Synthetic Music Random Generation Based on Nyquist

Wenhao Li [1, a], Zhengmo Ma [2, b] and Zijian Zou [3, *]

[1] Tianjin Foreign Language School, Tianjin, China

[2] Hangzhou Foreign Languages School, Hangzhou, China

[3] International College, Guangdong University of Foreign Studies, Guangzhou, China

* Corresponding Author Email: zouz1@mail.broward.edu, [a] tfls_larry2022@126.com, [b] summer.light.ma@gmail.com,

Abstract. The idea of composing music based on programs has a long history, from some pioneers who first tried to fulfill this idea, to now, the variety of software and workshops that allow people to generate music in different approaches. On this basis, this study will briefly discuss the main idea and some methods used to compose the music in this paper. We aimed to randomly generate a piece of music using only Nyquist. Thus, each time when listen to this, there is a slight difference. With this in mind, this paper has used two ways: the first one is to use several functions in Nyquist, which would randomly choose numbers in a given list or range. The second way is a method based on the functions mentioned earlier, called the random walk, which lets the program stochastically choose one item in a given list of a range of numbers every time to add or subtract to the origin number given by programmers. By using these two techniques, one can generate random sounds and random rhythmic patterns, and when we put these sounds and patterns into scores to generate various melodies, we can stochastically increase or decrease the overall pitch of that score. Finally, the scores are combined to create the whole piece of music. Although there are limitations, this study has provided some methods that can be useful for generating stochastic music.

Keywords: Synthetic Music, Nyquist, the random walk.

1. Introduction

Algorithmic composition, sometimes also known as “automated composition”, is to generate music on a computer with little human intervention by using formal processes [1]. By applying compositional processes in computers, composers can use this skill to make music with a brand-new approach. Algorithmic composition also has a history. Two pioneers in this new field are Hiller and Xenakis [2].

One of the earliest examples of computer-generated music was by Lejaren Hiller and Leonard Isaacson at the University of Illinois in 1955-1956. Using the Illiac high-speed digital computer, they successfully wrote the basic materials and stylistic parameters, resulting in the Illiac Suite [2]. They applied a method called "generator/modifier/selector". Firstly, certain "raw materials" are generated by a computer. Secondly, these musical materials are modified according to different functions. Lastly, the best results are selected from these modifications according to different rules to generate a full piece.

Musicomp was easy to use for programmers to create their own music style because the program was written as a library of subroutines. Iannis Xenakis, who is very famous in the field of computer music. He used mathematic knowledge like statistics and probability to make computer-generated music more live ensembles [3]. For instance, in his “stochastic” system, it would infer a score from a list of note densities and probability weights provided by the programmer, leaving the specific decision up to the random number generator [3]. In the examples of Hiller and Xenakis, there were already two systems, stochastic and rule-based systems. Later, we will be seeing a new category, which is artificial intelligence [2].

Artificial Intelligence, or a more familiar name, AI, has been a relatively new technology involving many fields, including algorithmic composition. The range of methods used to achieve the synthesis of algorithms is very broad, including many that are very different from artificial intelligence, but

also borrow mathematical models of complex systems [4]. For example, some methods are grammar, symbolic, knowledge-based Systems, Markov chains, and Artificial Neural Networks etc. [4]

Afterward, a new advanced language for algorithmic composition was generated by some programmers and musicians called Nyquist, which was supported by a series of computer languages and implementations [5]. Nyquist IDE based on Java helps programmers to write codes to generate sounds, which means that people can write straightforward code quickly in one editor, and they can compose a piece of complete music about any style by any functions, expressions, and libraries of Nyquists. In addition, Nyquist codes can be transferred to Audacity which is an audio editor to play these sounds from Nyquist and saves them as MP3 or WAV files [6].

So, Nyquist lays a foundation on many scholars who favor music composition to attempt to use them to create any types of songs they want or some innovative works. For example, according to the article from Li at the University of Illinois at Urbana-Champaign, the research is about how to transfer music pieces into different scales. The research method is to combine the advantages of Nyquist programming and data type "score" with their idea of music [7]. Finally, they can modify pieces into different styles according to the scales. Another example from the article written by a group of researchers is to compose real-time adapting weather music. Their project renders the help of people's emotions by writing and controlling the code from media to create new scores [8].

Therefore, the motivation of the study is about how to compose a piece of random synthetic music by utilizing the language of Nyquist. There are different pitches, rhythms, and intonation of notes generated and played randomly. Meanwhile, all sounds of music are synthesized by internal SAL code, not from external audios. Hence, it would be very adventurous and unexpected for listeners that each note totally from synthetic instruments matched each other with a random direction of melody. In this article, we will present our data and methods for choosing any types of Nyquist functions and notes to compile. Then our final codes, flowchart, and the demo are shared and discussed for the result of the study. Finally, the article will talk about some weaknesses and future possibilities, and then get an exact conclusion.

2. Data and method

In the study, all the audio and effect resources used are pre-integrated in the Nyquist IDE. The wavetable parameters for the oscillator include sine wave, triangle wave, and saw wave. For drum simulation, a white noise generator is also used. The construction of the final output used the score system in Nyquist. To add randomness to the output, the internal random number generator is applied to many parts of the work. No external plugins or samples are used in composition. When the code is run in Nyquist IDE (the version used in the study is 3.22), it will generate and play an audio file with a length of 128 seconds, including several different parts: a pluck, two different leads, a bass, a set of drum consisted through a kick and a hi-hat. Every part is played with the same fixed tempo. Except for the second lead and the hi-hat, all the sounds are played in a fixed rhythm.

The method of generating the whole music file can be separated into several steps: setting up basic variables and random lists, defining the note's function, constructing each part's score separately, and combining every score. The basic variables are a basic pitch and three notes randomly selected in the octave starting from the basic pitch. In order to make the pitch for the whole piece more pleasant, the basic pitch is selected in a relatively small range of 6, from F#2 to C3 (42 to 48 in Nyquist).

Most of the notes with a specific pitch have their pitch parameter from the list of the four pitches described above. Other random lists include the wavetable list deciding the timbre of the notes, the time list helping decide the time of hi-hat, and the second lead. They are generated through the internal function make-heap, which allows picking elements from a list randomly.

Besides these methods, two specially designed random algorithms are applied for the rhythm of the hi-hat and the pitch of the second lead. For the hi-hat, in order to keep them on the backbeat and add some double notes, a random function to determine the IOI (inter-onset time) of the notes is used, which is shown in the code. For the second lead, an algorithm for "random walking" is used [9],

indicating that instead of determining each note's pitch directly, its interval to the last note played is randomized. The realization of this used a list including many different intervals, make-random function, make-line function, and make-accumulate function.

The note functions generally follow that same structure: an oscillator at first, followed by a pawl function to shape the original signal, and a parameter to adjust the volume at the end. As mentioned above, the wavetable for the oscillator is randomized to create more variability. Moreover, some sounds have special treatment. The first lead has an LFO to modify its amplitude, and the second lead has another LFO to modify its frequency [10-12]. For drum sound, a low-pass filter after a white noise generator is used to simulate the kick sound, and a high-pass filter is used to simulate the hi-hat sound. To-mono function is applied to the kick sound to make it centralized.

The scores for each part are generated separately, with the same length determined through the score-dur parameter. As an isolated case, the score for the second lead has its length doubled to create a longer melody line. There is no tempo parameter for the whole piece since it is determined through the IOI and duration of notes. The smallest unit of IOI designed is 0.1, and 0.2, 0.4, and 0.8 are prepared for longer notes. These time parameters have ratios of 2, which ensures the beat stays stable. The random lists defined at first are now applied in the score generators, which creates a huge diversity.

The final step is combining the scores together. For playing them simultaneously, score-merge is applied. Score-transpose is also used to create a shift of pitch in the scores, adding coloration to the piece. After every single part is finished, score-append is used to connect them together, generating the score-total, which is ready to play.

3. Results and discussion

The first part of generating the music was to set variables for patterns and define different sound functions, they are for the rhythm and timbre of sounds. They are important for the generation of scores. The two main ways for randomness in this project are the random walk and functions in Nyquist. Nyquist functions like make-random and make-heap are used for rhythm randomness. For sounds, we define our own functions and return sounds by changing parameters in origin sound functions in Nyquist, adding random numbers, and multiplying them with those sound functions to create timbre randomness. Each time, the timbre generated is different. For example, in Fig 1 and Fig 2, the two graphs are different, but they are generated by the same function we defined called note 1. After finishing the first part, we already have all the patterns and sounds that are needed for building scores to generate melody.

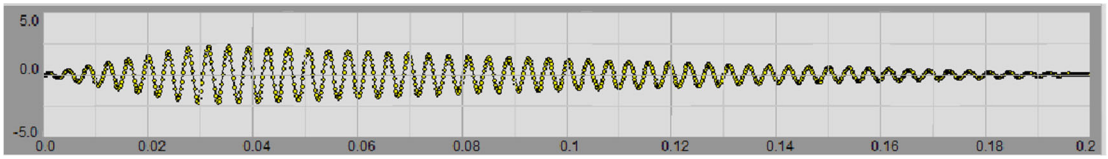

Figure 1. First time playing the note 1 function (Photo/Picture credit: Original)

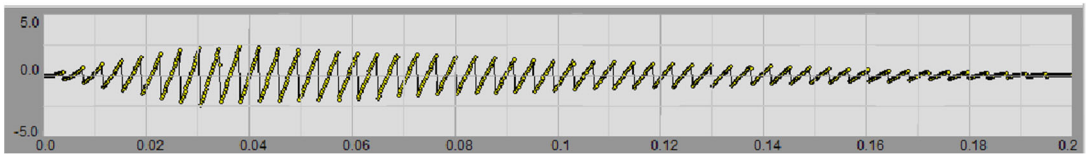

Figure 2. Second time playing the note 1 function (Photo/Picture credit: Original)

The second part is mainly scoring generation and combination, which means generating different melodies and then merging them into a single piece of music. The different patterns and sounds are put into different scores, they are pieces of melodies in data type. Some functions are used to randomly make the pitch a little higher or lower, as a result, the music can be different every time. After these scores are created and changed, they are combined with each other to generate the full piece of music.

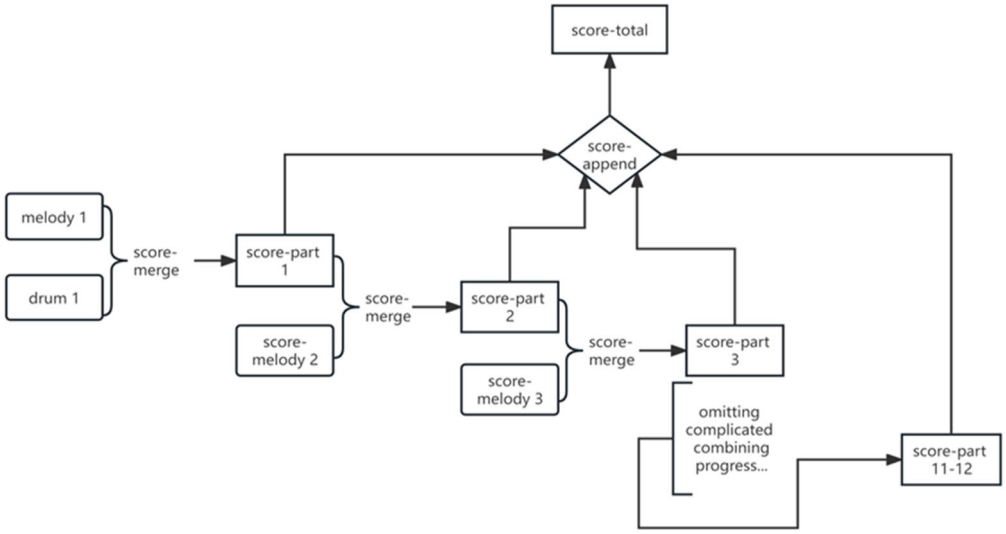

Figure 3. A flow chart for the combining process (Photo/Picture credit: Original).

The final result of the music is successful after combining all the scores together since it includes some innovative and irregular tones and sounds that listeners could not guess the next notes when they're listening. The codes of this composition have a logical process of not only creating patterns and defining sound function but also setting up some melody and drum scores that it is relatively understandable for code readers.

Initially, each score of melody has its unique generations like pitch, interval, duration, and score length. Drum scores are also mediated to be the rhythm parts. They quote notes in the second part and have their own score names. Besides, it uses score-merge, score transpose, and score-append functions to combine each score to create new scores. Some original scores are merged into a first part, and then the first part is merged into a second part with another melody or drum score. These scores merge in the same way. Finally, there are 10 score parts to be combined into a score total by appending. In addition, there are flow chart to express the progress of score combining in Fig. 3.

After loading the code to play, there are too many irregular notes like bricks to constitute a building each note from synthesized instruments became a crucial fragment to construct the part of the music. It sounds inharmonious and elusive when people listen for 2 or 3 seconds. When the music is played continually, it sounds reasonable for people because they can find the rhythm and repeated melodies gradually. Even though the notes in the music are irregular and random, it is not totally disordered. The high and low frequencies are clearly distributed, and it is ended by fading out to bring some mysteries for listeners.

In addition, when it is played at next time, it sounds different from the first time and the melodic lines are changed because of the note randomness. But the rhythm of the drums is unchanged. So, whatever it is played at any time, the sense of hearing will not be influenced by randomly changed melodies.

4. Limitations and prospects

Although our program completes the goal of generating a random music piece, it still has many drawbacks. For the first point, a structure of pattern-score is used in the program, indicating that the music is completely based on a fixed sequence allowing Nyquist to play. It is true that this sequence, predetermined by humans, is pleasant in common sense, but also limits the variability of the piece. As a random music generator, it is expected to produce more creative arrangements that human composers have hardly ever tried.

Additionally, the timbres of the sounds are far too simple and stable. Nyquist actually supports very complex audio design through basic blocks, but in the program, only the very simple oscillator and wavetables are used. Also, the shaping process is based on the PWL function, which is comprehensible but limited, since the modification is linear. Timbre is a very important element of electronic music, and the similar timbre in the pieces generated by this program leads to a consequence that the pieces also have a similar feeling, which contradicts the goal of adding more variability. Last but not least, the tempo of the piece, determined by time parameters in the score, is completely fixed.

Moreover, the beat of the main elements, including the pluck, the first lead, the bass, and the kick, stays the same all the time. These two points contribute to a fixed rhythm for all the different pieces the program generated. Rhythm can affect the song's emotion dramatically. Thus, a fixed rhythm here causes the emotion to be very limited.

According to the limitations analyzed above, there are many possible ways to continue improving the generating program. In order to solve the problem of a too-stable arrangement, we can continue using the pattern-score-based structure but add the pattern elements to a "pattern list" and select them randomly to avoid the previous problem. There are still many methods to improve the flexibility of arranging, including using machine learning to study similar works done by human composers.

For designing more creative timbres, a practical way is to use resampling. We can use the current oscillator but resample it and make it into a wavetable in Nyquist several times. This can help the timbre to be more variable and unexpected. Sound design with artificial intelligence may have some examples to study, which is also a method worth considering. The tempo is actually a relatively simple part of improvement. A basic time for calculating tempo (like 0.1 in the current program) could be set by the start part of the code, and other timing methods should be based on the basic static variable.

Besides the methods listed above, there is also some more creative way, of reconstructing part of the process. One of the possible ways is to change the original pitch system. It is now based on the equal temperament; however, it could also be based on the frequency in Hertz, which is closer to the physical essence of an oscillator. Although it could be harder for this method to make a pleasant song, frequency is a freer parameter.

5. Conclusion

To conclude, this study is about generating random music that will sound different whenever playing based on Nyquist. To be more detailed, we achieve this by using functions to make pitches, patterns, and melodies sound more random; for instance, the random walk method is used based on these functions, and some of them can be directly used, for example, they can randomly choose an item in a list or to randomly rise or fall a pitch. By using these techniques, we can synthesize a random piece of music. However, there are some limitations. For example, the scores can't be played in random order, the timbre of sounds is simple, and the tempo of the piece is not random. In the future, pattern lists can be added, timbres can be designed more randomly can creatively, and the tempo can be set at the beginning of the code with random functions inside. Overall, this study has provided some methods that can be helpful for synthesizing stochastic music.

Author contribution

All the authors contributed equally, and their names were listed in alphabetical order.

References

[1] A. Alpern, Hampshire, **95**, 120 (1995).

[2] J. Maurer, A Brief History of Algorithmic Composition. Retrieved from: https://ccrma.stanford.edu/~blackrse/algorithm.html (1999).

[3] I. Xenakis, *Formalized music: thought and mathematics in composition*, (Pendragon Press, Chicago, 1992).

[4] J. D. Fernández, and F. Vico, J. Arti. Intel. Res., **48**, 513-582 (2013).

[5] B. D. Roger, Comp. Music J., **21(3)**, 71–71 (1997).

[6] R. B. Dannenberg, *The Nyquist Composition Environment: Supporting Textual Programming with a Task-Oriented User Interface*, (Nyquist, CMU, 2008).

[7] J. Li, High. in Sci., Eng. & Tech., **39**, 209-214 (2023).

[8] Y. Huang, Z. Jia, C. Lin and Y. Shi, High. in Sci., Eng. & Tech., **39**, 280-285 (2023).

[9] B. D. Roger, *Dannenberg. Introduction to Computer Music* (ICM Edition, CMU, 2021).

[10] P. Cook, A NIME Reader: Fifteen years of new interfaces for musical expression, **11**, 1-13 (2017).

[11] I. Shapiro and M. Huber, Markov chains for computer music generation. Journal of Humanistic Mathematics, **11(2)**, 167-195 (2021).

[12] E. Frid, Multi. Tech. & Interac., 2019, **3.3**: 57.