# Dicy2 for Max
**composing and interacting with generative agents**

Version 3.0

December 9, 2022

# Contents

# 1   Dicy2 library for Max

## 1.1   Introducing Dicy2

**Dicy2 for Max** is a suite of Max abstractions which use machine learning for interactive generation of musical sequences. It can be integrated into musical situations ranging from the production of structured material within a compositional process to the design of autonomous agents for improvised interaction. It is available as a plugin for Ableton Live and a library for Max.

There is a discussion group for the **Dicy2 library for Max**, which can be found at the Ircam Forum: https://discussion.forum.ircam.fr/c/dicy2/.

## 1.2   Getting ready

### 1.2.1   Requirements

- Mac OS High Sierra (10.13) or greater

- Max 8.5 or higher

- MuBu for Max, **v.1.10.5 or higher**, which can be installed from the Package Manager.

### 1.2.2   Installation

Dicy2 is a Max package, which can be downloaded from:
https://forum.ircam.fr/projects/detail/dicy2/
Just unzip the package and put it in the /Documents/Max 8/Packages folder.

### 1.2.3   What's in the package

The Dicy2 distribution contains:

/patchers:

- Two core abstractions: dicy2.server.maxpat and dicy2.agent.maxpat.

- A set of easy abstractions to manage interactive audio with Dicy2.

/examples:

- Five tutorial patches covering basic concepts, a number of use cases, and detailed breakdowns of audio patching for advanced users.

- A 'Performance Strategies' folder, containing real-world examples built from collaborations with partner artists.

/externals: These max externals are used internally by some of the Dicy2 abstractions.

- spat5.file.info max external

- spat5.shell

/misc:

- Dicy2_server.app

The Dicy2_server.app is the background app which does the heavy lifting of Dicy2, and communicates wih the Dicy2.server for its communication with the Max environment.

/media:

This contains files used in the examples and tutorials.

/extras:

- the Dicy2-Overview patcher, accessible from the Extras menu, contains links to all of the content in the Dicy2 package.

**Warning**    The first time you use Dicy2, you will probably get a pop-up message asking you to allow `dicy2.server.app` to run. Click OK.

Video tutorials are available on Ircam's Youtube channel.

## 2 Dicy2 Concepts

Dicy2 uses machine learning to generate structured variety from musical source material. This basis material is used to train the dicy2.agent object, which uses machine learning techniques to create an internal map of temporal relationships within the basis material. The user then sends queries to the agent (called a Scenario), and Dicy2 returns segments of musical material in response. Using pattern recognition algorithms, it looks for a 'best match' between the Scenario and the material stored in the memory, taking into account both musical similarities and the temporal structure of the Memory.

To do this, the original source material is segmented, and each segment is given a label. A segment can be a MIDI note, a segment of an audio file, or anything in your original material that can be readily divided. Segments are labeled by similarity: similar segments are labeled identically, and dicy2 builds its model based on these labels. This combination of segments and labels is known as a Memory.

Once the Memory is built, you can send Dicy2 sequences of these labels (called a Scenario), and it will return optimized content that constitues the 'best match' for the label sequence. The content (which is output as a list) can then be sent to further patches for rendering. The illustration below presents a schematic of the Dicy2 workflow, showing the most common helper abstractions.
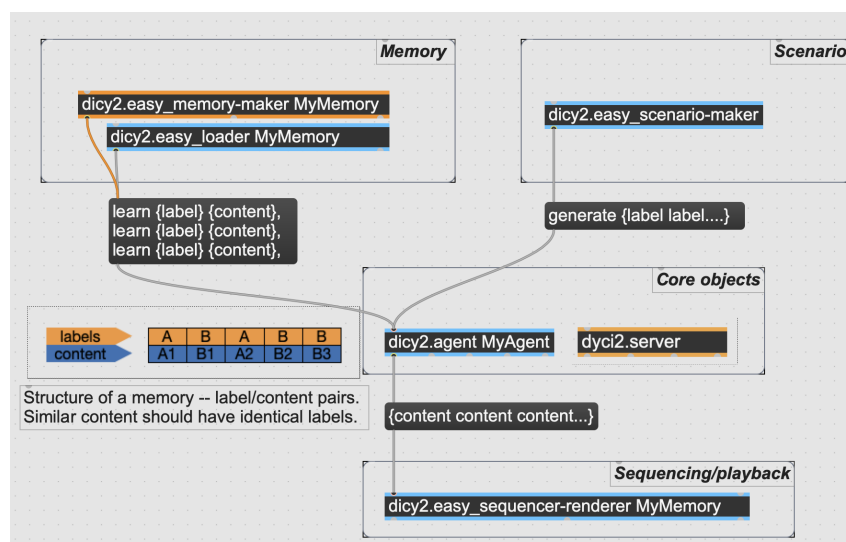


Figure 1: Basic Dicy2 data flow

## 2.1 Core Objects

### 2.1.1 The Server

Dicy2 runs in a background application called Dicy2_server.app, not in the Max application itself. The dicy2.server Max object is an abstraction that communicates with this app via Open Sound Control. The only function of the dicy2.server is to act as a communication hub between the Dicy2.agent Max abstractions and the background app.

Sending the dicy2.server the 'initialize' message will launch the background app, and open up OSC ports for communication with the background app and the Agents. The server and the background app can be stopped by sending the server the 'exit' message.

*Very important: only one copy of the dicy2.server abstraction should be open per instance of Max at any given time. Having multiple copies open will lead to errors!*

*Also very important: you should always have a [freebang] object send an 'exit' message to the Server in order to shut down the background app.*
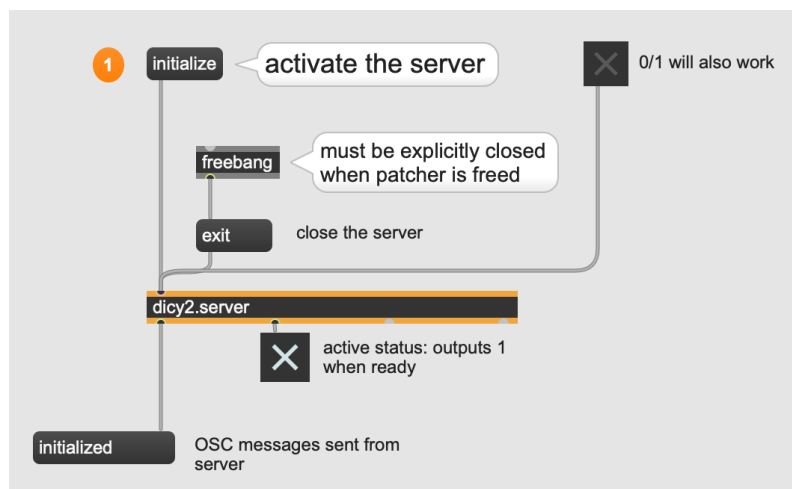


Figure 2: The Server

### 2.1.2 The Agent

The dicy2.agent abstraction is where you will have most of your interaction with the Dicy2 environment. You will train the Agent on a memory, and send Scenarios to the Agent, which will generate optimized responses of musical

6

material. You may have any number of Agents operating simultaneously, all communicating with the same Server. Each Agent requires a unique name as an argument. It expects a Scenario in its left inlet, and returns a content sequence from its left outlet.

## 2.2   Memory and Scenario

### 2.2.1   Memory

The Memory is a temporally structured sequence of events (which could be MIDI notes, audio segments, or really anything), each of which has a label. Events with the same label usually have one or more closely related parameters. A Memory can be either be developed offline or generated from input in real time, and labels can either be created by hand or generated by machine listening to audio content.

**The Memory**
The Memory is the sequence of labels and segmented content used to drive playback, and from which the agent creates a model used to generate response sequences.

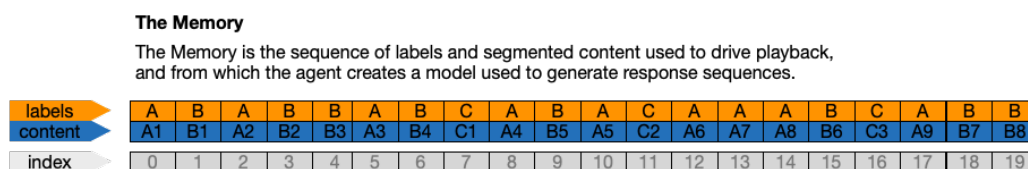| labels | A | B | A | B | B | A | B | C | A | B | A | C | A | A | A | B | C | A | B | B |
|--------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| content | A1 | B1 | A2 | B2 | B3 | A3 | B4 | C1 | A4 | B5 | A5 | C2 | A6 | A7 | A8 | B6 | C3 | A9 | B7 | B8 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

Figure 3:  Structure of a Memory

A Memory is any combination of content/label pairs.  When working with audio, it is easiest to use the dicy2.easy_memory-maker abstraction (discussed below), but to train an Agent on a Memory, all that is needed is to send it pairs in the format {label} {content}. The label can be either an integer or a symbol, but the content must be sent to the agent as a single symbol.

If your content is limited, you can write it by hand and store it in any kind of text file.  For example, here is part of the word-based memory from the server_and_agent tab in tutorial 0, which is stored in a coll object:

    29, adverb When;
    30, pronoun I;
    31, verb "am driving";
    32, preposition in;
    33, pronoun my;
    34, noun car;

Here, the first item in each line is a part-of-speech label, and the second the

7

content.(Note that in line 31, "am driving" is in quotes to make it a single symbol.) In this context, a 'dump' message to the coll will send out one line at a time. Prepending each line with the 'learn' message will train the Agent.

### 2.2.2 Scenarios

A Scenario is a sequence of labels presented to the agent, prepended by the 'generate' message. The Agent will always return a sequence of the same length as the number of items in the Scenario. In addition to labels, the dicy2.agent understands two special messages as part of a scenario:

- pick: This message, used in place of a label, gives the Agent a free choice for that member of the sequence. This is *not* a random choice, but one that will be optimized within the context of the scenario overall.

- free: the message 'free {int}' is the equivalent of sending {int} number of pick messages, giving the agent a wholly free choice of optimized sequences.

When the dicy2 agent receives a scenario, it returns optimized sequences of events from the Memory. The response of the agent is based not only on the current scenario presented to it, but the entire history of sequences that have been previously sent – you can think of the scenario as something you are constantly adding to. When the Agent receives a Scenario, Dicy2 combines machine learning and pattern recognition to locate sub-sequences in the Memory which have the highest amount of continuity with both the 'past' of the memory and the 'future' of the scenario. Scenarios can be generated by manually creating sets of labels, or automatically – either algorithmically, by machine listening to an audio input, or a combination of both.

For the scenario C A B B C C B A:

Dicy2 will look for a point in the memory where the past of the last event in the history (in this case the sequence B A B) is followed by the closest approximation to the scenario.
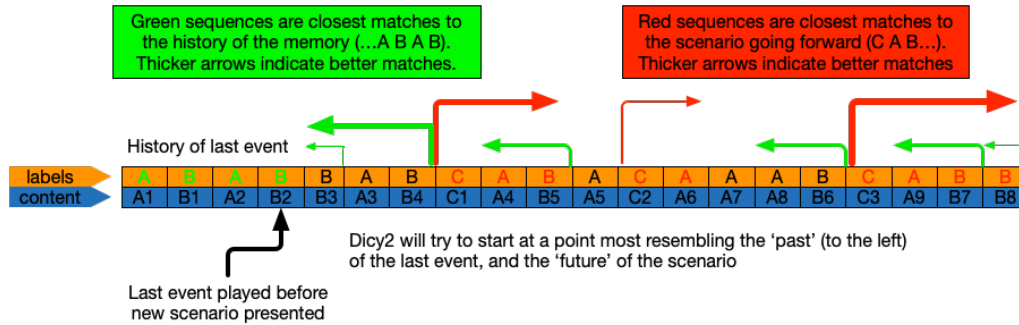
Green sequences are closest matches to the history of the memory (...A B A B). Thicker arrows indicate better matches.

Red sequences are closest matches to the scenario going forward (C A B...). Thicker arrows indicate better matches

History of last event

| labels | A | B | A | B | B | A | B | C | A | B | A | C | A | A | A | B | C | A | B | B |
| content | A1 | B1 | A2 | B2 | B3 | A3 | B4 | C1 | A4 | B5 | A5 | C2 | A6 | A7 | A8 | B6 | C3 | A9 | B7 | B8 |

Dicy2 will try to start at a point most resembling the 'past' (to the left) of the last event, and the 'future' of the scenario

Last event played before new scenario presented

Figure 4: How an Agent responds to a Scenario

## 2.3 Compositional approaches

### 2.3.1 Offline

When presenting scenarios to the Agent, there are two general overall approaches, offline and real-time. In an offline scenario, a structured sequence of labels is presented to the agent, and the returned content can be considered as a musical gesture, section, or even an entire composition. (See tutorial 00, offline_scenarios).

When the Agent outputs a response to a Scenario, it keeps a record of that response in its History. Each event is assigned a 'date', which is essentially a numbering of the order in which the events were generated. The Agent also has an internal clock, which can be used to point to any given date in the History. (It is important to note that the Agent has no internal sense of the passage of time, as the actual playback of events takes place outside of the Agent.)

When an Agent is presented with a Scenario, it is critical to keep in mind what we saw in Figure 3, that the Agent will optimize its response based on both the future of the Scenario *and* the past of the Memory at the point at which the scenario is generated, which will always be at the position of the date pointer. This means that Dicy2 will generate different responses depending on where in its History the date pointer is located. When presenting a new scenario, it is possible to do so at any date in the history, and the results will be optimal relative to what has already been written to the history at that
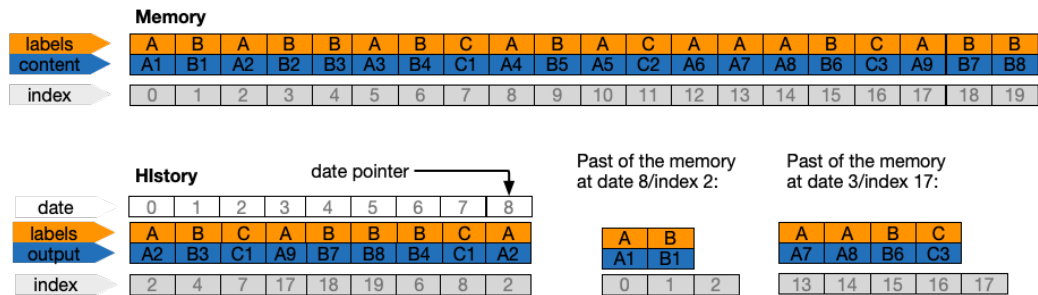
9

start date.



Figure 5: Memory and history

It is possible to change the start date at which you present a scenario by appending the '@absolute {int}' message to your Scenario. This will set the date at which the scenario will be presented, and will overwrite anything in the History from that point, up to the size of the Scenario itself. Because Dicy2's algorithm is dependent on what has previously been generated, changing the presentation date will change the sequence returned by your Scenario. (You can see this demonstrated in tutorial 00, scenario_timeframes.)
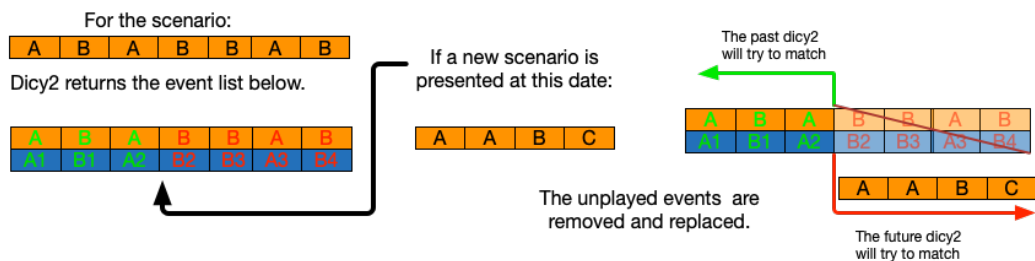


Figure 6: Overlapping time frames

### 2.3.2 Real-time

In many applications, Dicy2 is used to generate scenarios in real-time, from either MIDI input or analysis of incoming audio. In this case, it is possible that a new scenario will be presented before the playback engine has rendered all the items generated by the previous one. In order to have the new scenario properly optimized to the last played event to ensure continuity, it is necessary to update the Agent's internal clock so that it knows the date

10

in the history which was last played. In interactive applications, you can synchronize the clock as each event in the current response is rendered using the 'step' message, which increments the internal time pointer by 1. This guarantees that a new scenario will be correctly optimized. (See tutorial 0, real_time_scenarios.)
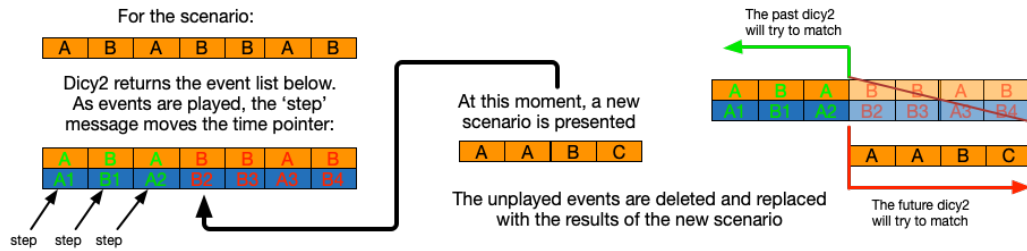


Figure 7: Using the step message to update the time pointer
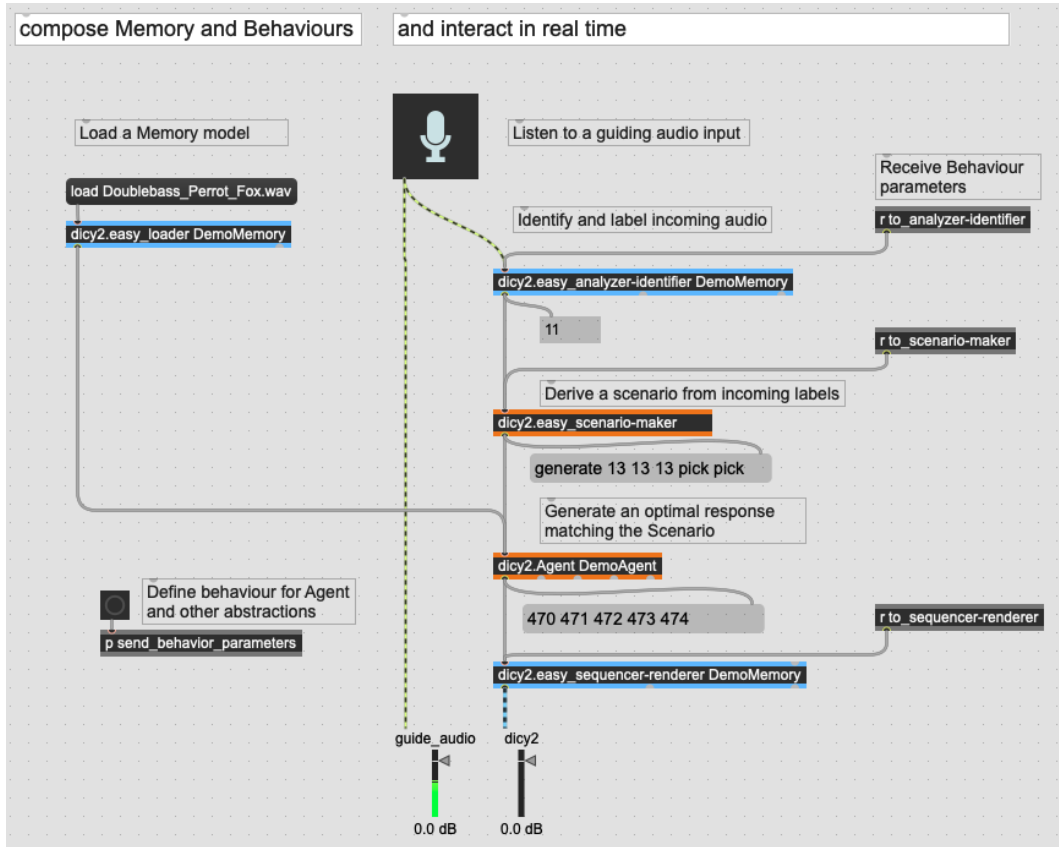
# 3 Audio Workflow



Figure 8: Dicy2: Basic setup for interactive audio

One of the most common use cases for Dicy2 is one in which segments from a prerecorded audio file is used as accompaniment to a live performer. In this situation, the audio file is segmented and analyzed using one or more audio descriptors, such as pitch, noisiness, etc., and each segment is labeled based on those descriptors. The Agent is then trained on the soundfile, creating a memory model based on the temporal sequence of the labels. Real-time audio, from either a live or prerecorded source, is then played into the system, and it is analyzed using the same descriptors as were used to define the memory, and labels are output corresponding to the labels in the original prerecorded file. These labels are then used to generate Scenarios, which in turn drive the Dicy2 agent to play back matching segments, optimized to the temporal structure of the original file.

The above illustration shows a typical Dicy2 audio setup. One can think

12

of the workflow as follows:

- Memory creation, which involves creating an audiofile to use as a basis for a Memory, choosing analysis and clustering parameters, and saving the Memory.

- Behaviour creation, setting parameters for the analyzer-identifier, the scenario-maker, the sequencer-render, and the Agent itself.

- Performance, when a guiding audio is sent to the analyzer-identifier to activate the entire system. Parameters of all the objects can of course be adjusted on the fly as part of the performance.

# 4   Audio and Machine Listening Abstractions

The Dicy2 library provides several utilities to simplify this process, most of them based on Ircam's MuBu library:

- dyci2.easy_memory-maker, for automatic segmentation, analysis, and labeling of an audio file.

- dyci2.easy_analyzer-identifier, for analyzing and classifying incoming audio.

- dyci2.easy_scenario-maker, for generating scenarios from incoming labels.

- dyci2.easy_sequencer-renderer, for audio playback.

- dyci2.easy_loader, for quick loading of saved Memories.

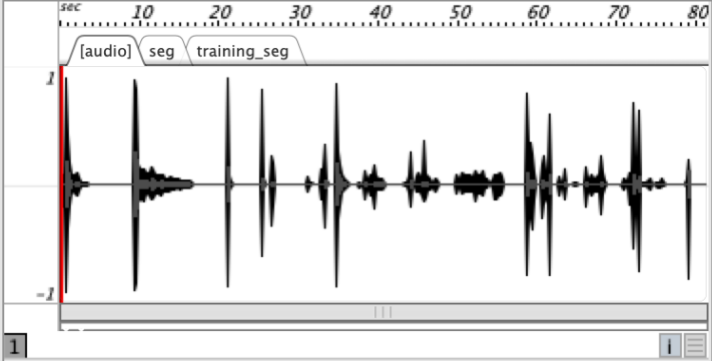We will now have a look at these tools in detail.

## 4.1   Using the Memory Maker

The dicy2.easy_memory-maker segments, classifies, and labels an audiofile, and saves this data for later use. It takes one mandatory argument: a Memory name.



Figure 9:  the dicy2.easy_memory-maker

Firstly, one loads an audiofile into the memory-maker, either by dragging a soundfile to the topmost pane, or using the 'read' button. Once this is done, the file is ready to be analyzed.

### 4.1.1 Segmentation and analysis

In the 'Audio segmentation and analysis' pane, you must select at least two descriptors: one to determine segmentation (which is typically Loudness, but for extremely legato material, other options may work better), and then one or more additional descriptors, which will determine the labeling of the segments. If you are looking to match melodic and harmonic color, the best starting point for analysis is Chroma. To reinforce the melodic aspect, add PitchClass, and to emphasize the timbre, add HarmonicSpectralCentroid.

Audio segmentation can be something of a dark art, and it is quite possible that the default segmentation setings will not work adequately with your audio material. Use the 'advanced segmentation parameters' button to fine-tune these settings if necessary. Details on how they operate can be found in the MuBu documentation help file for pipo.onseg.

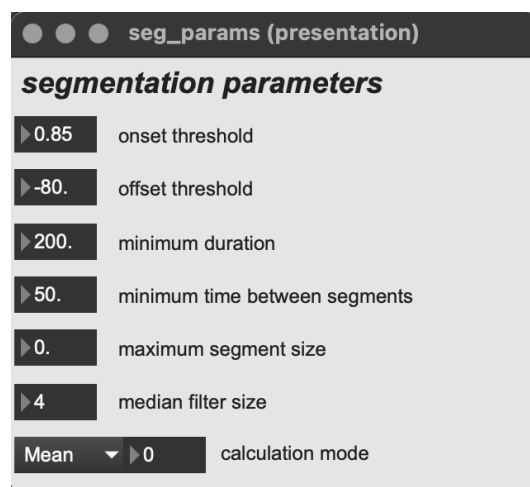Figure 10:  the dicy2.easy_memory-maker

Before completing the analysis, in the 'Classify segments into clusters' pane, there is one more important parameter: the number of classes. This parameter is very important: the greater the number of classes, the more detailed the analysis will be, but each class will have fewer members.

What this means in the context of Dicy2 is that there is a tradeoff between

15

the amount of variety the agent can generate and the number of classes. With a large class count, each class will have more individual character, but as the agent will have fewer choices in that class, the results will be more predictable. If there are fewer classes, the temporal model will have a greater number of options when responding to a scenario, but with too few there is less of a sense of distinction between classes. For the typical analysis mentioned above, Chroma alone works best with 20 classes, and Chroma+HarmonicSpectralCentroid with 30.

Once the segmentation module has sliced the soundfile into segments, and assigned each segment a set of descriptor values, this module looks at the descriptor values for each segment, and groups groups segments based on their relative distance in a data space whose dimensions are determined by the number of analysis descriptors, assigning each class a numerical label. These are the labels the dicy model will use when receiving and responding to queries.

This is often an iterative process, and once you have completed your analysis, it is worthwhile to examine your classes, which you can do by clicking on the 'inspect classes' button, which will open up the Class Inspector:
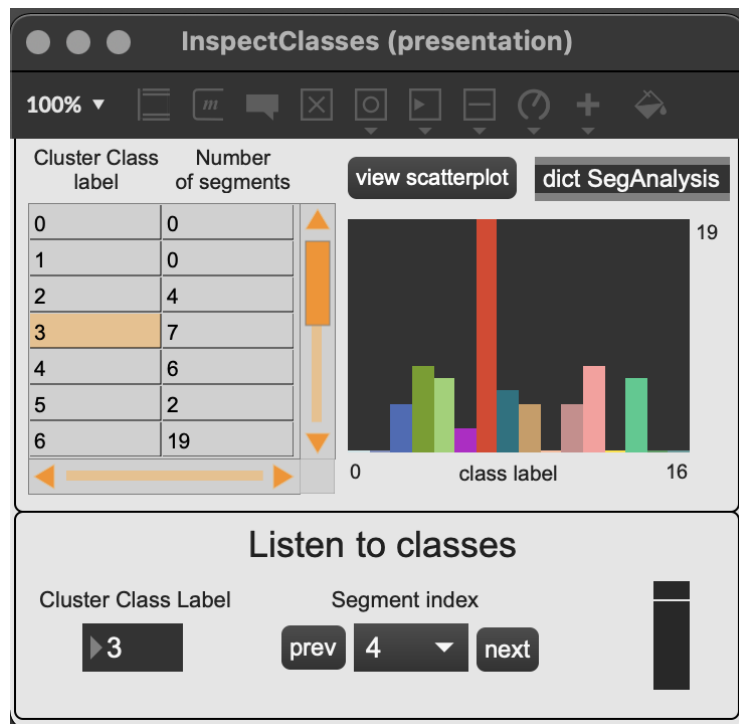


Figure 11: inspecting your classes

On the left is a jit.cellblock, listing the class labels, and the number of segments in each class – you will notice some classes are empty. The multislider on the right is a histogram which displays visually the number of members in each class. In the bottom pane one can listen to them by selecting the label, and choosing from the 'Segment index' menu. Listening to the classes can give you a sense of how much similarity and variation there is amongst the members of each class. You can also view the classification in a scatterplot:

Figure 12: classification scatterplot

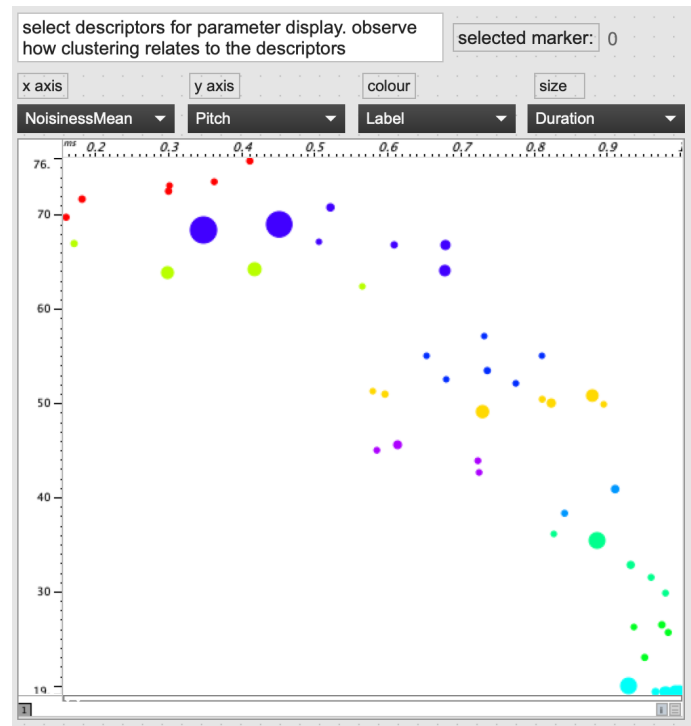Once you have tuned your classification and segmentation to your liking, you can train the the Agent to classification data to with the 'dump' button. (What the Agent actually learns is a sequence of label markerID pairs, designed for eventual playback by a mubu.concat object.) Finally, you can write the Memory to disk. When you do this, you will get two save dialogs in order. That is because an audio-based memory is comprised of three separate documents:

- Your original audio file.

- A .mubu file, which holds the segmentation and analysis data for your audio.

- A .json file, which is a Max dictionary. This stores the parameters used by the clustering algorithm, and will be used by the analyzer-identifier to identify and label incoming audio.

It is usually best practice to give all of these files the same name, with their respective extensions, as the dicy2.easy_loader utility is designed to handle identically named files.

Finally, you will notice that there is a field labeled 'memory name' in the bottom right corner. This is an argument which must be shared by all other easy abstractions which will be using this Memory. (It need not be the same as any Memory you create, as any Memory can be loaded into they system) In practice, this means:

- dyci2.easy_memory-maker

- dyci2.easy_analyzer-identifier

- dyci2.easy_loader

- dyci2.easy_sequencer-renderer

## 4.2   Using the analyzer-identifier

The dicy2.easy_analyzer-identifier listens to a stream of incoming audio, segments it, and using the descriptor and clustering information in the Memory's dictionary, classifies them with labels corresponding to those in the Memory. These labels are sent out of the left-hand outlet, in a manner determined by the timing mode (see below).
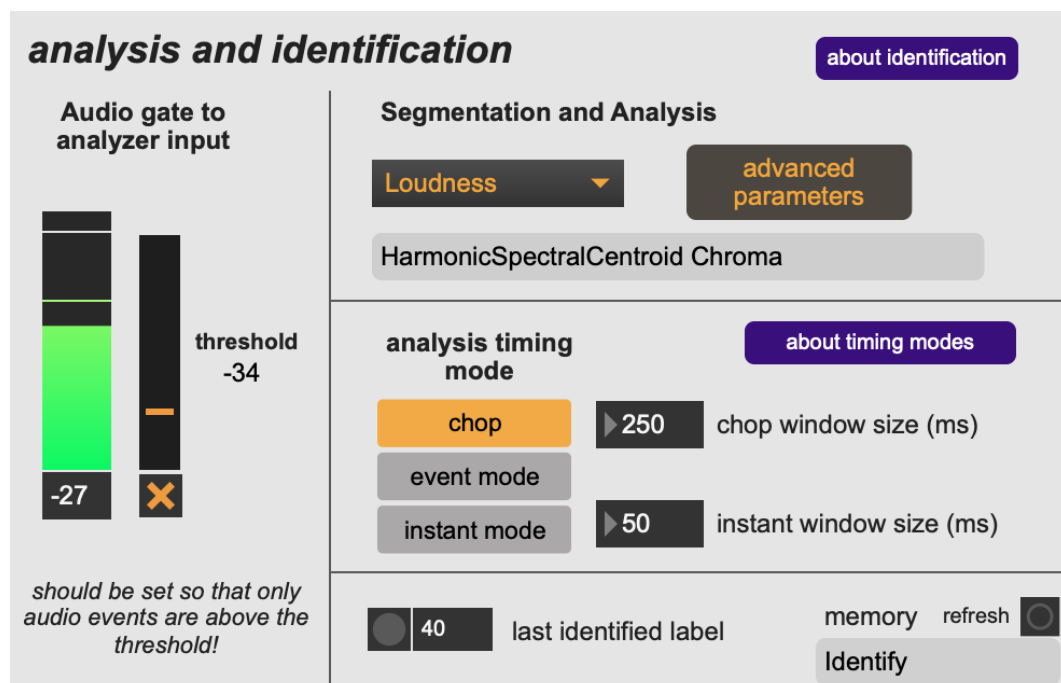


Figure 13:  The analyzer-identifier

### 4.2.1  Segmentation and Analysis

This section is set automatically to match the segmentation and analysis parameters of the Memory. It is theoretically possible to analyze with different descriptors from those in the Memory, but is not likely to give you good results. (For a sophisticated approach to this idea, see the Ex7_Playing_With_Mappings patch in the Performance Strategies folder.)

### 4.2.2  Gating

It is particularly important to apply a noise gate to your incoming audio stream, so that unwanted audio material is not sent to the analyzer for processing and labeling. This can be done in the Audio gate pane, and the toggle is illuminated when audio is passing through to the analyzer-identifier.

### 4.2.3  Timing Modes

The analysis timing modes determine the rate and method by which the analyzer-identifier identifies and labels incoming audio. There are three possible modes:

- Chop: the incoming audio is divided into equal segments, regardless of audio content. The segment size can be set with the 'chop window size' parameter.

- Event mode: incoming audio is segmented using the descriptor chosen in the 'segmentation and analysis' pane, and the label is output when te segment is completed.

- Instant mode: the analyzer looks at the incoming audio stream and analyzes a short time window to estimate the likely timing and label of the next event. The size of this analysis is set by the 'next window size' parameter.

Like the memory-maker, the analyzer-identifier has an option to fine-tune the segmentation parameters for incoming audio, if needed.

## 4.3  Using the scenario-maker



create scenarios from incoming labels

about scenario creation

**generation methods**

Repeat label
Start with label then pick
Mix label and pick
Last n labels
Mix last n labels and pick
free

**method parameters**

▶ 100    chance to output a scenario
         for each new label input (%)

         trig & output scenario

▶ 4      scenario length (n)

▶ 50     chance of 'label' in mix
         options (%)

output scenario (query to Agent):

40 40 40 pick

Figure 14:  The scenario-maker

As the analyzer-identifier only sends out one label at a time, it is useful to have a tool to generate a full scenario from a single label. Remember, the Agent works with time structures, and thus expects a Scenario to have several items – to send it a single label is meaningless. It is important to emphasize that you can devise any method you choose to generate scenarios from single labels, and the scenario-maker can be thought of as a starting point for more sophisticated methods of your own. The scenario-maker expects a single label as input. you can also send it a bang, which is treated as a 'pick' message.

### 4.3.1   Generation Methods

- repeat label: Just like it says: generates scenario length copies of the label.

- start with label then pick: The module will generate a sequence beginning with the last received label, and allow the agent to create an optimized sequence from there.

- Mix label and pick: randomly selects between the last label and the 'pick' message, which allows the agent to choose the optimized decision. The '% chance' number box determines what percentages of labels will be chosen.

- last n labels: Presents the last n labels received from the analyzer. the number of labels is determined by the 'scenario length' parameter.

- mix last n labels and pick: randomly selects between the list of last n labels and the 'pick' message. The '% chance' number box determines what percentages of labels will be chosen.

- free : same as n times pick for any incoming label or bang

### 4.3.2 Method Parameters

These settings determine aspects of the generation method's behaviour.

- chance of output: for each incoming label, determine the percentage liklihood that it will generate a scenario. Labels can come in very quickly, and in order for the renderer to complete playing the previous Scenario, it's often helpful to reduce the rate of new scenarios sent to the Agent.

- scenario length: the number of items sent out in response to a label (or bang) received by the scenario-maker.

- chance of 'label': the percentage chance that a given item will pass a label or a pick message. At 100%, no pick messages will be passed. This parameter only effects generation methods 1, 2, and 4 (zero-based).

## 4.4 Using the sequencer-renderer

It is important to keep in mind that the dicy2.agent sends out sequences of content in the form of lists, and the sequential playback of those lists is handled entirely outside of Dicy2. The dyci2.easy_sequencer-renderer is a utility for basic audio playback based on the mubu.concat  object. It receives a sequence of segments from the Agent, and plays them back in order.  It takes two arguments: the memory name, (which should be shared with other dicy2.easy abstractions as mentioned above), and an optional number of output channels. It has three outlets:

- Left: audio out through an mc multichannel output.

- Center: status messages, including number of remaining segments, the current marker ID, and information regarding duration of the the current event and the entire sequence.

- Right: sends a 'step' message after each event is played, designed to be connected to the Agent's internal clock pointer.

Like all of these utilities, the sequencer-renderer represents one possible approach to playback, and we encourage you to modify and elaborate these patchers, or develop your own entirely, to suit your own needs.
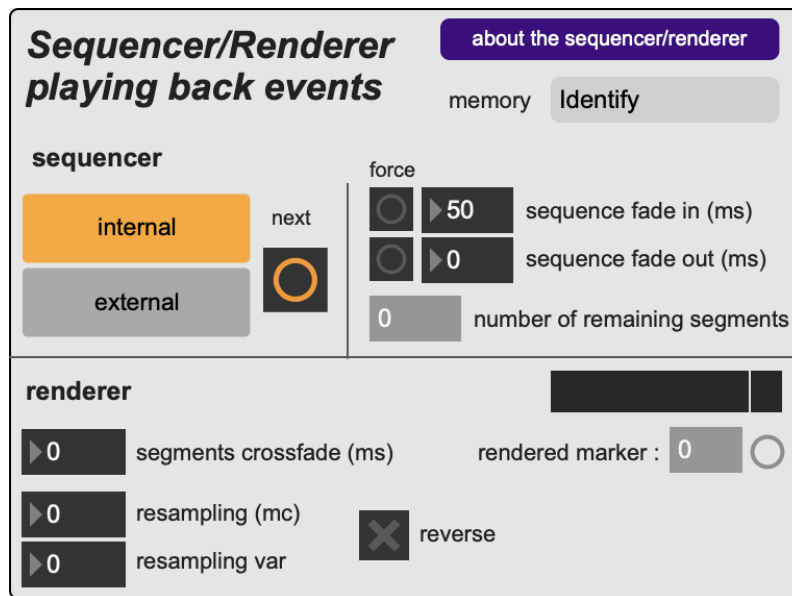


Figure 15: The sequencer-renderer

### 4.4.1 Sequencer

The Sequencer pane has two options:

- Internal: internal sequencing, based on duration of the segments. As soon as one is finished, the next segment in the list will start.

- External: segment triggering is controlled by bangs coming in to the right-hand inlet, so you can use an external clock of your own devising. A good example of quantizing audio output to a tempo grid can be found in the Ex5_Synchronizing patcher in the Performance Strategies folder.

### 4.4.2   fadein/out

These two values determine the fade in and out times of the entire sequence the sequencer-renderer has received. the fade in is timed in milliseconds, and the fade out by a number of events, so you can fade out over a certain number of events without having to calculate the duration of those events.

### 4.4.3   Renderer

This section exposes a few of the many options available to the mubu.concat object.

- Segments crossfade: The fadein/out time of an individual segment in ms.

- Resampling: Pitch transposition of the event, in +/- cents.

- Resampling var: Variation of the transposition in percent.

- Reverse: Play the segment backwards.

## 4.5   Using the Easy Loader

The dicy2.easy_loader is a simple abstraction for loading all the files required for an audio-based Memory.
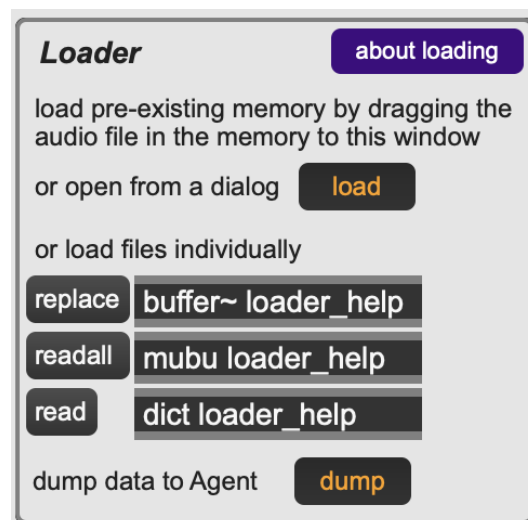


Figure 16:  The easy_loader

As we have mentioned, an audio-based Memory is comprised of three documents:

- A source audio file.

- A .mubu file containing analysis and segmentation data for the audio file.

- A .json file in Max dictionary format containing clustering and analysis data.

The easiest way to work with the easy_loader is to save all three documents with the same name, for example:

- SomeMemory.wav

- SomeMemory.mubu

- SomeMemory.json

The loader uses the audiofile (.wav, .aif, or .aiff) as the key to all three files. Dragging the audio file portion of the memory, or opening it with the 'load' button, will load the audio file, plus the associated MuBu and dictionary files, and automatically dump the analysis data to the Agent.

## 4.6    Behaviours

Since it it possible to load different Memories into the same set of audio objects on the fly, It can often be convenient to use remote messages to the various easy abstractions to wrap all the parameters into a single package – this is called a Behaviour. There is no official format for Behaviours, it is just a series of messages.
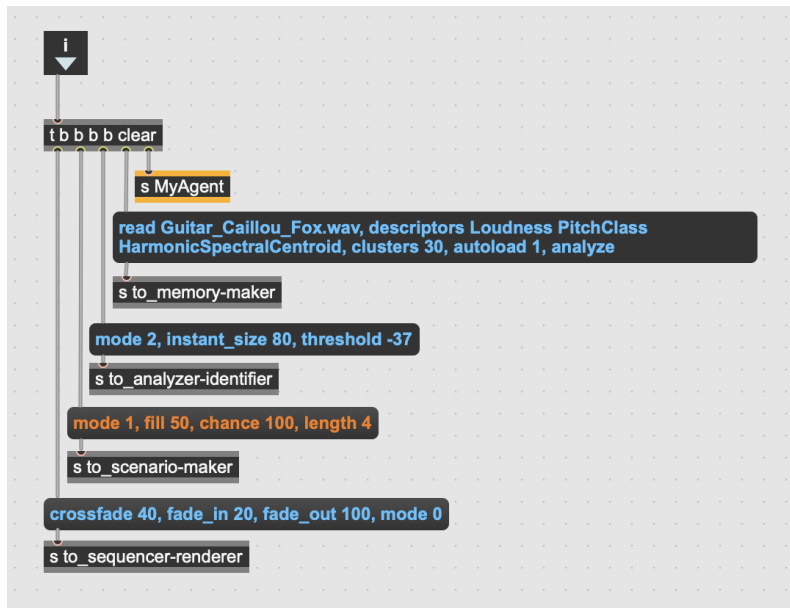
Figure 17: A typical Behaviour

In this particular example, the memory-maker creates a memory in real time using the parameters in the message box, and dumps the data to the agent via the analyze message. Message lists are then sent to the other easy abstractions to adjust their parameters. (Although this message creates a memory, one could easily create a Behaviour that simply loads a pre-existing Memory via the easy_loader.) Behaviours are thus a way to easily switch between 'presets' for an entire Dicy2 audio chain. You can see examples of Behaviours at work in the 'Ex2_Some_Behaviors' patch in the 'Performance Strategies' folder.

# 5   Further Exploration

Although this document has concentrated on real-time audio interactions, the basic Dicy2 workflow of developing a memory and passing scenarios can be applied to almost any set of temporally structured data, such as MIDI and text files, some instances of which you can find in the example patchers. You can also find more in-depth coverage of developing scenarios offline, as well as examples of using Agents to control other Agents, sharing Memories between Agents, and much more. We encourage you to examine the examples and the performance strategies carefully, as they can provide a springboard for further work of your own based on the Dicy2 system.

26

# 6   More about Dicy2

## 6.1   Some references

If using Dicy2, please quote **Nika, J., Muller, A., Borg, J., Ostrowski, M., & Assayag, G. (2022, December). Dicy2 for Max. Ircam. 2022** or
**Nika, J., Déguernel, K., Chemla, A., Vincent, E., & Assayag, G. (2017, October). Dyci2 agents: merging the" free"," reactive", and" scenario-based" music generation paradigms.  In International Computer Music Conference** (1). Additional article references are given in the "References" section (2; 3; 4; 5; 6).

- Video presentation about Dicy2 in French

- Video presentation about Dicy2 in English

- Some videos of collaborations with musicians using Dicy2 or its previous versions

## 6.2   Authors

Dicy2 is a library for Max and a plugin for Max for Live of the Ircam Musical Representations team, designed and developed by Jérôme Nika, Augustin Muller (Max library), Joakim Borg (Python generative engine / Gig RepMus API), and Matthew Ostrowski (tutorial patchers and videos, abstractions) in the framework of the projects ANR-DYCI2, ANR-MERCI, ERC-REACH directed by Gérard Assayag, and the UPI-CompAI Ircam project.

The audio use cases have been designed and developed with Diemo Schwarz and Riccardo Borghesi, and use the MuBu(7) and CatArt(8) environments of the ISMM team of Ircam.  Max4Live plugin by Manuel Poletti. Contributions / thanks: Serge Lemouton, Jean Bresson, Thibaut Carpentier, Georges Bloch, Mikhaïl Malt, Axel Chemla−Romeu-Santos, Vincent Cusson, Tommy Davis, Dionysios Papanicolaou, Greg Beller, Markus Noisternig.

## 6.3   Artistic collaborations

### 6.3.1   An instrument designed through artistic productions

Dicy2 integrates scientific and musical research results accumulated through productions and experiments with Rémi Fox, Steve Lehman, the Orchestre National de Jazz, Alexandros Markeas, Pascal Dusapin, Le Fresnoy - Studio

National des Arts Contemporains, Vir Andres Hera, Gaëtan Robillard, Benoît Delbecq, Jozef Dumoulin, Ashley Slater, Hervé Sellin, Rodolphe Burger, Marta Gentilucci... After having evolved research prototypes crystallizing the contributions of these various projects for several years, a collaborative work carried out during the year 2022 has led to the finalization of a release of Dicy2 as a plugin for Ableton Live and a library for Max.

### 6.3.2   Example and tutorial files

This distribution includes agents and sound files from past productions with our friends and collaborating musicians and composers who helped bring Dicy2 to life (courtesy of the artists). Please do not use these agents and files in any context other than these tutorials to respect their work and generosity.

**List of files**

- – *Doublebass_Perrot_Fox.wav*,
  - – *Guitar_Caillou_Fox.wav*,
  - – and *Voice_Daumergue_Fox.wav*

  were respectively recorded by Alex Perrot, Thomas Caillou, and Manu Daumergue during Rémi Fox's residency at Ircam for the concerts and first album of "C'est pour ça".

- – *Balafon_Lehman_ExMachina.wav*,
  - – and *SaxPlayingMode_Lehman_ExMachina.wav*

  were recorded by Steve Lehman for "Ex Machina" with Orchestre National de Jazz.

- *Texture_Maurin_ExMachina.wav* was recorded by Fred Maurin for "Ex Machina" with Orchestre National de Jazz.

- *Fox_Sax_1/2/3.aif* come from a performance of "C'est pour ça" at Ircam.

- *Piano_Markeas_Music-Of-Choices.wav* was recorded by Alexandros Markeas for "Music of Choices".

- *Soprano_Gentilucci.wav* was recorded by Marta Gentilucci during her residency "Female Singing Voice's Vibrato and Tremolo: Analysis, Mapping and Improvisation" at Ircam.

- *Nox3_LucidDreams.wav* comes from the song "Lucid Dreams" by Nox.3.

## 6.4 More

- Please write to `jerome.nika@ircam.fr` and `augustin.muller@ircam.fr` for any question, or to share with us your projects using Dicy2!

- Interested developers can check out the generative core of Dicy2, implemented as a Python library.

- License: GPL3

# References

[1] J. Nika, K. Déguernel, A. Chemla, E. Vincent, G. Assayag, *et al.*, "Dyci2 agents: merging the" free"," reactive", and" scenario-based" music generation paradigms," in *International computer music conference*, 2017.

[2] J. Nika, M. Chemillier, and G. Assayag, "Improtek: introducing scenarios into human-computer music improvisation," *Computers in Entertainment (CIE)*, vol. 14, no. 2, pp. 1–27, 2017.

[3] J. Nika, D. Bouche, J. Bresson, M. Chemillier, and G. Assayag, "Guided improvisation as dynamic calls to an offline model," in *Sound and Music Computing (SMC)*, 2015.

[4] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "Omax brothers: a dynamic yopology of agents for improvization learning," in *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pp. 125–132, 2006.

[5] J. Nika and J. Bresson, "Composing structured music generation processes with creative agents," in *2nd Joint Conference on AI Music Creativity (AIMC 2021)*, p. 12, 2021.

[6] T. Carsault, J. Nika, P. Esling, and G. Assayag, "Combining real-time extraction and prediction of musical chord progressions for creative applications," *Electronics*, vol. 10, no. 21, p. 2634, 2021.

[7] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, R. Borghesi, *et al.*, "Mubu and friends−assembling tools for content based real-time interactive audio processing in max/msp," in *ICMC*, 2009.

[8] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton, "Real-time corpus-based concatenative synthesis with catart," in *9th International Conference on Digital Audio Effects (DAFx)*, pp. 279−282, 2006.