Chapter 5

Design and implementation of Melodizer

This chapter will cover the details of the design and implementation of the melodizer object in OpenMusic. It will discuss the representation of musical elements that Melodizer will work with, and will give a quick reminder on voice objects and how they can be edited. It will then talk about the design of the melodizer object, and discuss the editor window in detail. Finally, it will detail the implementation.

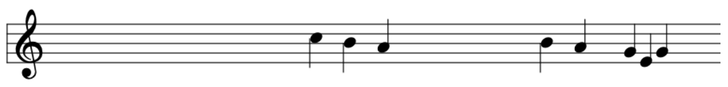
5.1 Representation of the musical elements

Two representations were considered for the musical elements: using Chord-Seq objects or Voice/Poly objects.

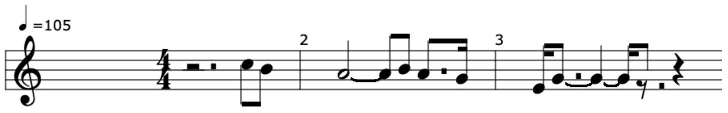
- Chord-Seq objects have a time-based approach to representing duration and rhythm. This makes it easy to access the starting time and duration of musical events, which is more practical for writing constraints on rhythm. However the visual representation is not easy to read. Notes are represented as quarter notes on a staff, and their duration is expressed as the distance between the note and the next note.

- Voice and Poly objects have a proportion-based approach, expressing duration as a fraction of a whole note and rhythm trees to represent rhythm. The visual representation is like the classical way of writing music, on a staff and with the shape of the note determining its duration.

Since the visual representation of Voice/Poly objects is easier for the composer to use, these objects will be used to represent musical elements in Melodizer even though it makes writing constraints involving rhythm more tedious to write. Figure 5.1 shows a simple melody represented using a Chord-Seq object (figure 5.1a) and a Voice object (figure 5.1b).



(a) Simple melody represented with a Chord-Seq object



(b) Simple melody represented with a Voice object

Figure 5.1: Comparison of the melody from "West Coast" by Imagine Dragons [7] represented with a Chord-Seq object and with a Voice object

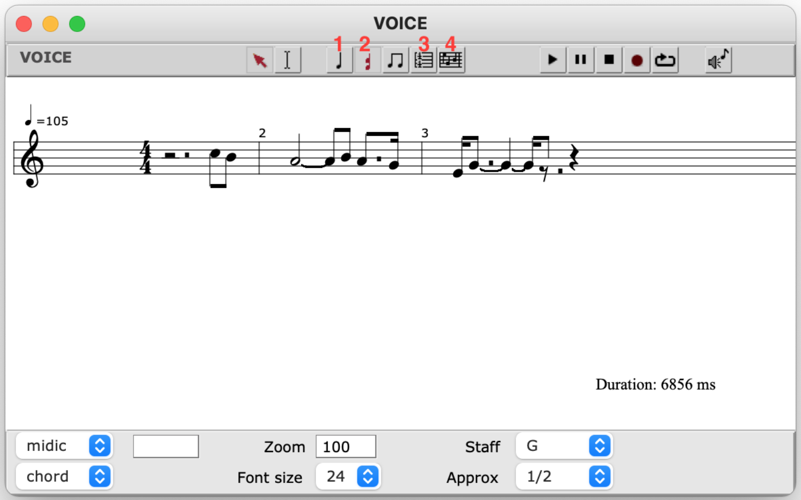


Figure 5.2: A voice object editor

5.2 Reminder on Voice objects

Since Voice objects are an important part of Melodizer, in particular the editor interface of these objects, a reminder is welcome in order to fully understand the implementation of the tool.
Voice objects represent a musical voice on a staff, with a tempo, a time signature, a rhythm and a pitch. The main attributes are tree, the rhythm-tree of of the voice, chords, the list of pitches for the notes, and tempo, the tempo of the voice. Each of these attributes can be set by connecting a data box to the appropriate inlet of the voice object, or they can be edited from the editor window of the object by adding notes, changing their duration,... The editor window of a Voice object is depicted in figure 5.2. Modifications to the voice can be done using the editor, the main ones are listed below :

- The pitch of a note can be adjusted by selecting it using the note tool (1) or the chord tool (2) and dragging it up/down or using the arrow keys of the keyboard.
- Measures : Measures can be added by clicking on the measure tool (3) and then clicking while pressing the “Cmd” key down where the new measure needs to be, or deleted by selecting it and pressing the “backspace” key.
- A rest can be transformed into a note and vice versa by selecting it using the chord tool and pressing the “esc” key.
- An event can be split into a group of smaller events of equal value by selecting it using the chord tool and entering a number representing the number of smaller events that need to replace this event.
- Notes can be added to a chord by selecting it using the note tool or the chord tool and pressing the “cmd” key.
- A voice or measure can be transposed by selecting it using the voice tool (4) or the measure tool respectively, and using the arrow keys.

Poly objects have a similar editor interface, the main difference being that there can be multiple voices. All the manipulations stated above are also valid for these objects.

5.3 Melodizer object design

The end tool is an OpenMusic object with an interface allowing to dynamically change input and parameters to model a variety of musical problems. Its attributes belong to one of three categories. Each of them will be discussed separately. The code for the tool can be found in appendix C. There is one inlet and one outlet for each attribute of the melodizer object, and their name appears when trying to connect something to them. Inlets allow to give input to the object, while outlets allow to take output from the object.

5.3.1 Input attributes

The first category of attributes is the input attributes. They can be set by the user, either through the editor window or by connecting data boxes to the inlets of the object, and allow to specify the problem precisely.

- **input-chords** is the voice object containing the chord sequence containing the chords that the melody will harmonize with.
- **input-rhythm** is the voice object containing the rhythm of the melody that the user wishes to find, or the voice object containing the melody that the user wants to vary.
- **key** is the key of the tonality that the user has chosen for the melody. It has a default value of C.
- **mode** is the mode of the tonality that the user has chosen for the melody. It has a default value of "major".
- **tool-mode** is the mode the tool is currently in. It has a default value of "melody-finder".
- **variety** is the solution variety desired by the user, expressed in musical events. It has a default value of 0.

5.3.2 Inner workings attributes

The second category of attributes is the attributes that allow to transfer data to different parts of the melodizer objects (e.g. from the interface to the CSP). They are not accessible to the user and are there for functionality.

- **optional-constraints** is a list of strings representing optional constraints that have been selected by the user and that should be added to the CSP for the search. It has a default value of an empty list.
- **global-interval** is the interval that the melody should cross if a pitch orientation constraint is selected. It is expressed in absolute value and has a default value of 1.
- **result** is a temporary holder for the return value of the CSP. It is a list containing the search-engine, the decision variable arrays, and the options objects.
- **stop-search** is a boolean to know if the user wishes to stop the search or not. It has a default value of nil and a value of t means that the user wants to stop the search.

5.3.3 Output attributes

The last category of attributes is the output attributes. They are the result of the search for a melody, and allow to use the results outside of the Melodizer tool in OpenMusic, for example to integrate in a musical piece. They can be accessed through the corresponding outlet of the melodizer object.

- **solution** holds the last proposed solution of the CSP.
- **solutions-list** is the list of all the solutions saved by the user for the current search.
- **motifs-list** holds the list of motifs saved by the user.
- **phrases-list** holds the list of phrases saved by the user.

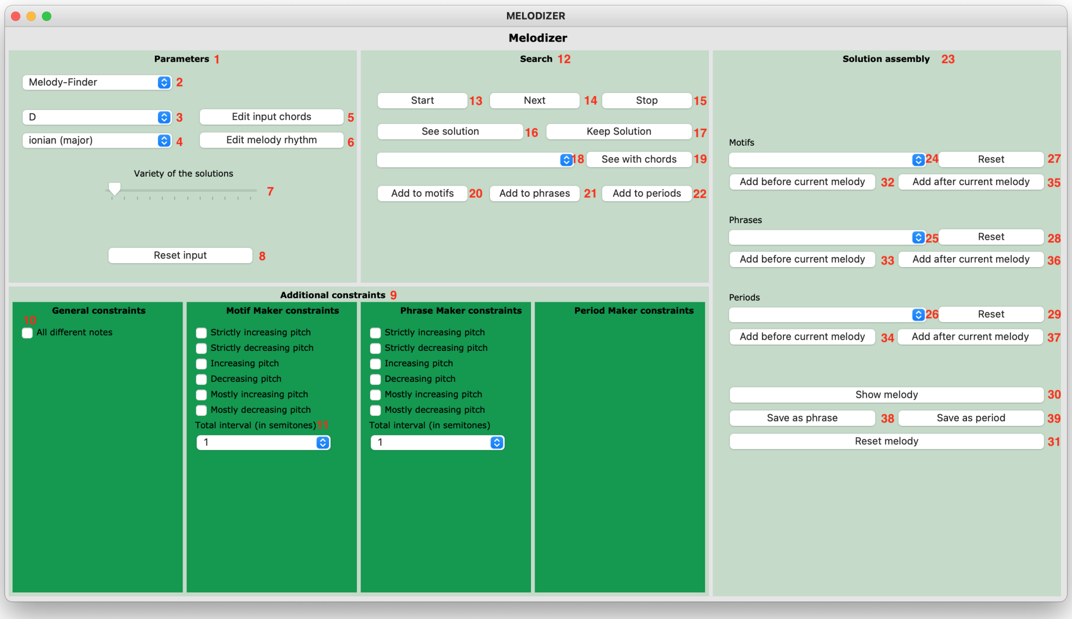


Figure 5.3: Editor window of the melodizer object

- **periods-list** holds the list of periods saved by the user.
- **output-solution** holds the selected solution for the current search in the pop-up menu.
- **output-motif** holds the selected motif for the current search.
- **output-phrase** holds the selected phrase for the current search.
- **output-period** holds the selected period for the current search.
- **melody** holds the melody constructed by the user using the solution-assembly panel.

5.4 Melodizer object editor

The editor window of the melodizer object is the most important part of the tool because it is the interface between the user and the CSP. It allows the user to interact with the problem by specifying optional constraints and with solutions to make them fit their needs. It is composed of four main components, that will be described individually. Figure 5.3 shows the interface of the melodizer object.

5.4.1 Input panel

The first panel that matters when using melodizer to generate melodies is the input panel (1) that allows the user to set the different parameters of the problem. This section covers the different elements that can be found in that panel.

Tool mode selection The first element we see is the tool mode selection pop-up menu (2). It allows the user to set the mode of the tool according to the use they want to make out of it. Currently, there are two modes : melody-finder, that allows the user to generate melodies based on input chords and a rhythm, or the variation-maker mode that allows to generate alternative melodies based on a given melody. Selecting a mode will modify the tool-mode attribute of the object to match the selection.

Tonality selection The two elements below that (3 and 4) allow the user to specify a tonality for the melody that they are looking for. Selecting a value will update the corresponding field of the melodizer object (key/mode).

Input edition The two elements to the right of the tonality selection (5 and 6) open the editor window for the input voice objects for the rhythm/melody and the chord sequence. Any modification is automatically saved since the object itself is edited.

Solution variety selection The slider in the middle of the panel (7) allows the user to set a proportion of musical events that must be different in every solution. A value of 0 means that two solutions can be identical. Selecting a value updates the variety field of the melodizer object. Currently, this feature is not fully supported, but the slider is there for future use.

Reset button The button at the bottom of the panel (8) allows to reset the voice objects for the rhythm/melody and the chords to start with a new blank voice object. Pressing the button will erase the input-chords and input-rhythm fields and replace their value with a new voice object.

5.4.2 Additional constraints panel

The panel at the bottom of the editor (9) is the additional constraints panel. It allows the user to specify additional constraints that they want to add to the problem. Depending on what they are looking for, there is a set of constraints available. Currently, there are no additional constraints for the variation-maker mode. Checking a box (10) adds the name of the corresponding constraint to the optional-constraints field of the melodizer object, and unchecking it removes it from the list. The mostly increasing/decreasing constraints (11) require to provide additional data, by selecting a minimum interval that the melody should cover. Selecting a value will update the global-interval field of the melodizer object. For now, constraints are not mutually exclusive meaning that it is possible to select different pitch orientation constraints even though they are not compatible (e.g strictly increasing and decreasing). Selecting several of them is likely to result in a problem with no solutions.

5.4.3 Search panel

The panel in the center of the top half of the editor window (12) is the search panel. It allows the user to control the search for a solution of the CSP with the parameters specified.

Start button The button on the left most side of the panel (13) is the start button. It allows the user to create the CSP by calling the melody-finder function or the variator function (see section 5.5.3), depending on the current mode of the tool. After pressing the button, any changes to the input panel or the additional constraints panel are not taken into account until the search is restarted by pressing the start button. Pressing the start button resets the solution and solutions-list attributes of the object as well as the stop-search field and updates the solution-display menu. It then stores the result of the function call in the result field of the object.

Next button The button to the right of the start button (14) allows the user to search for a solution. If the CSP has not been created, i.e the result field is nil, it raises an exception. Otherwise, it resets the stop-search field and creates a separate thread to call the search-next function (see section 5.5.3 as well). It then sets the solution field to the return value of the search-next function, and opens the editor window of the solution field to show the solution to the user. The user can modify the solution in the editor and any changes are saved.

Stop button The button to the right of the next button (15) allows the user to stop the search if it is taking too long. When it is pressed, the stop-search field of the melodizer object is set to t and the next time the search will be stopped by the options (max. 500ms after the pressing of the button), it will stop and raise an exception saying that the search has been stopped.

See solution button The button below the start button (16) allows the user to reopen the solution field editor window. It can still be modified and any changes are automatically saved. If there is no current solution, an exception is raised.

Keep solution button The button to the right of the see solution button (17) allows the user to save the current solution to the list of solutions for the current CSP. If there is no current solution, an exception is raised. Otherwise, the current solution is added to the solutions-list field of the melodizer object and the corresponding pop-up menu (18) is updated.

Solution list display The element under the see solution button (18) allows the user to see the list of solutions they saved. Selecting a solution will set the output-solution field of the object to the selected voice object, and open the editor window of the selected element, not the output-solution field. Any changes are saved.

See with chords button The button to the right of the solution list display (19) allows the user to see the solution together with the input chords. If there is no selected solution, i.e the output-solution field is nil, it raises an error. Otherwise, it creates a Poly object with the output-solution and the input-chords fields and opens its editor window. Any changes made at this stage are not saved to the output-solution window as it is a temporary object.

Add to motifs button The button on the bottom right of the panel (20) allows the user to save the selected solution (output-solution field) to the list of motifs (see Solution assembly panel). As always, if the output-solution field is nil, it raises an error. Otherwise, it creates a poly object with the output-solution and input-chords fields, and adds it to the motifs-list field. It then opens its editor window. Any changes are saved since the poly object here is not a temporary object.

Add to phrases/periods buttons These buttons (21 and 22) have the same behaviour as the add to motifs button, except that they add the solution to the phrases-list/periods-list fields respectively.

5.4.4 Solution assembly panel

The panel on the right of the window (23) allows the user to combine different motifs, phrases and periods that they have generated to write more complex pieces of music.

Motifs/Phrases/Periods list display The pop-up menus (24, 25 and 26) display the list of motifs/phrases/periods that have been saved by the user. Selecting one of them will set the output-motif/output-phrase/output-period field to the selected poly object, and open its editor window. Any changes are saved automatically.

Reset buttons The buttons to the right of the pop-up menus (27, 28 and 29) allow the user to erase the list of saved motifs/phrases/periods. Clicking on it will reset the output-motif/output-phrases/output-periods and motifs-list/phrases-list/periods-list fields of the object, and update the corresponding pop-up menu.

Show melody button The button below the list of motifs, phrases and periods (30) allows the user to see the melody that they are currently creating by assembling the different elements they created. If there is no melody, it throws an error. Otherwise, it opens the editor window of the melody field of the melodizer object. It can be edited like every other Poly object, and changes are automatically saved.

Reset melody button The button on the bottom of the panel (31) allows to erase the melody that the user was creating, e.g. when it has been saved to start a new one. When it is pressed, it sets the melody field to nil.

Add before/after current melody buttons The buttons below the pop-up menus showing the list of elements saved by the user (32 to 37) allow to add the selected element before/after the current melody, respectively. When they are pressed, if there is no melody (the melody field is nil) it throws an error. Otherwise, it calls the concat function from OpenMusic to append the melody and the element together.

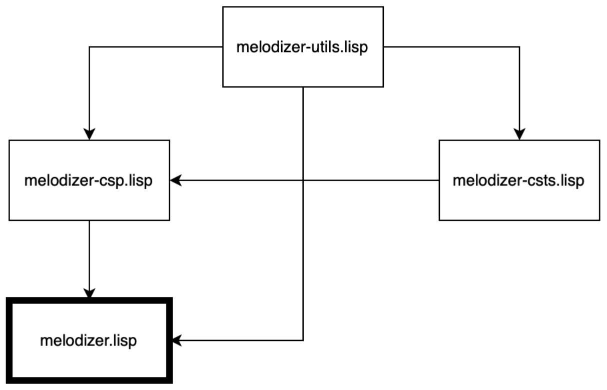


Figure 5.4: Relationship between the different files

Save as phrase/period buttons The last two elements that need to be discussed are the buttons underneath the show melody button (38 and 39). They allow to save the melody constructed by the user as a phrase or period respectively. When they are pressed, they check wether the melody is nil or not. If it is, it throws an exception, otherwise it adds the melody field to the phrases-list or the periods-list field respectively.

5.5 Melodizer object implementation

The code for Melodizer is spread into 4 different files to keep the main features separated. Figure 5.4 shows how these 4 files are interconnected. An arrow from 1 file to another means that functions from the first file are called from the code in the other file. The "melodizer.lisp" file contains the code for the melodizer object as well as the editor, the "melodizer-csp.lisp" file contains the code for the CSPs described in previous chapters, the file "melodizer-csts.lisp" contains the constraints used for the CSPs, and the "melodizer-utils.lisp" file contains generic utility functions.

5.5.1 Melodizer-utils

This file contains a variety of functions that have very different purposes. Here is a list of the main functions and their purpose:

- Conversion functions (from MIDI to MIDIcent and vice versa, from the name of a note to its value and vice-versa,...)
- A function to update a pop-up menu from the interface
- A function to get the starting time of all notes from a Voice object
- A function to get the series of tones and semitones defining a specific mode
- A function to get the notes that can be played on top of a given chord
- A function to get the mode and inversion of a chord based on its notes.

5.5.2 Melodizer-csts

This file contains all the constraint functions used in the CSPs described in chapter 3. Details will not be repeated here, but here is a list of the available constraints:

- All the notes of the melody are different
- Strictly increasing/decreasing pitch
- Increasing/decreasing pitch
- Mostly increasing/decreasing pitch

- At least n notes of the melody take the value they have in the input
- Notes are always higher than the lowest note from the chords and lower than the highest note of the chord + 2 octaves
- The intervals between consecutive notes follow a specific rule
- Notes are in the tonality specified
- The melodic interval in the context of a given chord is a maximum of an octave
- Notes played on top of a chord follow a set of rules

5.5.3 Melodizer-csp

This file contains the code for the two CSPs used in Melodizer, as well as the function that is searching for the solutions. The searching algorithm is the same for both CSPs, so there is only one searching function. The algorithm for the CSPs is explained in chapter 3 and the algorithm for the search is detailed in chapter 4.

5.5.4 Melodizer

This file contains the code for the object itself, as well as the code for the editor window and all of its components. The main components are:

- The declaration of the melodizer class and all the attributes of the object
- The creation of the main view
- The creation of the input panel
- The creation of the search panel
- The creation of the optional constraints panel
- The creation of the solution assembly panel

The code for this file is too long to show here, but can be found in appendix C.