (ritardo di 0 millisecondi).
- **Del**: premette di ritardare un bang. Si può impostare come argomento il valore numerico (espresso in millisecondi) del ritardo. Senza argomenti, il ritardo di default è di 5 millisecondi.

PILNOTE: FUNZIONAMENTO

L'utente di Pilnote è chiamato a preparare la propria esecuzione memorizzando nel sistema due collezioni di dati (fig.2):

1. Una lista che rappresenti la successione di note in input, che da ora in avanti chiameremo 'melodia pilota'. L'indice di questa lista, invece, la chiameremo 'step' e ha valore iniziale uguale a 1;
2. Una lista di note e/o di controlli per ogni step, ovvero per ogni nota della melodia pilota. Possiamo immaginarla come una lista di liste.

Queste due collezioni quindi hanno lo stesso numero di elementi; nel caso della prima sono note, o per meglio dire indicazioni di pitch, nel caso della seconda sono liste di note o controlli. Ad ogni modo, l'elemento allo step X di una collezione corrisponde all'elemento allo step X dell'altra.

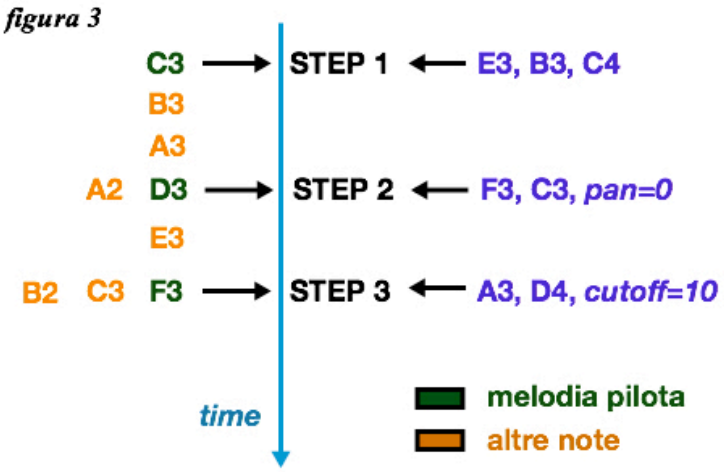
figura 2

Una volta preparata l'esecuzione con la memorizzazione e il caricamento dei dati, l'esecuzione è pronta. Pilnote resta in ascolto dell'input MIDI scelto dall'utente; la prima nota della melodia pilota, allo step 1, è quella che dà il comando al sistema di generare i dati memorizzati allo step 1. Questa nota, in questo preciso momento è detta la 'nota attesa'. Non appena l'utente suona la nota attesa, Pilnote genera le note corrispondenti sull'output MIDI scelto, e istantaneamente si mette di nuovo in ascolto dell'input in attesa della nota successiva della melodia pilota (in questo caso allo step 2). Suonando questa nota vengono generate le note corrispondenti allo step 2, e il flusso continua in questo modo fino al raggiungimento dell'ultimo step. Dopo che sono state generate le note corrispondenti all'ultimo step, il sistema ritorna in attesa della nota allo step 1, e si riparte da capo.

Per quanto riguarda invece la dinamica, alle note generate viene attribuito lo stesso valore di *velocity* (termine usato nel protocollo MIDI per indicare l'intensità) della nota eseguita in input allo step corrispondente.

Un'ulteriore precisazione va fatta per l'input: appurato che solo la sequenza di note della melodia pilota permette di generare le note corrispondenti e di conseguenza procedere nell'avanzamento del brano, nulla vieta all'esecutore di suonare qualsiasi altra nota, tra due note successive della melodia pilota, che non siano la nota attesa. In questo modo è possibile fare improvvisazioni o abbellimenti o ad ogni modo suonare note che non facciano parte dello scheletro essenziale della melodia (fig.3).

figura 3



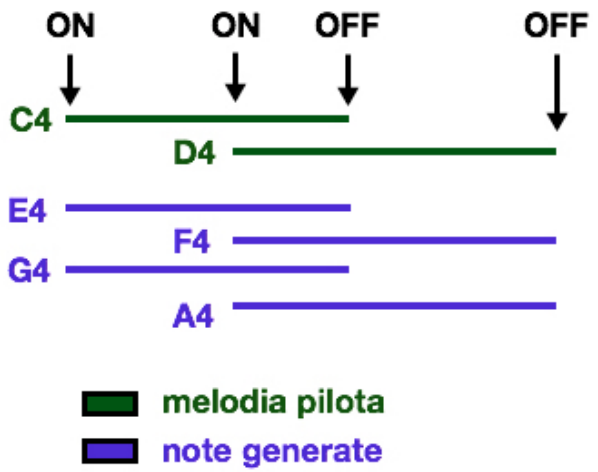
Inoltre, anche se a mettere in azione il sistema è una successione di note, è possibile suonare anche accordi, contenenti le note della melodia pilota, senza che le note di questi accordi interferiscano con la durata delle note generate, qualora dovessero sovrapporsi. Si è evitato infatti che, qualora una o più note generate coincidessero con le note eseguite fisicamente sul dispositivo, il rilascio di queste ultime, immettendo sul canale MIDI dei messaggi di NOTE OFF, facesse rilasciare anche le note generate aventi la stessa intonazione.

Si è spiegato come è caratterizzato l'output riguardo all'inizio delle note, ma non si è ancora parlato del loro rilascio. Si offrono all'utente due modalità:

1. Assegnare alle note generate un valore di durata costante per tutte.
2. Le note generate in simultanea con l'esecuzione della nota X sono rilasciate soltanto nell'istante esatto in cui viene rilasciata X.

Questa seconda modalità permette di suonare *legato* non solo le note della melodia pilota ma anche le note generate corrispondenti. Per essere precisi, se due note successive della melodia pilota sono suonate legate, saranno legate anche le note generate corrispondenti. È stato implementato il legato nella sua forma più realistica, quella che tiene conto di una sovrapposizione, seppure molto breve, delle due note. Ancora, precisiamo che se le due note subiscono una sovrapposizione, le note generate corrispondenti seguono lo stesso andamento (fig.4).

figura 4



Si prevede l'implementazione di una terza modalità secondo cui le note che vengono generate continuano a suonare anche dopo il rilascio della nota che le ha evocate, fino a che non viene suonata la nota successiva della melodia pilota, allorchè vengono generati i messaggi di NOTE OFF delle prime note e, in un tempo rapidissimo, i NOTE ON delle nuove note.

È possibile anche bloccare una nota della melodia pilota e le note generate corrispondenti per un tempo indefinito, e nel frattempo andare avanti a suonare il resto della successione.

PILNOTE: MAPPA DEL SOFTWARE E IMPLEMENTAZIONE

Pilnote è formato da una *finestra principale*, dalla quale si può accedere tramite pulsanti ad altre due finestre, chiamate *insert_melody* e *insert_data* per l'inserimento dei dati. Queste tre finestre in realtà sono patch in modalità *locked* a cui sono stati nascosti gli oggetti e le connessioni tra gli oggetti tramite la funzione "hide on lock". Oltre a queste tre finestre user-friendly ce ne sono altre sette, affatto comprensibili per un utente inesperto, alle quali si può accedere soltanto passando alla modalità *unlocked:*

1. *engine,* accedibile dalla finestra principale, contiene il motore di Pilnote, il cuore pulsante del software;
2. *follow,* contenuta in *engine,* contiene l'implementazione della modalità 'follow' grazie a cui le note generate hanno la stessa durata delle note in input che le evocano (le note generate in corrispondenza in una nota X in input vengono rilasciate nell'istante esatto in cui viene rilasciata X). Questa modalità permette anche di suonare legato, che in musica è l'azione di suonare due note consecutive legando i due suoni, evitando che ci sia un'interruzione tra i due. L'implementazione del legato, di conseguenza, è contenuta in questa sottopatch;
3. *store_melody,* contenuta in *insert_melody,* provvede alla memorizzazione della successione di note in input in una opportuna struttura dati;
4. *store_data,* contenuta in *insert_data,* provvede alla memorizzazione dei dati da generare in output in una opportuna struttura dati;
5. *check_melody,* contenuta in *insert_melody*, provvede al monitoraggio del suono durante l'operazione di revisione, da parte dell'utente, della melodia pilota, nella finestra *insert_melody;*
6. *check_data,* contenuta in *insert_data*, provvede al monitoraggio del suono durante l'operazione di revisione, da parte dell'utente, dei dati da generare, nella finestra *insert_data;*
7. *bookmarks,* contenuta nella finestra principale, provvede al funzionamento dei segnalibri, che permettono di navigare tra gli elementi dell'input per posizionarsi in qualunque momento in un punto a piacere del brano.

- FINESTRA PRINCIPALE: locked -

La finestra principale permette, tramite i tre riquadri con la cornice azzurra nella parte superiore, di preparare l'esecuzione. Il riquadro più piccolo, al centro, permette di selezionare un dispositivo tra quelli attualmente connessi al computer. Nel caso in cui il dispositivo desiderato non sia visibile, si può aggiornare il menu a tendina premendo il pulsante "**Refresh**".

Il riquadro a sinistra riguarda la melodia pilota: è possibile caricare un file .txt contenente la melodia memorizzata (cliccando il bottone con la scritta "**open a .txt file**"), oppure scriverne una nuova. Cliccando su "**edit/write a new melody**", infatti, comparirà la finestra *insert_melody,* che come abbiamo già visto è deputata all'inserimento dei dati. In basso, un menu a tendina permette di selezionare l'output MIDI della melodia pilota e il canale MIDI, nel caso in cui si voglia usare il software come router per indirizzare l'input proveniente da un dispositivo MIDI verso un modulo sonoro collegato al computer. Tuttavia l'utente può anche non essere interessato a questa opzione, qualora il dispositivo sia già direttamente collegato a un modulo sonoro esterno.

Il riquadro a destra invece riguarda i dati da generare: analogamente al riquadro di sinistra, il bottone "**open a .txt file**" apre una finestra di sistema per il caricamento di una collezione di dati memorizzato su file, mentre cliccando su "**store your notes & control**" si aprirà la finestra *insert_data*. In basso, un menu a tendina permette di selezionare l'output MIDI dei dati e il canale MIDI.

A sinistra, sotto la scritta **Bookmarks**, si possono impostare fino a tre segnalibri, per posizionarsi in un punto a piacere del brano in qualunque momento. Basta inserire nel riquadro a destra il numero di step a cui si vuole avere un accesso rapido e, quando lo si desidera, cliccare sul tasto **GO.** In alternativa si può assegnare un numero di control change per ogni segnalibro (per l'implementazione dei segnalibri si rimanda al capitolo sulla sottopatch *bookmarks*).

Sotto i segnalibri è presente un riquadro con alcune opzioni che riguardano l'esecuzione. Un interruttore permette di selezionare una tra le due modalità di esecuzione già citate:

- *follow*: le note generate in corrispondenza in una nota X della melodia pilota, vengono rilasciate nell'istante esatto in cui viene rilasciata X;
- *free*: tutte le note generate hanno un valore di durata fissato; per scegliere tale valore basta inserirlo nel riquadro numerico di fianco alla scritta **duration.**

Clunch è un sistema che permette di interrompere l'algoritmo di matching tra l'esecuzione e la melodia pilota. Una volta assegnato un numero di control change per questa funzione, durante

ogni azionamento del controllo assegnato, l'esecuzione della nota attesa dal sistema non avrà alcun effetto (Pilnote rimarrà ancora in attesa di tale nota). Il meccanismo è molto simile alla frizione dell'automobile (da qui l'utilizzo del termine *clunch*): durante la pressione del pedale della frizione, le ruote vengono staccate dal motore, perciò ogni tentativo di accelerare è vano. Durante la pressione del controllo associato al clunch, si attiverà il **led** tondo posto a sinistra.
Al centro della finestra si trova un piccolo riquadro azzurro. Un interruttore accende l'algoritmo di matching: la sua funzione è solamente di evitare il conflitto con le altre finestre di inserimento dati eventualmente ancora aperte. A destra è visualizzato il numero di step: durante l'esecuzione si aggiorna, si incrementa. Questo oggetto è di sola lettura, serve per monitorare il numero di step corrente. Il tasto START al centro porta il sistema allo step 1, quindi alla situazione di partenza.

Ora tutto è pronto per l'esecuzione: sulla tastiera musicale più in alto, con i tasti rossi e la scritta NEXT NOTE, è evidenziata la nota attesa dal sistema, ovvero la prossima nota della melodia pilota. Suonando quella nota verranno generati i primi dati, cioè note e controlli (se memorizzati).
La tastiera al centro, con i tasti verdi e la scritta NOW PLAYING, mostra le note che sono in esecuzione in tempo reale.
La tastiera più in basso, con i tasti blu e la scritta YOUR NOTES, evidenzia le note che vengono generate da Pilnote di volta in volta.
A destra, su due colonne, vengono visualizzati i controlli generati con i relativi valori.

- FINESTRA PRINCIPALE: unlocked -

A parte alcune funzioni per la scelta dei dispositivi, gran parte del contenuto della finestra principale è costituito da:

- oggetti che permettono di visualizzare e monitorare cosa sta accadendo (ad esempio qual è prossima nota attesa, quali sono le note e i controlli generati); questi oggetti ricevono le informazioni da altre patch che contengono l'implementazione vera e propria, tramite coppie di **send/receive**;
- pulsanti e interruttori che permettono all'utente di utilizzare il software o modificare a piacimento le impostazioni; questi oggetti inviano messaggi ad altre patch che contengono l'implementazione vera e propria, tramite coppie di **send/receive**.

Il riquadro centrale in alto per la scelta del dispositivo è realizzato molto semplicemente con un oggetto **umenu** che riceve le varie opzioni da un oggetto **midiinfo**. Il tasto Refresh triggera un messaggio contenente 0 a due oggetti **midiinfo**: uno è quello appena citato, che riceve lo 0 dall'inlet di destra per inviare a **umenu** la lista degli input disponibili, che viene inoltrata a un oggetto send chiamato 'pilotDevice' per rendere fruibile la scelta del dispositivo anche ad altri punti del software; l'altro riceve 0 dall'inlet di sinistra, e invia la lista degli output disponibili agli oggetti **umenu** dei riquadri laterali.

Tra il riquadro centrale e quello di destra ci sono due oggetti che si occupano di far suonare le note in arrivo dai dispositivi MIDI: si tratta di un **notein** collegato a un **noteout**. Come abbiamo detto, questa funzione è sempre attiva, ed è utile anche per scegliere le note da memorizzare nelle finestre *insert_melody* e *insert_data.* **Noteout** inoltre riceve dal riquadro di sinistra l'output e il canale MIDI a cui dirottare (eventualmente) le note. **Notein** invece riceve dall'oggetto umenu del riquadro centrale il dispositivo MIDI scelto per l'input.

I pulsanti "open a .txt file" triggerano l'invio di un messaggio '*read*' agli oggetti **coll** corrispondenti, che stanno nella patch *engine*.

I pulsanti "edit/write a new melody" e "store your notes & control" triggerano l'invio di un messaggio '*open'* a un oggetto **pcontrol**, che permette di dare il comando rispettivamente alle sottopatch *insert_melody* e *insert_data* di aprire la finestra corrispondente.

Il pulsante START invia un bang al messaggio start presente nella patch *engine.*

L'interruttore ON/OFF invia uno 0 o un 1 (a seconda che sia selezionato ON o OFF) alla patch *engine*.

L'oggetto number che mostra lo step corrente riceve il valore numerico dal secondo outlet dell'oggetto **coll** (in *engine*) che contiene i dati.

Tutti i messaggi che riguardano i segnalibri, quindi i numeri di step in uscita, i control change assegnabili e i bang in arrivo dai tasti GO, confluiscono tutti nella sottopatch *bookmarks*, ognuno in un apposito inlet o coppia di **send/receive**.

Nel riquadro nero in basso a sinistra, l'interruttore per la scelta della modalità di esecuzione follow/free e l'indicazione di durata per la modalità 'free' vengono inviati alla patch *engine*.

La funzione 'clunch' è implementata nel seguente modo: l'oggetto **midiin,** a destra degli oggetti **kslider,** riceve dal receive pilotDevice l'informazione su quale dispositivo di input è stato scelto dall'utente, e invia tutti messaggi ricevuti da questo input all'oggetto **midiparse**. Questo filtra tutti i messaggi inoltrando a un oggetto **unpack**, da un apposito outlet, solo i messaggi di control change come coppie di elementi (liste di due elementi): il primo per il numero di control change,

il secondo, per il valore. **Unpack** spacchetta queste liste inviando il primo elemento di ogni lista dall'outlet di sinistra e il secondo dall'outlet di destra. I valori, quindi, escono dall'outlet di destra e vengono scritti in un oggetto **message**. I numeri di control change invece arrivano a un oggetto **sel** che invia un bang solo se il numero ricevuto è uguale a 0 (di default) oppure a un numero che viene ricevuto dall'inlet di destra, cioè dall'oggetto **number** presente nel riquadro nero in basso a sinistra di fianco alla scritta "**>>cc**". Quindi, quando viene ricevuto un control change che corrisponde a quello scelto, il valore, che è stato scritto nell'oggetto **message**, viene triggerato dal bang prodotto da **sel**, e inviato, tramite una coppia di **send/receive**, alla patch *engine*. Tale valore viene inviato anche all'oggetto **led** del riquadro nero per monitorare l'azionamento della funzione 'clunch'.

L'oggetto **kslider** con la scritta NEXT NOTE, che mostra per ogni step la prossima nota della melodia pilota attesa dal sistema, riceve il pitch dall'oggetto **coll** contenuto in *engine* che contiene la melodia pilota.

L'oggetto **kslider** con la scritta NOW PLAYING, che mostra le note in esecuzione in tempo reale, riceve pitch e velocity da un oggetto notein. La velocity è utile per avere anche l'informazione di rilascio della nota, in virtù del fatto che l'oggetto kslider interpreta una nota con velocity uguale a zero come un messaggio di NOTE OFF.

L'oggetto **kslider** più in basso con la scritta YOUR NOTES, che mostra le note generate da Pilnote, riceve i messaggi di pitch e velocity dalla patch *engine*.

I due messaggi che mostrano i controlli generati da Pilnote sono oggetti **message** che ricevono i dati dalla patch *engine,* e vengono scritti tramite un messaggio parametrico contenente il comando 'append'; questo comando permette di aggiungere una nuova parola in coda al messaggio, senza sovrascriverne il contenuto. Ogni volta che lo step si incrementa, l'oggetto number che contiene lo step triggera due messaggi contenenti il comando 'set [empty]', per resettare gli oggetti **message** prima dell'arrivo dei nuovi controlli.

- INSERT_MELODY: locked -

In primo piano è visibile una rappresentazione di una tastiera musicale. Si tratta di un oggetto di Max chiamato **kslider**. Dovendo memorizzare una melodia, questo oggetto **kslider** è sato impostato come monofonico, cioè verrà sempre evidenziata una sola nota per volta.

Il procedimento per la memorizzazione è molto semplice. Occorre:

- Selezionare la nota che si vuole memorizzare. Questo può avvenire con due modalità differenti:

- Tramite un dispositivo MIDI collegato al computer (selezionabile dalla finestra principale nel riquadro centrale della parte superiore). In questo caso la nota selezionata verrà evidenziata in verde automaticamente sull'oggetto **kslider**.
- Cliccando sull'oggetto **kslider** alla nota desiterata. Per ascoltare il suono della nota selezionata occore cliccare su ON l'interruttore alla destra di **kslider** alla voce "**sound from mouse click**". Il tasto STOP appena sotto permette di fermare il suono, dato che un clic sull'oggetto **kslider** simlua la pressione del tasto, a cui però non corrisponde nessun rilascio del tasto stesso.

➢ Assicurarsi che lo step indicato sotto a **kslider** sia quello desiderato. Per cambiarlo, cliccare sull'oggetto number di fianco alla scritta "**step**", digitare il numero desiderato e premere [invio], oppure navigare con le freccette fino al raggiungimento del valore desiderato.

➢ Cliccare sul tasto STORE

Ogni volta che si clicca sul tasto STORE il valore dello step nell'oggetto number viene incrementato di uno, per velocizzare il processo di inserimento.

Per inserire un nuovo step tra due note successive della melodia pilota, si deve premere il tasto INSERT NEW accertandosi che lo step sia impostato al valore corretto. Così facendo, tutte le posizioni maggiori o uguali di quello step vengono incrementate di 1 e allo step scelto ci sarà una posizione vuota, che andrà riempita memorizzando una nota musicale.

Nel pannello verde, in basso a destra, ci sono altre tre funzionalità per la gestiona del file .txt su cui viene memorizzata la struttura dati:

- "**write in a .txt file**" apre una finestra di sistema per la scrittura su file .txt della struttura dati corrente;
- "**open editor**", per aprire il file .txt e apportare modifiche dirette;
- "**clear all data**" per cancellare completamente il contenuto della struttura dati;

Nel pannello azzurro sottostante è possibile revisionare la melodia. Ci si può posizionare liberamente a qualunque step digitando il valore nell'oggetto number vicino alla scritta "**go to**", e incrementare di uno con la freccetta della tastiera QWERTY per procedere nella successione di note. Dato che da parte dell'utente non viene data nessuna indicazione riguardo al rilascio delle note, queste suonano per un tempo indefinito. Il tasto STOP a fianco permette di fermare il suono.

- INSERT_MELODY: unlocked -

L'oggetto **notein** posizionato in alto invia a **kslider** le note MIDI provenienti da qualunque dispositivo connesso, passando per uno **stripnote**, che filtra (non lascia passare) i messaggi di NOTE OFF. L'uso di **stripnote** in questo caso è puramente per fini estetici, non funzionali. Dato che **kslider** è impostato in modalità monofonica, grazie a **stripnote** vengono evidenziate i tasti soltanto al momento della loro pressione e non al momento del loro rilascio. Avendo collegato **notein** a **kslider**, ogni nota ricevuta da un dispositivo MIDI verrà visualizzata su **kslider** di colore verde.
Per attivare il suono durante la selezione di note tramite clic sull'oggetto **kslider**, occorre inviare i valori di pitch delle note in questione alla sottopatch *check_melody* (di cui parleremo in seguito). L'oggetto **radiogroup** alla voce "sound from mouse click", quando viene selezionato ON invia uno 0 al selettore **gswitch2** , che permette ai valori di pitch in uscita da **kslider** di arrivare a *check_melody*. Quando invece viene selezionato OFF, vengono dirottate su un'uscita vuota. Il pulsante STOP triggera l'invio del messaggio *'stop'* a *check_melody* per interrompere il suono. Dato che in questo caso i dati riguardano una melodia, ogni volta che si seleziona una nota viene inviato un messaggio *'stop'* a *check_melody* per interrompere il suono della nota precedente.
Quando la selezione avviene tramite dispositivo MIDI, ogni nota ricevuta triggera un messaggio contenente 1 all'oggetto radiogroup, che viene automaticamente impostato su OFF. Questo perché, quando si suona un dispositivo MIDI connesso al computer, i suoni sono sempre attivati in virtù degli oggetti presenti nella finestra principale, perciò lasciare l'interruttore "sound from mouse click" su ON causerebbe la generazione di suoni doppi.

- STORE_MELODY -

La memorizzazione della melodia è demandata alla sottopatch *store_melody*, alla quale per ogni nota vengono inviati il valore di pitch, il bang proveniente dal tasto STORE, il numero dello step. Un quarto inlet riceve il bang proveniente dal tasto INSERT NEW. In *store_melody* il pitch e lo step vengono impacchettati in una lista grazie a un oggetto **pack**. Ogni volta che viene ricevuto un bang, questo viene mandato su quattro collegamenti: il primo viene inviato a **pack**, che quindi invia la lista a un messaggio parametrico che imposta un ulteriore messaggio nel seguente modo: *'store* [step] [pitch]'; questo messaggio viene triggerato dal secondo bang, che

viene ritardato con un oggetto **del**, e viene inviato all'oggetto **coll** contenuto in *engine*, che contiene la struttura dati della melodia pilota. Gli altri due bang servono per creare un indice corrispondente nell'oggetto **coll** che memorizza i dati. Per fare ciò, un bang triggera un messaggio impostato come '*delete* [step]', l'altro, ritardato, triggera un messaggio impostato come '*insert* [step]'; in questo modo, anche nel caso di modifiche ad uno stesso step della melodia pilota , si cancella e si aggiunge di nuovo uno step nell'oggetto **coll** dei dati, per evitare di compromettere la corrispondenza degli step nei due oggetti **coll**.
Anche l'inserimento di un nuovo indice viene gestito in *store_melody*: in questo caso l'unico parametro è lo step, di conseguenza non è necessario l'utilizzo di **pack**. Ogni volta che viene ricevuto un nuovo valore di step, un messaggio parametrico imposta un ulteriore messaggio nel seguente modo: '*insert* [step]'. Quando invece viene ricevuto un bang dal tasto insert new, questo messaggio viene inviato sia all'oggetto **coll** che contiene la melodia pilota, sia all'oggetto **coll** che contiene i dati da generare.
I tre pulsanti in corrispondenza di "write in a .txt file", "open editor", "clear all data" sono collegati rispettivamente ai messaggi '*write'*, '*open'* e '*clear'*, che vengono inviati all'oggetto **coll** contenente la melodia pilota.
La revisione della melodia nel pannello azzurro ha origine dall'oggetto **number.** Questo invia il numero di step su due collegamenti: uno triggera un messaggio '*stop'* che viene inviato a *check_melody* per interrompere il suono della nota precedente; l'altro all'oggetto **coll** contenente la melodia pilota, che invierà in uscita la nota richiesta. Quest'ultimo collegamento è ritardato con un **pipe** di 10 millisecondi per permettere al messaggio '*stop'* di arrivare a destinazione prima che l'oggetto **coll** invii la nota richiesta. Quando la nota, o meglio il pitch della nota richiesta, viene inviato dall'oggetto **coll** contenuto in *engine,* entra in un **send** denominato 'fromMelody' ed esce da due oggetti **receive**: uno in *insert_melody,* collegato a **kslider** del pannello azzurro per evidenziare la nota corrente; l'altro in *check_melody* per la produzione del suono corrispondente.

- CHECK_MELODY –

La patch *check_melody* si occupa di far suonare le note della melodia pilota in fase di revisione, per cui arrivano dall'oggetto **coll** le indicazioni di pitch della sequenza di note ma mancano le indicazioni di intensità e rilascio da parte dell'utente. Arrivano anche i valori di pitch in uscita dall'oggetto **kslider** nel pannello verde, infatti la situazione è la stessa: arrivano i valori di pitch delle note corrispondenti ai tasti selezionati ma mancano le indicazioni di intensità e

rilascio da parte dell'utente. All'interno troviamo un oggetto **makenote** con il valore di velocity impostato come argomento a 110. La durata è impostata a un valore molto alto (1000 secondi), in modo tale che il suono venga fermato o dall'arrivo della nota successiva o da un clic sul tasto STOP; in entrambi i casi viene inviato un messaggio '*stop'* a **makenote**, che interrompe il suono. Come di consueto **makenote** è collegato a un **noteout** che riceve dalla finestra principale anche il device di uscita e il canale MIDI. Il collegamento a **makenote** dei valori di pitch provenienti dall'oggetto **coll** è interrotto da un selettore **gswitch2**, comandato all'apertura e alla chiusura dall'interruttore "melody sound" presente in *insert_data.*

- INSERT_DATA: locked -

In questo caso per ognuno dei pannelli colorati abbiamo due oggetti **kslider,** quello sopra per la melodia, che si presume sia già stata memorizzata, quello sotto per le note da generare. Dovendo memorizzare un numero finito di note, quest'ultimo è sato impostato come polifonico.
Il procedimento per la memorizzazione delle note è analogo al caso di *insert_melody*. Occorre:

- Selezionare la nota/le note che si vogliono memorizzare. Questo può avvenire con due modalità differenti:
  - Tramite un dispositivo MIDI collegato al computer (selezionabile dalla finestra principale nel riquadro centrale della parte superiore). In questo caso la nota/le note selezionate verranno evidenziate in verde automaticamente sull'oggetto **kslider**.
  - Cliccando sull'oggetto **kslider** alla nota desiterata. Per ascoltare il suono della nota/delle note selezionate occore cliccare su ON l'interruttore alla destra di **kslider** alla voce "sound from mouse click". Il tasto KEYBOARD RESET appena sotto permette di resettare **kslider** e fermare il suono, dato che un clic sull'oggetto **kslider** simlua la pressione del tasto, a cui però non corrisponde nessun rilascio del tasto stesso.
- Assicurarsi che lo step indicato sotto a **kslider** sia quello desiderato. Per cambiarlo, cliccare sull'oggetto number di fianco alla scritta "step", digitare il numero desiderato e premere [invio], oppure navigare con le freccette fino al raggiungimento del valore desiderato.
- Cliccare sul tasto STORE per ogni nota da inserire

Per la memorizzazione dei controlli:

- Impostare gli oggetti **number** alle voci "control change" e "value";

- Assicurarsi che lo step indicato sotto a **kslider** sia quello desiderato. Per cambiarlo, cliccare sull'oggetto number di fianco alla scritta "**step**", digitare il numero desiderato e premere [invio], oppure navigare con le freccette fino al raggiungimento del valore desiderato.
- Cliccare sul tasto **STORE.**

L'oggetto **kslider** superiore mostra la melodia pilota memorizzata per ogni step. Si può scegliere se attivare il suono o meno, tramite l'interruttore "**melody sound**".

Il bottone in corrispondenza di "**clear this step**" elimina tutti i dati presenti allo step selezionato.

Nel pannello verde, in basso a destra, ci sono altre tre funzionalità per la gestiona del file .txt su cui viene memorizzata la struttura dati:

- "**write in a .txt file**" apre una finestra di sistema per la scrittura su file .txt della struttura dati corrente;
- "**open editor**", per aprire il file .txt e apportare modifiche dirette;
- "**clear all data**" per cancellare completamente il contenuto della struttura dati;

Nel pannello azzurro sottostante è possibile revisionare i dati. Ci si può posizionare liberamente a qualunque step digitando il valore nell'oggetto number vicino alla scritta "**go to**", e incrementare di uno con la freccetta della tastiera QWERTY per procedere nella successione di note. Dato che da parte dell'utente non viene data nessuna indicazione riguardo al rilascio delle note, queste suonano per un tempo indefinito. Il tasto **STOP** a fianco permette di fermare il suono.

L'oggetto **kslider** superiore mostra la melodia pilota memorizzata per ogni step. Si può scegliere se attivare il suono o meno, tramite l'interruttore "**melody sound**".

- INSERT_DATA: unlocked -

L'oggetto **notein** posizionato in altoa destra invia a **kslider** le note MIDI provenienti da qualunque dispositivo connesso. Avendo collegato **notein** a **kslider**, ogni nota ricevuta da un dispositivo MIDI verrà visualizzata su **kslider** evidenziata di verde.

Per attivare il suono durante la selezione di note tramite clic sull'oggetto **kslider**, occorre inviare i valori di pitch delle note in questione alla sottopatch *check_data* (di cui parleremo in seguito).

L'oggetto **radiogroup** alla voce "**sound from mouse click**", quando viene selezionato ON invia uno 0 al selettore **gswitch2** , che permette ai valori di pitch in uscita da **kslider** di arrivare a

check_data. Quando invece viene selezionato OFF, vengono dirottate su un'uscita vuota. Il pulsante STOP triggera l'invio del messaggio *'stop'* a *check_data* per interrompere il suono.
Quando la selezione avviene tramite dispositivo MIDI, ogni nota ricevuta triggera un messaggio contenente 1 all'oggetto **radiogroup**, che viene automaticamente impostato su OFF. Questo perché, quando si suona un dispositivo MIDI connesso al computer, i suoni sono sempre attivati in virtù degli oggetti presenti nella finestra principale, perciò lasciare l'interruttore "sound from mouse click" su ON causerebbe la generazione di suoni doppi.

- STORE_DATA –

La memorizzazione dei dati è demandata alla sottopatch *store_data*. In *store_data* il pitch e lo step vengono impacchettati in una lista grazie a un oggetto **pack**. Ogni volta che viene ricevuto un bang in seguito alla pressione del tasto STORE, questo viene mandato su due collegamenti: il primo viene inviato a **pack**, che quindi invia la lista a un messaggio parametrico che imposta un ulteriore messaggio nel seguente modo: *'merge* [step] *n* [pitch]'; questo messaggio viene triggerato dal secondo bang, che viene ritardato con un oggetto **del**, e viene inviato all'oggetto **coll** contenuto in *engine*, che contiene i dati da generare. Per i controlli il procedimento è analogo: per memorizzare un controllo vengono ricevuti, tramite appositi inlet, il numero del control change e il valore. Viene creata una lista con un oggetto **pack**, e poi con due bang, di cui uno ritardato, viene inviato all'oggetto **coll** un messaggio con la seguente formattazione: 'merge [step] cc [control change] cv [valore]'. L'inserimento dei caratteri 'n', 'cc' e 'cv' serve per creare una sorta di parsing all'interno della struttura dati: quando questa viene letta, gli elementi vengono analizzati a due a due, e a seconda che il primo elemento sia 'n', 'cc' o 'cv', il secondo elemento viene trattato rispettivamente come una nota, un control change o un valore riferito al control change (si veda il capitolo sulla patch *engine*).
Ogni volta che viene ricevuto un nuovo valore di step, vengono impostati tramite messaggi parametrici due ulteriori messaggi con la seguente formattazione: '*insert* [step]' e '*delete* [step]'. Non appena viene ricevuto un bang dal tasto "clear this step", vengono all'oggetto **coll** che contiene i dati da generare, ma il primo viene ritardato con un oggetto, così facendo viene prima cancellata l'intera riga della collezione e poi ne viene creata una nuova allo step corrispondente.
I tre pulsanti in corrispondenza di "write in a .txt file", "open editor", "clear all data" sono collegati rispettivamente ai messaggi '*write'*, '*open'* e '*clear'*, che vengono inviati all'oggetto **coll** contenente la melodia pilota.

Nel pannello azzurro si ha la revisione dei dati**.** Il funzionamento dell'oggetto **kslider** superiore, che offre un promemoria della melodia pilota per ogni step, funziona esattamente allo stesso modo della patch *insert_melody* (si rimanda al capitolo dedicato). L'oggetto **number** vicino alla voce "go to" invia il numero di step su due collegamenti: uno triggera un messaggio '*stop'* che viene inviato a *check_data* per interrompere il suono delle note precedenti; l'altro all'oggetto **coll** contenente la melodia pilota, che invierà in uscita la nota richiesta. Quest'ultimo collegamento è ritardato con un **pipe** di 10 millisecondi per permettere al messaggio '*stop'* di arrivare a destinazione prima che l'oggetto *coll* invii le note richieste. Quando i pitch vengono inviati dall'oggetto **coll** contenuto in *engine,* entra in un **send** denominato fromNotesPitch ed esce da due oggetti **receive**: uno in *insert_data,* collegato a **kslider** del pannello azzurro per evidenziare le note correnti; l'altro in *check_data* per la produzione dei suoni corrispondenti.
Sulla destra delle tastiere **kslider** troviamo due messaggi che servono per mostrare per ogni step i controlli memorizzati; il sistema è identico a quello presente nella finestra principale di Pilnote, perciò si rimanda al capitolo dedicato a tale finestra.

- CHECK_DATA –

La patch *check_data* si occupa di far suonare le note (da generare) in fase di revisione, per cui arrivano dall'oggetto **coll** le indicazioni di pitch delle note ma mancano le indicazioni di intensità e rilascio da parte dell'utente. Arrivano anche i valori di pitch in uscita dall'oggetto **kslider** nel pannello verde, infatti la situazione è la stessa: arrivano i valori di pitch delle note corrispondenti ai tasti selezionati ma mancano le indicazioni di intensità e rilascio da parte dell'utente. All'interno di *check_data* troviamo un oggetto **makenote** con il valore di velocity impostato come argomento a 80. La durata è impostata a un valore molto alto (1000 secondi), in modo tale che il suono venga fermato o dall'arrivo della nota successiva o da un clic sul tasto STOP; in entrambi i casi viene inviato un messaggio '*stop'* a **makenote**, che interrompe il suono. Come di consueto **makenote** è collegato a un **noteout** che riceve dalla finestra principale anche il device di uscita e il canale MIDI.

- ENGINE -

Come già accennato, la patch *engine* è il cuore del software. Vi sono contenute le strutture dati che memorizzano i dati e l'algoritmo di matching che permette di sincronizzare l'esecuzione con la generazione dei dati stessi.

Tutto parte dall'oggetto **notein**, in alto a sinistra, che riceve dall'oggetto **receive** denominato 'pilotDevice' l'informazione sul dispositivo MIDI scelto dall'utente per l'input. **Notein** è collegato a uno **stripnote**: questo è molto importante perché si vuole che il matching tra l'esecuzione e la melodia pilota avvenga solo in corrispondenza dell'inizio delle note (quindi di messaggi di NOTE ON), non del loro rilascio. Quindi, se il messaggio in arrivo è un NOTE OFF questo viene filtrato, se invece si tratta di un NOTE ON la velocity viene inviata ad oggetti deputati alla formattazione delle note generate (questo per attribuire anche alle note generate la stessa velocity delle note eseguite dall'utente), mentre il pitch viene inviato all'inlet di sinistra dell'oggetto == che contiene la funzione matematica di uguaglianza. Questo oggetto è di importanza cruciale, perché implementa il matching tra l'esecuzione e la nota attesa della melodia pilota. L'oggetto == confronta il numero ricevuto dall'inlet di sinistra con il numero ricevuto dall'inlet di destra, ma il comportamento dell'oggetto all'arrivo dei due input è diverso: quando un numero raggiunge l'inlet di destra viene semplicemente impostato come secondo operando, invece ogni volta che l'inlet di sinistra riceve un numero questo viene impostato come primo operando, e immediatamente viene anche triggerata l'operazione di uguaglianza, con conseguente invio del risultato dall'outlet. I risultati dell'operazione sono due: 1 se i numeri sono uguali, 0 altrimenti. L'output di == è stato connesso con un oggetto **sel** che invia un bang solo se l'input è 1, in altre parole invia un bang se e solo se si verifica un matching. Torneremo su questo oggetto più avanti, dopo aver introdotto gli oggetti **coll**.

Come già anticipato più volte in questo testo, *engine* contiene i due oggetti **coll** che contengono le strutture dati della melodia pilota (che per comodità chiameremo '**coll** di sinistra') e dei dati da generare (che chiameremo '**coll** di destra'). Nella figura 5 si possono osservare due file .txt di esempio. Gli oggetti **coll** possiedono un puntatore, che permette di navigare nella collezione. Sebbene le due collezioni si corrispondano, i due puntatori non punteranno allo stesso step durante l'esecuzione. Infatti, ogni volta che vi è un matching, cioè una corrispondenza tra la nota attesa e la nota eseguita dall'utente, l'oggetto **coll** di destra deve generare i dati relativi alla nota attesa, e contemporaneamente il **coll** di sinistra deve rendere noto al sistema il pitch della nota attesa successiva. Quindi il puntatore del **coll** di sinistra deve puntare sempre alla posizione successiva rispetto al puntatore del **coll** di destra. Come esempio, immaginiamo che l'esecuzione sia giunta allo step 50, quindi la nota attesa fa riferimento allo step 50 ed è un A4 (la sull'ottava centrale). Il **coll** di destra punta allo step 50, mentre il **coll** di sinistra punta allo step 51. Non appena l'utente suona un A4, viene inviato un messaggio '*next'* a entrambi gli oggetti **coll**, di conseguenza:

- il **coll** di destra invia i dati memorizzati allo step 50, che vengono generati, e il puntatore si sposta alla posizione 51;
- il **coll** di sinistra invia al sistema l'informazione di pitch della prossima nota attesa, memorizzata allo step 51, e incrementa il puntatore che si sposta allo step 52;

Prima di iniziare l'esecuzione l'utente clicca sul tasto START. In *engine* questo bang, che viene ricevuto tramite l'oggetto **receive** denominato 'toStart', triggera un messaggio '*start'* che viene inviato ad entrambi gli oggetti **coll**, con la conseguenza che i puntatori di questi si collocano allo step 1. Lo stesso bang, però, triggera anche un messaggio '*next'*, questa volta diretto al solo **coll** di sinistra, il quale invia il pitch relativo alla prima nota della melodia pilota (step 1) all'inlet di destra dell'oggetto **==,** e sposta il proprio puntatore sullo step 2.

Ogni volta che si verifica un matching, l'oggetto **sel** invia un bang che triggera un messaggio '*next'* diretto ad entrambi gli oggetti **coll**; e così il cerchio si chiude.

L'oggetto **coll** di destra, al momento della memorizzazione dei dati, viene popolato con un parsing di questo tipo:

[step], [['n' o 'cc' o 'cv'] [numero]]*;

dove []* indica che il contenuto delle parentesi quadre può ripresentarsi tante volte quante lo desidera l'utente. In altre parole, dato che nella stessa struttura dati abbiamo la coesistenza di tre tipi diversi di informazioni (pitch, numero di control change, valore relativo al control change), ogni elemento è accompagnato alla sua sinistra da un carattere che identifica il tipo di dato che l'elemento rappresenta.

figura 5

file .txt che contiene la melodia pilota

Pink Panther m...

```
1, 49;
2, 50;
3, 51;
4, 52;
5, 49;
6, 50;
7, 51;
8, 52;
9, 49;
10, 50;
11, 51;
12, 52;
13, 51;
14, 50;
15, 49;
16, 48;
17, 49;
```

Insertion Point Line: 1

file .txt che contiene i dati da generare

Pink Panther data

```
1, n 44 n 69;
2, n 45 n 70;
3, n 46 n 63 n 82 cc 47 cv 98;
4, n 47 cc 47 cv 0;
5, n 44 n 57 cc 5 cv 10 cc 10 cv 100 n 84;
6, n 45 cc 5 cv 0 cc 10 cv 0;
7, n 46 n 63;
8, n 47 n 69 n 50 n 91 n 60 n 52;
9, n 44;
10, n 45 n 76;
11, n 46 n 83 n 55;
12,;
13, n 46 n 65;
14, n 45 n 54 n 68 cc 64 cv 100;
15, cc 64 cv 0;
16, n 43;
```

Insertion Point Line: 1

Quando vengono generati i dati, dunque, l'oggetto **coll** invia una lista siffatta di elementi, che deve essere opportunamente smistata. Per fare ciò, la lista passa attraverso un oggetto **zl iter** che suddivide la lista in liste più piccole formate da soli due elementi (2 è impostato come argomento di **zl iter**). Queste piccole liste raggiungono un oggetto **route** che, a seconda della stringa ricevuta come primo elemento ('n', 'cc' o 'cv') indirizza il secondo elemento rispettivamente al primo, al secondo o al terzo outlet.

I valori di pitch, che escono dal primo outlet di **route**, entrano in un selettore **gswitch2**, le cui due uscite corrispondono alle due modalità di esecuzione, 'follow' e 'free' (infatti i comandi su quale collegamento attivare arrivano dall'interruttore free/follow posto sulla finestra principale, tramite una coppia di **send/return**).

Il caso 'free' è piuttosto semplice: i valori di pitch entrano in un **makenote,** che riceve l'indicazione di velocity direttamente da stripnote (cioè dall'esecuzione da parte dell'utente) e la durata dall'oggetto number recante la scritta duration sulla finestra principale.

Il caso di 'follow' invece è piuttosto complesso, per questo è stata creata una sottopatch apposita che prende in input i valori di pitch (da **route**), di velocity (da **stripnote**), il bang in uscita da **sel** e anche il pitch della nota attesa.

In entrambi i casi, i valori di pitch e velocity in uscita da **makenote** e dalla sottopatch *follow* confluiscono in un oggetto **noteout** che provvede alla fuoriuscita delle note generate sull'output desiderato.

I numeri di control change e i rispettivi valori, in uscita rispettivamente dal secondo e terzo outlet di route, entrano in un oggetto pack che li impacchetta in liste di due elementi. Per poter essere fruibili, occorre creare delle liste di due elementi così formate:

[[control change][valore]]

Dato che, in *store_data*, questi due elementi vengono memorizzati nell'oggetto **coll** con questa stringa:

'merge [step] cc [control change] cv [valore]'

l'elemento [control change] arriva per primo all'oggetto **pack**, perciò va memorizzato facendolo entrare nell'inlet di destra. Non appena arriva anche l'elemento [valore], nell'inlet di sinistra, **pack** invia la stringa di due elementi, che quindi sarà così formata:

[[valore][control change]]

Grazie all'oggetto **zl rev** possiamo invertire l'ordine degli elementi, ottenendo la stringa corretta. Questa viene inviata a **midiformat,** e quindi a un midiout per la fuoriuscita dei controlli sull'output desiderato.

- BOOKMARKS -

La patch è formata da due insiemi di oggetti disgiunti. La parte a sinistra si occupa di impostare i tre segnalibri allo step desiderato e di azionarli su ricezione del rispettivo bang. Si tratta di tre moduli identici, uno per ogni segnalibro. L'elemento di complessità sta nel fatto che il puntatore dell'oggetto **coll** contenente la melodia pilota durante l'esecuzione punta sempre allo step successivo rispetto a quello puntato dall'oggetto **coll** contenente i dati. Dunque, volendo impostare il sistema a essere pronto a partire da qualsiasi punto, occorre impostare i due puntatori distanziati di una posizione. Inoltre bisogna triggerare il **coll** della melodia pilota a inviare la nota che corrisponde allo step desiderato, in modo tale che il sistema sappia qual è la nota attesa, che quidi può anche essere visualizzata dall'utente sull'oggetto kslider che reca la scritta NEXT NOTE.

Per esempio, se il segnalibro deve preparare Pilnote a partire dallo step 50, occorre:

- muovere il puntatore del **coll** contenente i dati allo step 50;
- muovere il puntatore del **coll** contenente la melodia pilota allo step 51;
- inviare al **coll** contenente la melodia pilota un messaggio con il numero 50, in modo che invii in output la nota contenuta allo step corrispondente.

A tale scopo, il numero di step in arrivo ad un modulo viene usato per preparare, tramite messaggi parametrici, due ulteriori messaggi così definiti: '*goto'* [step]. In uno dei due casi, però, lo step viene incrementato di uno facendolo passare in un oggetto +. I due messaggi sono connessi tramite coppie di send/return agli oggetti **coll** corrispondenti, che stanno nella patch *engine*. Un terzo messaggio viene settato sul numero di step richiesto, e connesso con l'oggetto send che porta al **coll** della melodia pilota. Questi tre messaggi vengono triggerati ogni qualvolta arriva un bang al modulo corrispondente, e vengono inviati sui rispettivi collegamenti.

La parte destra della patch ottempera all'uso dei segnalibri tramite controlli assegnabili. Esattamente come avviene nella finestra principale per l'implementazione della funzione clunch, l'oggetto **midiin** riceve dal receive pilotDevice l'informazione su quale dispositivo di input è stato scelto dall'utente, e invia tutti messaggi ricevuti da questo input all'oggetto **midiparse**. Quest'ultimo filtra tutti i messaggi inoltrando a un oggetto **unpack**, da un apposito outlet, solo i messaggi di control change come coppie di elementi (liste di due elementi): il primo per il numero di control change, il secondo, per il valore. **Unpack** spacchetta queste liste inviando il primo elemento di ogni lista dall'outlet di sinistra. I numeri di control change arrivano a un oggetto **sel** che invia un bang solo se il numero ricevuto è uguale a 0 (di default) oppure a un numero che viene ricevuto dall'inlet di destra, ovvero dall'oggetto **number** sulla finestra

principale di fianco alla scritta **"cc"** del segnalibro corrispontente. Quindi, quando viene ricevuto un control change che corrisponde a quello scelto, viene inviato un bang, tramite una coppia di **send/receive**, al modulo corrispondente sulla parte sinistra della patch.

- FOLLOW -

La modalità 'follow', come si è già accennato in altre parti del testo, consente di far coincidere la durata delle note generate con la durata delle note della melodia pilota corrispondenti. In altre parole, le note generate in simultanea con l'esecuzione della nota X sono rilasciate soltanto nell'istante esatto in cui viene rilasciata X. Come conseguenza di ciò, è possibile suonare *legato*, ovvero suonare due note successive legando i due suoni, senza che tra i due ci sia interruzione. Questo implica, nella pratica, che possa esserci una sovrapposizione, seppure breve, tra le due note. La complessità di questa patch deriva proprio dalla gestione di questa sovrapposizione tra le note generate. Infatti nel momento in cui viene suonata la seconda nota, tenendo suonata la prima, il sistema ha generato nuove note, e quindi si è già "dimenticato" delle note precedenti, così come della nota attesa allo step precedente, e anche i puntatori nel frattempo sono stati spostati. Al momento del rilascio della prima nota, il sistema ha bisogno di sapere quali tra tutte le note generate in essere devono essere rilasciate simultaneamente al rilascio di questa nota, e quali devono continuare a suonare fino al rilascio della seconda. Per fare ciò è stato implementato un meccanismo di memorizzazione temporanea di note generate e note attese corrispondenti, costituito da due locazioni:

- Quando viene suonata la prima nota, vengono generate alcune note, che vengono memorizzate nella prima locazione insieme alla nota della melodia pilota corrispondente (per esempio allo step 50). Quest'ultima diventa la 'nota attesa' per questa locazione, che chiamiamo locazione 1.
- Viene suonata la nota successiva della melodia pilota (che diventa la nota attesa della locazione 2), con conseguente generazione di altre note (step 51). Queste vengono memorizzate nella locazione 2.
- Le locazioni restano sempre in ascolto di tutti i messaggi di NOTE OFF provenienti dall'utente. Non appena l'utente rilascia la nota attesa della locazione 1, le note memorizzate in essa vengono rilasciate, ossia vengono prodotti dei messaggi di NOTE OFF per le note memorizzate nella locazione 1.

- Se viene suonata la prossima nota attesa dal sistema, quella dello step 52, questa e le note generate corrispondenti vengono memorizzate nella locazione 1, sovrascrivendo la memorizzazione precedente, e via discorrendo.

In realtà l'implementazione prevede l'utilizzo di tre locazioni. Il motivo risulterà chiaro in seguito.

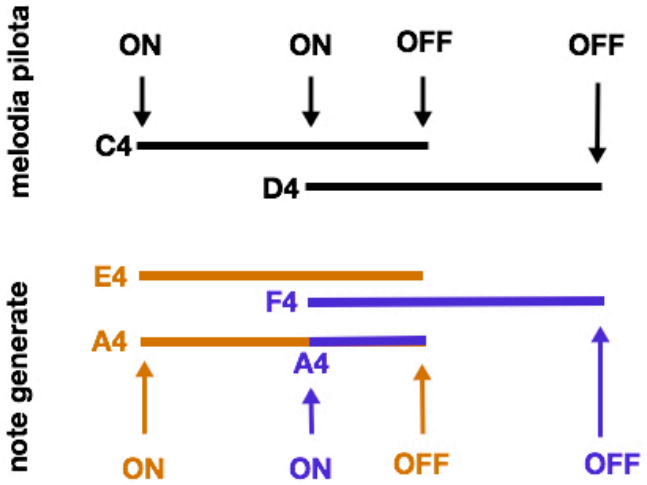
Un ulteriore problema, che è stato gestito, riguarda le note ribattute. Potrebbe capitare di avere memorizzato una nota che deve essere generata sia in uno step, sia nello step successivo. In questo caso, suonare legato comporta un problema con il rilascio della seconda nota, infatti quando verrà rilasciata la prima, questo messaggio di NOTE OFF farà rilasciare anche la successiva, dato che hanno la stessa intonazione (fig. 6). Inoltre, si è osservato come in alcuni moduli di sintesi sonora (non in tutti), alla ricezione di due NOTE ON successivi per lo stesso pitch, il secondo viene praticamente ignorato. Per ovviare a questi problemi, all'interno della patch è stato creato un sistema che confronta le note generate ad ogni step con le note generate allo step precedente, in cerca di note comuni. Se ve ne sono, si applica la seguente procedura:

- Si genera un NOTE OFF per ognuna delle note comuni ai due step (corrente e precedente)
- Si rigenera un NOTE ON per ognuna delle note comuni
- Si informa la locazione che memorizza le note generate allo step precedente di non generare messaggi di NOTE OFF per le note comuni

Si può notare come questa procedura non comporti nessun problema nel caso in cui le due note vengono suonate staccate (cioè non legate).

Questa patch contiene inoltre l'implementazione delle note tenute: è possibile tenere

figura 6



suonata una nota della melodia pilota, e di conseguenza le note generate corrispondenti, e nel frattempo proseguire l'esecuzione degli step successivi, senza che la funzione del legato ne risulti compromessa. Questo è il motivo per cui sono stati utilizzate tre locazioni di memoria e non due: avendo a disposizione tre locazioni, è possibile dedicare una locazione alle note tenute, mentre le altre due garantiscono l'efficacia del legato per le note seguenti. Per riuscire a implementare efficacemente questo meccanismo, occorre tenere presente che :

- Ogni nota che viene suonata potrebbe diventare una nota tenuta, ma quest'informazione non è nota a priori dal sistema, e rimane a discrezione dell'esecutore fino al momento del rilascio della nota stessa
- Mentre è in essere una nota tenuta, una nota successiva della melodia pilota che viene nel frattempo suonata, potrebbe candidarsi per diventare una nota tenuta, dopo il rilascio della nota tenuta in essere.

Per tenere conto di queste problematiche, è stato introdotto il concetto di lucchetto: ogni volta che una nota attesa viene suonata, e non vi è in essere alcuna nota tenuta, questa nota si appropria del lucchetto, vale a dire che diventa una potenziale nota tenuta. Se questa nota diventa effettivamente una nota tenuta, il lucchetto impedisce alla locazione corrispondente di essere sovrascritta, e la funzione del legato per le note successive della sequenza si giocherà tra le altre due locazioni. Se invece la nota viene rilasciata prima della nota successiva, il lucchetto viene passato alla locazione che corrisponde a quest'ultima.In caso di nota tenuta, la cui locazione detiene il lucchetto, occorre determinare quale delle altre due locazioni è la candidata ad ottenere il lucchetto dopo il rilascio della nota tenuta, e con quale criterio. L'implementazione attuale prevede un criterio di tipo First-In-First-Out, ovvero la locazione corrispondente alla prima nota che viene suonata dopo la nota tenuta, "prenota" la ricezione del lucchetto. Se si aggiunge una terza nota sovrapposta alle altre due, la terza locazione si metterà in coda per l'acquisizione del lucchetto. A questo punto, se la prima nota candidata dovesse essere rilasciata, la locazione corrispondente viene rimossa dalla coda, e la seconda candidata diventa prima candidata (fig. 7)

Procediamo ora con l'analisi dell'implementazione. Per semplicità, visto considerato il gran numero di cablaggi di questa patch, sono stati nascosti dallo screenshot (disponibile nell'appendice) le connessioni che riguardano le inizializzazioni degli oggetti.

La patch si può immaginare divisa a metà verticalmente: nella parte sinistra (in alto) ci sono gli oggetti che implementano il routing delle note verso la locazione corretta e di conseguenza la gestione del lucchetto. Nella parte destra ci sono le tre locazioni, ovvero tre moduli uguali di oggetti interconnessi, a loro volta divisi in due parti: la parte alta è dedicata alla memorizzazione

figura 7

delle note della melodia pilota, nella parte bassa alla memorizzazione delle note di cui bisogna generare il rilascio nel momento corretto.

I valori di pitch delle note da generare giungono alla patch dall'inlet numero 2. Da qui entrano in uno gswithc2 a tre vie, che le direziona :

- in basso, all'uscita della patch per la generazione dei messaggi di NOTE ON, con velocity uguale a quella della nota attesa corrispondente eseguita dall'utente. Questo collegamento è ritardato con un pipe per permettere alle note ribattute di generare un NOTE OFF prima del secondo NOTE ON.
- alla prima locazione disponibile per essere memorizzate fino al momento corretto per il rilascio.

Gswitch2 sposta l'output della connessione di una posizione verso destra ogni volta che riceve un bang, oppure, se riceve un numero compreso tra 0 e 2, viene spostato sull'outlet corrispondente (l'outlet 0 è quello di sinistra, l'outlet 1 è quello al centro, l'outlet 2 è quello di destra).

Il lucchetto è stato implementato in questo modo: l'oggetto **counter** in alto a sinistra invia, ogni volta che arrivano nuove note da generare, un numero compreso tra 0 e 2 (in loop), e immediatamente si autoincrementa. Questo numero passa attraverso un **sel**, che riceve il numero della locazione che detiene il lucchetto dall'inlet di destra. L'outlet di destra di **sel** è collegato all'inlet che comanda il routing di **gswitch2**, cosicchè **counter** può dare indicazione a **gswitch2** su quale uscita attivare solo se questa non coincide con la locazione lucchettata. Se invece il numero in uscita da **counter** è proprio il numero della locazione lucchettata, **sel** invia un bang allo stesso **counter** che salterà al numero successivo.

Lo stesso discorso vale per le note della melodia pilota: queste arrivano dall'inlet 3 e entrano in un **gswitch2** a tre vie con la stessa inizializzazione di quello appena citato. I due oggetti **gswitch2** si muovono sempre in parallelo, infatti ricevono entrambi i comandi dall'outel destro di **sel**. Ad ogni corrispondenza, la nota della melodia pilota raggiunge la parte superiore di una locazione, e nel contempo triggera un messaggio contenente il numero della locazione corrispondente (0, 1 o 2). Se non ci sono in essere note tenute, allora la nota detiene il lucchetto, e infatti il numero della locazione serve per informare l'oggetto **sel** citato in precedenza riguardo alla posizione del lucchetto. Se invece c'è già una nota tenuta (che quindi detiene il lucchetto), il numero di locazione viene inserito nella coda FIFO, rappresentata dall'oggetto **bag**. Vi è un secondo oggetto **counter**, a destra di **bag**: la sua utilità consiste nel contare quante note della melodia pilota sono in esecuzione (cioè suonanti) in ogni istante. Questo **counter** non è impostato per contare in modalità loop, infatti:

- viene inizializzato a zero;
- ogni volta che si verifica una corrispondenza viene incrementato di uno;
- ogni volta che registra un rilascio viene decrementato di uno.

In questo modo risulta molto utile per decidere se la locazione corrispondente a una nota in arrivo deve essere lucchettata oppure se, essendoci già una nota tenuta, deve essere inserita nella coda FIFO.

La parte superiore di una locazione memorizza la nota (o, per meglio dire, il suo pitch) della melodia pilota che corrisponde alle note memorizzate nella parte inferiore della locazione stessa. In particolare, questo pitch è memorizzato in un oggetto ==, che rimane in ascolto di tutte le note eseguite dall'utente tramite un oggetto **notein**. Un altro oggetto == controlla che il messaggio in arrivo abbia velocity uguale a zero, ovvero che sia un NOTE OFF. Non appena arriva un messaggio il cui pitch è uguale a quello memorizzato e la cui velocity sia uguale a zero, viene prodotto un bang che fa svuotare l'oggetto **bag**, nella parte inferiore della locazione, che contiene le note generate (più precisamente i pitch) in attesa di essere rilasciate. Questi pitch vengono inviati dall'apposito outlet, e nel contempo triggerano un messaggio contenente 0 che viene inviato dall'outlet dedicato alla velocity. Una volta fuoriusciti dalla sottopatch, questi valori di pitch verranno incapsulati in un messaggio MIDI con velocity zero, e verranno interpretati come messaggi di NOTE OFF.

La gestione delle note ribattute è implementata nella parte inferiore delle tre locazioni:

- Ogni volta che vengono ricevuti i pitch delle note da memorizzare, questi, oltre a raggiungere l'oggetto **bag**, raggiungono anche un oggetto thresh, che raggruppa elementi ricevuti singolarmente e in una breve frazione di tempo in un'unica lista.

- Questa lista viene inviata all'oggetto **zl sect** della stessa locazione e a quello della locazione successiva (in termini di successione dei numeri identificativi; per esempio per la locazione 1, la locazione successiva è la 2). La locazione successiva, per come è stata implementata la patch, contiene le note generate allo step successivo, mentre la locazione precedente contiene le note generate allo step precedente.
- L'oggetto **zl sect** riceve la lista dei pitch della locazione a cui appartiene e quella della locazione precedente, e confronta le due liste in cerca di valori comuni.
- Se ne trova, li invia all'inlet di destra dell'oggetto **zl filter** della locazione precedente; quest'ultimo filtrerà i pitch provenienti dall'oggetto **bag** inoltrando solo quelli non comuni alle due locazioni, generando quindi messaggi di NOTE OFF soltanto per le note che non sono state ribattute.

Data l'esistenza di questi scambi di dati tra le varie locazioni, è stato necessario rendere compatibile il meccanismo di gestione delle note ribattute con il meccanismo di gestione delle note tenute (il passaggio del lucchetto). Per fare ciò, ci si è serviti dell'oggetto **counter** che conta quante note della melodia pilota sono in esecuzione simultaneamente. Se il conteggio è minore di 3, allora la situazione è riconducibile a quella di un normale legato, e dunque non va corretta. Se invece il conteggio è uguale a 3, cioè siamo in presenza di una nota tenuta, occorre escludere dalle connessioni tra una locazione e l'altra quella con il lucchetto. Per questo motivo ogni collegamento è stato sdoppiato tramite un **gswitch2** per connettere ogni locazione con le altre due. Solo e soltanto quando il conteggio è uguale a 3, le connessioni vengono dirottate a comando in modo tale da escludere dagli scambi la locazione lucchettata.

CONCLUSIONI

Le possibilità creative offerte da Pilnote non sono ancora state esplorate in modo soddisfacente. Si è sperimentato come esso sia molto efficace nella creazione di armonizzazioni, come ad esempio le armonizzazioni per le sezioni di ottoni, o per coro. Considerando questo ambito di applicazione, può essere anche uno strumento molto utile agli arrangiatori per velocizzare il processo di scrittura e verifica di un'armonizzazione, anche senza saper suonare il pianoforte.

Avendo a disposizione un numero illimitato di note da poter generare e una rapidità di azione pressochè istantanea, rende possibile eseguire parti che per un musicista singolo non sono riproducibili con nessun altro strumento musicale esistente. Si provi a immaginare, per esempio, un'armonizzazione a sei voci, a parti late (anche "molto late"), suonata con estrema velocità, usando solamente una mano su una tastiera, oppure un flauto MIDI. Anche considerando un pianista, che sicuramente ha possibilità esecutive, in termini di polifonia, maggiori rispetto agli altri strumenti, ci troviamo di fronte a un limite teorico di dieci note di polifonia (senza contare il pedale) , che comunque può avere un'estensione limitata e una velocità di esecuzione molto bassa. Con ciò non si intende inneggiare a Pilnote come uno strumento sostitutivo di altri strumenti. Sarebbe assurdo, considerato che l'intera architettura si basa su alcuni limiti, come il fatto di avere un output che coincide ritmicamente con l'input, o il fatto di dover memorizzare i dati prima dell'esecuzione. Sono elementi caratterizzanti che rendono Pilnote definibile come un supporto informatico all'esecuzione, più che come uno strumento musicale. Ma le sue caratteristiche lo rendono capace di creare soluzioni musicali uniche, inedite.

La possibilità di generare controlli offre uno strumento di modifica automatica del suono sincronizzata con l'esecuzione, del tutto originale. Essa può coinvolgere non solo il suono prodotto dall'esecutore di Pilnote, ma anche altri musicisti con cui ci si trova a condividere l'esecuzione. Come ogni meta-strumento, somiglia molto a un direttore d'orchestra [4]. E proprio come i direttori d'orchestra (almeno fino al secolo scorso) è libero da metronomo. Tutte le considerazioni fatte finora hanno valore se si continua a rimanere nell'ottica per cui la novità di Pilnote è di poter aggiungere elementi a un esecuzione liberando l'esecutore dalla meccanicità del clock.

Inoltre, il prototipo qui presentato è solo l'inizio, il cuore di un progetto espandibile e integrabile. Come già annunciato, tra gli sviluppi futuri c'è l'implementazione di una terza modalità di durata delle note generate, che fa coincidere il rilascio con l'inizio della nota attesa indipendentemente dalla durata della nota corrispondente della melodia pilota. Come auspicato

da Ludovico e Malcangi, può essere integrato al formato IEEE 1599, in modo da poter arricchire l'esecuzione di altri elementi multimediali o rendere Pilnote un utile strumento per l'apprendimento [2][4]. Dopodichè si apre l'ampio capitolo dell'integrazione con l'audio: lasciando inalterato l'algoritmo di base, si può abilitare Pilnote a ricevere un input audio invece che di tipo simbolico. Oppure ancora, uscendo dalla sfera strettamente musicale, si può implementare il matching con il parlato al posto delle note, utilizzando gli ormai avanzati sistemi di riconoscimento vocale.

La parte più bella è quella che deve ancora venire.

BIBLIOGRAFIA

[1] Haury, J.: *La Pianotechnie ou notage des partitions musicales pour une interprétation immédiate sur le Métapiano*, in Journées d'Informatique Musicale, Grenoble, 1er - 3 avril 2009

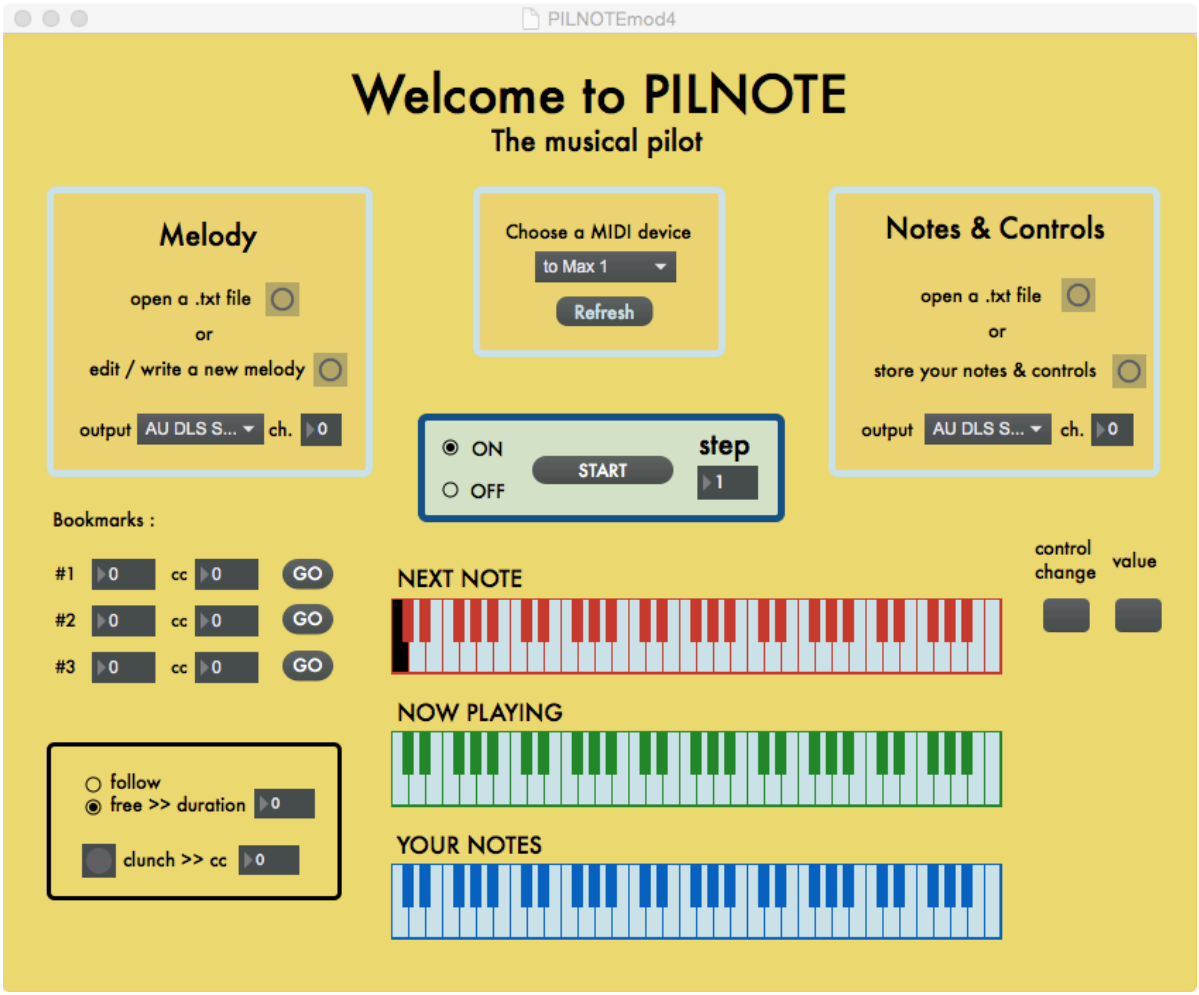
[2] Ludovico, L.A., Malcangi, M.N.: *An IEEE 1599 Framework to Play Music Intuitively. The Metapiano Case Study*, in Helfert, M., Restivo, M.T., Uhomoibhi, J., Zvacek, S. (eds.) Proceedings of the 6th International Conference on Computer Supported Education (CSEDU 2014), vol. 1, pp. 409-414. SciTePress - Science and Technology Publications, Barcelona (2014)

[3] Cipriani, A., Giri, M.: *Musica Elettronica e Sound Design. Teoria e pratica con MaxMSP,* ConTempoNet Editore, 2013.

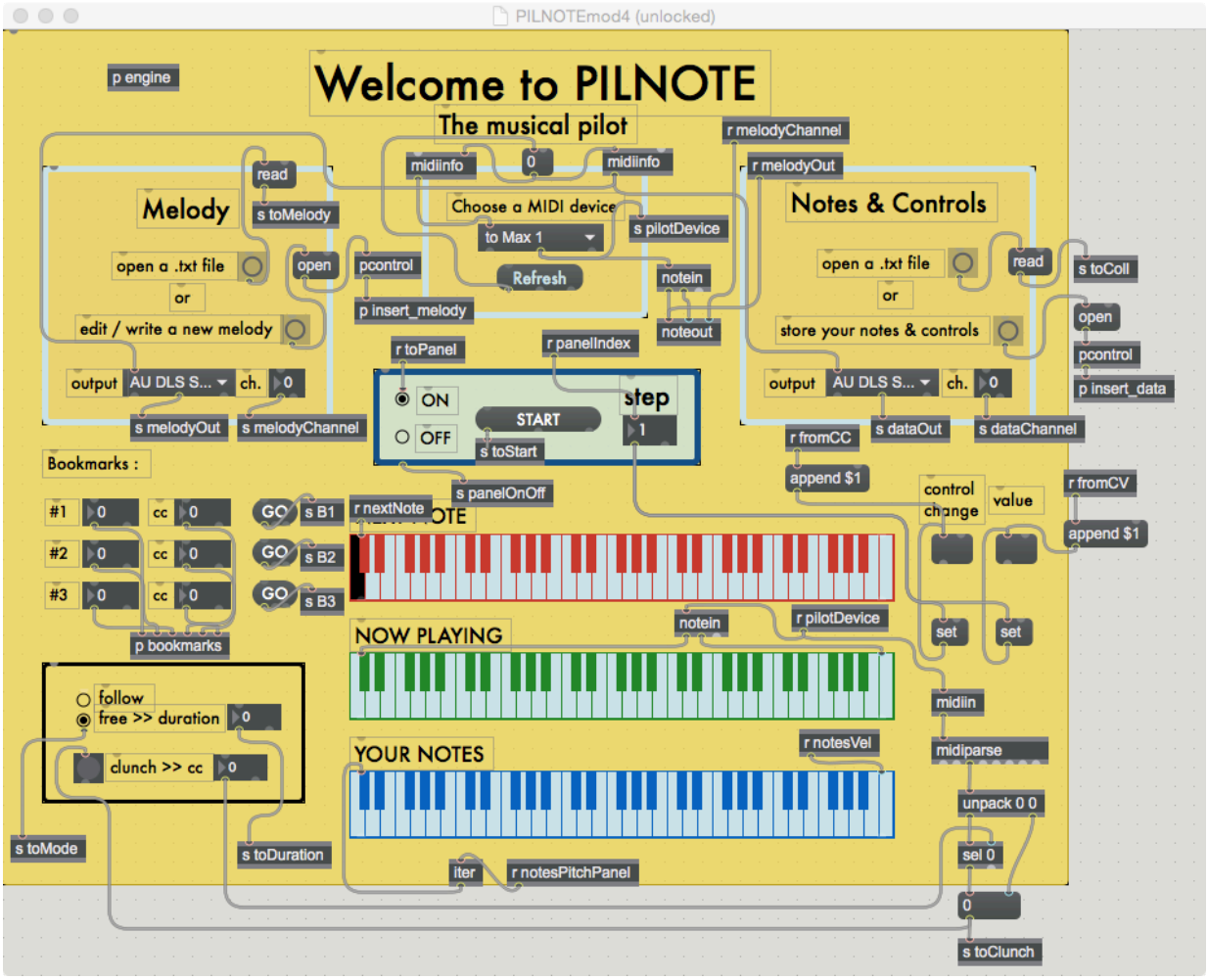
[4] Ludovico, L.A., Malcangi, M.N.: *Meta-instrument and Natural User Interface: a New Paradigm in Music Education*, in: Cermáková, K., Rotschedl, J. (eds.) Proceedings of the 13th International Academic Conference, Antibes, France, 15 - 18 September 2014, pp. 283-290. International Institute of Social and Economic Sciences (IISES), Antibes (2014)

APPENDICE: SCREENSHOTS DELLE FINISTRE E DELLE PATCH

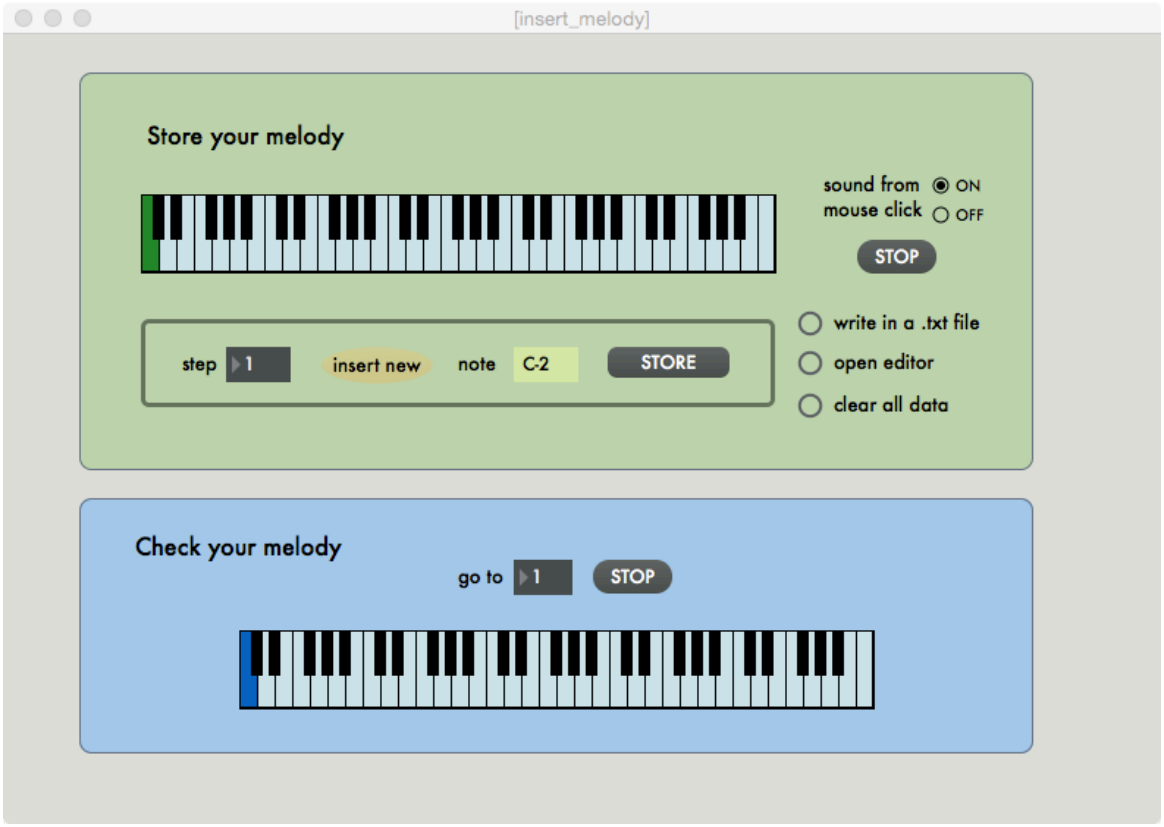
- FINESTRA PRINCIPALE: locked -



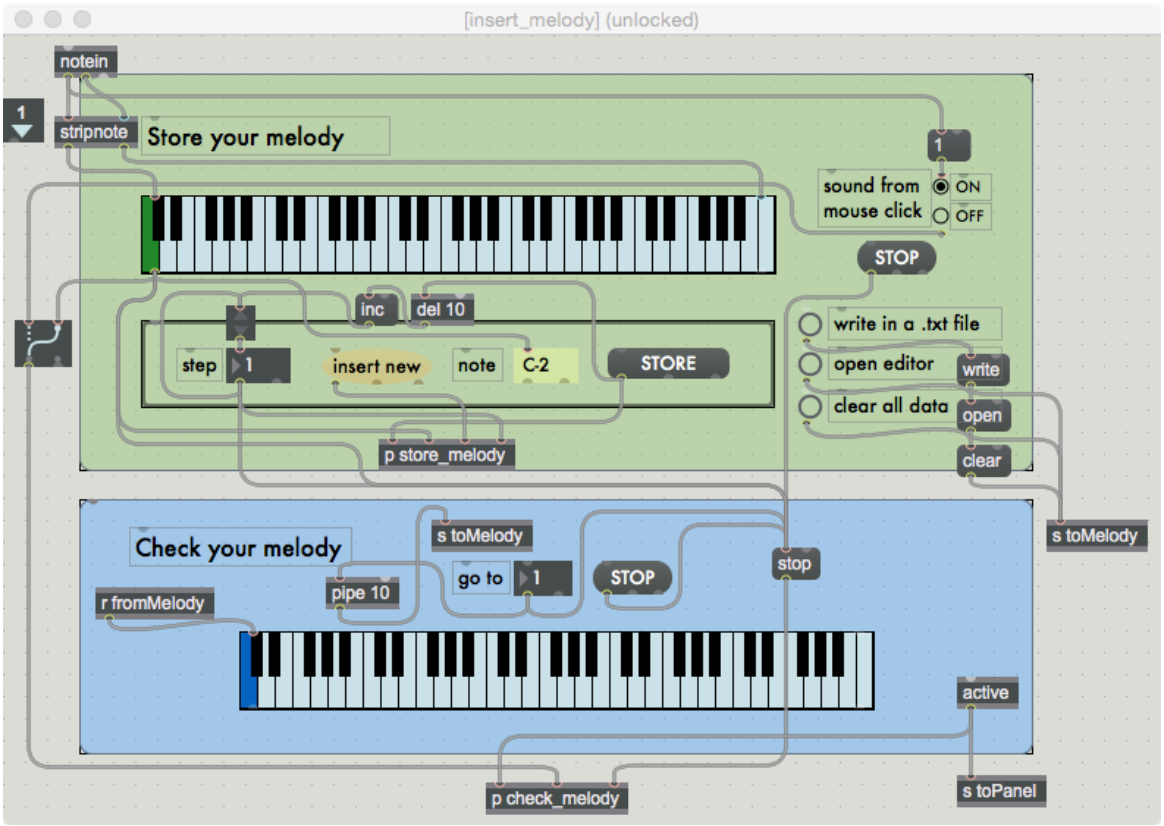
- FINESTRA PRINCIPALE: unlocked -



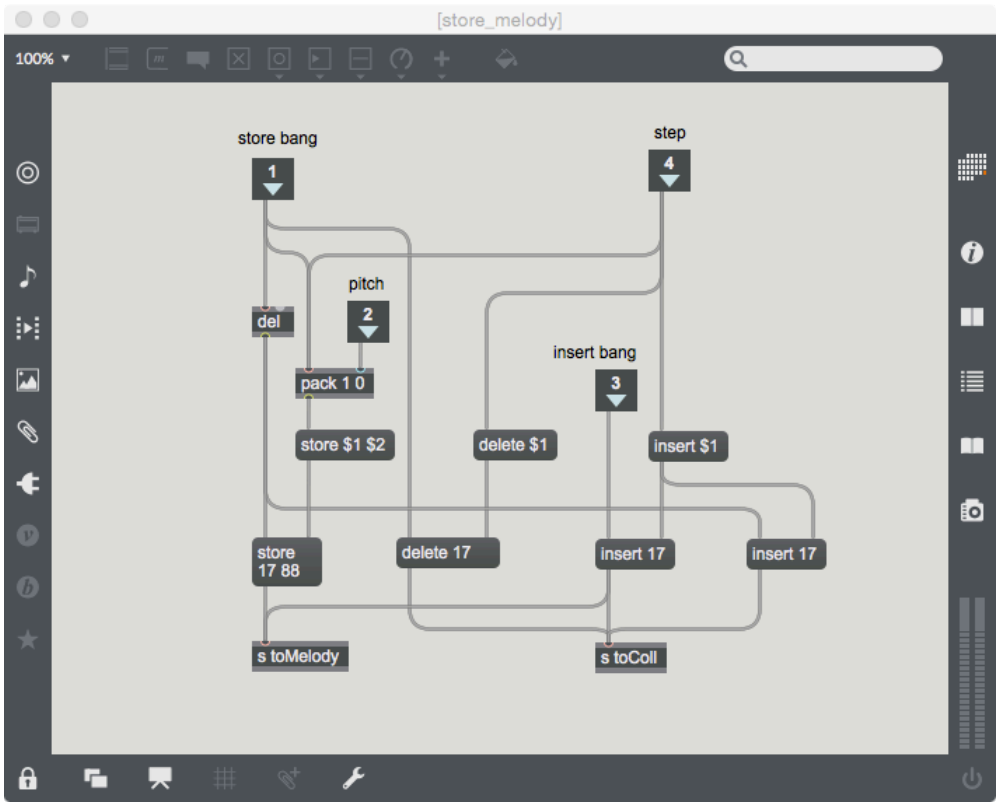
- INSERT_MELODY: locked -



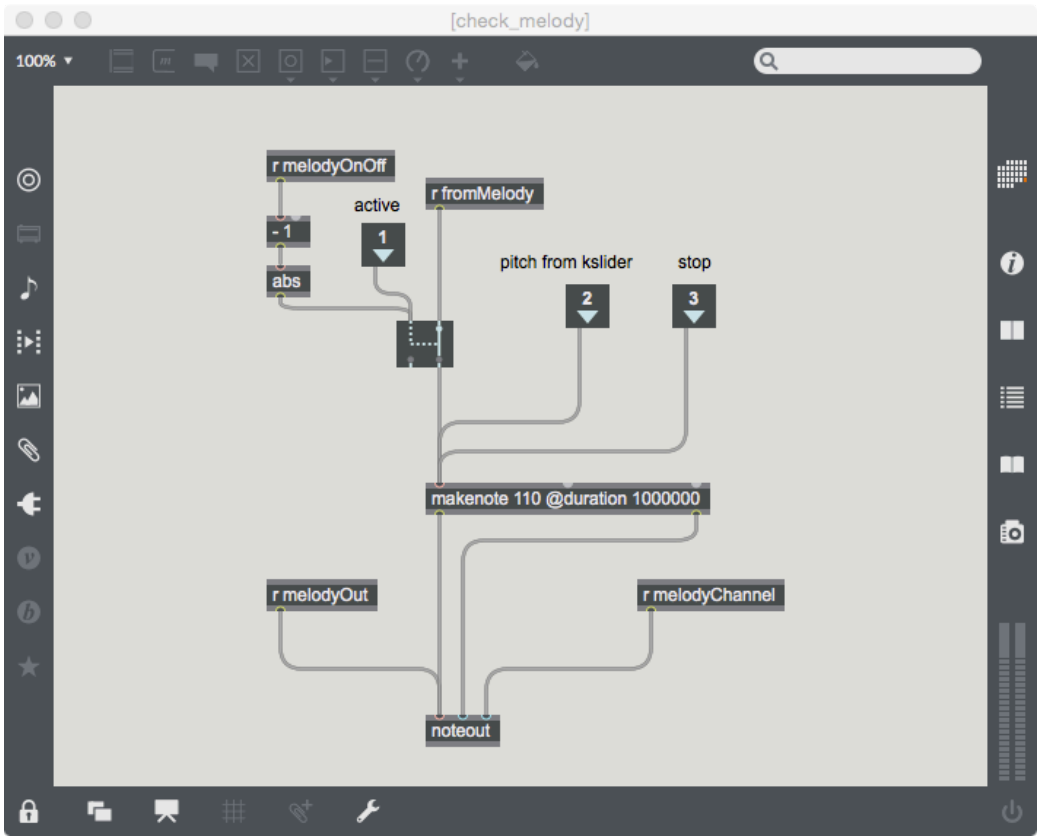
- INSERT_MELODY: unlocked -



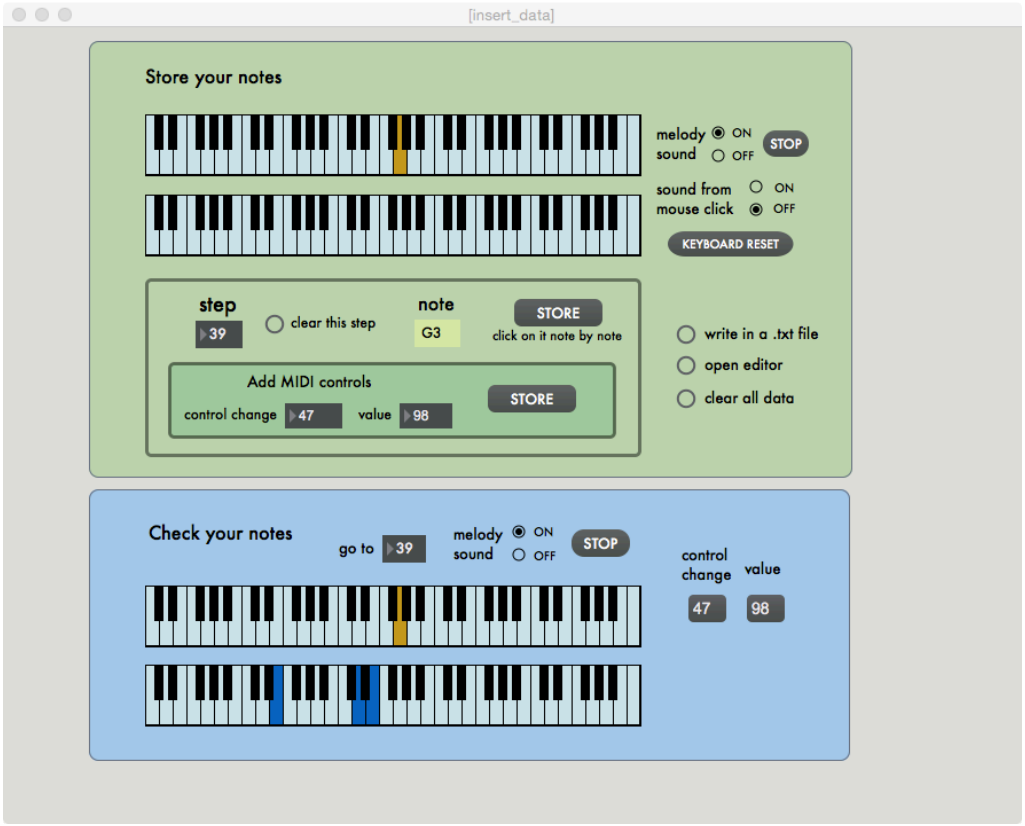
- STORE_MELODY -



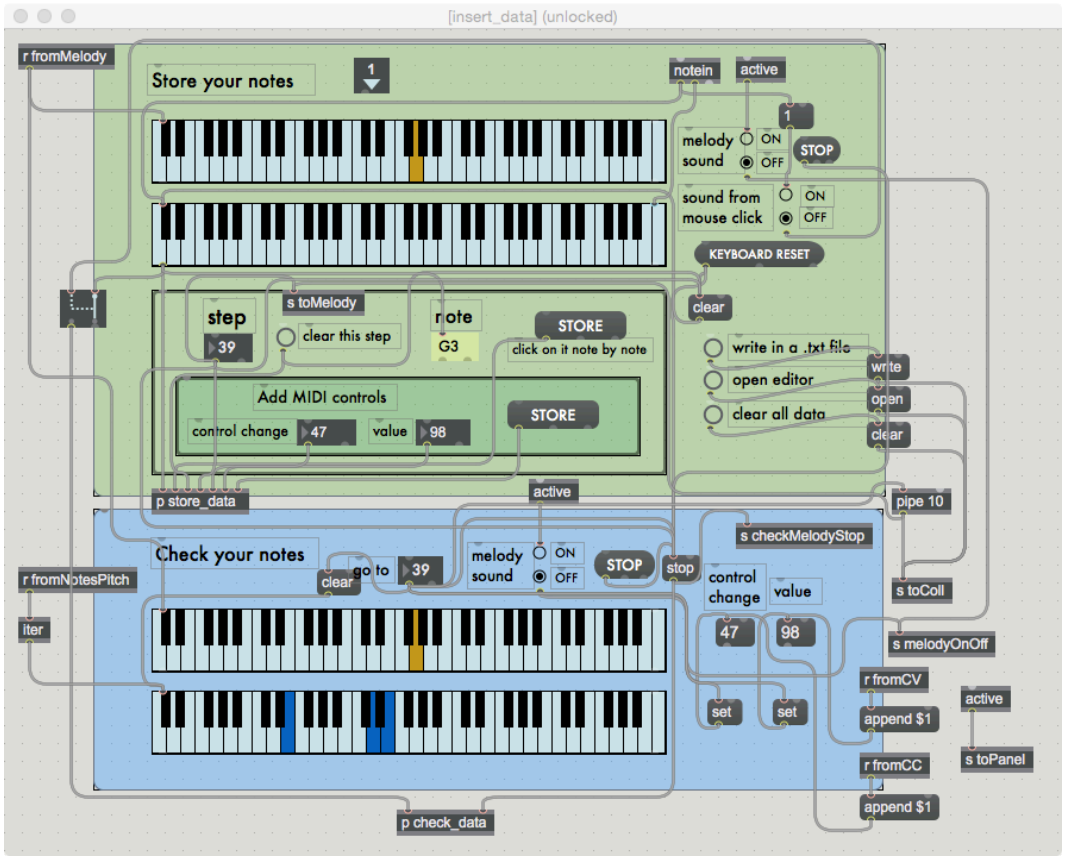
- CHECK MELODY -



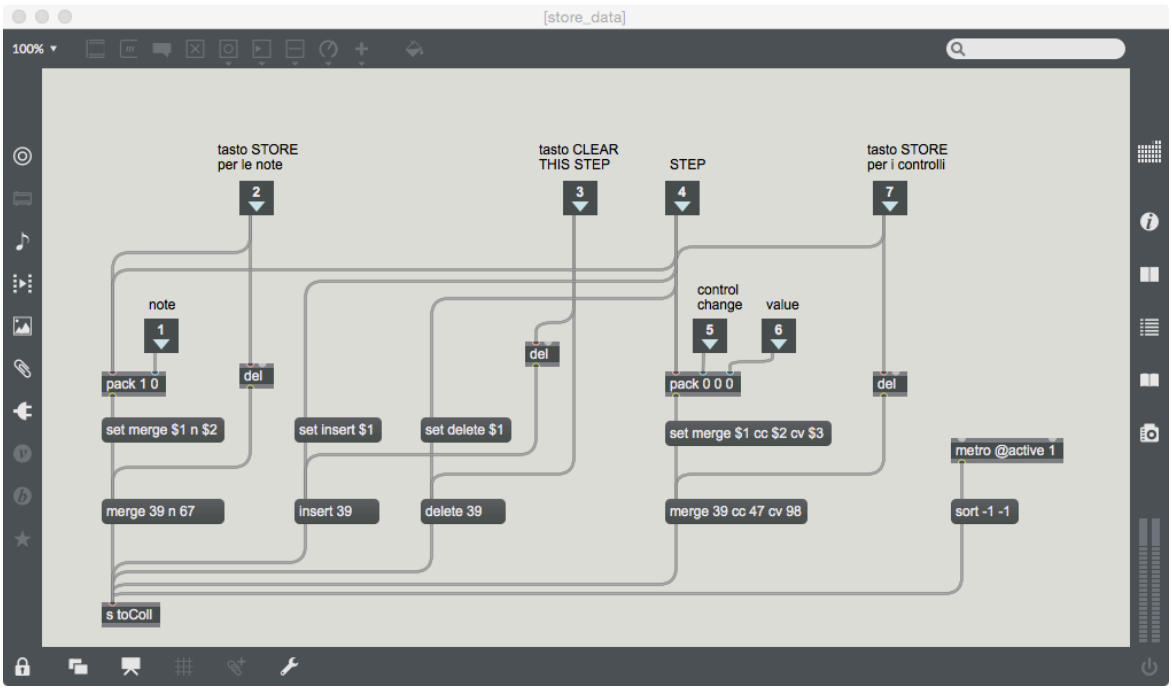
- INSERT_DATA: locked -



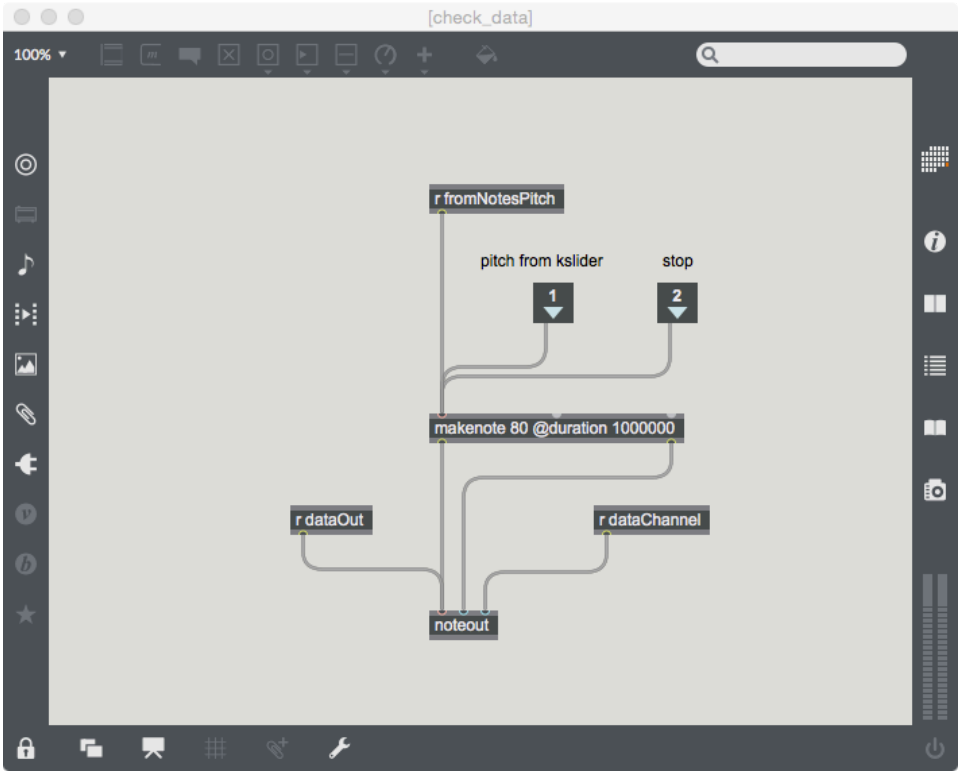
- INSERT_DATA: unlocked -



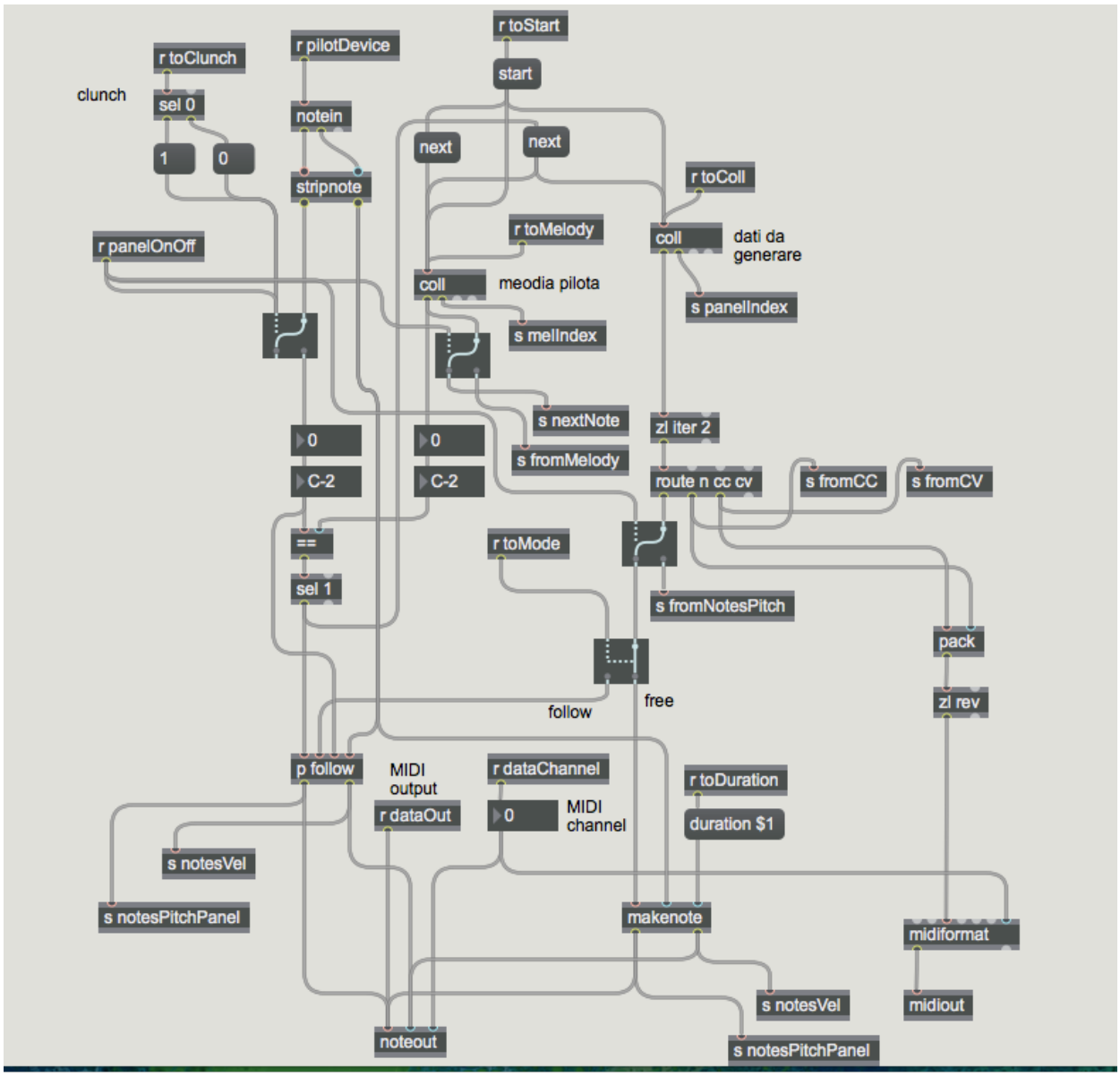
- STORE_DATA -



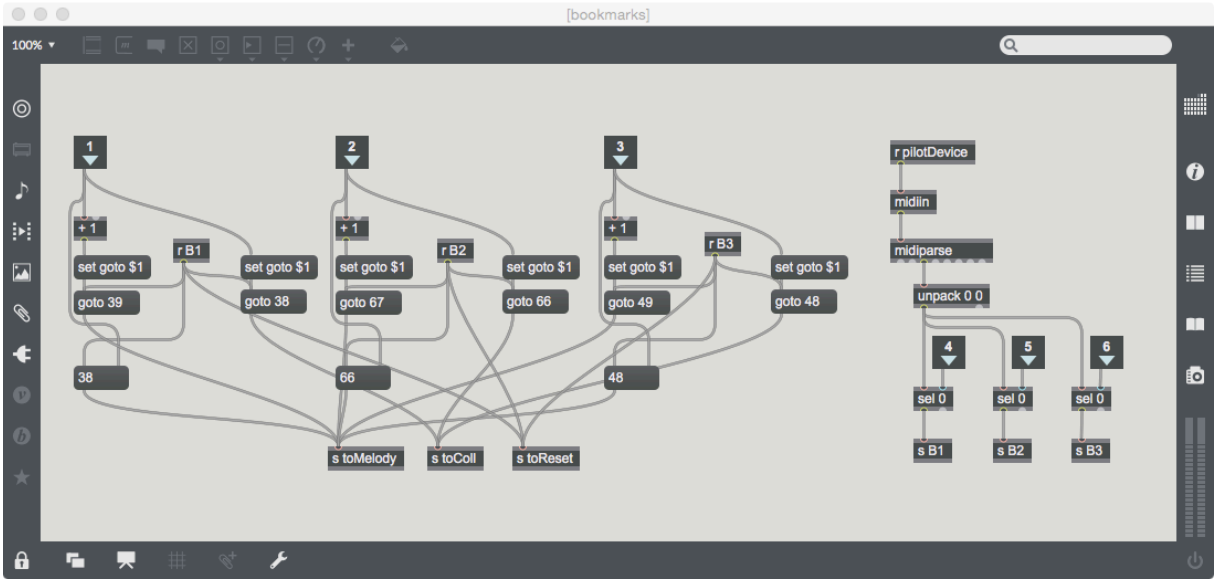
- CHECK_DATA -



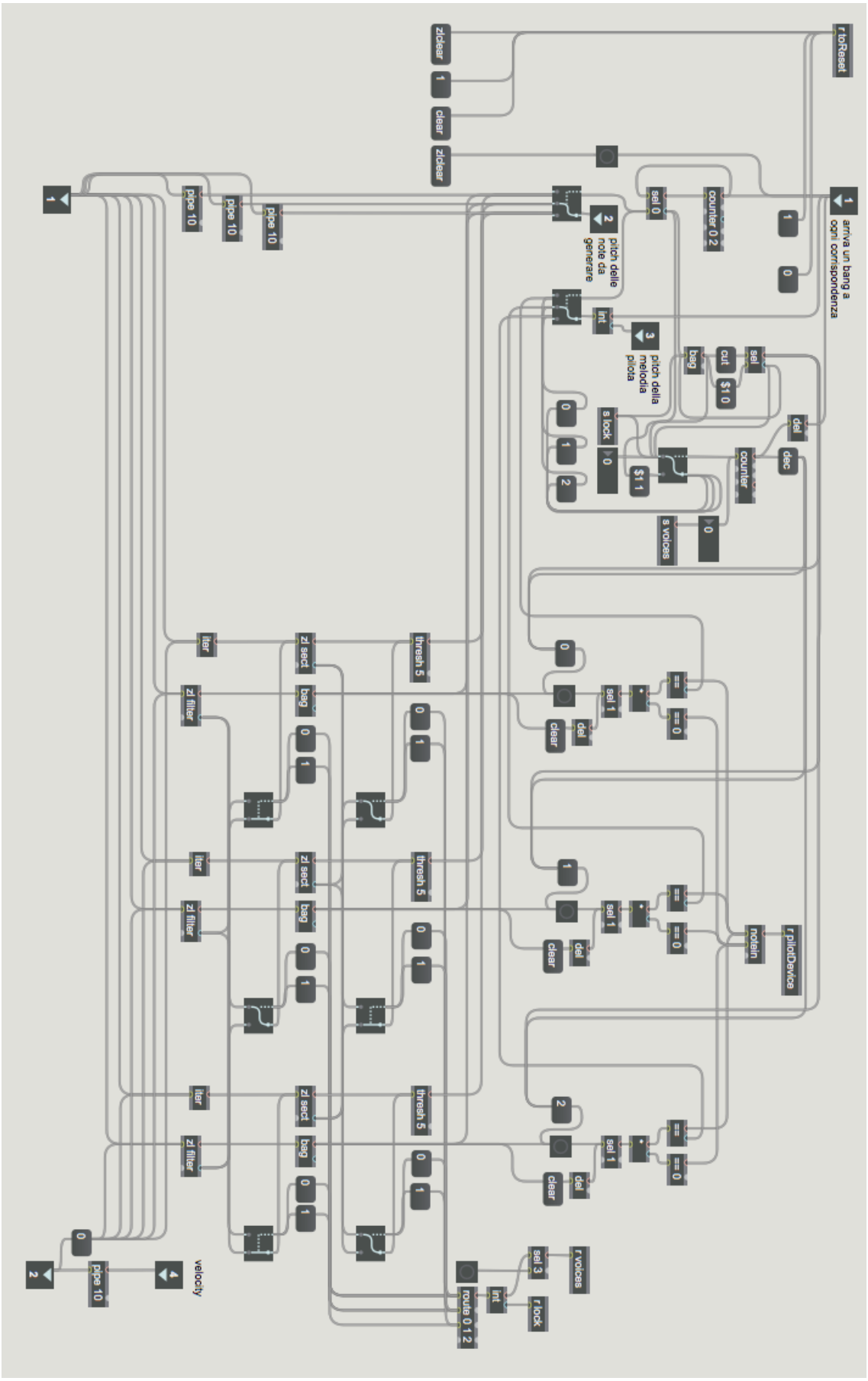
- ENGINE -


r toStart
r toClunch
r pilotDevice
clunch
start
sel 0
notein
next
next
1
0
stripnote
r toColl
r toMelody
coll
dati da generare
r panelOnOff
coll
meodia pilota
s panelIndex
s melIndex
s nextNote
zl iter 2
0
0
s fromMelody
C-2
C-2
route n cc cv
s fromCC
s fromCV
==
r toMode
sel 1
s fromNotesPitch
pack
free
follow
zl rev
p follow
MIDI output
r dataChannel
r toDuration
r dataOut
0
MIDI channel
duration $1
s notesVel
s notesPitchPanel
makenote
midiformat
s notesVel
midiout
noteout
s notesPitchPanel

- BOOKMARKS -


[bookmarks]
100%
1
+ 1
set goto $1
goto 39
r B1
set goto $1
goto 38
38
2
+ 1
set goto $1
goto 67
r B2
set goto $1
goto 66
66
3
+ 1
set goto $1
goto 49
r B3
set goto $1
goto 48
48
s toMelody
s toColl
s toReset
r pilotDevice
midiin
midiparse
unpack 0 0
4
5
6
sel 0
sel 0
sel 0
s B1
s B2
s B3

- FOLLOW -


r toReset
ztclear
1
clear
ztclear
1
arriva un bang a
ogni corrispondenza
sel 0
counter 0 2
2
pitch delle
note da
generare
int
3
pitch della
melodia
pilota
bag
sel
cut $1 0
s lock
0
1
2
$1 1
del
dec
counter
s voices
pipe 10
pipe 10
pipe 10
iter
zl sect
bag
zl filter
thresh 5
sel 1
== 0
del
clear
r pilotDevice
notein
r voices
sel 3
int
r lock
route 0 1 2
velocity
pipe 10

RINGRAZIAMENTI

Il mio percorso di studi universitario si conclude dopo 6 anni accademici dal suo inizio. Volendo ripercorrere cronologicamente gli eventi che mi hanno portato fin qui, non posso che iniziare ringraziando i miei genitori, che mi hanno sostenuto e sopportato, sotto molti aspetti: economico, psicologico, motivazionale. Allo stesso modo ringrazio tutta la mia famiglia.

Il 2010 fu l'anno della maturità, conseguita in un Liceo che porterò sempre nel cuore, per la qualità dell'insegnamento e delle persone. Visto che, nei miei programmi attuali, con questa laurea si conclude il mio percorso di studi, ringrazio i miei compagni e professori del Liceo, perché è anche grazie a loro che sono qui.

Molti dei compagni di classe di allora sono tuttora miei grandissimi amici, e li ringrazio doppiamente per riempire di amore i miei giorni e le mie notti.

Ringrazio anche gli Sugarcandy Mountains, band di scapestrati nata ai tempi del liceo e tutt'ora attiva, per avermi dato l'opportunità di esprimere la mia creatività lavorando con altre persone.

Nel settembre successivo cominciai il corso di Informatica dell'Università degli Studi di Milano Bicocca. Avevo un bel gruppo di studio, grazie a cui ho capito molte cose di questo strano mondo (quello dell'informatica). Li ringrazio molto, soprattutto Fabio De Nigris, che per me fu come un fratellone. Dopo due anni di corso, capii che avevo bisogno di raddrizzare il tiro per comprendere nei miei studi anche la componente musicale, che per me è sempre stata una infinita fonte di motivazione.

Nel 2012 mi iscrissi a questo corso di Laura, non ancora consapevole se mi avrebbe dato quello che andavo cercando. Non legai in modo particolare con alcuno, dovendo seguire corsi del primo, del secondo e del terzo anno per recuperare il tempo perso. Tuttavia posso dire di aver conosciuto molte persone interessanti e di cuore.

Nel 2013 fu la volta di Saluzzo. In questa cittadina piemontese ai piedi del Monviso si nasconde una scuola magica. Interruppi gli studi universitari per beneficiare di una borsa di studio che non si sapeva fino a quando sarebbe stata disponibile. Passai il test di ammissione brillantemente e passai uno splendido anno immerso nel mondo della produzione audio, il mio mondo, circondato da persone molto simili a me. Ringrazio i miei compagni di corso e i professori. Ironia della sorte, dopo un anno questa borsa cessò di esistere.

Grazie a questa scuola ebbi l'opportunità di lavorare in uno studio di registrazione a Torino davvero unico. Ringrazio il titolare, Tiziano Lamberti, che per me è stato un grande maestro. Non mi insegnò tanto un lavoro, bensì mi insegnò un mestiere.

Nel 2014 mi iscrissi nuovamento a questo corso per concludere il terzo anno. Nonostante avessi molti esami arretrati, in un anno e tre mesi sono riuscito a superare 13 esami e la tesi. Ancora, non legai con nessuno in modo particolare, eccezion fatta per Eleonora Dolif, con la quale affrontai il progetto di Basi di Dati. Fu un ostacolo insormontabile, infatti ringrazio molto Eleonora per averlo reso, con la sua compagnia, molto più leggero.

Un ringraziamento speciale per Michele Botti, che mi ha aiutato nella traduzione dal francese della pubblicazione di Jean Haury.

Da ultimi, ma non per importanza, ringrazio i miei relatori. In questa collaborazione, seppur breve, mi sono sentito riconosciuto e supportato fin dall'inizio, con onestà e correttezza. Esprimo la mia più sincera gratitudine.