Projet Sample Orchestrator 2
D4.4.3 Augmented Rendering Prototype final

Version	Date	Contributeurs	Validation	Contenu
V01	Février 2013	Laurent Bonnasse-Gahot (principal chercheur et développeur, rédaction), Benjamin Lévy, Gérard Assayag	Gérard Assayag, Hugues Vinet	Distribution prototype final et documentation.

Résumé:

Ce document présente la dernière version du prototype d'harmonisation/arrangement à la volée.

Présentation

SOMax est un agent autonome capable de générer du contenu musical en se basant sur une connaissance extraite d'un corpus pré-analysé ou d'un matériel directement appris à la volée. Au-delà des fonctionnalités présentes dans OMax, ce nouveau prototype, plutôt spécialisé en midi même s'il est capable d'audio, est capable de préserver une pulsation et de se synchroniser avec une entrée extérieure. La navigation dans une mémoire musicale donnée peut être guidée par différentes contraintes rythmiques, mélodiques et/ou harmoniques créant des situations d'accompagnement ou d'arrangement temps-réel. Un ensemble de commandes externes des principaux objets SOMax permet de piloter et de contrôler dynamiquement le comportement d'un ou de plusieurs agents ainsi que leurs interactions avec le musicien humain et avec les autres agents qui peuvent " jouer ensemble ". Ces commandes, également accessibles grâce à l'interface graphique de l'objet (voir Fig. 1), sont décrites dans les fichiers d'aide associés au prototype (maxhelp et maxref). Ces deux fichiers sont reproduits en annexe du présent document.
Quatre patches d'exemple brossent les diverses utilisations de l'objet somaxPlayer, cœur du prototype :
(1) Le patch *somaxConductor_midi_example.maxpat* (voir Fig. 2) permet d'utiliser SOMax avec un contrôleur MIDI (typiquement un clavier). En plus des informations MIDI directement disponibles, le programme est capable de calculer en temps réel la couleur harmonique (chromagramme) ainsi que d'extraire la pulsation du jeu du musicien.
(2) Le patch *somaxConductor_audio_example.maxpat* (voir Fig. 3) permet d'utiliser SOMax avec une entrée audio. Le programme est capable d'extraire la hauteur des notes dans un cas monophonique, de calculer en temps réel la couleur harmonique (chromagramme) ainsi que d'extraire la pulsation à partir du flux audio.
(3) Le patch *somaxConductor_recording_example.maxpat* (voir Fig. 3) présente la possibilité du logiciel d'apprendre son corpus au fur et à mesure qu'il écoute un musicien. Ce matériel est alors directement disponible à la machine pour jouer.
(4) Le patch *somaxConductor_twoAgents_example.maxpat* (voir Fig. 3) illustre la possibilité d'utiliser plusieurs agents somaxPlayer. Dans cet exemple, le second agent se synchronise à la pulsation du premier, et les deux agents s'écoutent mutuellement du point de vue de la contrainte harmonique.

Fonctionnalités

Principales fonctions réalisées :
- recombinaisons préservant la pulsation
- détection et suivi de tempo depuis une entrée audio ou midi
- synchronisation à une pulsation extérieure

– segmentation en phrases avec gestion des silences
– double utilisation de la transposition locale et globale
– adaptation à de nouvelles tonalités
– découverte de nouvelles relations à l'intérieur d'un corpus
– navigation dans un corpus sous différentes contraintes rythmiques, mélodiques et/ou harmoniques (fonction arrangement / harmonisation / chorus / alignées temporellement)
– écoute harmonique non symbolique (mémoire échoique, couleur texturale), permettant de s'adapter à n'importe quel contexte musical midi ou audio
– accompagnement, harmonisation d'une mélodie
– génération de mélodies/chorus avec écoute harmonique et rythmique
– apprentissage hors-ligne (scripts Matlab), notamment pour le traitement de corpus de grande taille, et possibilité de créer des bases directement dans le logiciel, à la volée, ou encore d'enrichir des bases existantes.
– interaction musicale multi-agent avec écoute mutuelle harmonique et rythmique
– commandes externes qui permettent de programmer et contrôler dynamiquement le comportement d'un agent

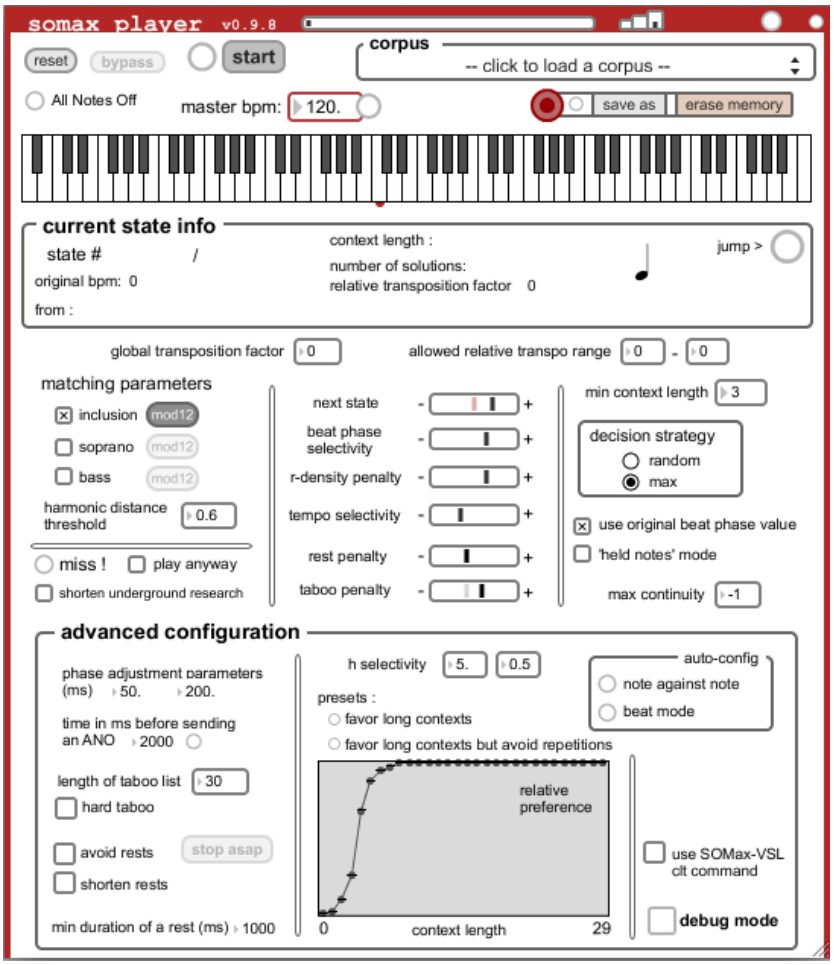


FIGURE 1 – User interface of the somaxPlayer object.

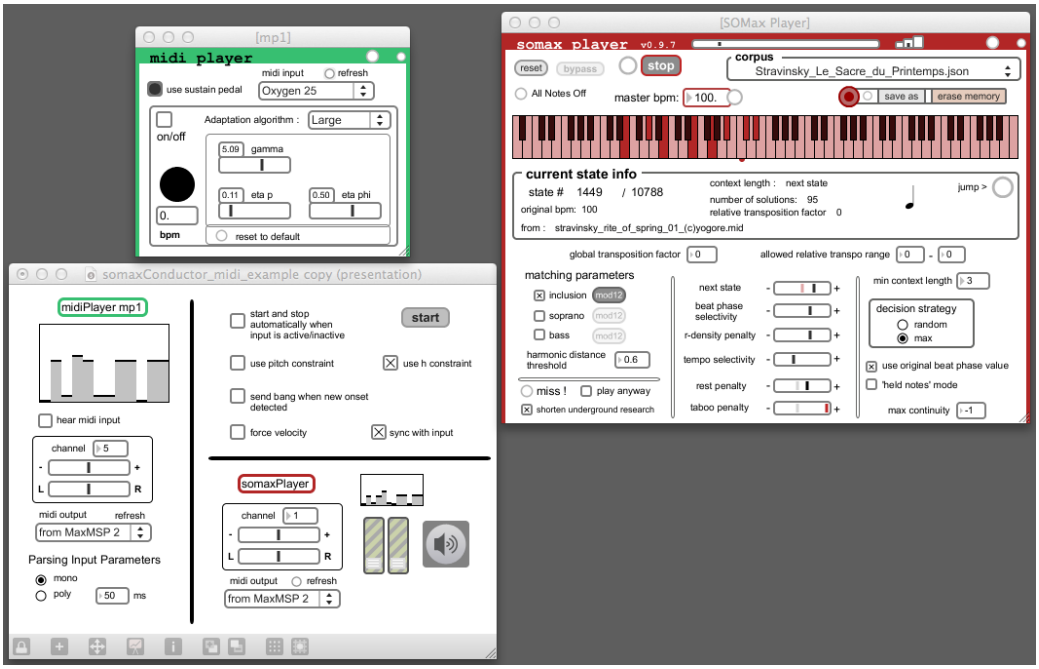


FIGURE 2 – somaxConductor_midi_example.maxpat

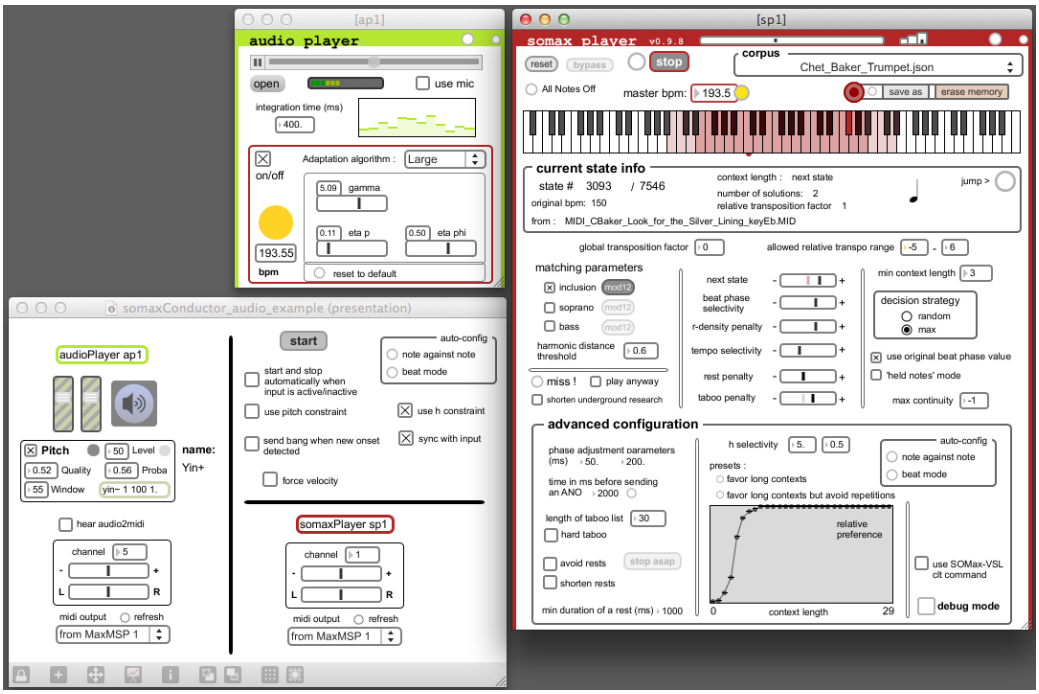


FIGURE 3 – somaxConductor_audio_example.maxpat

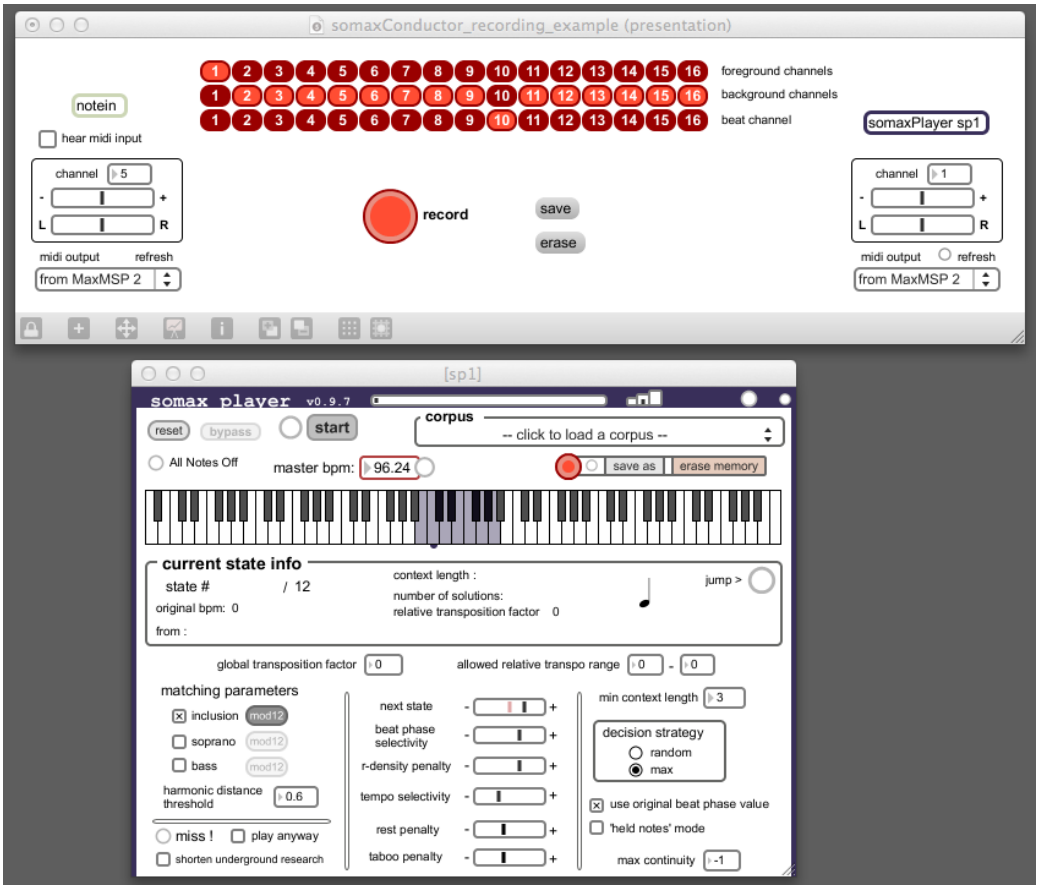


FIGURE 4 – somaxConductor_recording_example.maxpat

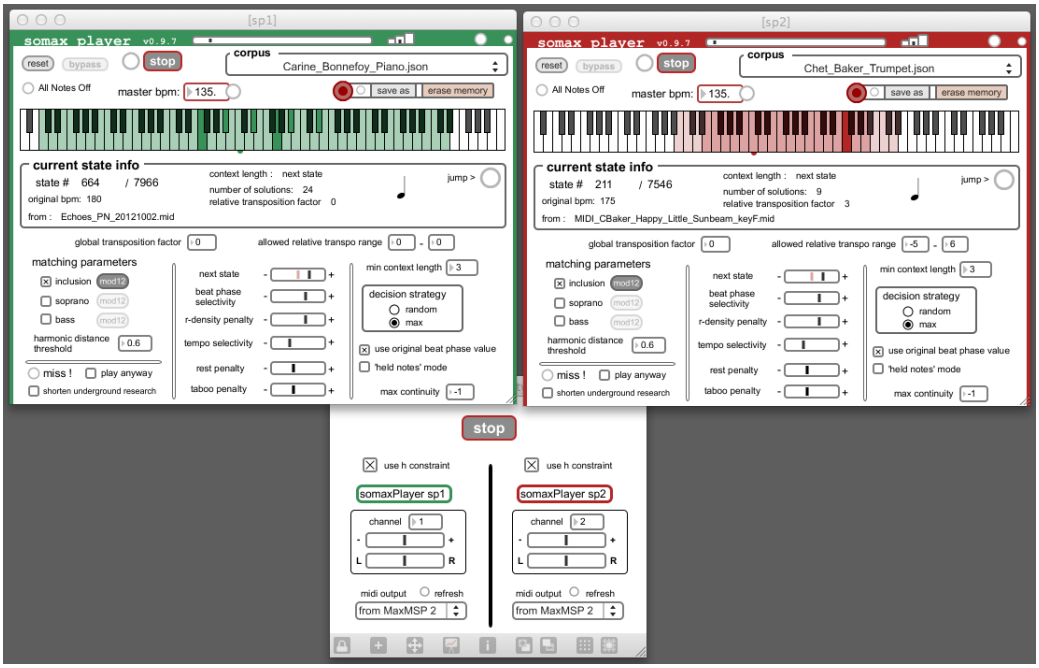


FIGURE 5 – somaxConductor_twoAgents_example.maxpat

somaxPlayer

Generates music using some active listening

open somaxPlayer reference

somaxPlayer is a musical agent that can generate musical content based on knowledge extracted from a pre-analysed corpus or from material that is learned directly on the fly. Developed within the context of the ANR project SOR2, SOMax is part of the OMax project (http://repmus.ircam.fr/omax/home). In addition to OMax's features, it is able to keep a pulse and synchronize with an external input. Navigation within the given musical memory can be guided through different rhythmic, pitch and/or harmonic constraints. Recombinations possibilities are also strongly increased by the use of transposition.

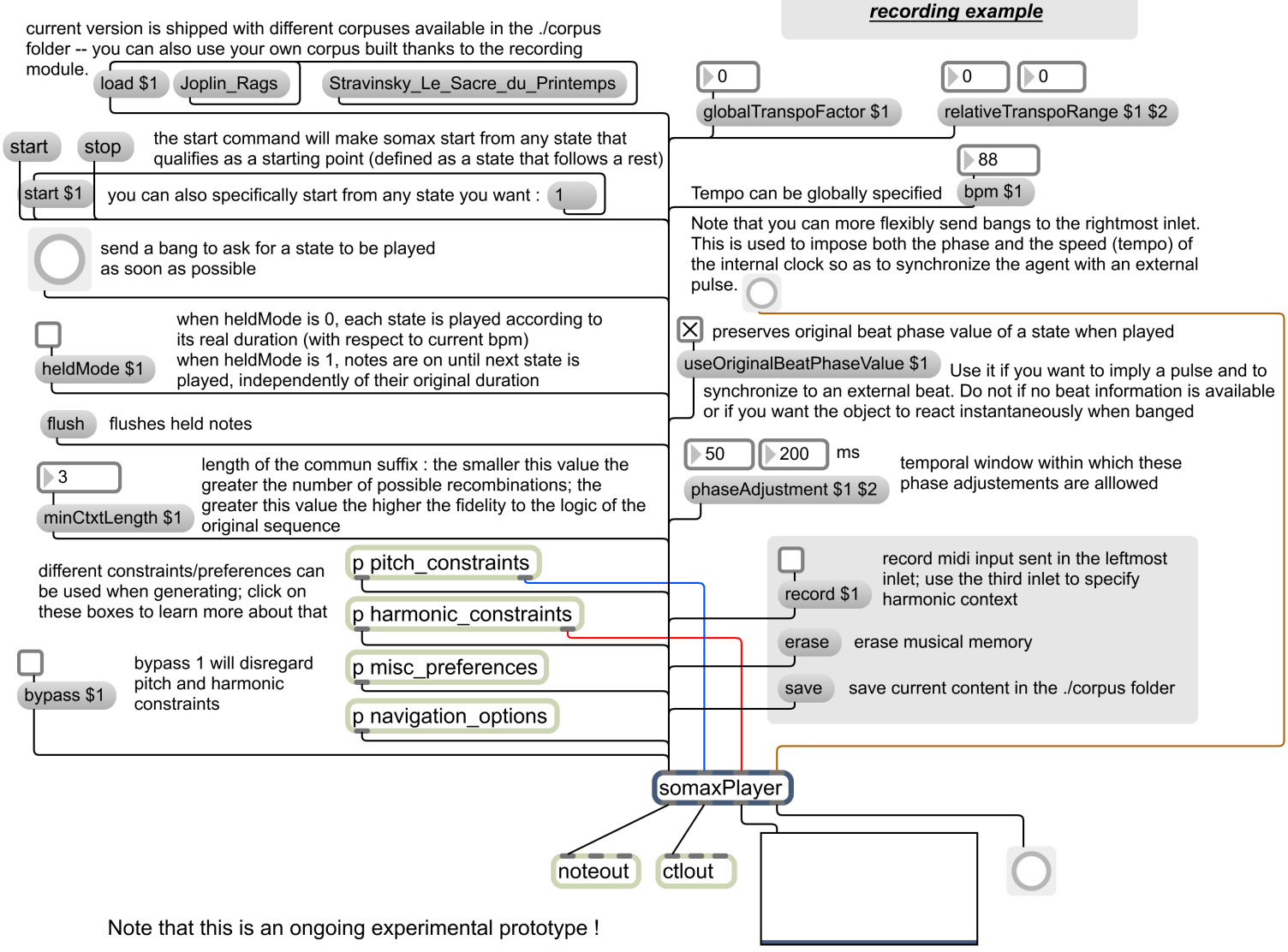
See Also:

Objects:

This help file briefly introduces the main external commands. Example patches illustrate basic ways of using them. Note that clicking on the object reveals a user interface that you can also use in order to change the different parameters. These external commands make it possible to program and dynamically control the behavior of the agent.

a few example patches

midi example
audio example
two agents example
recording example



Note that this is an ongoing experimental prototype !

Questions, comments, suggestions should be sent to : `laurent.bonnasse-gahot@ircam.fr`

somaxPlayer

Generates music using some active listening

Description

somaxPlayer is a musical agent that can generate musical content based on knowledge extracted from a pre-analysed corpus or from material that is learned directly on the fly. Developed within the context of the ANR project SOR2, SOMax is part of the OMax project (http://repmus.ircam.fr/omax/home). In addition to OMax's features, it is able to keep a pulse and synchronize with an external input. Navigation within the given musical memory can be guided through different rhythmic, pitch and/or harmonic constraints. Recombinations possibilities are also strongly increased by the use of transposition.

Arguments

Name	*Type*	*Opt*	*Description*
name	symbol	opt	Sets the name of the agent.

Messages

bang		In left inlet: Asks for a new state to be played as soon as possible. In right inlet: Imposes both the phase and the speed (tempo) of the internal clock of the agent so as to synchronize it with an external pulse.
bassMatching	mode [int]	In left inlet: Selects bass matching mode. 0: no constraint, 1: the lowest note of the state must match the lowest note of the list of pitches, 2: same, using pitch classes (ie modulo 12)
beatPhaseSel	value [int]	In left inlet: Sets beat phase selectivity, ie how much you want the chosen state to have an original beat phase value that corresponds to current beat phase value. Useful to keep a pulse while preserving micro-timing details; also useful if you want to try and capture interactions between melody or harmony and rhythm; 0: not al all; 1: a little; 2: quite a bit; 3: a lot
bpm	value [int]	In left inlet: Sets bpm (beats per minute).
bypass	off/on (0 or 1) [int]	In left inlet: Toggles bypass mode on and off. When bypass is 1, somaxPlayer disregard any pitch or harmonic constraints.
color	[float]	In left inlet: Sets somaxPlayer color in RGBA format (default is 0.7 0.15 0.15 1.0).
decisionStrategy	mode [int]	In left inlet: Selects decision strategy mode. 0: final state is chosen randomly among all satisfying states with a probability proportional to its evaluation; 1: same but considering only the best satisfying states (then watch out for loops !)
erase		In left inlet: Erases musical memory.
float	chromagram [float]	In third inlet: Specifies harmonic context used for annotation when recording, and harmonic constraint when playing. This harmonic vector consists in a list of 12 float values (chromagram), specifying the relative energy of each pitch class (from C to B) within the musical content under consideration. Note that chroma can be computed in real time from midi content (look at the midi2chroma object), or can be

		directly extracted from audio (look at the audio2chroma object).
flush		In left inlet: Flushes held notes.
globalTranspoFactor	semitones [int]	In left inlet: Transposes musical memory by n semitones (up for positive values of n and down for negative ones).
hSelectivity	slope [float] inflection point [float]	In left inlet: Sets parameters of the function that converts similarity between harmonic contexts into a preference of being chosen. Similarity (Pearson correlation) is transformed to a weight through a hyperbolic tangent; these two parameters correspond respectively to the slope and the inflection point of this function.
hThreshold	value [float]	In left inlet: Sets minimal similarity requirement. Similarity (Pearson correlation) between the chroma used for constraint (entered in the third inlet) and the one in memory (harmonic context) has to lie above this threshold.
heldMode	off/on (0 or 1) [int]	In left inlet: Toggles held notes mode on and off. When heldMode is 0, each state is played according to its real duration (relative to current bpm); when heldMode is 1, notes are on until next state is played, independently of their original duration.
int		In left inlet: Sends MIDI notes that you want to record and use as musical material. MIDI data should be sent as a list of three int values: pitch, velocity, and channel. In second inlet: Specifies pitch constraints as a list of MIDI pitches. -1 value indicates no constraint.
jump		In left inlet: the jump command will make somaxPlayer `jump' (ie go to any other place within the considered musical memory other than the original continuation) as soon as possible.
load	name [symbol]	In left inlet: Loads corresponding corpus.
maxContinuity	value [int]	In left inlet: Forces somaxPlayer to jump (as soon as possible) after n consecutive states; -1 means no such constraint.
minCtxtLength	value [int]	In left inlet: Specifies minimal length of the commun suffix underlying jumps within the musical memory. The smaller this value the greater the number of possible recombinations; the greater this value the higher the fidelity to the logic of the original sequence.
nextState	value [int]	In left inlet: Sets next state probability (ie reading original material vs. recombining it) 0: other things being equal, next state is as probable as states with preferred context length; 1: next state is more favored; 2: next state is much more favored; -1: next state is less favored; -2: next state is much less favored;
phaseAdjustment	before [int] after [int]	In left inlet: Specifies temporal window (in ms) within which phase adjustments are allowed.
pitchMatching	mode [int]	In left inlet: Selects pitch matching mode. 0: no constraint, 1: played state must contain (or be contained in) the list of pitches, 2: same, using pitch classes (ie modulo 12)

playAnyway	off/on (0 or 1) [int]	In left inlet: Activates playAnyway mode. When constraints are not fulfilled, you can decide to play anyway, the state being chosen according to your preferences.
rdensSel	value [int]	In left inlet: Sets rhythmic density selectivity, ie how much you want the chosen state to have a rhythmic density (defined as the number of note onsets within the last 1000 ms) that corresponds to the current one. 0: not al all; 1: a little; 2: quite a bit; 3: a lot
record		In left inlet: Records MIDI input sent in the left inlet (using harmonic context specified in third inlet and pulse information sent in right inlet). Useful to create a new musical memory on the fly or to enrich an existing corpus.
relativeTranspoRange	lowest interval (-11 to 0) [int] highest interval (0 to 11) [int]	In left inlet: The command relativeTranspoRange followed by two int sets the range of allowed internal transpositions (relative to the current global transposition factor). The two values represent respectively the lowest and highest interval in semitones. This makes it possible to find links within the musical memory based on patterns that are identical up to some transposition factor. somaxPlayer will automatically transpose musical memory so as to keep coherence with current musical flow while adapting to new external situations (such as harmonic constraints).
restPenalty	mode [int]	In left inlet: Sets the way rests are favored/penalized while navigating the musical memory. -2: rest states are much favored -1: rest states are favored 0: other things being equal, a rest state is as probable as any other state 1: rest states are penalized 2: rest states are much penalized 3: rest states are not allowed
save		In left inlet: Saves current memory into files (available in the ./corpus folder). It can be reused later thanks to the load command.
shortenRests	off/on (0 or 1) [int]	In left inlet: Toggles shortenRests mode on and off. When activated, this mode makes rests end on next following beat.
shortenUndergroundResearch	off/on (0 or 1) [int]	In left inlet: Activates the shortenUndergroundResearch mode. When constraints are not fulfilled, you can decide to accelerate the silent undeground navigation, so as to arrive more rapidly to a state that satisfies the constraints (in that case, playAnyway should then be 0).
sopranoMatching	mode [int]	In left inlet: Selects soprano matching mode. 0: no constraint, 1: the highest note of the state must match the highest note of the list of pitches, 2: same, using pitch classes (ie modulo 12)
start		In left inlet: Starts somaxPlayer from any state that qualifies as a starting point (defined as a state that follows a rest).
start	state [int]	In left inlet: Starts somaxPlayer from a specific state.
stop		In left inlet: Stops somaxPlayer immediately and flushes held notes.
tabooLength	value [int]	In left inlet: Sets the size of the taboo list.
tabooPenalty	mode [int]	In left inlet: Sets the way taboo states (the n states that were

		just played) are favored/penalized while navigating the musical memory. Penalizing those states makes it possible to avoid loops and explore in a more thorough way the memory. Conversely you might want to create loops by favoring states that were played in a recent past. -2: taboo states are much favored -1: taboo states are favored 0: other things being equal, a taboo state is as probable as any other state 1: taboo states are penalized 2: taboo states are much penalized 3: taboo states are not allowed
tempoSel	value [int]	In left inlet: Sets tempo selectivity, ie how much you want the chosen state to have an original tempo that corresponds to current tempo. 0: not al all; 1: a little; 2: quite a bit; 3: a lot
useOriginalBeatPhaseValue	off/on (0 or 1) [int]	In left inlet: When activated, this mode tries to preserve original beat phase value of a state.
winSize	value [int]	In left inlet: Changes the size of the user interface window. 0, small; 1, medium (default); 2, big;

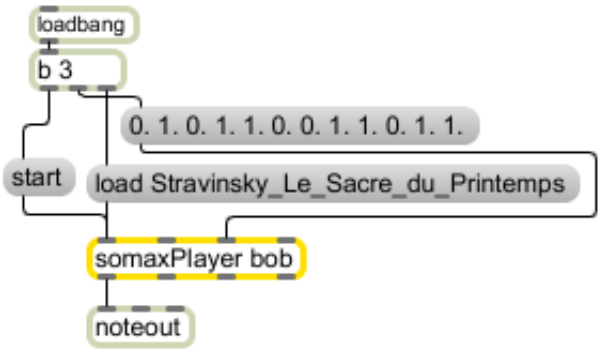
[Information for box attributes common to all objects]

Output

bang: Out right inlet: Internal pulse of the agent.

list: Out left outlet: MIDI note message that consists of a list of three int values: pitch, velocity and channel.
Out 2nd outlet: MIDI control message that consists of a list of three int values: controller value, number and channel.
Out 3nd outlet: Current harmonic context (chromagram of 12 float values).

Examples



This small example should result in somaxPlayer playing around with the famous texture of Les Augures printaniers.

See Also

Name	*Description*
somaxConductor	