Algorithmic Interactive Music on the Web Browser

201071767

ABSTRACT

Algorithmic Interactive Music is not yet very common, and the Web Browser is the media which can best help it in becoming mainstream. The project consists in a Web Music Application dedicated to the generation and manipulation of Musical structures in real time, to be used in the context of an Improvisation or Live Performance. A Background on Algorithmic Music, as well as a comparison between my Project and other existing works, is given. The Application is meant to foster the Musical creativity of the user by allowing low and abstracted level of control over creation and manipulation of Musical patterns. The Generative Algorithms produce plausible and engaging melodies; nevertheless, the focus of the Application is on how the user can actively and creatively manipulate them.

1. BACKGROUND

1.1 Overview of Algorithmic Music

In order to provide an overview of the wide domain of Algorithmic Music, a distinction of convenience has to be made between software Algorithmic Music, where the user enjoys Music produced by a software, and traditional Algorithmic Music where the composer exploits mathematical procedures as tools or techniques for Music Composition. While software Algorithmic Music necessarily implies mathematical processes for Composition, traditional Algorithmic Music has had a long history started long before the arrival of Electronics and computers.

1.2 Software Algorithmic Music

Software Algorithmic Music can be broadly subdivided into 3 subcategories; Generative Music, Reactive Music and Interactive Music. For the scope of this Project, we are concerned about Generative and Interactive Music only.

Generative Music involves no interaction between the listener and the algorithm responsible for Music Composition. At most, the listener can start/stop the music, without any mean or tool to control and/or bias the system [1:628-629]. Examples of Generative Music are Brian Eno's *Generative Music 1* (1996) [1:631] and Alex Bainter's *Generative.fm* (2019) [2].

Interactive Music assumes a peculiar position inside Algorithmic Music, since it expects the user -who is simultaneously listener, performer and composer- to directly manipulate the algorithmic software and to assume an active role in Musical Composition; interactivity is achieved with the help of a User Interface, broadly defined as any mean of converting several kinds of user input gestures into data [1:628-629]. Among examples of Algorithmic Interactive music Systems we find the Procedural Sequencer Eurorack module *Tuesday* [3] and the software *TidalCycles* [4] by Alex McLean.

Furthermore, Hybrid Systems, which implement characteristics typical of both Generative and Interactive Algorithmic Music, exist; examples are the web-based application *In C* by Tero Parviainen (2017) [5] and the mobile applications *Bloom* (2008), *Trope* (2009) and *Scape* (2012) by Brian Eno and Peter Chilvers [6].

1.3 Traditional Algorithmic Music

From J.S. Bach's fugues (e.g. use of inversion, retrograde, augmentation), to the 12 tones rows permutations of Serialism, to Terry Riley's minimalist, aleatoric and improvisational approach of *In C* (1964) , history is full of examples where Music Composition is expressed as a process which implies mathematical generation and manipulation of musical patterns [7:248-249].

For the scope of this Project it is worth referring, in addition to aforementioned Brian Eno, to the activity of formal minimalist composer Tom Johnson, especially the works *Rational Melodies* (1982) [8] and *La Vie est si courte* (1998) [9]. Even more important is Laurie Spiegel's paper *Manipulations of Musical patterns* (1981) [10]. A Musical Pattern can be defined as “relationships between sonic events [...] taking place in time” [7:245], and patterning is the act for which a composer symbolize and manipulate source material by adopting 12 different transformational procedures; transposition, reversal, rotation, phase offset, rescaling, interpolation, extrapolation, fragmentation, substitution, combination, sequencing and repetition [10].

2. INTRODUCTION

2.1 Project requirements and overview

The project consists in a Web Music Application for Algorithmic live Music performances and/or improvisations (see video “`Free improvisation.mov`”). In order to foster the user’s Musical inspiration and creativity, it allows the generation, concatenation, elaboration and combination of Musical structures. The main aim of the Project is the creation of a Hybrid Algorithmic Music System which uses features typical of both Generative (contemplative approach) and Interactive (explorative approach) Music Systems and which allows the user in-depth control over the possible manipulations of Musical patterns, without the need for he/she to be a coder or programmer. In fact, “the process of creating music involves not only the ability to design [...] patterns of sound, but a working knowledge of all the processes of transformation which can aesthetically be applied to them” [10]. The main motivation underpinning the Project is that, even though a community of Musicians/Programmers is slowly gathering around this very appealing opportunities, there is currently still a lack of such type of web-based Music Applications and of any kind of standard commonly-accepted reference for them.

The Project’s algorithms for Musical generation tend to produce plausible and engaging melodies; nevertheless, the focus of the Application is on how the user can manipulates these tunes, rather than on generating human-like melodies, emulating composer-specific compositional techniques or phrases typical of any particular Musical style. A constrained random generation Algorithm is used for generating melodic content, while a stochastic binary subdivision Algorithm [11:2] is used for generating rhythmic durations. The software libraries used are; Tone.js for high-level Web Audio API abstraction (synthesisers, effects and scheduling) [12], Chance.js for constrained, weighted and seedable generation of pseudo-random numbers [13], Tonal.js for Musical scales data abstraction [14]; the Software Architecture framework used for web-based User Interface is Angular 7 [15].

2.2 Comparisons with other Software Systems

2.2.1 *Web-based / Multiplatform*

Web-based Applications tout-court have two important qualities comparing to other common platforms. They are cross-platform; this means that publication is instantaneous, user access is easy and, above all, mass distribution is simplified and so is selling for the programmer/artist and exploring for the audience [1:638-639]. Additionally, they are immune from the problem of “platform segmentation” -for which Designers and Programmers require the use of different and specific programming languages and paradigms for Operative System of different platforms, which drastically increases time and cost [1:637]-.

Generative.fm [2], being purely Generative, does not involve user interaction. Furthermore, it uses distinct compositional techniques for each piece; differently, my Project is aimed to offer the user general purpose compositional tools, not restricted to or biased by any particular Musical style.

The Interactive Music App *In C* by Tero Parviainen [5], technically an App for playing one’s own unique version of Terry Riley’s *In C*, reveals a very simple User Interface while giving the user control over combination of patterns between different instruments only, with no way of regulating the generation or manipulation of patterns. My Project, by implementing a more sophisticated User Interface, allows what just described.

2.2.2 *Mobile applications*

While affected by the "platform segmentation" issue [1:637], mobile Algorithmic Music applications were the first ones to merge characteristics from both Generative and Interactive Music. *Scape* [16] uses a particularly simple yet very effective and eloquent User Interface which implements the compromise between complexity of control and ease of use by allowing the user to experiment with already made material while reserves to planned, precise and meticulous composition a secondary and less important place. In fact, since the expressive and sonic behaviour of musical instruments is limited and decided in advance by Eno and Chilver, it may be said that the app's authors themself -and not the user- are the real composers of the Music, while the user is more an "arranger" [1:632-633]. Also, the behaviour of instruments and interaction between them, determined by a set of "rules", remains "behind the scenes" [16] and is hidden from the user. Again my System, by implementing a less abstracted User Interface, gives no algorithmic control over the synthesis parameters of the various instruments, but in turn allows more control over the low-level generation and manipulation of musical compounds.

2.2.3 *Live coding*

The programming language *TidalCycles* allows a very deep level of control over both generation and manipulation of musical patterns, but the user needs to be a coder (or at least to have some programming experience) in order to use its command-line interface. The User Interface of my Project, in spite of requiring the user to have a minimum Musical background, allows him/her to achieve a similar level of control as *TidalCycles* without he/she to have coding experience.

2.2.4 Eurorack Modules / *Embedded Systems*

While having no manipulation capabilities, the Eurorack module *Tuesday* is capable of generating very interesting outcomes by giving the user the possibility of independently control the algorithms for melody and rhythm generation. Additionally, the System is deterministic; provided that other parameters do not change, the same melodic or rhythmic algorithm will, in fact, always produce the same Musical patterns when set to the same value. Furthermore, melody density can be controlled by selecting the probability of events to be played as notes, rather than muted as rests.

Even though not in an Hardware System domain, these three fundamental low-level features are also implemented in my System. The independent melody and rhythm generation has been further extended to independent melody and rhythm manipulation; the deterministic approach has been implemented by generating recallable pseudo-random number series, where a random number generator always produces the same values when fed with the same seed [13].

2.3 **Comparisons with traditional Algorithmic Music**

2.3.1 *Low-level Compositional strategies*

Many of the possible low-level manipulations of my System are inspired by Laurie Spiegel's *Manipulations of Musical patterns* [10]. I hereby explain them as applied to my System (see also video "`Basic functions and Manipulations of Musical patterns.mp4`").

TRANSPOSITION - Add/subtract an offset to/from a parameter. For example, the following Musical elaborations can be made; melodic progressions (e.g. `phrase -> octave offset/melodic offset`), rhythmic augmentation or diminution (`phrase -> rhythmic vertical offset`), or playing a part at a different tempo (`global -> rate`).

RESCALING - Expansion or contraction of minimum and maximum ranges for constrained random melodic and/or rhythmic generation (e.g. `global data -> pitches range`, `global data -> rhythms vertical/horizontal range`). Providing the same melodic/rhythmic seed, the software generates respectively the same pitches and/or rhythmic durations; hence, in the generated Musical data, "Distances are changed, but not ratios" [10].

REPETITION, SEQUENCING, COMBINATION - A `phrase` can be cloned from another `phrase`. A `pattern` can be cloned from another `pattern` contained either in the same `phrase` or in a different `phrase`. Furthermore, the `loop` facility allows the user to indefinitely loop the phrase or combination of series of phrases whose number has been entered in `phrases arrangement`. Every time the `generate` button is pressed, changes can be heard on beginning of next loop.

SUBSTITUTION - It is similar to sequencing, and can happen at a high level, by substituting, inside `phrases arrangement`, the position of a `phrase` with the position of another phrase, or low-level, by substituting a `pattern` with another pattern, belonging either to the same phrase or to another phrase.

FRAGMENTATION - Changing `number of events in pattern` without changing rhythmic or melodic seeds. This makes each pattern a fragment of a stream of pitches or rhythms, from which only some, in crescent order from the first, are chosen.

PHASE OFFSET - Two or more `parts'` loops played by two or more instruments are synced together via the main transport but, if they have different lengths, they will not start always at the same time. Furthermore, two or more parts can be played at a different `playback speed`.

EXTRAPOLATION/INTERPOLATION - To change the number of notes in the same scale, that is changing `scale type` (e.g. from `Major pentatonic` to `Major` or vice versa). By maintaining the same melodic seed, the software will produce a melody with the same shape and direction, but with different notes.

2.3.1 *High level Compositional strategies*

Because of its general-purpose orientation, openendness and versatility, the low-level patterns manipulations afforded by the System *can* be modelled and adapted by the user, at his/her own taste, to common higher-level compositional strategies which may or may not be the ones used by famous Composers. Strategies which involve clear mathematical processes or algorithms are easier to model and reproduce.

For the sake of presenting a use case, hence, I hereby use my System in order to produce example Compositions using algorithms by Tom Johnson. *Rational melody XV* [8] is about self-replicating melodies; after an entire phrase of *k* notes is played, the same phrase is re-proposed by playing a note and skipping the next one, then the same phrase is proposed again by playing a note and skipping the next two, so on and so forth until all phrases have *k* notes (see also video `"Tom Johnson, use case n. 1"`). Finally, in *La vie est si courte* [9] different instruments play the same melody at half or double speed, process which in my System can be represented by changing the `rhythmic vertical offset` parameter (see also video `"Tom Johnson, use case n. 2"`).

3. SYSTEM DEVELOPMENT

3.1 System overview

In terms of Software Architecture, each `instrument` is an Angular 7 component, composed by the following files; a .ts (Typescript, subtype of Javascript, responsible for the Software logic and functions implementations), .html (Hyper Text Markup Language, describes the structure and content of the Web Page) and .css (Cascade Style Sheet, responsible for the visual layout and styling of the Web Page). During Project Development, many versions of the System were created; the Design of the most recent ones (mostly suggested by System tests, see section 3.4) focused on real-time tools (e.g. proper functioning of the `loop` facility).

3.2 Design

When reading this section, please refer to video `"Basic functions and Manipulations of Musical patterns.mp4"`.

3.2.1 *Instruments*

The User Interface includes 4 instruments; `SYNTH, MEMBRANE SYNTH, AM SYNTH` and `FM SYNTH`, each of which has 3 sections; `Oscillators`, `Effects` and `Part`. `Oscillators` is the only module which characterises each instrument with a particular timbre and synthesis engine; `Effects` and `Part` are identical across different instruments.

3.2.1 *Hierarchy of Part and Musical structures*

`Part` is the section which is responsible for the generation and manipulation of Musical patterns. The User Interface reflects the hierarchy of Musical structures, which have mainly four levels of parent/children relationships; `Part, Phrase(s), Pattern(s)` and `Event(s)` (as listed from the highest level to the lowest level). An `Event` can either be a note or a rest, depending on `notes likelihood percentage`, `notes likelihood seed` (pseudo-random and deterministic, controllable from its parent Phrase) and `probability of event firing` (random and non deterministic, controllable from global data). For each `Event`, its rhythmic duration and pitch (if it is not a rest) is determined respectively by the `rhythmic/melodic seed` of its parent `Pattern`. A `Pattern` can have any number of `Event(s)`, and a `Phrase` can have any number of `Pattern(s)`.

3.3 **Implementation**

3.3.1 *Determinism and Pseudo-randomness*

Usually, a Deterministic System, which always produces the same output (or stream of outputs) given the same starting condition(s), implies by definition the absence of randomness. Nevertheless my System, for the sake of repeating or modifying patterns, two fundamental elements are needed; a random number generator, and some mean to recall any outcome or series of outcomes the user liked. In fact, in order to be Deterministic as well as retaining some capabilities for random number generation, a System can only implement pseudo-randomness and not true-randomness; a Pseudo Random Number Generator based on the *Mersenne Twister* algorithm allows this process, and it is in fact implemented in *Chance.js* [13]. For each `Pattern`, two distinct random number generators objects are instantiated; one is fed with the `melodic seed`, the other is fed with the `rhythmic seed`.

3.3.2 *Rhythm generation - Binary subdivision Algorithm*

The metric structure of western popular music exhibits an extremely strong tendency – it is binary. Mostly, whole notes are divided into half notes, quarter notes, eighths, sixteenths, etc. Many common algorithms which produces random rhythmic durations fail because do not respect this simple convention. Furthermore, patterns started with an event having rhythmic duration belonging to a binary subdivision level *n* (see Table 1), infrequently continue with rhythmic durations belonging to a $n-1$ or $n+1$ level, very rarely extend to a $n-2$ or $n+2$ level, even more rarely extend to a $n-3$ or $n+3$ level, so on and so forth [11:3-4]. Doing so, rhythmic "emphasis is constantly returned to the primary stress cycle" [11:4], which is the duration of the first event.

This is achieved in software by first generating the rhythmic duration for the first `Event` in the `Pattern` (it is picked from the matrix in Table 1, within both vertical and horizontal ranges declared by the User). Then, the binary subdivisions for each next Event is calculated via a random number generator with a weighted (random normal) distribution of probabilities [REF] calculated within the inputs `rhythmic vertical` (binary subdivisions) and `rhythmic horizontal` (rhythmic duration multiplier by 1, 3, 5 or 7) `ranges` (see Table 1).

3.3.3 *Notes density*

Similarly to *Tuesday* [3], for each `Phrase` the user can choose the likelihood for each Event to be a note rather than a rest (`notes likel. percentage`), and recall any particular series of pseudo-randomly generated results (see 3.3.1) with a seed (`notes likel. seed`).

The Software implementation is as follows; first, an entire `Phrase` is generated. Then, a random choice algorithm (which only provides binary results - e.g. true or false, head or tail, ecc.) yields *n* results (*n* is the total number of `Events` in a `Pattern`). For each `Event` in the `Pattern` whose correspondent random choice is negative (false), the `Event` is deleted from the `Phrase`, and it will behave as Musical rest.

3.4 **Testing**

Throughout Project development, Software Testing was mainly carried with 2 different approaches; in the stages of low-level software implementation, unit and integration tests were made in order to check that single portions of code worked as expected. Since success or failure are the only outcomes in this kind of testing, no System improvements were suggested, and variables' values were printed in the Browser console. Then, at a higher-level of Software Implementation, System tests were made either by me or others, in order to understand whether the System met its requirements. In doing so, focus was given to the User Interface and to the actual possibility and intuitiveness of making a live algorithmic music performance. When improvements were suggested by System testing, low-level implementations had often to be modified and unit/integration tests were needed again.

4. FUTURE WORK

Some late System testing revealed the following possible important improvements to be made;

Implement smaller rhythmic durations as divisors (/3, /5, /7, rather than *3, *5, *7 only - see Table 1). The possibility of playing triplets will be an important improvement.

MIDI Velocity (dynamic intensity) control and generation (now there is no user control over velocity and all the notes have random velocity within half and full scale as ranges). A control over the amount of accent over the first Event(s) of a Pattern or Phrase will be appropriate.

History feature; give the user the possibility to undo/redo any data entry; this will be especially useful for recalling previously-entered seeds and thus further enhancing creativity.

Extend algorithms for melody generation also to signal generation (`Tone.Signal` [12]) in order to control synthesis parameters of instruments like it is possible to do with *Tuesday* via CV (control voltages).

Improve the User Interface by relying on a Graphic Designer for advanced CSS styling.

5. CONCLUSION

The aims and objectives, which have been fixed in Semester 1, have been entirely met by the up-to-date implementation of my System. My knowledge of Web Application Design has highly improved, and so has my ability to model and program Interactive Musical Systems. Finally, being the Web Browser the most effective media to publicise Algorithmic Interactive Music as well as to contribute to establish the habit of its use among people, I hope with my Project to support the foundation of a Web-based Algorithmic Interactive Music standard.

Acknowledgments

I would like to thank Scott McLaughlin for his precious support to this Project. Also, I deeply thank my parents (my mum Piera and dad Luca) for giving me the possibility to continue my University education. Finally, many thanks to my special friend Alice who gave me strength in moments of weakness, loved me sincerely and teached me how to love and take care of myself.

6. REFERENCES

[1] Y. Levtov, 'Algorithmic music for mass consumption and universal production', in *The Oxford handbook of Algorithmic Music*, ed. by Alex McLean and Roger T. Dean, (Oxford: Oxford University Press, 2018), pp. 627-644, (pp. 628-629).

[2] A. Bainter. (2019). *Generative.fm - Endlessly unique ambient music* [Online]. Available: https://generative.fm/

[3] This is not rocket science. (2019). *Tuesday - Procedural Sequencer* [Online]. Available:http://www2.thisisnotrocketscience.nl/eurorack/tuesday/

[4] A. McLean. (2018). *TidalCycles userbase.* [Online]. Available:https://tidalcycles.org/index.php/Welcome

[5] T. Parviainen. (2017). *In C - Play your own unique version of Terry Riley's "In C" with the help of five automated bot performers* [Online]. Available:https://teropa.info/in-c/

[6] B. Eno, P. Chilvers. (2018). *GenerativeMusic.com | Apps by Brian Eno and Peter Chilvers*. [Online]. Available:http://www.generativemusic.com

[7] T. Magnusson, A. McLean, 'Performing with Patterns of Time', in *The Oxford handbook of Algorithmic Music*, ed. by Alex McLean and Roger T. Dean, (Oxford: Oxford University Press, 2018), pp. 245-265.

[8] T. Johnson. (2018). *Tom Johnson: Illustrated Music #6, Rational Melody XV* [Online]. Available:https://www.youtube.com/watch?v=Bqn7cLC6bak&t=103s

[9] T. Johnson. (2018). *Tom Johnson: Illustrated Music #9, La Vie est si courte* [Online]. Available:https://www.youtube.com/watch?v=F7KBTIXNkc8&t=80s

[10] L. Spiegel. (Oct. 1981). "Manipulations of Musical patterns", *Proceedings of the Symposium on Small Computers and the Arts*, No. 393, pp.19 - 22.

[11] P. S. Langston. (1988). *Six Techniques for Algorithmic Music Composition*

[12] Y. Mann. (2017). *Tone.js*. [Online]. Available:https://tonejs.github.io/

[13] V. Quinn. (2013). *Chance.js* [Online]. Available:https://chancejs.com/

[14] *tonal* [Online]. Available:http://danigb.github.io/tonal/index.html

[15] Google inc. (2019). *Angular documentation* [Online]. Available:https://angular.io/guide/quickstart

[16] Brian Eno, 'Scape by Brian Eno and Peter Chilvers - Available for iPad', [YouTube video], 25 Set 2012, <https://www.youtube.com/watch?v=8zNL1KRrUVk>, [accessed 4 May 2019], 1:08.

7. TABLES AND FIGURES

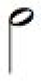
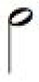
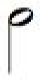
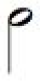
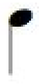
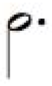
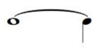
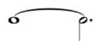
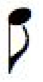
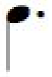
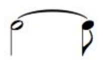
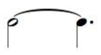
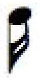
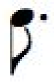
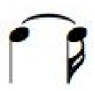
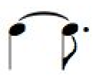
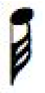
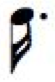
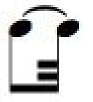
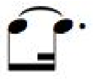
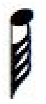
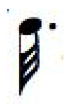
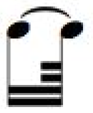
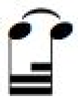
	column 0	column 1	column 2	column 3
binary subdivision level 0	(4)	(12)	(20)	(28)
binary subdivision level 1	(2)	(6)	(10)	(14)
binary subdivision level 2		(3)	(5)	(7)
binary subdivision level 3		 (3)	 (5)	 (7)
binary subdivision level 4				
binary subdivision level 5				
binary subdivision level 6				
binary subdivision level 7				
binary subdivision level 8				

Table 1. Rhythmic durations matrix (numbers in brackets represent the multiplier - e.g. (2) = the rhythmic duration is multiplied by 2).