Somax2 User’s Guide

version 2.5

March 28, 2023

Contents

1 Somax2 for Max **3**
- 1.1 Introducing Somax2 . . . 3
- 1.2 Getting Ready . . . 3
  - 1.2.1 Requirements . . . 3
  - 1.2.2 Installation . . . 4
  - 1.2.3 What's in the package . . . 4

2 Somax2 Concepts **7**
- 2.1 The Corpus and Navigation Model . . . 7
- 2.2 Interacting with Somax2 . . . 9
  - 2.2.1 Slices . . . 9
  - 2.2.2 Influences . . . 9
  - 2.2.3 Peaks . . . 9
  - 2.2.4 Putting all together . . . 10
- 2.3 Somax2 User Interface . . . 12

3 Somax2 Objects **13**
- 3.1 Somax2 Architecture . . . 13
- 3.2 Server . . . 13
  - 3.2.1 Server App . . . 14
- 3.3 Player . . . 15
  - 3.3.1 Player App . . . 16
- 3.4 Influencers . . . 18
  - 3.4.1 Audio Influencer App . . . 19
  - 3.4.2 MIDI Influencer App . . . 20
  - 3.4.3 Audio Corpus Builder . . . 22
  - 3.4.4 MIDI Corpus Builder . . . 24
- 3.5 Recap . . . 25

4 Mastering Somax2 Interaction **27**
- 4.1 Basic Workflow . . . 27
  - 4.1.1 Interaction Parameters . . . 27

5 Tutorials and Further Explorations **32**
- 5.1 Somax2 Overview patch . . . 32
- 5.2 App Tutorials . . . 33
- 5.3 Max Tutorials . . . 35
- 5.4 Performance Strategies . . . 37
- 5.5 Templates . . . 39
- 5.6 Help Center . . . 39

5.7 Credits 40
5.8 More and Contacting the Team 40

1. Somax2 for Max

1.1 Introducing Somax2

Somax2 is an application and a library for live co-creative interaction with musicians in improvisation, composition or installation scenarios. It is based on a machine listening, reactive engine and generative model that provide stylistically coherent improvisation while continuously adapting to the external audio or MIDI musical context. It uses a cognitive memory model based on music corpuses it analyses and learns as stylistic bases, using a process similar to concatenative synthesis to render the result, and it relies on a globally learned harmonic and textural knowledge representation space using Machine Learning techniques.

Somax2 has been totally rewritten from Somax, one of the multiple descendant of the well known Omax developed in the Music Representation team over the years and offers now a powerful and reliable environment for co-improvisation, composition, installations etc. Written in Max and Python, it features a modular multi-threaded implementation, multiple wireless interacting players (AI agents), new UI design with tutorials and documentation, as well as a number of new interaction flavors and parameters.

In the new 2.5 version, in addition to a self-contained app patch, it is also now designed as a Max library, allowing the user to program custom Somax2 patches for everybody to design one's own environment and processing, involving as many sources, players, influencers, renderers as needed (these objects will be explained further). With these abstractions, implemented to provide complete Max-style programming and workflow, the user could achieve the same results as the Somax2 application but, thanks to their modular architecture, it is also possible to build custom patches and unlock unseen behaviours of interaction and control.

This guide will take you through the general philosophy and process of Somax2, then present the Somax2 core objects. In order to quickly learn the basic use of Somax2, please go through the 'somax2.overview.maxpat' that you'll find in the Somax2 folder.

Get more info, help and resources (demos, medias, discussions etc.) at the Somax2 research project page: `http://repmus.ircam.fr/somax2` .

1.2 Getting Ready

1.2.1 Requirements

- macOS 10.13 or later;
- Max 8;
- (Python 3.9 or later – only needed for manual installation).

Note: It's currently not possible to run Somax natively on M1 machines: if you're using a M1 Mac, make sure to run Max under Rosetta (see `https://docs.cycling74.com/max8/vignettes/apple_arm64`)

1.2.2 Installation

Easy Installation

This is the procedure recommended for most users, unless you explicitly want to modify the python code.

- Go to Releases at `https://github.com/DYCI2/Somax2/releases` and download the latest version of Somax2 (Somax-2.5.x.dmg);
- Copy the extracted /Somax-2.5.x folder into the /Packages folder in your Max folder (by default, this is /Documents/Max 8/Packages).

Manual Installation

If you want to modify the python code, you will need a manual installation. This assumes you already have python 3.9+ installed.

Step 1: Install Somax

- Clone the master branch of the repository at `https://github.com/DYCI2/Somax2` or go to Releases at `https://github.com/DYCI2/Somax2/releases` and download the latest version of the Somax source code;
- Add the max/somax subfolder of /Somax-2.5.x to your Max path through Options -> File Preferences in Max. Make sure that the 'subfolders' option is checked.

Step 2: Install Python Requirements

- In a terminal, cd to the /Somax-2.5.x root folder and install the requirements with `'pip3 install -r python/somax/requirements.txt'`.

1.2.3 What's in the package

The Somax2 distribution contains many files and folders, among them some important ones are:

- /corpus: folder with all the distributed Somax2 set of corpus;
- /docs: folder for tutorial patches and ready-to-play performance strategies;
- /externals: folder for the externals used inside Somax2;
- /extras: here you will find the 'somax2.overview.maxpat', the patch that will help you start exploring everything;
- /help: folder for the maxhelp files related to every Somax2 abstraction;
- /patchers: the actual Max patches that constitute Somax2;
- somax2.maxpat: application patch for Somax2, with one player;
- somax2.overview.maxpat: the overview patch that will guide you anywhere you need;
- /templates: folder for the template patches from one to four players.

To start diving into the world of Somax2, we encourage starting by taking a look at the following sections, and using the 'somax2.overview.maxpat' as the main guide to redirect you wherever you need.

Somax2 Overview

Located in the Somax2 root folder, or in the /extras folder, the 'somax2.overview.maxpat' is the starting point to begin exploring the world of Somax2. As shown in Figure 1.1, from here you will be able to access all the different tutorials, as well as to get access to ready-to-play patches defining specific performance strategies. Templates from one to four players are also available, as well as maxhelps for all the Somax2 objects.

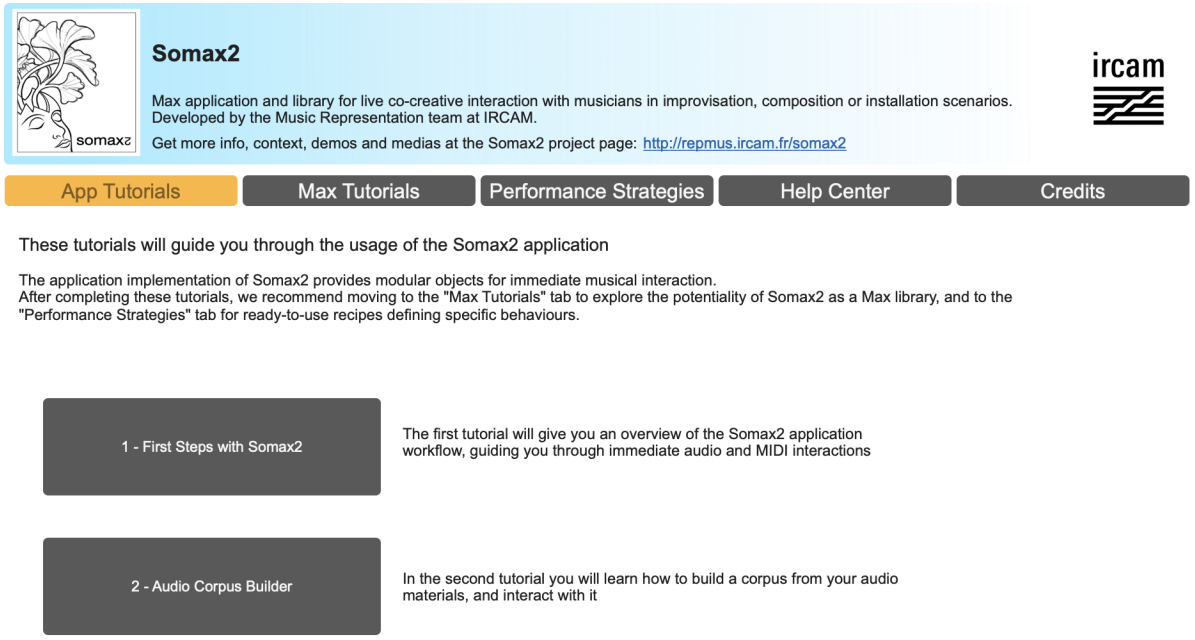


Figure 1.1: The 'somax2.overview.maxpat' is the starting point for the Somax2.5 package.

App Tutorials

Accessible from the 'somax2.overview.maxpat', but also available in the /docs/tutorial-patchers folder, the two patches 'app1 - First Steps with Somax2.maxpat' and 'app2 - Audio Corpus Builder.maxpat' will guide you through the basics of the Somax2 application.

Max Tutorials

Seven patches will guide you through the usage of the Somax2 Max library. These tutorials will start by reviewing the very basic Max workflow of the Somax model, and incrementally move to complex ways of interaction, thanks to the modular architecture of the Somax2 abstractions. Find these from the 'somax2.overview.maxpat' patch or in the /docs/tutorial-patchers folder.

Performance Strategies

Three ready-to-play patches will showcase specific behaviours of interaction of Somax2, from mimetism and harmonization strategies to installative modes. Available from the 'somax2.overview.maxpat' or in the /docs/tutorial-patchers folder.

Templates

In the /templates folder you will find four Somax2 application patches, already configured with a range of one to four Players. Use these templates as the starting point of your Somax application patches.

Help Center

From the Help Center tab of the 'somax2.overview.maxpat' or in the /help folder you will get access to specific and detailed maxhelp files for the Somax2 abstractions. Use these when you have any doubts.

2. Somax2 Concepts

Somax is an interactive system which improvises around a given musical material, aiming to produce a stylistically coherent improvisation while jointly listening to and adapting to live musical input . The system is trained on musical material selected by the user, from which it constructs a 'corpus' that will serve as a basis for the improvisation. Somax can use either audio or MIDI files as its musical material (or a combination of the two), and it is able to listen and adapt to both audio and MIDI input from the external world.

Somax may serve as a co-creative agent in the improvisational process, where the system after some initial tuning is able to continuously listen and adapt to the musician in a self-sufficient manner. Of course, the input doesn't have to come from a live musician; any type of audio and/or MIDI input works, be it from an audio file, score editor, synthesizer, DAW, or another Somax player. Somax also allows detailed parametric controls of its output and can even be controlled as an instrument in its own right. Also, the system isn't necessarily limited to a single agent or a single input source - it is possible to create an entire ensemble of agents where the user can control how the agents listen to various input sources as well as to each other.

The goal of this section is to provide a brief introduction to Somax and provide the reader with the fundamental knowledge about how its interaction model works, which in turn should serve as a basis for making informed choices when tuning and interacting with the system. For a hands-on introduction to Somax and its user interface, also see the interactive tutorials provided with the Somax distribution.

2.1 The Corpus and Navigation Model

As previously mentioned, the Somax system generates its improvisation material based on an external set of musical material, the «corpus». This corpus can be constructed from one or multiple MIDI files, or a single audio file, freely chosen by the user. In contrast to many other generative approaches, the system does not construct a model that eventually is independent of the material that was used to train it. Rather, the model is constructed directly on top of the original data and provides a way to navigate through it in a non-linear manner. One way of seeing this is to consider that some fine-grained aspects of the musical stream are somehow too complex to be modeled, but will be preserved – to a certain extent – when weaving into this musical material.

In order for Somax to build a corpus from given music material, the first step is to segment the musical stream into discrete units or «slices», which are vertical (polyphonic) segments of the original file, where the duration of a slice is the distance between two note onsets.[1] Each slice is analyzed and classified with regards to a number of musical features related to its harmony/texture, individual pitches, dynamics, etc., and these features along with the pattern structure they infer over the musical sequence will serve as the main basis for controlling the navigation model. Thus a navigation model is made of a musical memory

[1] In non-quantized MIDI files it is rare for any two notes that are perceived as simultaneous to be exactly simultaneous. Since one goal of Somax is to be able to maintain and reproduce the original timings within slices as recorded, notes with almost simultaneous onsets will still be grouped together in a single slice but with their internal timing offset preserved.

(basically the corpus), plus a dynamic pattern matching and selection scheme to navigate into memory and reconstruct a generative signal. Slicing is illustrated in figures 2.1 in the case of MIDI, where the polyphony is broken down in vertical columns with ties between notes, prior to analysis and classification of the content. This representation allows to weave back the polyphonic structure at generation time.

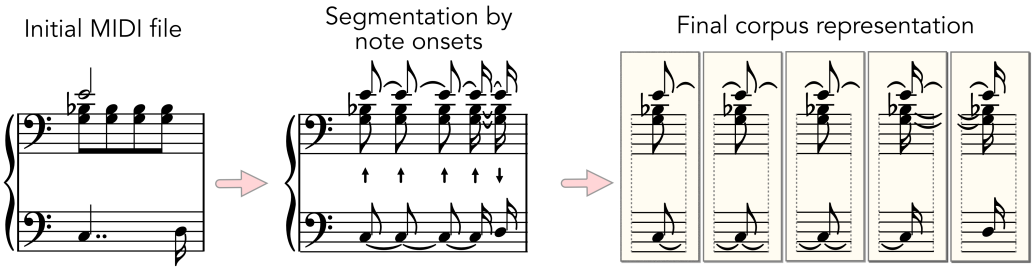


Figure 2.1: Constructing a corpus by segmenting MIDI data into «slices».

The procedure of mapping memory with patterns is actually repeated for each music feature (harmony, melody etc.) in the analysis, effectively resulting in a multilayer representation where each layer roughly corresponds to one feature, i.e., one layer for harmony, one for pitch, etc.

When a musician interacts with the system, a similar process of segmentation and multilayer analysis and classification is computed in real-time on the input stream, and at each point in time the result of this process is matched to the information in the navigation model, generating activations, or «peaks», at certain points in the sequential memory where the input corresponds to the model. Each peak corresponds to a point in the memory, so the entire set of peaks can effectively be seen linearly as a one-dimensional curve over the corpus' time axis (see Figures 2.3 and 2.4 below for examples). The peaks in each layer are merged and scaled according to how the system has been tuned, and finally the output slice is selected from the sequential memory based on the distribution of the peaks, typically at a location that is a good match with the overall context at this particular moment.

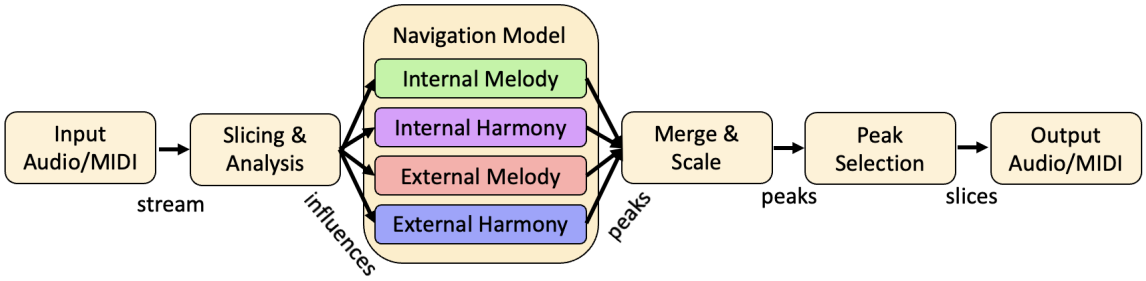


Figure 2.2: An overview of the steps through which the system generates its output at each given point in time.

The generated output of this process is a co-improvisation that recombines existing material in a way that is coherent with the sequential logic and statistical properties of the original material while at the same time adapting in real-time to match the input from the musician. This is because the multilayer peak profiles is shaped not only by the input, but

also by the output of the navigation itself, which is called "self-listening". Self-listening conditions its own set of layers, that combine in turn with external listening ones. This process of attempting to balance the internal logic of the corpus with the external logic of the musician often provides a mix of coherency and unexpectedness in a way that convincingly gives the impression of an active agent in the improvisational process.

2.2 Interacting with Somax2

When interacting with Somax2, there are three main concepts that are important to understand: «slices», «influences» and «peaks».

2.2.1 Slices

A slice, as previously mentioned, is a short segment of the corpus and serves as the smallest building block of the output of the system. The slice can be manipulated to some extent (transposed, filtered with regards to voices/channels, stretched etc.) but will always maintain most fundamental properties of the original corpus.

2.2.2 Influences

When Somax listens to a musician, this musical stream is segmented and analyzed with respect to its musical parameters similarly to how the corpus was constructed, but with a slightly different set of methods to be able to operate in real-time. The result of this process are discrete chunks of multilayer data or «influences», which the system uses to be able to compare the input to the memory, where the main purpose of the influence is to act as the guide that determines the output of the system. The concept of an influence may initially seem like an implementation detail, but will become increasingly important for more complex configurations with multiple agents and/or multiple input sources. The main takeaway is that the system cannot listen directly to a musical input stream, but will need to translate it into influences, and that the process of tuning the listener can be a very important factor for the quality of the co-improvisation. Thus influences are the main data that circulate between the Somax agents, hence the names ‘audioinfluencer’ and ‘midiinfluencer’ for the input modules. Typically, a Somax Player "listens" to a certain amount of cumulated influences (audio inputs, MIDI inputs, other agents’ productions) and takes decisions based on these influences.

2.2.3 Peaks

Finally, a «peak» is, again, a point in the corpus where the input corresponds to the model, or simply a match between an incoming influence and a corresponding slice that would serve as an output candidate. Each peak has a height, corresponding to a probability (or viability) of that particular slice as an output candidate. Unlike influences (which are visible in the interface) and slices (which are correlated to the audible output of the system), peaks are never interacted with directly, they’re only part of the internal state of the system, but perhaps the most vital part. Each peak effectively corresponds to a slice in the corpus that could serve as an output at the current point in time, given the latest influence. Having at each point in time a reasonable number of peaks is thus vital for the quality of the output, since having no peaks means that the output has not taken the musician’s influences into

account, and on the opposite side, in most cases a large number of peaks indicate that the matching is imprecise.

2.2.4 Putting all together

To put the concept of peaks in context, let's dive a bit deeper in how the system works. While the musician is playing, Somax is at each detected onset segmenting / analysing the input into influences, carrying information about the pitch, harmony, etc. of what the musician currently is playing. This process is carried out by agents of the system called «influencers». This information is routed to a «player», which handles the entire process of matching and generating output. The influence is routed in multiple layers by the player, as briefly mentioned, where each layer corresponds to one musical dimension (e.g. harmony, melody) of the influence. In each layer, a model of the corpus with respect to the particular layer's musical dimension exists, and upon receiving an influence, the model will look for sequences in the corpus that match the sequence of most recent influences from the input, and, in each of those places, insert new peaks.

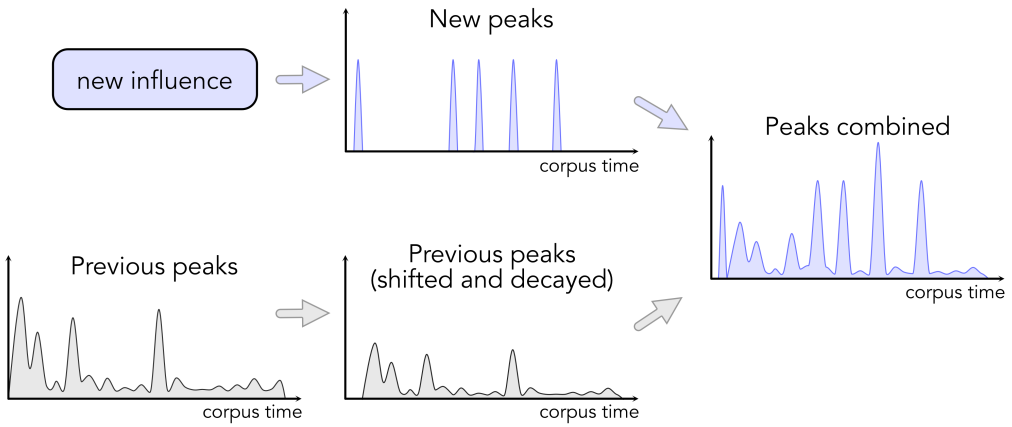


Figure 2.3: The process of shifting and decaying previous peaks in a single layer upon receiving new influences (the process of matching the incoming influence to the corpus has been omitted for clarity).

The system is also simulating a type of short-term memory inside this model by not immediately discarding peaks from previous influences, but rather shifting them along the time axis of the memory and decaying their height corresponding to the amount of time that has passed, followed by merging them with the new incoming set of peaks. This means that sequences continuously matching several consecutive influences will be more highly prioritized over others, as illustrated in Figure 2.3. Another powerful property of this system is that it allows some form of fuzzy pattern recognition since positions that are not perfect matches at time t, can still become good matches at time $t + i$, due to the propagation mechanism. Finally, the peaks from all layers will be merged together into a single set of peaks which the system will use to probabilistically determine which slice is the best output candidate[2]. The result of this multilayer peak merging process is an output that will not just strictly match the harmony and pitch of the influence but rather improvise around the most recent history of influences with regards to the corpus own structure,

[2]Actually, in addition to this, there are a number of parameters that scale the height of the peaks individually with regard to a number of other musical parameters of choice, but this is thoroughly documented in the help files and tutorials in Max and will not be discussed here

with both fidelity and agency. In addition, as already said, there are also two layers which listens to the output of the player itself, as feedback layers, that can be used to balance the player's consistency with the input with its continuity in its own performance. The balance between the different layers as well as control over the shift/decay time of old peaks, length of sequences to match in the memories, etc. are all available in the 'somax.player.app' and 'somax.player.ui' user interface, as shown later in Section 3.

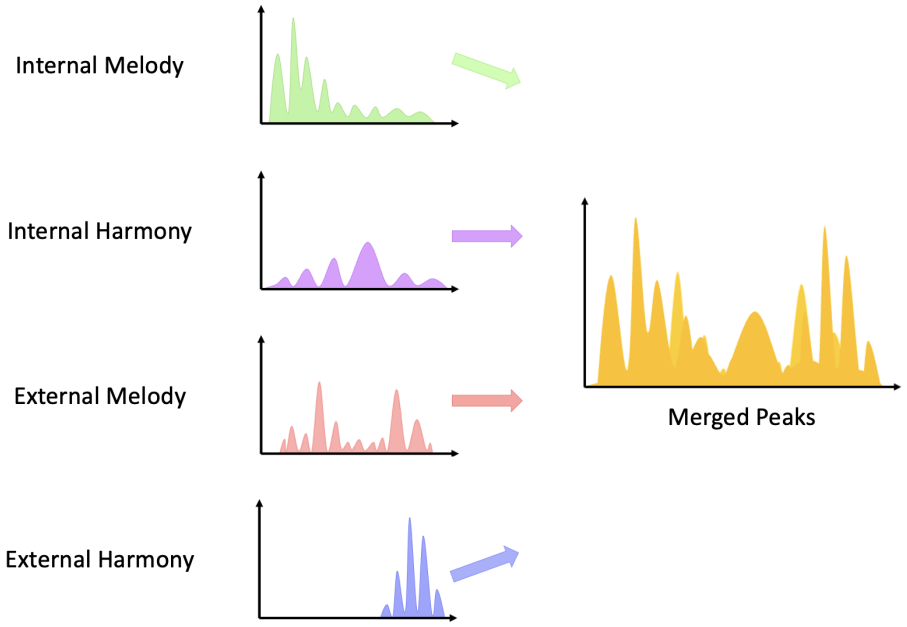


Figure 2.4: The four layers of peaks corresponding to different musical dimensions such as internal melody, internal harmony, external melody and external harmony, being merged into one set of peaks before the final scaling and peak selection. Here, all four layers are weighted equally, but it is possible to balance the contribution from each of the layers.

If the concept of peaks isn't perfectly clear to you after reading this – don't worry! Go to the tutorials and start experimenting with the system while keeping one thing in mind: if the number of peaks is continuously zero or continuously too high[3], this is likely an indicator that the system is working poorly and should be retuned. If not – you're probably doing quite well.

Another important aspect of the interaction with Somax is its relation to time. According to the user's preference, each player can be assigned to either operate continuously in time as an autonomous agent, maintaining the pulse and exact within-slice timings of the original corpus (while possibly adapting to the tempo and/or phase of the input), or operate reactively, generating output synchronously as requested by the input's events. In the continuous case, this means that the player improvises freely over time while still taking the influences of the musician into account, while in the reactive case, it synchronizes strictly (note-by-note) or loosely (depending on tuning) with the input. Of course, the player is in the latter mode not strictly limited to the input from where it receives its influences, but could be connected to a third source of some sort, for example any type of step-sequencer or other generative approaches, thus giving the user multiple options for controlling the

[3] exactly how large "too high" is varies with the type of layer and context, but larger than 10% of the total number of slices in the corpus with no transpositions active could serve as a reference of "too high" that is valid for most layers and contexts

temporal domain of the system.

2.3 Somax2 User Interface

Understanding the generation and interaction concepts presented above only from a theoretical standpoint might seem pretty difficult. But thanks to the user interface provided by the Somax2 application objects, you can have immediate hands-on and start playing with Somax2 right away. In Figure 3.16 you could see the basic objects of the Somax2 application. These are:

- the Server;
- the Player;
- the Audio Influencer;
- the MIDI Influencer.

This is of course just one possible configuration of the Somax2 objects, as a big advantage of this application is that each object communicates with each other in a wireless way, without the need of patch chords. In this way you can easily adapt the patches at your need, and build your own configuration without having to worry about messy patch cables.

In the next Section we are going to break down these application objects, giving an overview of all of them, as well as explaining their main parameters of interaction.

Figure 2.5: The 'somax2.maxpat' included in the distribution gives you quick access to all the objects in the Somax2 application: server, player, audio influencer and MIDI influencer.

3. Somax2 Objects

3.1 Somax2 Architecture

The new Somax2.5 has been redesigned both as a Max application and a Max library. Almost every object in the package has a ‘core’ version and an ‘.app’ version (i.e., ‘somax.player’ and ‘somax.player.app’).

As their names say, core objects are pure Max abstractions, intended for users that want to use Somax2 in a fully Max-like programming style.

On the other hand, .app objects are abstractions providing a user interface to control the parameters, as well as other utility tools to immediately use the abstractions. They are really wrappers around the core objects (that are still contained inside any .app object) that provide the users instant access to a nice interaction, while maintaining the modularity of the core objects. Only .app objects appear in the ‘somax2.maxpat’ application that most people will use, unless they are skilled Max programmers and wish to build their own custom application patch.

Some of the objects have also a ‘.ui’ version (i.e., ‘somax.player.ui’), which is a compact user interface version of the core object, and that could be used as an alternative to this one. Note that the .ui objects doesn’t have the same utilities as the .app objects, and therefore they cannot be used as ready-to-play objects, but rather they are intended as visual feedback alternatives to the core objects. Since .ui objects are included in .app objects they will also serve in the following to show and explain some of the .app user interface details.

This chapter is intended to give you a brief overview of the main Somax2 objects in both the core and .app versions. For a further exploration, see the ‘somax2.overview.maxpat’ patch.

3.2 Server

The Somax Server is an external Python application through which all ‘somax.player’ objects communicate. The ‘somax.server’ Max object manages all the communication with the remote server using the OSC protocol, so in order to use a ‘somax.player’, it’s always necessary to first initialize the ‘somax.server’ object. The server is also responsible for managing and synchronize time between the players.

So when a ‘somax.server’ object is created in a Max patch, the usual workflow for it consists on the following step:

- initialize the server: by default, the ‘somax.server’ will automatically be initailized when created. However, if this has been created with the attribute ‘@autoinitialize 0’ you need to send it the message ‘initialize’;
- initialize the players: to initialize all the local ‘somax.player’ objects, send to the server the message ‘initializeplayers’;
- activate the server’s transport: to start the generation for all the players, use a ‘toggle’ to start or stop the player’s transport.

See the 'somax.server.maxhelp' shown in Figure 3.1 for a detailed description of it.

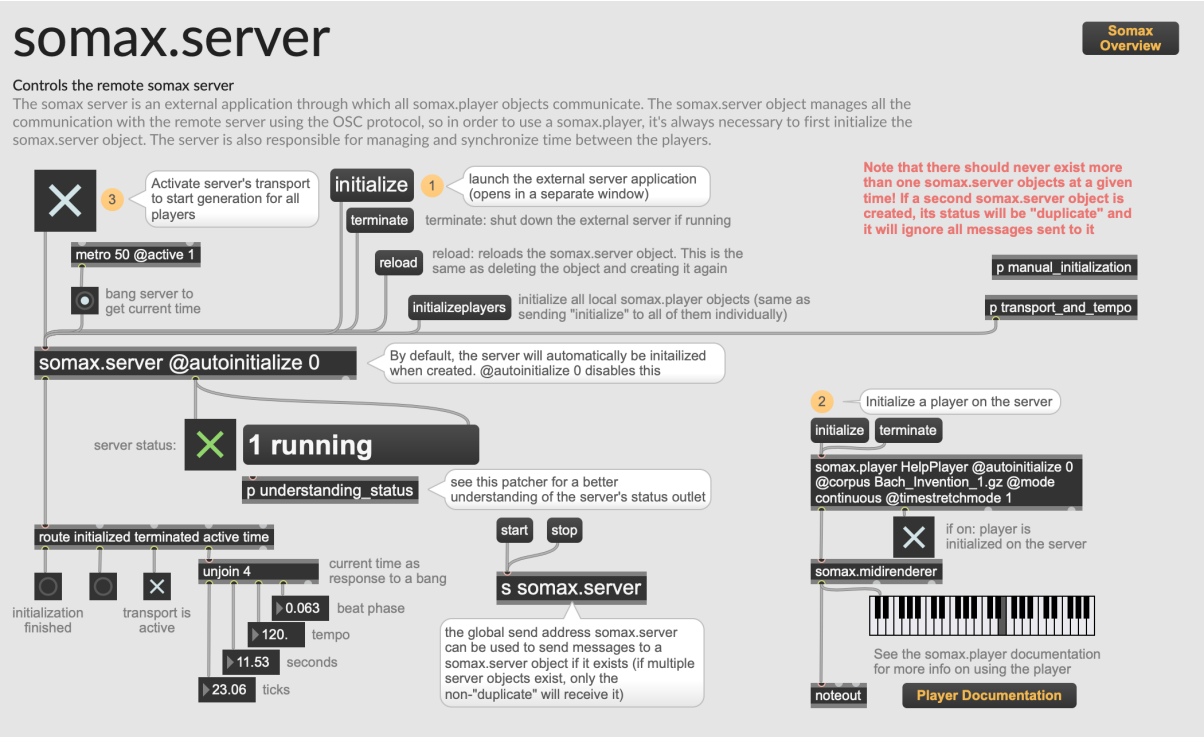


Figure 3.1: Help patch for 'somax.server.maxhelp' available in the /help folder and from the help center of the 'somax2.overview.maxpat'.

3.2.1 Server App

The 'somax.server.app' object is a wrapper that handles the communication with the remote Python server, where all the players are stored. The server also handles the scheduling of events and all aspects of timing and tempo, as well as the construction of new corpora through the Corpus Builders module. See the 'somax.server.app.maxhelp' for a detailed description of it.

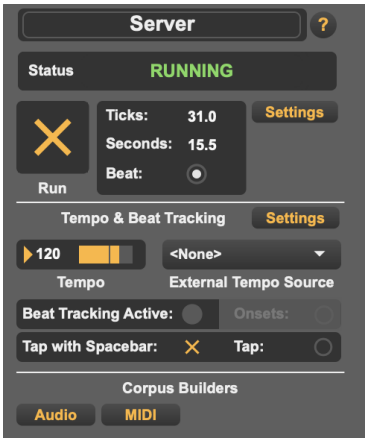


Figure 3.2: User interface of the 'somax.server.app' with controls for all the parameters, as well as the Corpus Builders modules.

3.3 Player

The 'somax.player' is the main generative object in the Somax package. The player reads a corpus (constructed through a corpus builder) and generates MIDI or audio content by playing segments (events) from the corpus in a non-linear manner. The main way of interacting with the player is by sending influence messages from a 'somax.midiinfluencer' and/or a 'somax.audioinfluencer'. These serve as the harmonic and melodic context which the player is trying to match.

The player exists in three different kind of objects, each one targeting a particular use case:

- somax.player: this is the core Max object. To use it, you have to patch an influencer to its inlet, and connect its outlet to an audio or MIDI renderer, as the player itself does not output audio or MIDI, but data;
- somax.player.ui: this acts exactly the same as the somax.player, but gives you a user interface to control its parameters;
- somax.player.app: this is the application module of the player. It has at its core the basic somax.player object, but in addition to the user interface (same as the somax.player.ui) it provides modules for selecting influences, feedback of the matches and audio or MIDI playback. This is a ready-to-play object, that you can use in any patch in a complete wireless way (the routing of every signal in the .app objects doesn't need any patch chords).

See the 'somax.player.maxhelp' shown in Figure 3.3 for a detailed description of it.

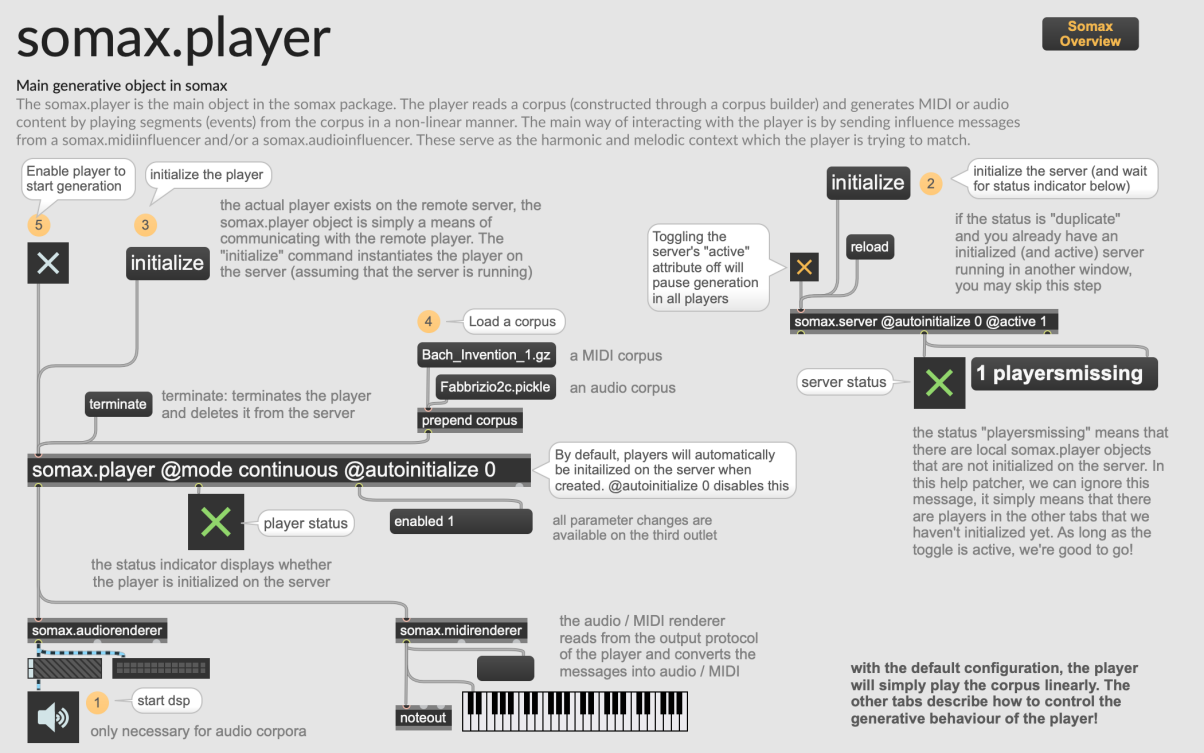


Figure 3.3: Help patch for 'somax.player.maxhelp' available in the /help folder and from the help center of the 'somax2.overview.maxpat'.

3.3.1 Player App

The 'somax.player.app' is a wrapper around the somax.player, encapsulating it together with other useful tools for routing influences, rendering and playing back audio or MIDI output. The somax.player.app has a graphic interface to control all its parameters and provide the user with better interaction. Thanks to this app object, Somax can send influences between influencers and players wirelessly (i.e., without Max patch cords). See the 'somax.player.app.maxhelp' for a detailed description of it.

Figure 3.4: The 'somax.player.app' object. Note that this provides a module for routing the influences, the controls on the player's parameters (which is what is shown in the 'somax.player.ui') and modules for outgoing influences and output audio or MIDI controls.

Note that the 'somax.player.app' is actually composed by multiple modules, as shown in Figure 3.4:

- Influence Sources: module used to select which influence the player should listen to (within the audio influencer, MIDI influencer or any other player located in the

patch). From here you can also select which kind of influence (onset, pitch, chroma) each influencer will send to the player;

- Player Controls: this corresponds to the 'somax.player.ui' object and it's the module where you can control the main parameters of the player for the intended interaction. Note that by pressing on the «Settings» button, a new window with the complete set of parameters pops up, as showed in Figure 3.5;
- Region Filter and Number of matches: this module gives a visual feedback on the last played state index of the corpus, as well as the number of matches on any layer. It's also possible to select a certain region of the corpus to be played;
- Outgoing influences: this module is actually an Influencer, as it will be described in the following paragraph. It's useful to monitor the data that the player is outputting, in terms of onset, pitch and chroma;
- Output Control: in the case of an audio player, as in Figure 3.4, you can select on which audio channels the player should playback its audio, as well as controlling its pan, gain, attack, release, and other settings available by clicking on the dedicated button. The same applies for a MIDI player, where you can select the MIDI output device, port and channel.

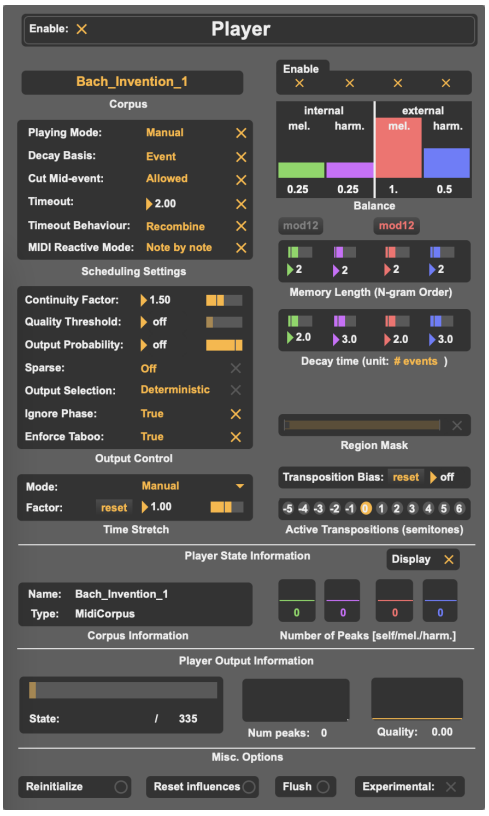


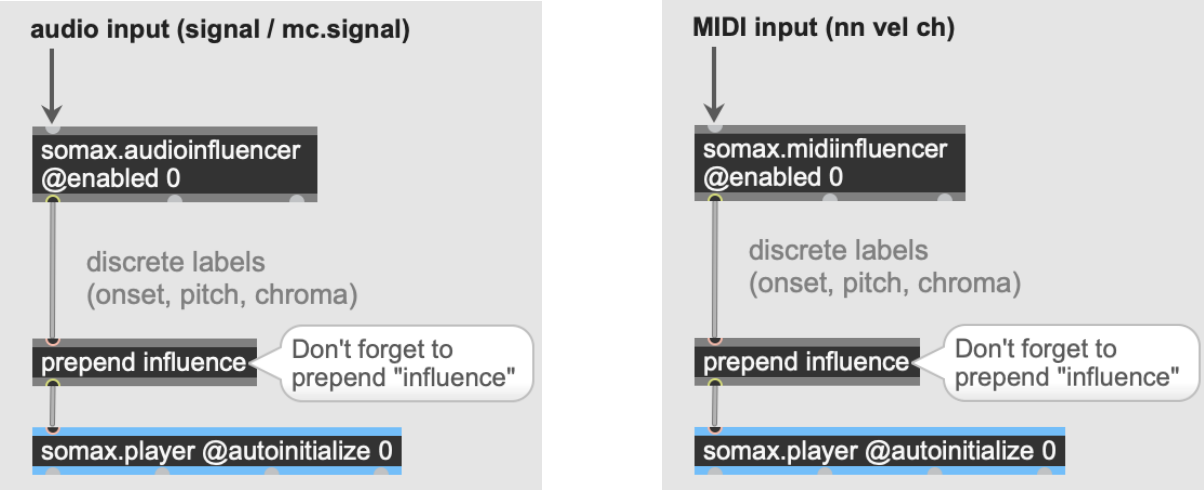
Figure 3.5: Clicking on the «Settings» button of the'somax.player.app' object will open this window with access to all the available player's parameters.

3.4 Influencers

The scope of the influencers is to analyze and segment audio or MIDI into discrete labels (onset, pitch and chroma) in real-time, for them to be sent to any Somax player.

The audio influencer segments and analyzes an incoming audio signal with respect to pitch, chroma and onset into discrete labels, which then can be used to influence a 'somax.player' in order to control the player's way of navigating through the corpus.

Similarly, the MIDI influencer analyzes incoming MIDI data with respect to pitch, chroma and onset into discrete labels, which then can be used to influence a 'somax.player' in the same way.



(a) somax.audioinfluencer **(b)** somax.midiinfluencer

Figure 3.6: Relation between the influencers and the somax.player. Note that it's needed to prepend the word 'influence' to any data output from them, before sending this data to the player.

As the player, the influencers exist in three kind of objects, following the same taxonomy:

- somax.audioinfluencer or somax.midiinfluencer: this is the core Max object. To adjust the parameters, you can send to it a wide set of messages (see 'somax.audioinfluencer.maxhelp' and 'somax.midiinfluencer.maxhelp' for a detailed description of these). You can connect the first outlet of this object to the inlet of a 'somax.player' after prepending the word 'influence' to it. You can also select which kind of influence you want, using a Max 'route' object;
- somax.audioinfluencer.ui or somax.midiinfluencer.ui: this acts exactly the same as the core influencer object, but gives you a visual feedback on the current analyzed data in terms of onset, pitch and chroma (see Figure 3.7) and gives you access to all its parameters by clicking on the «Settings» button;
- somax.audioinfluencer.app or somax.midiinfluencer.app: this is the application module of the influencer. It has at its core the basic influencer object, but in addition to the user interface (same as the ui object) it provides modules for listening to the audio or MIDI outputs, as well as wirelles routing of the influences to any player. The application modules are infact visible in the Influence Sources module of the 'somax.player.app' (the top module, as described before and shown in Figure 3.4).

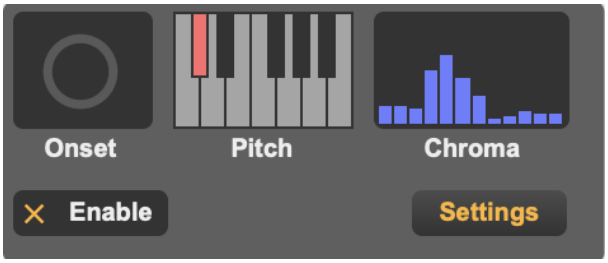


Figure 3.7: Compact interface of a 'somax.audioinfluencer.ui' or 'somax.midiinfluencer.ui' object, showing the analyzed onset, pitch and chroma values.

3.4.1 Audio Influencer App

The 'somax.audioinfluencer.app' object is a convenient wrapper around the audio influencer with some additional user interface to handle input and output. Thanks to this app object, Somax can send influences between influencers and players wirelessly (i.e. without max patch cords). See the 'somax.audioinfluencer.app.maxhelp' for a detailed description of it.

To fine-tune all the available parameters for audio segmentation, you can press the «Settings» button available on the user interface of the 'somax.audioinfluencer.app' (as shown in Figure 3.8). This will open a new window, as shown in Figure 3.9 where you can control each parameter, as well as selecting the method for onset detection. You will also have a visual feedback on the analyzed onset, pitch and chroma.

Figure 3.8: The 'somax.audioinfluencer.app' provides modules for selecting the audio input source, as well as playback and routing of audio output and MIDI output. The user can select the audio input from any channel of the computer soundcard, and route the stereo output of the audio influencer to any output channel, as well as controlling pan and delay values for latency compensation.

Figure 3.9: Clicking on the «Settings» button of the 'somax.audioinfluencer.app' and 'somax.audioinfluencer.ui' objects will open this window with access to all the available parameters for audio analysis.

3.4.2 MIDI Influencer App

Like the audio one, the 'somax.midiinfluencer.app' object is a convenient wrapper around the MIDI influencer with some additional user interface to handle input and output. Thanks to this app object, Somax can send influences between influencers and players wirelessly (i.e., without Max patch cords). See the 'somax.midiinfluencer.app.maxhelp' for a detailed description of it.

To fine-tune all the available parameters for MIDI segmentation, you can press the «Settings» button available on the user interface of the 'somax.midiinfluencer.app' (as shown in Figure 3.10). This will open a new window, as shown in Figure 3.11 where you can control each parameter, as well as selecting the method for pitch analysis. You will also have a visual feedback on the analyzed onset, pitch and chroma.

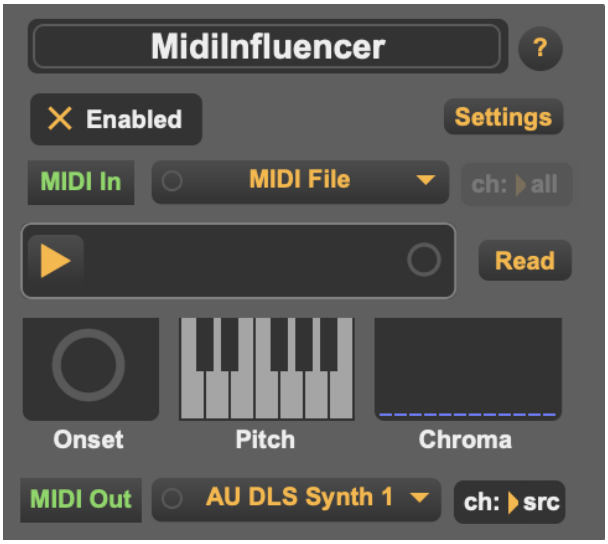


Figure 3.10: The 'somax.midiinfluencer.app' provides modules for selecting the MIDI input source, as well as playback and routing of MIDI output. The input can be a MIDI file or any MIDI channel, while the output can be sent to the standard Max MIDI device or sent to any external application, using the desired MIDI channel.

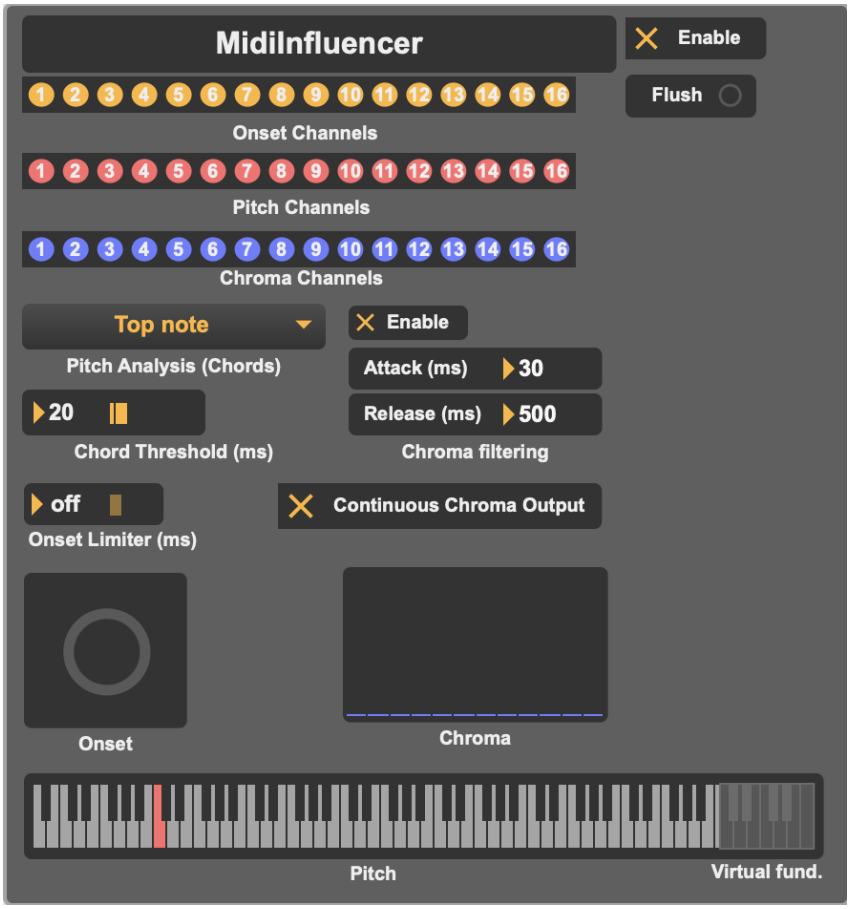


Figure 3.11: Clicking on the «Settings» button of the 'somax.midiinfluencer.app' and 'somax.midiinfluencer.ui' objects will open this window with access to all the available parameters for MIDI analysis.

3.4.3 Audio Corpus Builder

The 'somax.audiocorpusbuilder' is used to construct a corpus from an audio file. 'A somax.player' cannot read an audio file directly, it requires the audio to be analyzed and segmented before being able to parse it. The 'somax.audiocorpusbuilder' does exactly that — it segments the audio file and analyzes it with regards to a number of descriptors and exports the result as a separate file, that then can be loaded into a 'somax.player'.

As opposed to the objects seen so far, the audio corpus builder has only one kind of object: the core one. However, building a corpus implies tuning quite a lot of parameters. Especially for audio material, having a visual feedback of where the onsets are exactly located in the analyzed file, and how long are the resulting slices going to be, is extremely important in order to build a nice corpus. For this reason a user interface with full access to all the parameters of the segmentation, as well as visual feedback on the sliced sound file, is available by double clicking on the 'somax.audiocorpusbuilder' or by sending to it the message 'openwindow'. This interface is shown in Figure 3.13.

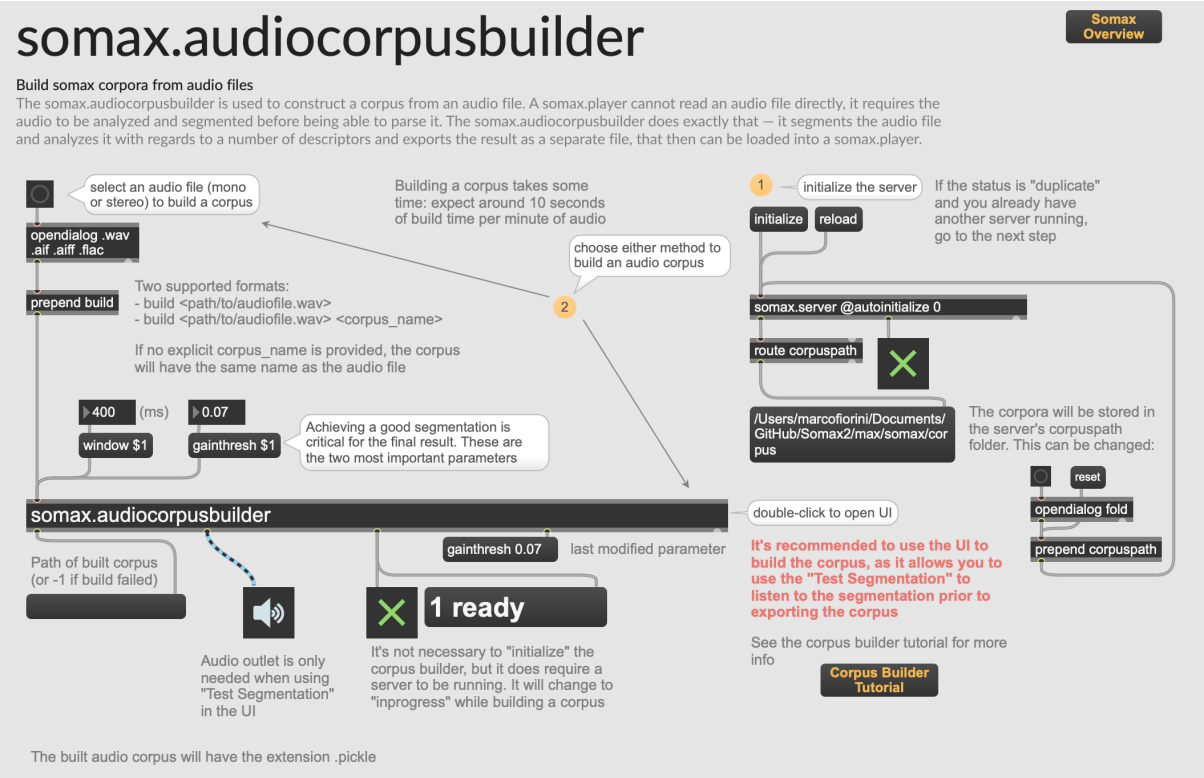


Figure 3.12: Help patch for 'somax.audiocorpusbuilder.maxhelp' available in the /help folder and from the help center of the 'somax2.overview.maxpat'.

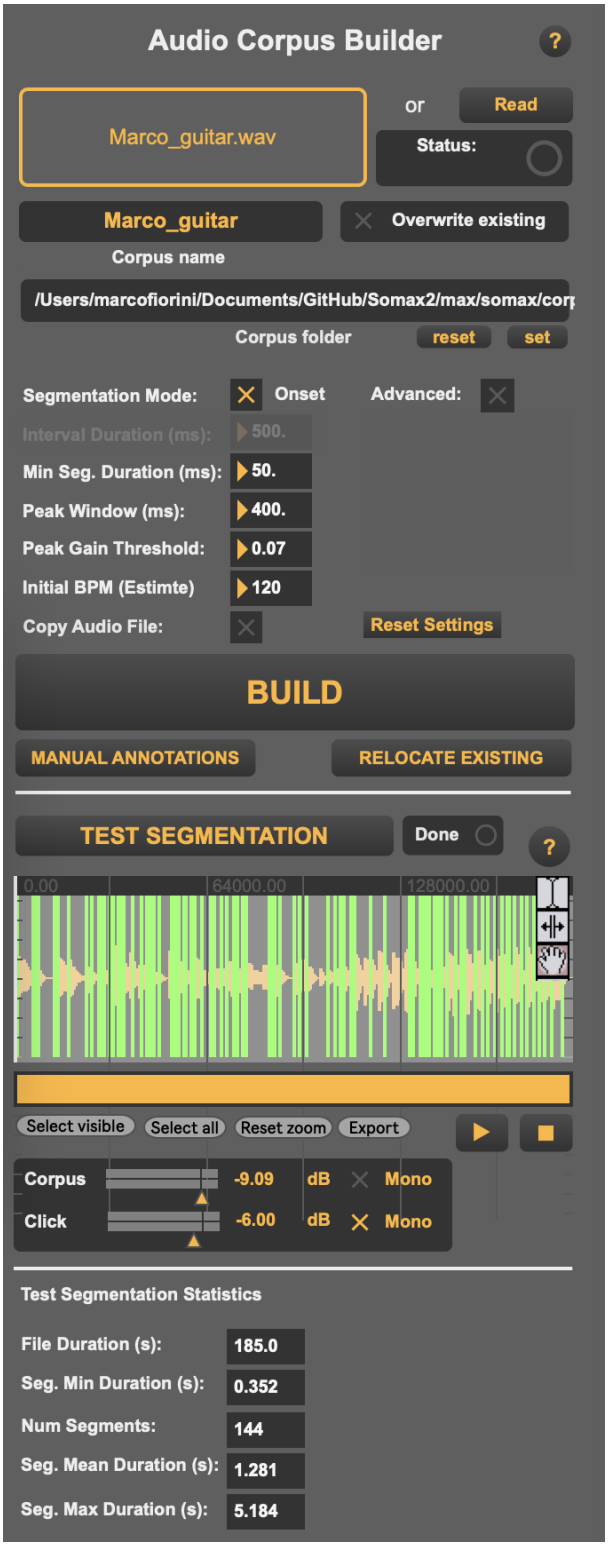


Figure 3.13: The 'somax.audiocorpusbuilder' provides a wide set of parameters for segmenting audio corpus, as well as module to visualize and listen to the segmentation before actually building the corpus.

For a more detailed description of it, see the 'somax.audiocorpusbuilder.maxhelp' or the audio corpus builder tabs in the 'somax.server.app.maxhelp'.

For an actual usage in context of it, see the 'app2 - Audio Corpus Builder.maxpat' App Tutorial or the 'max3 - Building a Corpus.maxpat' Max Tutorial.

3.4.4 MIDI Corpus Builder

Similarly to the audio one, the 'somax.midicorpusbuilder' is used to construct a corpus from one (or multiple) MIDI file(s). A 'somax.player' cannot read a MIDI file directly, it requires the MIDI to be analyzed before being able to parse it. The 'somax.midicorpusbuilder' does exactly that — it segments the MIDI file and analyzes it with regards to a number of descriptors and exports the result as a separate file, that then can be loaded into a 'somax.player'.

Just like its audio counterpart, the MIDI corpus builder has only one kind of object: the core one. But for the same reasons, a user interface with full access to all the parameters of the segmentation is available by double clicking on the 'somax.midicorpusbuilder' or by sending to it the message 'openwindow'. This interface is shown in Figure 3.15. Of course here you won't have a representation of the waveform, since you are dealing now with MIDI data. However, the MIDI corpus builder gives you access to a selection of the different descriptors for pitch and chroma to different MIDI channels. In this way it's possible to build a corpus from multiple MIDI files, as well as selecting the melody (pitch) and the harmony (chroma) from different MIDI channels, potentially associated to different MIDI files.

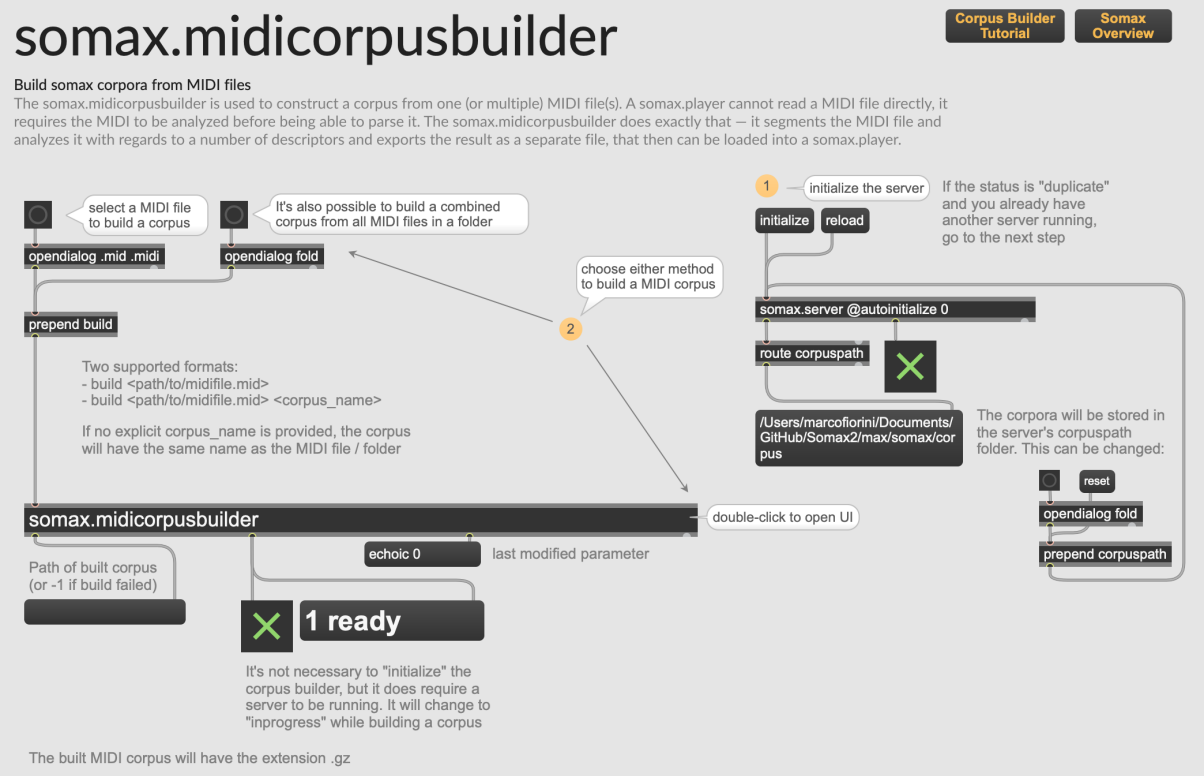


Figure 3.14: Help patch for 'somax.midicorpusbuilder.maxhelp' available in the /help folder and from the help center of the 'somax2.overview.maxpat'.

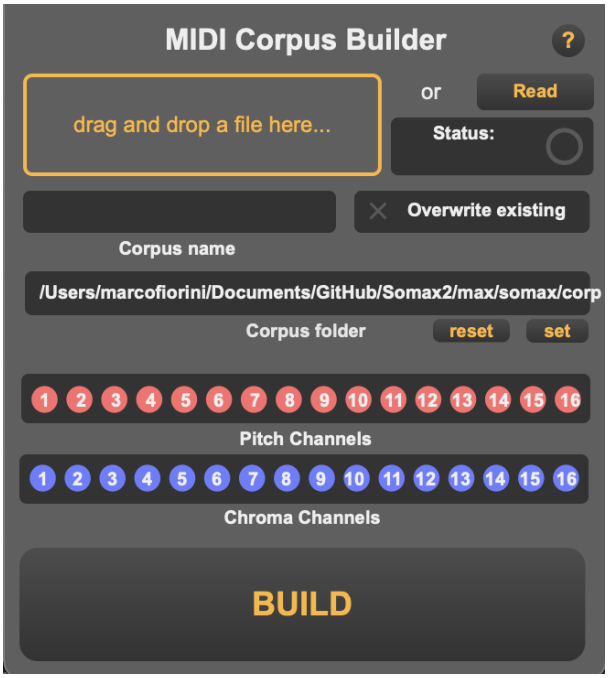


Figure 3.15: The 'somax.midicorpusbuilder' has parameters to specify the melodic (pitch) and harmonic (chroma) dimensions when building the MIDI corpus.

For a more detailed description of it, see the 'somax.midicorpusbuilder.maxhelp' or the midi corpus builder tab in the 'somax.server.app.maxhelp'.

For an actual usage in context of it, see the 'max3 - Building a Corpus.maxpat' Max Tutorial.

3.5 Recap

Now that each Somax2 application object has been presented, it should be easier for you to locate them in the default 'somax2.maxpat' shown in Figure 3.16. Thus, we can be even more precise in classifying the objects contained in this patch, as it should be now clear that all of those are .app objects, communicating in a wireless way with each other:

- somax.server.app;
- somax.player.app;
- somax.audioinfluencer.app;
- somax.midiinfluencer.app.

As previously mentioned, this is just one possible configuration that will give you immediate interaction. We encourage you to use the 'somax2.overview.maxpat' to explore all the suggested configurations and resources, notably the different template patches located in the /templates folder. Use these as a sort of test bench, to try out all the different means of interaction and explore the parameters. Once you will be confident with the objects, you could then create your own patches and achieve new ways of configurations and interaction. Remember to rely on the 'somax2.overview.maxpat' to redirect you wherever you need, and to explore all the App and Max Tutorials contained there, as well as studying the help files available from the Help Center, and the Performance Strategies.

Figure 3.16: The 'somax2.maxpat' included in the distribution gives you quick access to all the objects in the Somax2 application: server, player, audio influencer and MDI influencer.

4. Mastering Somax2 Interaction

4.1 Basic Workflow

Based on the theoretical model previously described in Section 2, the basic workflow of interacting with the Somax2 system, as shown in Figure 4.1, unfolds as following:

- Somax2 generates its improvisation material based on an external set of musical material, the «corpus». This corpus can be constructed from one audio or MIDI file freely chosen by the user, thanks to the dedicated corpusbuilder objects. The constructed corpus is then stored in a bigger database called corpora, accessible by the Player from a corpuspath folder. So, the corpus is the actual musical material loaded into a Player.
- The Audio and MIDI Influencers listen to a continuous stream of audio or MIDI input data (from any type of source, including live musicians) and segments it temporally, where each slice is analyzed with respect to onset, pitch and chroma, which then is sent to the Player. Thus, the influences are the triggers for the co-improvisational behaviour of the Player.
- The Player is the main agent of Somax2. It listens to the influencers and, based on that, it recombines the content of the corpus, generating some output.
- The Server is the core of Somax2, handling all the scheduling and communication with the background Python app and all instances of somax.player through OSC. It will open in an external window when you launch it.
- The ‘somax.player’ does not play back, it only provides a list of sequential events. Playback is handled independently via the somax.audio/midi renderer objects (or by implementing your own sequencer/playback patch). However, if you use the ‘somax.player.app’ object, rendering and playback of the output is handled here, as this is a ready-to-play application object.

4.1.1 Interaction Parameters

While the ‘somax.player’ has a wide set of parameters, fully documented in the «parameters» tab of the ‘somax.player.app.maxhelp’, the ‘somax.player.ui’ gives you access to a selection of main parameters, as shown in Figure 4.2, and described in the following section.

Playing Mode

Controls the player’s mode:

- Reactive: Output will be triggered whenever the player receives input from an influencer (note-by-note);
- Continuous: Output will be triggered continuously regardless of input.

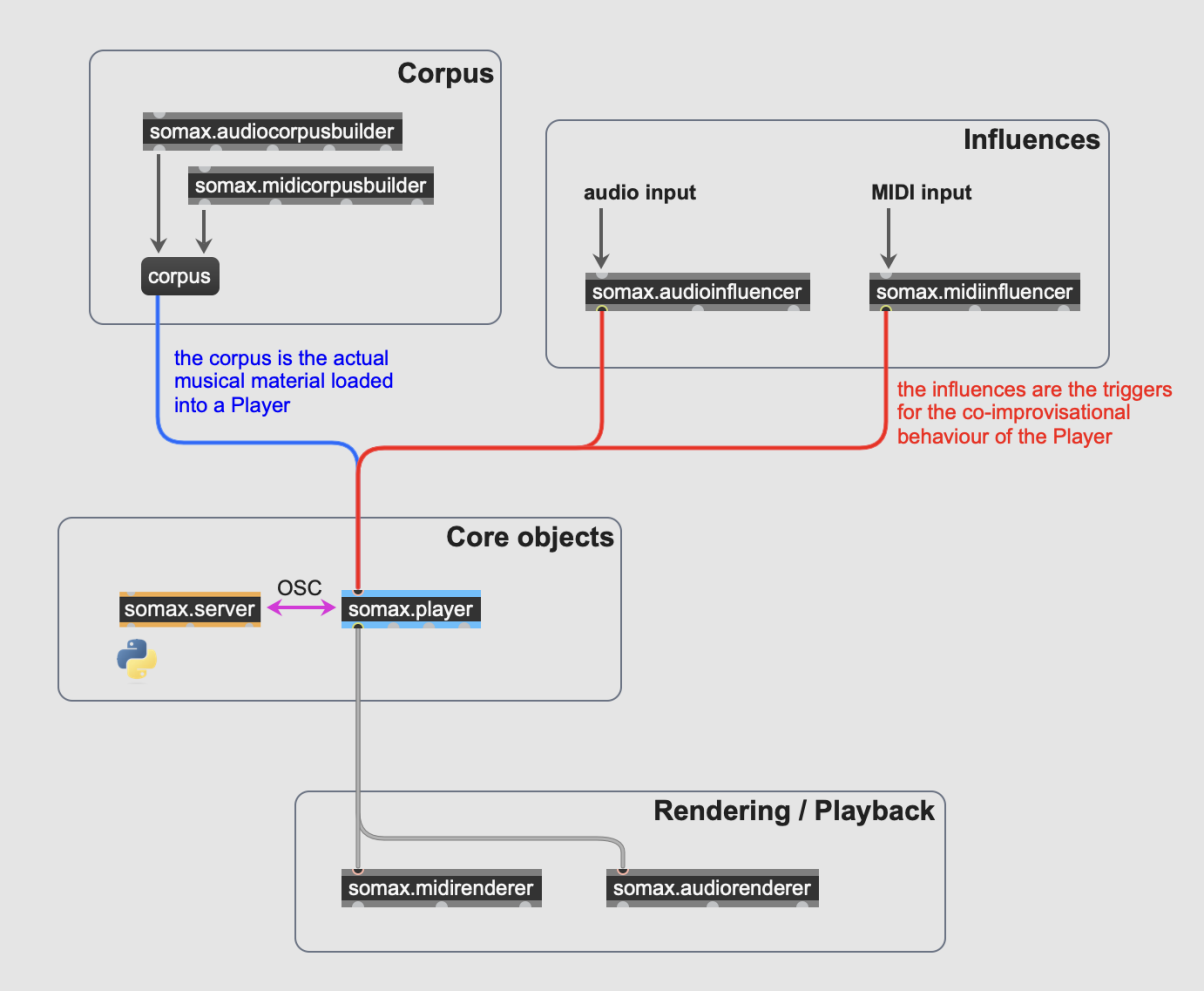


Figure 4.1: Basic Workflow of the Somax2 system.

Timeout

In reactive mode, this option controls whether the player should continue playing if no new trigger has arrived by the time the player has finished playing its current event. Setting this to a non-zero value will make the player continue for that number of seconds. It's also possible to disable this to make the player continue endlessly.

Continuity

Continuity controls the order of the current state index of the player's output:

- Continuity > 1 will prioritize continuation (and result in fewer jumps);
- Continuity = 1.00 will result in no alterations (no bias introduced by this parameter);
- Continuity < 1 will prioritizing jumping.

Quality / Sparse

The «Quality» Threshold sets a minimum score required for a match to qualify as output. When combined with «Sparse», this will ensure that no events are played unless they are considered good matches.

If «Sparse» is On:

Figure 4.2: The user interface of 'somax.player.ui' shows a set of main parameters to control the interaction.

- quality 0.0 plays any found match;
- quality 1.0 will almost never play;
- in-between values will act as a threshold, to select matches above this threshold and play them.

If «Sparse» is Off, it will replace the voids (no-play) by a default event, generally the next event, or a jump resulting from self-influence.

Cut

In reactive mode, output is generated each time a new trigger (onset) arrives. If the player is in the middle of playing an event when a trigger arrives, «cut» controls whether the currently played event should be interrupted or not:

- Allowed: Interrupt the current event and trigger the new event immediately when the new trigger arrives.
- Not Allowed: Delay the trigger so that the new event starts playing once the current event has been completed.

Probability

It will condition each generated output with a probability, so that it may or may not play the event. This parameter is inactive when set to 1.0 (off), but any value lower than 1.0

will result in less than 100% of the events being played. For example, when set to 0.2, only 20% of the generated events will be played

Internal and External Influences

Control the balance between different internal and external type of influences (layers). As it can be seen in Figure 4.3, the four colors (green, purple, red, blue) correspond to the four different layers introduced in Section 2.2.4:

- Green (internal melody): The feedback layer which listens to the melody (pitch) of the previous output of the player itself;
- Purple (internal harmony): The feedback layer which listens to the harmony (chroma) of the previous output of the player itself;
- Red (external melody): The melody layer which listens to melodic (pitch) influences from exernal sources (audio/MIDI influencers);
- Blue (external harmony): The harmony layer which listens to harmonic (chroma) from external source (audio/MIDI influencers).

Figure 4.3: User interface to control the Balance between the dimensions, length of matching sequences for each dimension as well as decay time of peaks (Memory lengths).

State

The last state that was output is visualized in the lower part of the interface, as well as a Region Filter to select the desired range of states you want the player to jump to. Here you have also a visual feedback of the occurrence of a match or not, through the «Match» light. If green, a match occurred and is being played, if red, you don't have a match, if yellow, the player is playing a default sequence, generally linear, which happens either because there is no match — but Sparse is Off , so it defaults to the fallback behaviour — or because it is playing the time-out sequences.

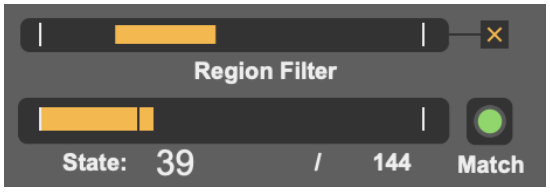


Figure 4.4: This part of the player's interface shows the last output state, as well as the region filter (if enabled), and if a match occured or not.

Number of Matches

The green, purple, red and blue indicators (see Figure 4.5 shows the number of matches in each layer. Note that this may contain overlapping matches. The white indicator shows the total number of matches when all layers have been merged. The merged result will not contain duplicates, and will in most cases be lower than the sum of the individual layers.

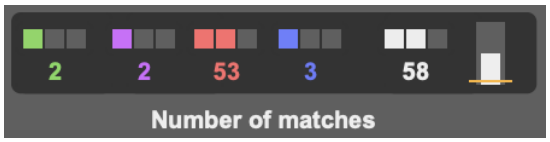


Figure 4.5: While the user doesn't interact directly with the peaks, they are still indicated in the user interface. Here, the colors green, purple, red, blue and white correspond to the number of peaks in the internal feedback layer (green: pitch, purple: harmony), external pitch layer (red), external harmonic layer (blue), and total number of peaks after merge (white) .

5. Tutorials and Further Explorations

Somax2 opens up a new world of surprising and ever-evolving ways of interaction. Here we have a few more suggestions, but we encourage you to explore the whole library and we hope you could achieve exciting results with it.

5.1 Somax2 Overview patch

Starting from the 'somax2.overview.maxpat' located in the root folder of Somax2, or in the /extras folder, and available from the Extras menu of Max, you can browse the vast set of resources included in the Somax2 distribution.

As shown in Figure 5.1, according to your needs, from here you can have quick access to tutorials for the Somax2 application, tutorials for the Somax2 Max objects, Performance Strategies and Templates, as well as a Help Center to have thorough information about any Somax2 object.

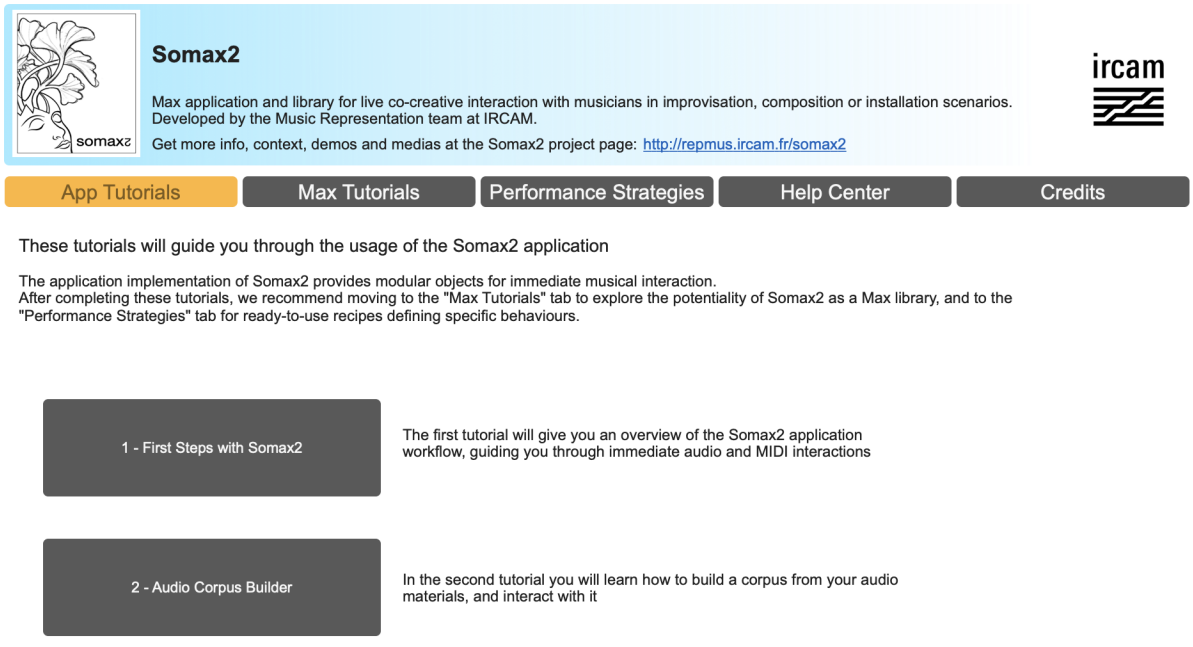


Figure 5.1: The 'somax2.overview.maxpat' is the starting point for the Somax2 package. From here you can have access to the App Tutorials (first tab), Max Tutorials (second tab), Performance Strategies and Templates (third tab), Help Center (fourth tab), as well as a direct link to the project page.

5.2 App Tutorials

From the first tab of the 'somax2.overview' you will have access to two interactive tutorials that will guide you through the basics of the Somax2 application.

The first tutorial ('app1 - First Steps with Somax2.maxpat', see Figures 5.2 and 5.3) will give you an introduction to the Somax2 app objects, as described in Section 3. This tutorial is intended to walk you in the very first steps of using the Somax2 application and it will give you immediate access to a quick interaction, using both audio and MIDI.

The second tutorial, 'app2 - Audio Corpus Builder.maxpat' as shown in Figure 5.4, will guide you to the usage of the audio corpus builder, to build a corpus from your own audio material and use it immediately in Somax2.

Use these detailed tutorials for a quick dive into the application version of Somax2.

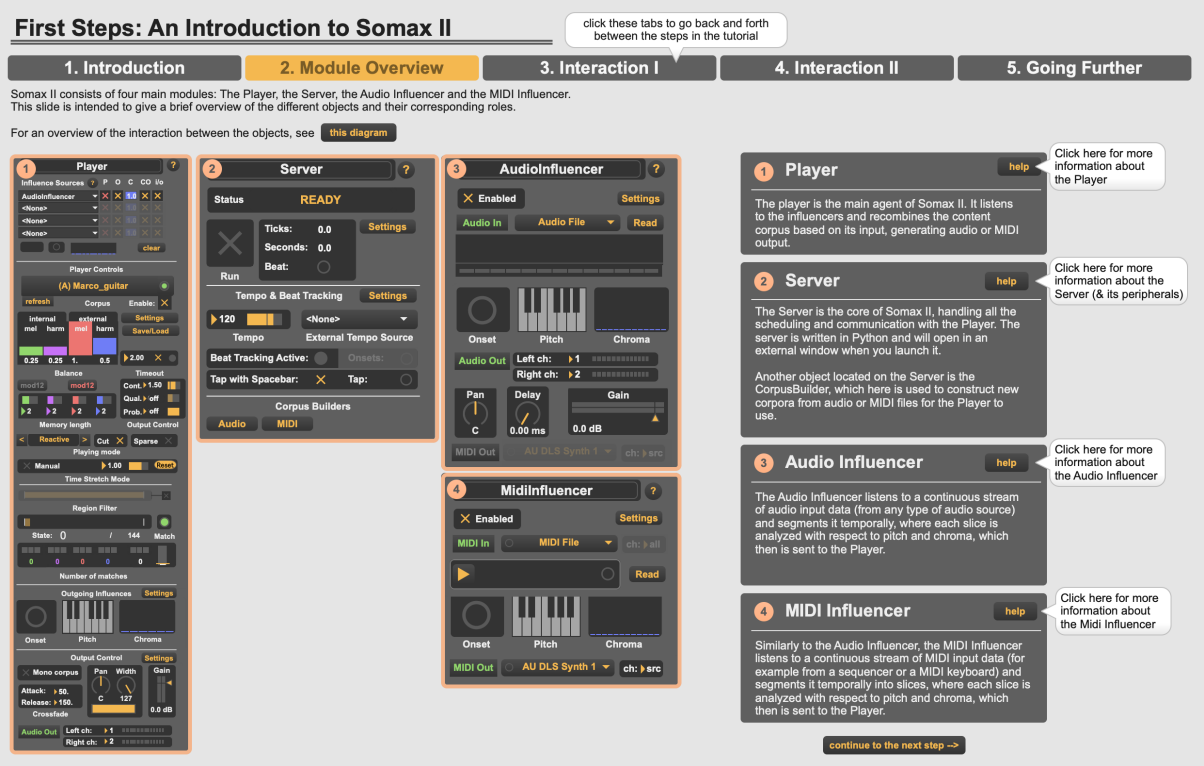


Figure 5.2: The tutorial 'app1 - First Steps with Somax2.maxpat' will give you a first overview of all the Somax2 app objects, and guide you through your very first steps.

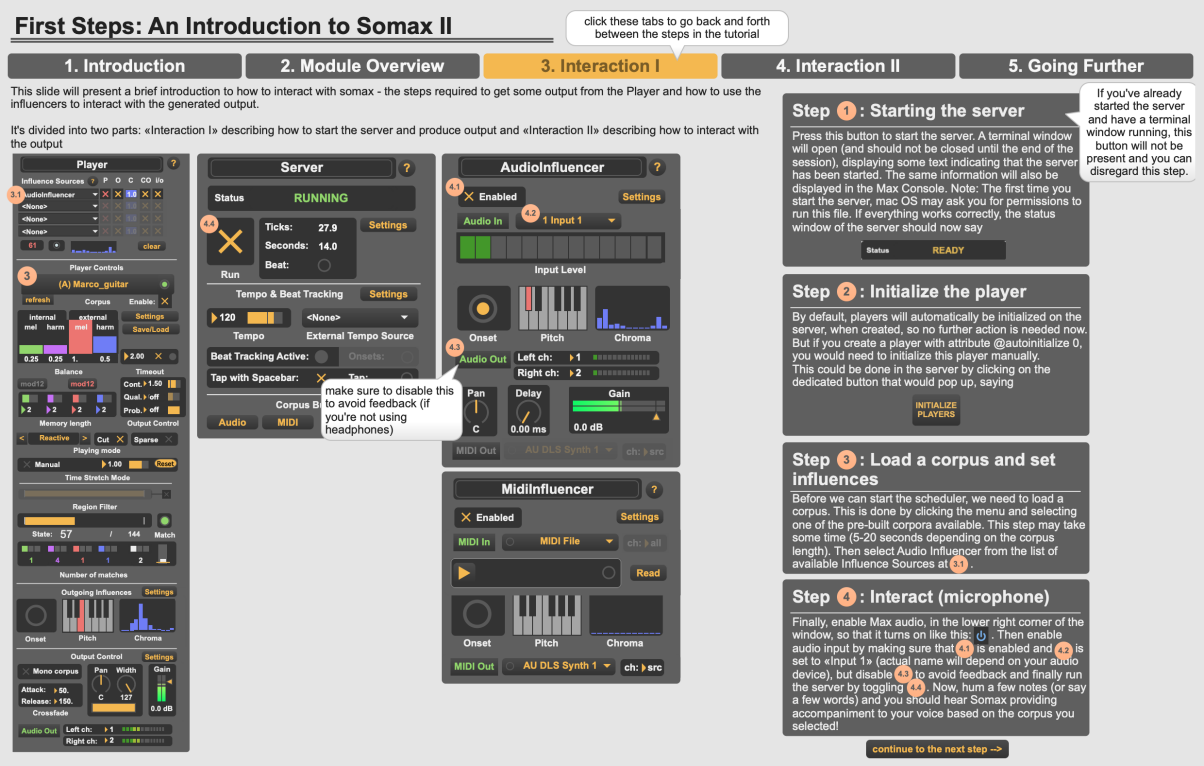


Figure 5.3: The tutorial 'app1 - First Steps with Somax2.maxpat' will guide you through audio and MIDI basic interaction.

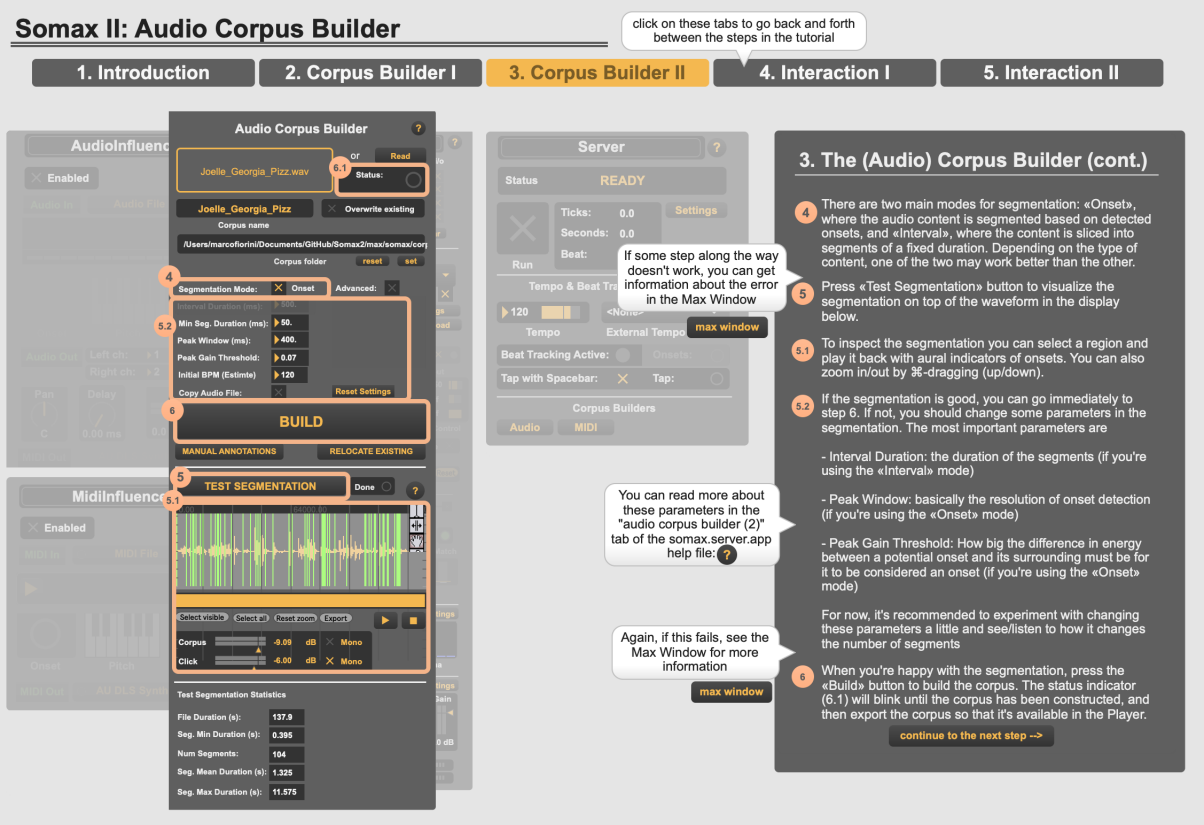


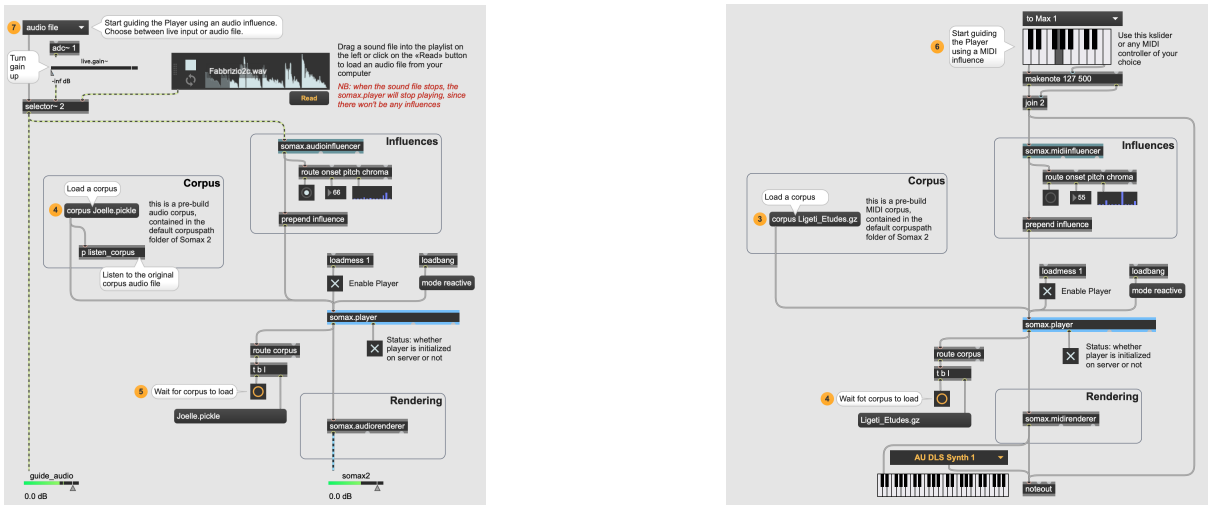
Figure 5.4: The tutorial 'app2 - Audio Corpus Builder.maxpat' will illustrate how to build an audio corpus from your own musical material, and use it immediately in Somax2.

5.3 Max Tutorials

From the second tab of the 'somax2.overview' you will have access to seven detailed tutorials, that will guide you through the basic steps of using Somax2 Max objects.

These tutorials are distributed according to an incremental level of complexity, so we suggest that you follow them in their order, to understand and explore the deep world of possibilities that the Somax2 Max library can give you. Each tutorial has several tabs, to give you a full overview of both audio and MIDI, according to the covered topic.

The following Figures will show you some brief extracts of these Max tutorial patches.



(a) Basic audio demo **(b)** Basic MIDI demo

Figure 5.5: The tutorial 'max1 - Basic Workflow.maxpat' will give you the bases for audio and MIDI Max interaction.

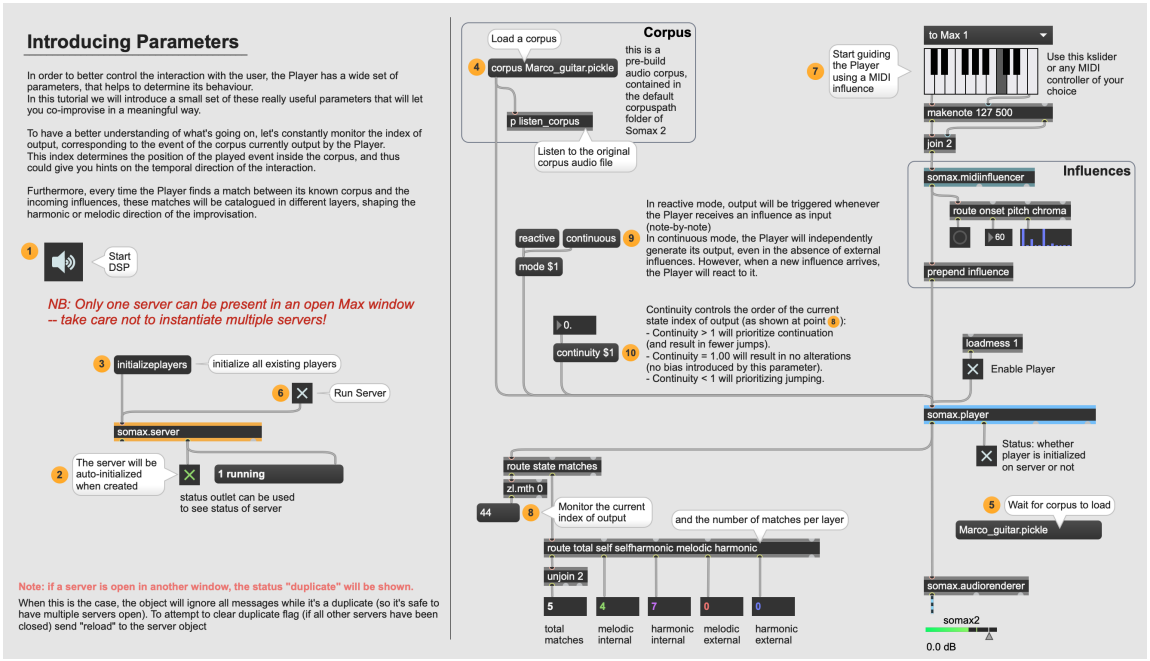


Figure 5.6: The tutorial 'max2 - Introducing Parameters.maxpat' will give you a first overview of using the interaction parameters, as described in Section 4.1.1. These parameters are then deeply explored in the tutorial 'max4 - Mastering Somax Interaction Parameters.maxpat'.

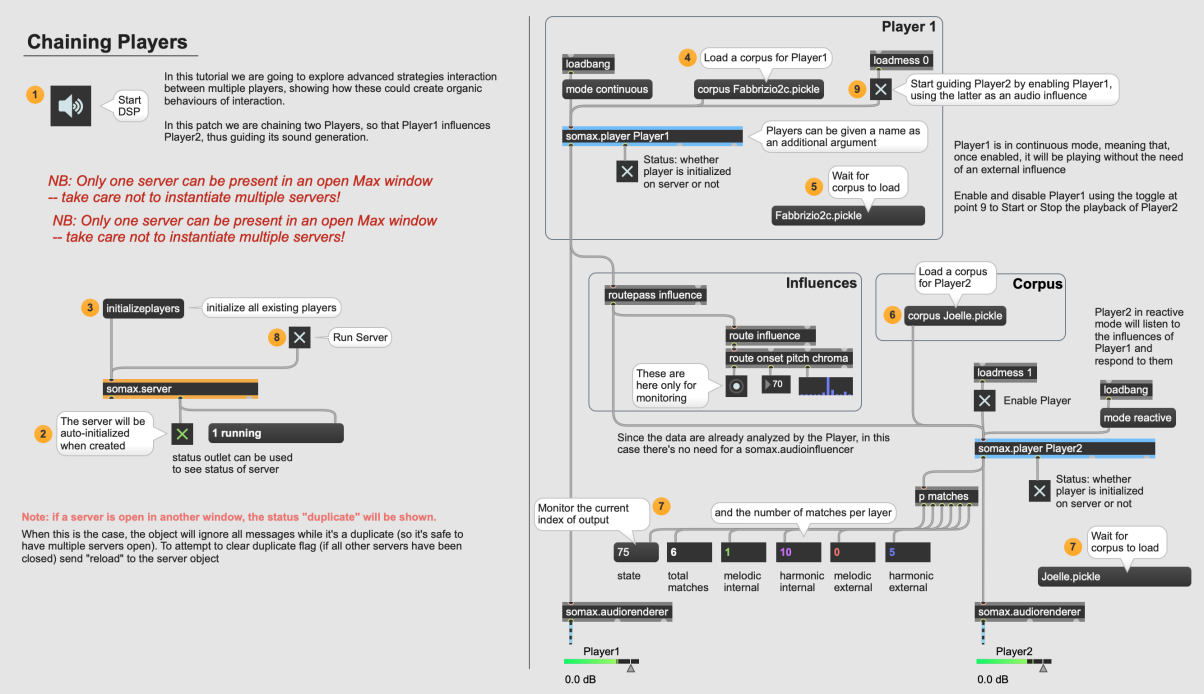


Figure 5.7: The tutorial 'max5 - Advanced Players Interaction.maxpat' shows how to achieve complex means of interaction, thanks to the modularity of the Somax2 Max library. In this particular example, a 'somax.player' is used to influence another player.

Figure 5.8: The tutorial 'max7 - UI versus App.maxpat' will show you how to configure the .ui version of Somax2 objects to work exactly as the .app ones.

5.4 Performance Strategies

Once you have familiarized with the concepts explained in this document, and explored the different tutorials, we encourage you to try the performance strategies patches.

These are ready-to-play patches that achieve specific Somax2 behaviours, like mimetism, harmonization or a never-ending installation mode. These behaviours are achieved through scripting the different Somax2 objects with initialization messages in Max, and have been specifically chosen from our team members' preferred configurations. You can find those patches in the /docs/tutorial-patches folder, or from the third tab of the 'somax2.overview.maxpat'.

The following Figures will give you an overview of these patches.

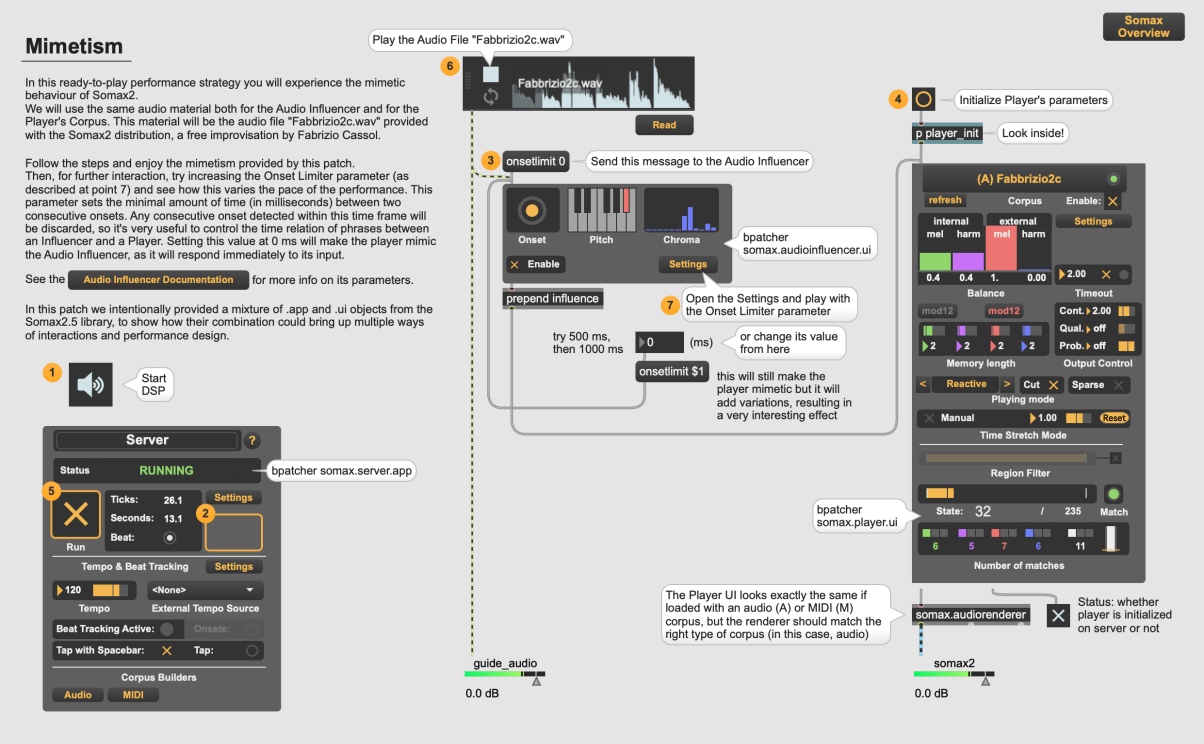


Figure 5.9: The patch 'performance - Mimetism.maxpat' will give you immediate mimetic results between an audio influencer and a player.

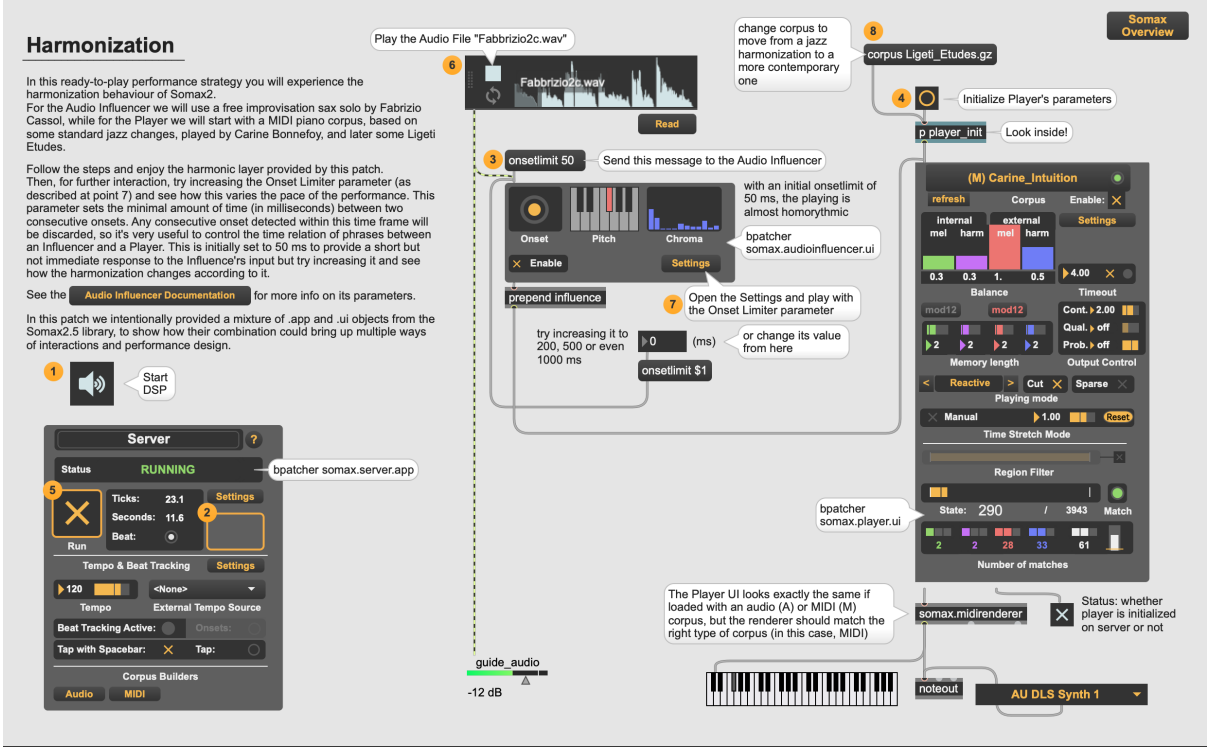


Figure 5.10: The patch 'performance - Harmonization.maxpat' will let you use the harmonic material of a player to instantly harmonize a melodic audio influencer.

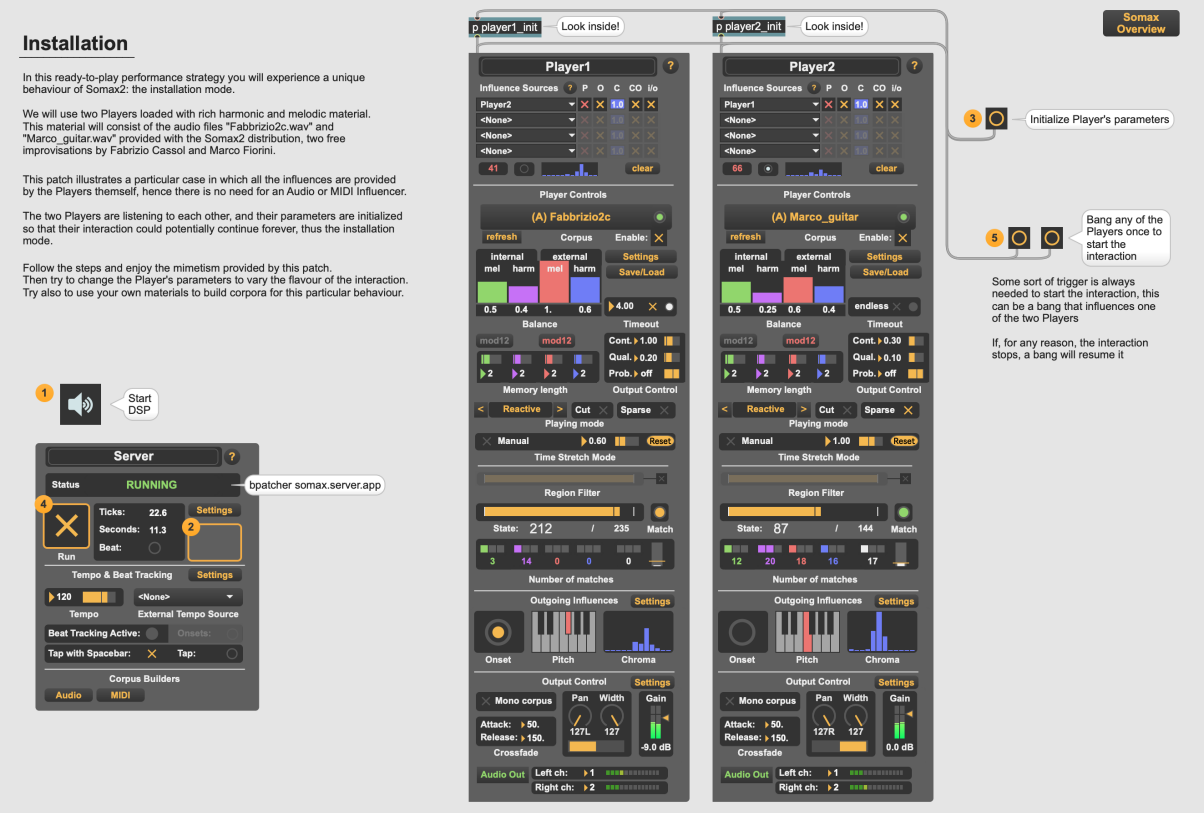


Figure 5.11: Thanks to the 'performance - Installation.maxpat' patch you could have an installative environment where the interaction between two players could potentially continue endlessly.

5.5 Templates

In the /templates folder you can find four patches, already set up with configurations from one to four players, and ready to play. These patches use only the application objects of Somax2 and can be used as a test bench to try all the different Somax2 features in various players configurations, as well as a starting point to create your own patches. Choose one configuration that you like, copy all the objects in a new patch and have fun experimenting with your own new patch.

Figure 5.12: Overview of the four different application template patches that provide easy access to configurations from one to four players.

5.6 Help Center

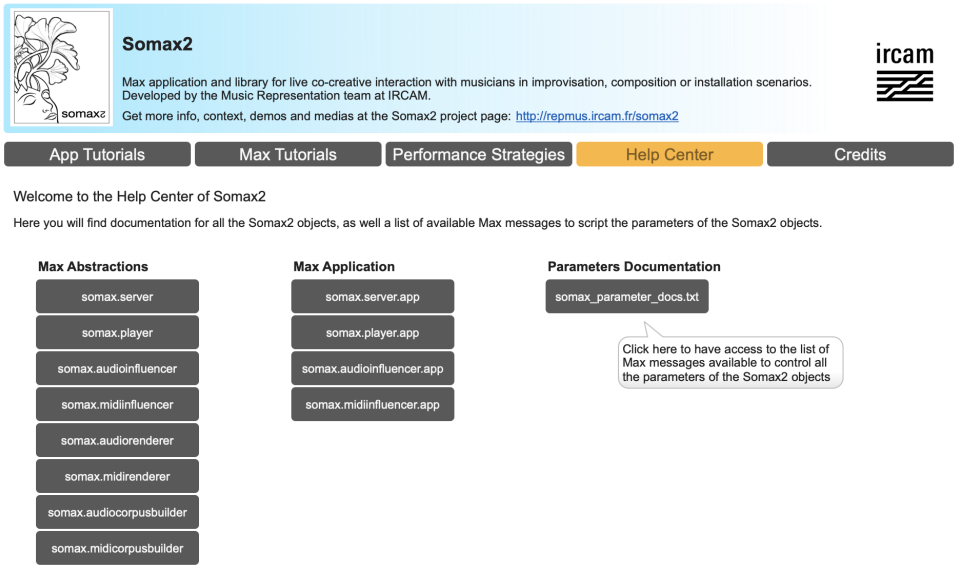


Figure 5.13: From the fourth tab of the 'somax2.overview' you will have access to an exhaustive Help Center. From here you can open the maxhelps of any Somax2 object, both as .app and Max versions. There is also a button to access a document listing all the possible Max messages you can use to control the different interaction parameters.

5.7 Credits

Somax2 is a totally renewed version of the Somax reactive co-improvisation paradigm born in the Music Representations Team at Ircam - STMS, a descendent of the well known OMax improvisation software.

It is part of the research projects ANR MERCI (Mixed Musical Reality with Creative Instruments) and ERC REACH (Raising Co-creativity in Cyber-Human Musicianship) directed by Gérard Assayag.

Somax 2.5 development by Joakim Borg, documentations and tutorials by Joakim Borg and Marco Fiorini.

Somax created by Gérard Assayag and Laurent Bonnasse-Gahot, adaptations and pre-version 2 by Axel Chemla Romeu Santos, early prototype by Olivier Delerue.

Thanks to Georges Bloch, Mikhaïl Malt and Marco Fiorini for their continuous expertise.

Thanks to Bernard Borron, Bernard Magnien, Carine Bonnefoy, Joëlle Léandre, Fabrizio Cassol, Marco Fiorini for their musical material used in Somax2 distribution corpus.

5.8 More and Contacting the Team

See project page at `http://repmus.ircam.fr/somax2` for info, help, demo videos, medias and various resources.