# Final Project: Probabilistic Arpeggiator

ECS7012P Music and Audio programming – Spring 2022
Tolly Collins – 200553283

**Abstract**

We design an arpeggiator-style melody generator to produce a melody to match a given harmonic base. The algorithm gives the user the choice to apply constraints over different musical aspects of the generative process, which results in an instrument which is reasonably successful in achieving the expectation balance between consistency and change. A MIDI looper is used to play back bass sequences and a 2D wavetable is used to provide a rich and variable sound.

## 1 Introduction

### 1.1 Background

This project explores the development of a generative arpeggiator, which should output a sequence of MIDI note commands that adhere musically both to an underlying harmony and to the wider metrical context. A significant area of research in generative music is in the application of Markov Models. In most cases, the probabilities within the model are calculated as a result of a statistical analysis of a musical corpus or genre [1]. These models are therefore useful when attempting to mimic a given style of music. The Markov condition states that the probability of an outcome must depend solely on the immediately preceding state of the system [2]. This principle can be extended to higher-order Markov Models where the probability of outcomes at a certain point in a sequence depend on the preceding $L$ states (for an $L^{th}$-order model) [1]. The following output can then be determined by procedures such as a random draw or simply selecting the most probable outcome.

Although results have been achieved with Markov Models that do imitate a musical style with a degree of success, there have been several limitations of this approach highlighted. A Markov model results in a random walk, and Whorley & Conklin [3] suggest that this process can never produce musically convincing results due to the diversity problem. The problem stems from the fact that the most statistically unlikely choices in the random walk will most often not be musically appropriate. While unexpectedness is often vital for creating a pleasurable listening experience [4], an unexpected choice in a Markov Model pollutes the probability space in the near future [3] which will often result in a musically unrealistic outcome. However, if the probabilities of the less likely outcomes are reduced too much, this results in a lack of diversity in the artefacts of the system. In essence, when we compose music we are searching for improbable yet musically appropriate outcomes, and it is very difficult to steer Markov Models in this direction.

Various attempts have been made to overcome the Markov Model issues, which often involve constraining the initial statistical sampling process. Snodgrass & Ontanon [5] aimed to provide an element of user control in the generative process which alters the probabilities in the model. Importantly, this control does not come in the form of cherry-picking or rating outcomes; rather it affects the original probabilities. Other approaches include applying an incremental process, where model outcomes are added to the statistical sampling space [2]. Pachet & Roy suggest that extra

context is essential in being able to generate a desired quality of model output, but this necessarily contradicts the Markov hypothesis. It is therefore perhaps necessary to deviate from the strict Markov condition, instead aiming to create an algorithm which is 'optimally Markovian' [2].

Beyond Markov Models, many other approaches have been tested such as neural networks, evolutionary processes and grammar-based processes [6]. Herremans et al. outline a concept map for the generation system, highlighting the compositional concerns of narrative (emotion), interaction (with the compositional process) and difficulty of performance. A significant challenge of both Markov and neural network systems is in satisfying the second condition of giving a satisfactory level of human control over the generative process. In general, the longer-term structure of the system's artefacts is important in satisfying these conditions which involves the musical elements of melody, harmony, rhythm and timbre.

## 1.2 Aims and Approach

The system implemented in this study aims to draw on the conclusions and outcomes of this prior work. The aim is to create a musical tool / instrument which produces a melodic sequence resembling that of an arpeggiator. However, rather than the user specifically providing a sequence of input notes and specific pattern parameters to the instrument, the user supplies a harmonic progression. This harmonic progression can be kept constant via looping functionality, or it can be played live. The arpeggiator then creates the melodic accompaniment via a generative process, which outputs each note individually in real time. Furthermore, this allows the generated sequence to evolve over time, rather than being fixed until a parameter or note is altered by the user.

Although the initial inspiration came primarily from the Markov Model approaches, it was decided that its limitations were too great for this use case for it to be the primary component of the system. Furthermore, the desire to create a user-friendly interaction experience with the instrument [6], it was important that a significant level of control could be imposed on the generative system if desired. However, the idea of an output note being sampled from a probability distribution does form the basis of this system. The idea of implementing constraints on the probability distribution forms the basis for the system design, with the user having access to controls which are focused on different musical elements such as rhythm, harmony, melodic intervals and dynamics.

The balance between unexpectedness and consistency was a major driving force in the design decisions taken. While the output should not be entirely predictable, complete randomness does not provide a pleasurable listening experience. In terms of the user's compositional experience, it would generally be more satisfying to feel that a significant level of control can be attained over the generated sequence, and that the direction in which the model goes as the sequence evolves can be controlled, at least to some degree. The results should be reproducible to some extent, while also allowing for unexpected occurrences. To satisfy these criteria, the decision was made to offer various levels of control to the user over different aspects of the generative process. The used can then make the decision over how much variation the model is allowed, and they can decide whether to take fine-grained or more generalised control. Examples of aspects that can be altered by the user include the 'seed' sequence, the initial probability distribution over note chroma class options, various coefficients in the generative algorithm and a generalised control of the degree of randomness. The implemented approach is outlined below.

# 2 Design

## 2.1 General Instrument Design

The instrument contains three musical elements. The first is an optional kick drum which plays on each tactus, acting both as a guide for the user when playing the bass element, and to help place the resulting sequence in the metrical context in a way that adds to the listening experience. The kick drum uses a sample from the website Freesound [7], which is a reasonably punchy EDM synthesized kick. Playing the sample is handled by the pre-defined `MonoFilePlayer` class, called each audio frame in `render()`. The kick drum is given an amplitude profile to emphasise the metrical structure, with a higher amplitude on the first and third beats of each measure.

The second musical element is the bass sound. This is synthesises using a 2-dimensional wavetable, inspired by soft synths such as Native Instruments' Massive [8] and XFer Records' Serum [9]. The pre-defined `Wavetable` class was adapted to hold two initial wavetables, and then the final wavetable provided for synthesis is a linear interpolation between them. The user has control over the interpolation ratio, meaning that they can choose to use just one of the original waves, or any degree of combination between the two, given the wavetable resolution. The interpolated waveform is then held in a buffer for quick real-time access, only requiring occasional re-calculations when the interpolation ratio is altered.

The design of the two waveforms in the 2D wavetable uses Valimaki's Differentiated Parabolic Waveform technique, which allows for cleaner-sounding digital sawtooth and square waves with significantly reduced aliasing artefacts [10]. This approach involves applying a second-order FIR lowpass filter to a squared version of a sawtooth-shaped sequence created with a linear ramp. The square wave is created by subtracting a phase-shifted version of the sawtooth wave from itself.

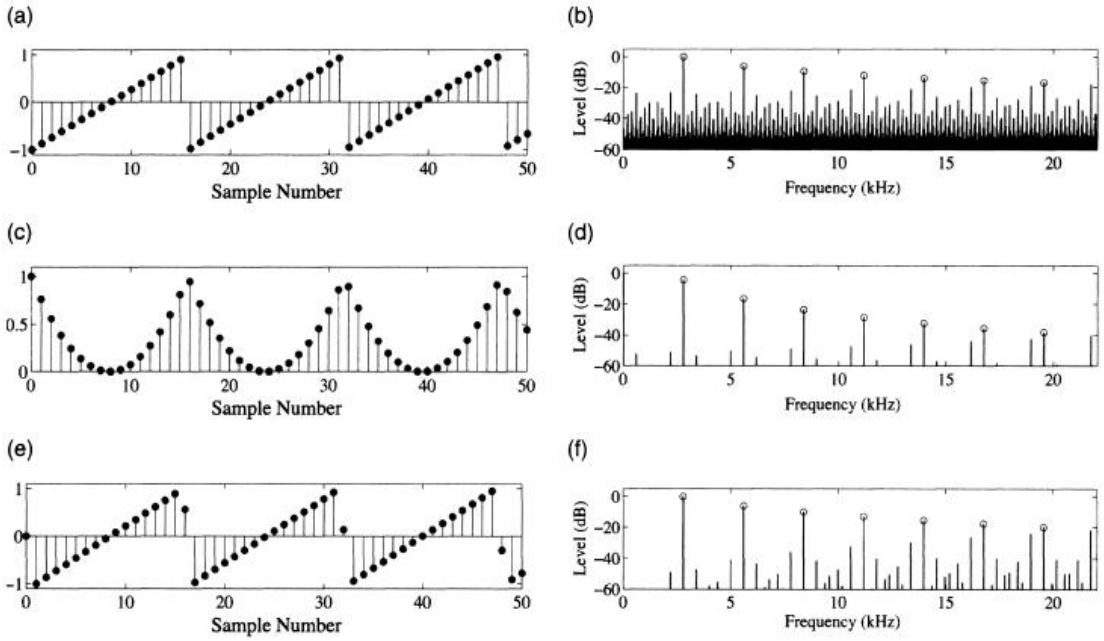



Fig 1: Taken from [10], we see the basic sawtooth wave, the squared version and the final differentiated waveform with relative amplitude spectra shown on the right.

The synthesised signal is then filtered using the digital Moog ladder filter emulation from Valifakis & Huovilainen [10]. This functionality of this filter was also put in a separate class, so it could be called directly from `render()`.

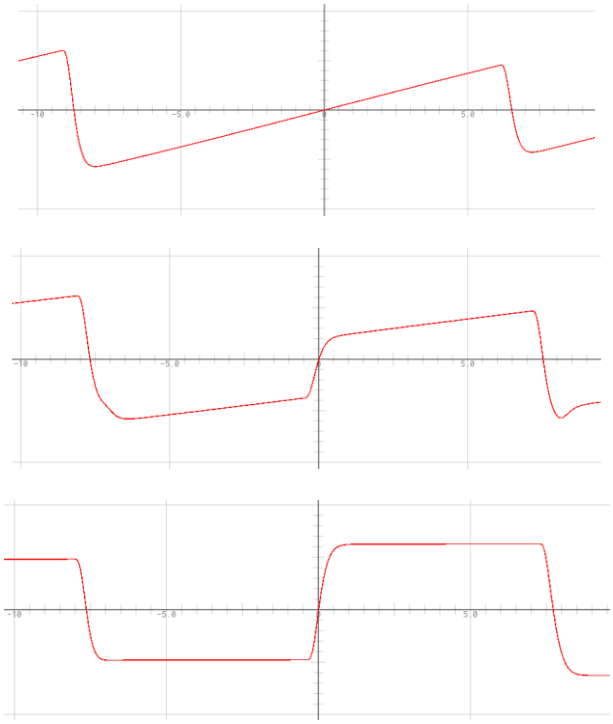


Fig 2: A screen shots from the Bela Oscilloscope of the resulting waveform with gentle filtering applied. The top is top waveform is heavily weighted towards sawtooth, the middle is a mixture of both, and the bottom shows the square waveform.

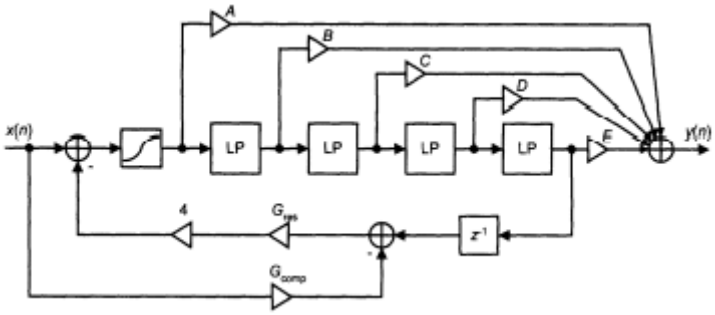



Fig 3: Block diagram for the digital Moog ladder filter emulation [10]

This bass sound was then layered with a sub-bass sound from a second oscillator which just used the square waveform with its pitch shifted to an octave below the played note. This gives the bass sound an added presence and depth.

These waveforms were also used for the oscillator for the arpeggiator sound. Althugh the output from the arpeggiator is at a higher pitch than the bass, the sounds were further differentiated through default settings of a high number of voices for the arpeggiated sound at a higher detune ratio, and the sound was set more towards the square wave than the bass. Both sounds had ADSR envelopes applied, with faster attack and decay settings for the lead sound for both the amplitude and the filter cutoff envelopes. These settings could all be controlled by the user through either the GUI or the QuNeo MIDI controller [11] that the system was designed to be played with.

The user has three main elements of the instrument to control: the bass notes, the sound parameters and the arpeggiator generation algorithm parameters. It is important that the user can adjust the two sets of parameters live rather than just having to set the at the start, therefore it was important to add the functionality for the user to be able to loop a bass sequence. A MIDI looper class was created which keeps a record of all MIDI note on commands in a circular buffer with a resolution of 960 samples per beat. When the looping button is pressed on the control pad, the loop is read from so that no new notes need to be played. In order to allow for the tempo to be changed at any point, the `Looper` read pointer is incremented fractionally according to the ratio between the audio sample rate and the tempo, and a new output is read whenever the pointer's value goes over 1.0. At all other times, a 'no-note' output is sent back to `render()`. When a note on command is read form the `Looper`, the MIDI `noteOn()` function is called in the same way as it would if an incoming MIDI message were received. Furthermore, the write functionality remains throughout, so the user can alter the sequence while it is looping. The bass synthesizer is set up in a continuous monophonic manner, where a note continues playing until a new note is played. If a note is held down, then any previous note on messages stored in the `Looper` over that sequence position will be overwritten. The looping functionality can be turned on and off at any point, as can any of the kick, bass or lead sounds.

## 2.2 User control interfaces

An important element of the user experience is being able to have easy and intuitive control over the instrument. The arpeggiator was designed to be combined specifically with the QuNeo MIDI controller, and all of its sensors were mapped to various instrument parameters.

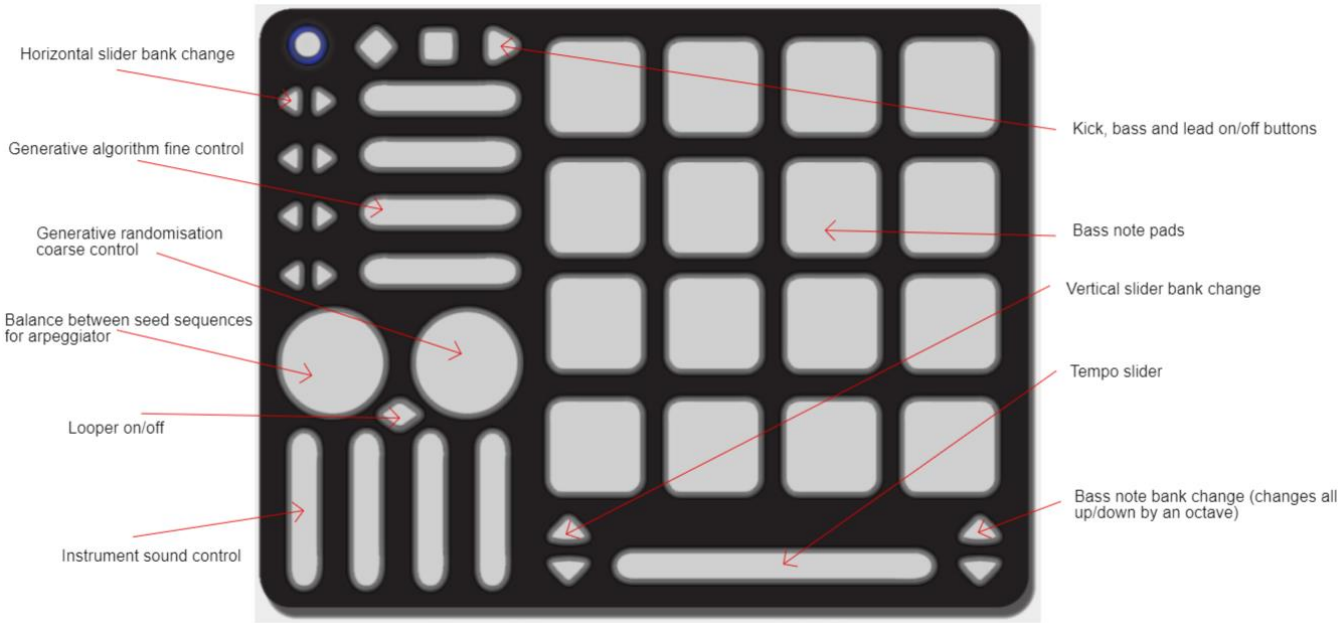



Fig 4: QuNeo control mappings

The three buttons at the top control the three sound elements of the instrument, turning them on or off individually. Turning the synthesizer off resets the previous sequence information held in the buffer to the seed sequence interpolation (explained further below). The horizontal sliders control the size of the coefficients in the generative algorithm. There are 9 controls in total, and so these

are assigned to banks which are selected via the controls to the left of the sliders. The circular rotary sensors control the arpeggiator seed sequence balance and give a coarser control over the degree of randomness. The rhombus button turns the looper on/off. The vertical sliders control the sound characteristics and the long horizontal slider at the bottom controls the tempo. Finally, the main pads are used for the bass notes. One of the motivations of using the QuNeo controller is that X-Y information from the pads can be transmitted, which allows for easy major/minor distinction while playing the same pad. If a pad is played on the left-hand-side, the arpeggiator is told to use the major mode with this harmonic root, with the minor mode corresponding to the right-hand-side of the pads.

A second motivation for using the QuNeo controller is the ability to control LEDs for every sensor. This allows, for example, the overall randomness ('temperature') control LEDs to be changed proportionally whenever a fine-grain slider is changed. This provides feedback on the total 'temperature' level as the fine-grain controls are altered. Furthermore, LEDs on the main pads are used in two different ways. The first is to signify which note (and which mode) is being played, which continues when the bass is being played via the loop with CC control messages being sent from the main system. The second application is that successive pads flash in time with the tactus beat, giving the user a visual cue to follow. A significant proportion of the code in `render.cpp` is dedicated to implementing this controller functionality. For additional controls not mapped to the QuNeo, the browser-based GUI was used.

## 2.3 Generative Algorithm

The main element of this project is the generation of sequences based on a steerable random process, with the functionality of this process being placed in the class `ProbabilisticArp`. The `generate()` function is called on each 16th note, and either outputs a MIDI note, amplitude pair or outputs a non-note message. The sequence of steps in the algorithm will now be described.

First, the process is provided with two seed sequences, and the user can select different starting sequences via the GUI. These sequences are then interpolated between via a balance control, which can be updated in real time. The purpose of the seed sequence is not to be continued from as with, for example, a standard Recurrent Neural Network model. Rather, they provide a reference for elements of the generated sequence such as the contour, general pitch area and interval sizes. All of the output note decisions are also taken relative to the key and mode information received from the bass element, the most recent note played and the notes from the previous sequence at the same metrical position. The idea is that the playoff between these opposing factors is what gives the randomness a more musically coherent factor, and the challenge was to produce a system which achieves the correct level of expectation.

```
{{48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9}, {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9}, {55, 1.0}, {48, 0.9}, {51, 0.9}, {55, 0.9}, {48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9},
 {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9}, {55, 1.0}, {48, 0.9}, {51, 0.9}, {55, 0.9}, {48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9}, {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9},
 {48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9}, {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9}, {55, 1.0}, {48, 0.9}, {51, 0.9}, {55, 0.9}, {48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9},
 {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9}, {55, 1.0}, {48, 0.9}, {51, 0.9}, {55, 0.9}, {48, 1.0}, {51, 0.9}, {55, 0.9}, {48, 0.9}, {51, 1.0}, {55, 0.9}, {48, 0.9}, {51, 0.9}}
```

Fig 5: example of a seed sequence of 64 16th notes and accompanying amplitudes

For the main generative process, the first stage is for an initial set of weightings to be assigned to the 12 note classes. A pair of user-adjustable choices of initial values are interpolated between via the 'harmonic temperature' control value. The higher the harmonic temperature, the more likely less harmonically expected notes are to be selected.

a)
```
{7, 0, 4, -1, 9, 2, 11, 5, 8, 1, 3, 10, 6}
{7, 0, 3, -1, 9, 2, 10, 5, 8, 1, 6, 11, 4}
```

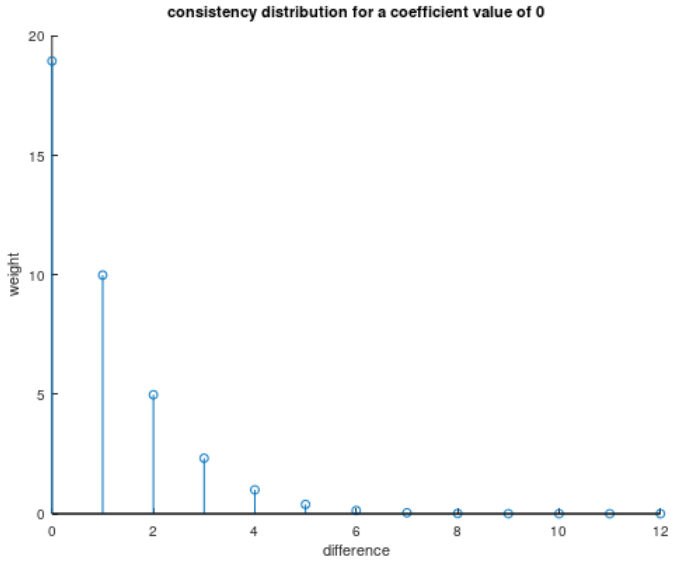
b)
```
{12, 12, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {8, 8, 8, 2, 12, 12, 8, 7, 6, 5, 4, 3, 2}
```

Fig 6: a) the 12 major and minor chroma classes relative to the root note (0) in order of expectedness. The value '-1' is used to signify a 'non-note'. b) a pair of respective low (left) and high (right) temperature weightings for each chroma option.

Then, a 'consistency' coefficient is applied to determine the degree to which the generated sequence follows the previously generated 4-bar sequence (or the seed sequence for the first iteration). This is determined by the formula:

$$w = \frac{13 - |p - x|}{9}^{8(1-c)}$$

$w$ is the weighting given to the chroma class x, relative to the previous pattern chroma value $p$ and the consistency coefficient $c$. If $c = 0$, then the generated sequence will be heavily biased towards following the previous sequence as close as possible, whereas a maximum value of $c = 1$ gives no effect. The '-1' non-note case is dealt with separately.



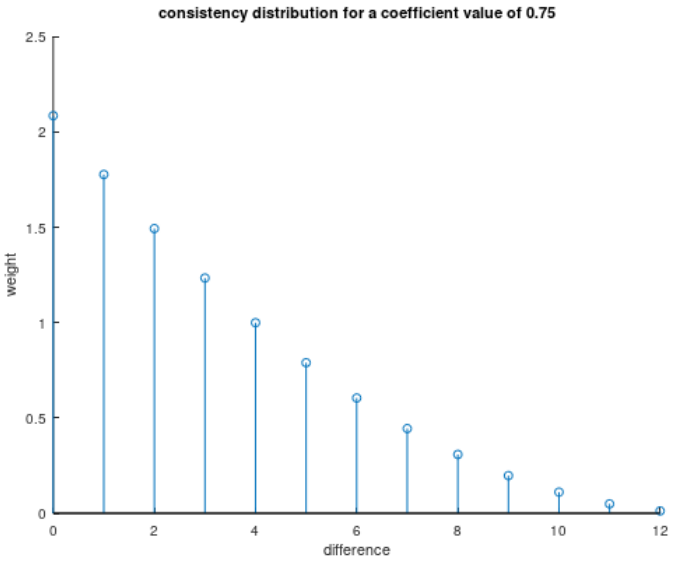



Fig 6: The weightings given to the values in the probability distribution according to the 'consistency' control. The top graph shows the multiplier for each chroma difference value for a consistency slider position of 0 (i.e. most consistent with outcomes heavily biased to be similar to the previous sequence), and the bottom graph shows the distribution of multipliers for a consistency slider position of 0.75.

The third control is 'sparsity'. Higher sparsity values increase the probability of a non-note, regardless of any other contextual factors.

The fourth control is 'rhythmic temperature'. This also affects the probability of a non-note occurrence, but in this case lower slider values increase the probability of a note / non-note occurrence matching that of the equivalent position in the previous 4-bar sequence.

The fifth control is 'movement'. A movement value of 0 has no effect on the outcome, but higher values increase the probability of a chroma interval between the generated note and the previous note to be small. One of the aims of implementing this control was to give an option to increase the probability of repeated notes in the generated sequence.

At this point, the weightings are normalised to a sum of 1 and a continuous sample space in the interval [0, 1] is defined with intervals for each possible chroma outcome proportional to the outcome of the weighting process described above. This sets the note class for the arpeggiator's output, but the absolute octave pitch value remains to be chosen.

The octave number is chosen in a similar manner, with some degree of control over the process given to the user. The sixth generation control determines the degree to which the melodic contour should follow that of the selected seed sequence, controlled by the 'contour temperature'. A lower temperature biases the distribution to favour octave choices which follow the same contour as before, whereas higher settings relax this restriction. This control allows the user to bring the output sequence towards or away from the seed sequence as it evolves over each 4-bar cycle.

The seventh control is similar to the sixth, but in this case low 'pitch temperature' values give a high weight to pitch values that are close to those in the seed sequence in the respective position.

The eighth control ('interval temperature') affects the probability of certain interval sizes. Low settings encourage small intervals and higher settings encourage wider intervals by adding weighting to octave choices which are further away from the pitch of the preceding note. As before, the resulting weightings are then transformed into an interval from [0, 1] which is randomly sampled from via a continuous uniform distribution.

The ninth and tenth user controls affect the amplitude of the selected note. The seed sequence provides initial amplitudes for each note, and the 'dynamic contour temperature' control determines the degree to which the dynamics should match the dynamics of the seed sequence. Finally, the 'dynamic temperature' control allows a degree of random deviation in dynamic level, with higher values allowing a wider sampling range for the random value added to the level.

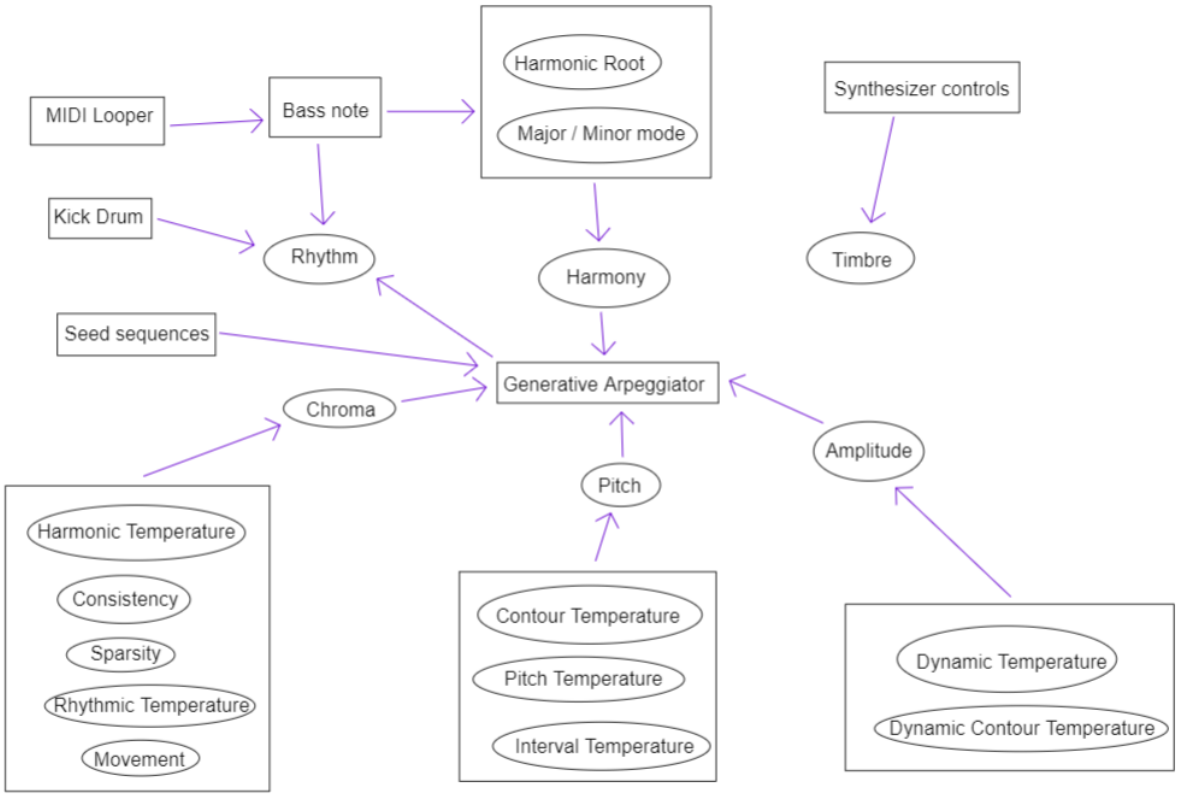



Fig 7: Conceptual diagram of the probabilistic Arpeggiator instrument.

## 3 Evaluation

The system was tested with four different seed sequences and three options for starting note chroma distributions. In each case, with the low temperature settings, the arpeggiator produced an outcome which was close to the seed sequence. The sequence did change slowly over time, but the algorithm ensured that it could not stray too far from the seed sequence notes. Adjusting the temperature controls allowed different aspects of the sequence to vary more. With high overall temperature levels, it was noticeable that the original seed sequence was not recognisable. However, it was interesting that the outputs produced from the arpeggiator did vary in style given different seed sequences, even when the randomisation level was increased.

One particular success of the algorithm was its ability to capture the rhythm of the seed sequence while varying from it in a way that felt musically coherent. The treatment of 'non-note' events meant that the rhythmic deviation felt controlled. When the harmonic temperature was turned down, the resulting sequence was always harmonically tied to the bass note which meant that it never seemed too unexpected. However, when the harmonic temperature was increased, the particular choice of starting distribution did have an effect on the feel of the output sequence. When more harmonically unexpected notes were given particularly high starting probability levels, the resulting output sequence could feel too random and the musical sense could be lost. There is scope to experiment further with these initial weight values, investigating the effect on the sequences produced by the instrument.

It was important to be able to choose between different seed sequences for the arpeggiator to add variety, and a large amount of the intrigue in using the instrument came from seeing how the output

sequences related to the original seed sequence. However, a possible improvement would be to give the user a way to input a new seed sequence while the instrument is playing, or to adapt the current selected sequence. This would make the instrument operate a little more like a conventional arpeggiator, and this could further enhance it as a creative tool.

The use of the QuNeo MIDI controller made the interaction with the instrument intuitive and dynamic. The LEDs provide useful feedback, but it is difficult to keep track of the uses of each slider. It would be beneficial to develop a full Graphical User Interface to accompany the instrument to give clearer visual prompts and feedback.

The overall sound from the instrument was pleasing, and there is further scope to add more waveform options to the 2D wavetable.

The use of classes kept the code well organised. There was some repetition between the Wavetable1D and Wavetable2D classes, and it may have been better practice either to make these both inherit from a common parent class, or for the Wavetable2D to be a child of the Wavetable1D. Significant care was taken to ensure that all of the elements were real-time safe and that in particular the generative algorithm was able to be calculated within the real-time deadline.

# 4 Conclusion

The main aim of this project was to create a generative algorithm which produces results which are both musically coherent and interesting. I believe that this aim has been achieved, with an instrument that offers varying degrees of control to the user. A general temperature control can be used to control the overall level of randomness in the generative process, but perhaps more interstingly the instrument design offers the user a way to select more fine-grain control over different musical aspects of the output sequence. Some potential improvements have been noted, particularly with introducing ways to create new seed sequences and creating a more user-friendly graphical interface.

Having real-time control over the degree of randomness goes some way to reducing the diversity problem inherent with many generative approaches. The use of fine-grained control, seed sequences and starting distribution weights makes the results reasonably consistent if desired, but it is still difficult to guide the arpeggiator in a precise direction. It would be worth considering how to add to or alter the way that the instrument is controlled in order to give the user a greater degree of control over exactly how much randomness they wish to have. For example, if the user could choose certain notes to remain fixed and the algorithm then altered the other notes,

## References


[1] Schulze & van der Merwe (2011) – Music generation with Markov Models. *IEEE Vol 18, No.3*

[2] Pachet & Roy (2010) – Markov Constraints: steerable generation of Markov Sequences. *Springer*

[3] Whorley & Conklin (2016) - Music Generation from Statistical Models of Harmony. *Journal of New Music Research*

[4] Narmour (1991) - The Top-down and Bottom-up Systems of Musical Implication: Building on Meyer's Theory of Emotional Syntax. *Music Perception*

[5] Snodgrass & Ontanon (2016) - controllable procedural content generation via constrained multi-dimensional Markov chain sampling. *IJCAI*

[6] Herremans et al (2017) – A functional Taxonomy of Music Generation Systems. *ACM Computing Surveys, Vol. 50, No. 5, Article 69*

[7] www.freesound.org - Kick sample - kick_gettinglaid.wav by DWSD [CC0 license]

[8] Native Instruments – Massive synthesizer https://www.native-instruments.com/en/products/komplete/synths/massive/

[9] Xfer Records – Serum synthesizer https://xferrecords.com/products/serum

[10] Välimäki & Huovilainen (2006) - Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal, 30(2), pp. 19-31.*

[11] QuNeo MIDI controller – Kieth McMillen Instruments https://www.keithmcmillen.com/products/quneo/