Altered Tunings in A ― 音律の歴史的な変遷と地域的な違いをテーマとした楽曲のネットワーク分散演奏 ―

石原 尚始、永江 孝規
芸術学研究科メディアアート専攻
インタラクティブメディア学科

Altered Tunings in A ― Distributed music play over the network inspired by the historical changes and regional differences ―

ISHIHARA Hisashi, NAGAE Takanori
Department of Media Art, Graduate School of Art, Department of Interactive Media

(Received October 29, 2021；Accepted January 12, 2022)

キーワード：調律、音律、基準音、シンセサイザー、クライアント・サーバー通信、ゲームエンジン

要旨

We developed a distributed client/server network system which plays music almost synchronously using asynchronous communication sending music data. The tuning can be changed during the performance from the standard frequency 440 Hz to another, such as 506 or 377 Hz. These changes imply that historically the tuning frequencies had been varied. The violoncello part repeats Morse signals identifying some airports, which means that the tuning could not be the same in different regions. The piano part plays twelve notes simultaneously to indicate the temperament.

1. はじめに

楽器を調律する際に用いられる基準音A、すなわち、88鍵ピアノの中央に位置するドの音のすぐ上のラの音の周波数は今日国際的に440Hzと定められており、ラジオや電話の時報の音にも使われている。この音はたとえばバイオリンでは第2弦、ビオラでは第1弦を開放弦で弾いたときに鳴るべき音とされており、おそらくは遠い古代、この音の高さ（音高）は人が演奏する弦楽器の弦の長さに由来して、必然的に決まったものと思われる。

この基準音Aは昔からおおよそ440Hzくらいの音であったようだが、それがきっかり440Hzであることにされたのは何事にも世界標準を定めることが要求されるようになった近代以降のことであり、国際標準機構ISOで発行（publish）されたのは比較的最近、1975年のことであった（2017年に審査(review)され、最終現行版であることが確認(confirm)された）[1]。Aが440Hzであることを明示するためにA440と表記されることもある。またDAW（digital audio workstation）など今日の音楽ソフトで標準的に使われるMIDI（Musical Instrument Digital Interface）では、この音高はA4（つまり、可聴音域で低いほうから4番目のA）と表記され、69番というMIDI番号が振られている。

以後特に断らない限り本稿中ではAとはA440のこととする。

著者の一人石原はオランダで毎年催される国際的な若手作曲者対象のフェスティバルGaudeamus Muziekweek [2]において、2012年にYongecomposerbal部門でAltered Tunings in Aと題する曲を上演した。この曲は元々、単純にAの周波数が時代や地域でどのように異なっているか調査する目的で作っていたが、その過程でアイデアがふくらみ､時間と場所を超える旅行、というテーマで作曲することになった。従って、以下a～eに示す、幾つかの異なるアイデアによって構成されている。

a. 場所と時代に基づいたAの周波数
b. 時代によって変遷する調律
c. チェロの刻むリズム
d. 440Hzを知らせるビープ音
e. ピアノパートにおける音程を聴かせるための音塊

本稿では、この2012年のコンサートで使われたピアノのパートを再度解釈し直し、ゲームエンジンを用いて演奏した結果について考察し報告する。ゲームエンジンはもちろん、今日私たちの認識では「楽器」ではなく、また音楽業界において普及したソフトウェアでもない。ゲームエンジンはただ単にゲームを開発するための環境である、ゲーム制作に特化したツールであるという認識が今なお根強い。

しかしながらMoogが初めてシンセサイザーを世に送り出したときにも、彼の発振器や変調器や増幅器などを組み合わせた電気回路の塊は「楽器」ではなかった。その塊に鍵盤をつなぎ、「これは楽器です」と称して売り出し、多くのミュージシャンを魅了したことによって初めて楽器となったのである。

ゲームエンジンはもはや単にゲームを作るだけのツールではなく、楽器としての機能も備えつつあるが、その機能を用いてゲームエンジンで開発されたアプリケーションは未だに少なく、ミュージシャンにもゲームエンジンは認知されていない。そこでゲームエンジンが今後本格的な楽器となり得る可能性の一端を示すことが本研究の目的の一つともなっている。

メディアどうしの融合は新たな表現分野を生み出す。ソフトウェアは、マルチメディア、ミクストメディア、メディアミックスなどという言葉に象徴されるように、媒体やコンテンツの垣根を越え、音楽や映像、ゲーム、インタラクティブコンテンツなどが一つになる方向へ支援する。ゲームエンジンが楽器となり得ることを立証できれば、それをそのまま楽器として用いても良いが、ゲームエンジンが本来得意とするゲームなりCGなりと組み合わせて用いることもできる。ゲームエンジンはさまざまなプラットフォームで多様なコンテンツ制作に用いられ、さらに今後も活発に開発用途を広げていくことが期待される。

2. 調律と音程の変遷に関する調査

2.1. 場所と時代に基づいたAの周波数

調律に用いられる基準音Aの周波数は歴史的に異なる。今日一般的なコンサートにおいて、多くの場合Aは440Hzから444Hzほどとなっているが、バロック音楽など古楽の演奏には、より低い415Hzといった周波数が用いられることもある。Aはそれぞれ地域と時代で、少しずつ上がったり下がったりしていて、演奏家もコンサートごとに合わせるのが一般的であって、特に合理的な理由は無い、といったところではないか。

また、ストラディバリウスなど、昔に作られたバイオリンは、今日のバイオリンよりも長く、440Hzでチューニングするには張力が強くなりすぎて楽器に負担が掛かる、とも言われる。つまり440Hzはそのバイオリンが作られた当時に比べて音高が高いと推測できる。

過去の音高が低かったことは予想できるが、実際の音高を調査するにはどうしたら良いか。それには当時の楽器を見るしかないのだが、実際のところ音高は、演奏家、地域、時代によって揺れており、当時Aを絶対的な音高で固定しなくてはならないという考え方もなく、楽器間でそろえる一般的な約束事も無かったため、コンサートに楽器を持ち寄るたびにその場で合わせるのが一般的だった。弦楽器は弦の張力を変えることで容易にチューニングできるが、音高を変えにくい管楽器は、長さを増やすことで（下にアジャスターのようなものを付ける事が多い）音高を低くするか、数本用意して一番合うものを使う、などの対策が取られた。

今日私たちが当時の音高を知るには、残された楽器から調査する以外にない。今日調律にもっぱら使われている音叉は1711年イギリスの音楽家John Shoreによってはじめて使用された[3]。弦楽器では音高は可変であり、管楽器は固定であるから、管楽器の音高を調査していくことになる。ルネサンス期に用いられた、確たる証拠となり得る管楽器には、コルネット、フルート、リコーダー、クラリネット、パイプオルガン、ピッチパイプなどがある。これらのうち、制作された地域や年代が正確に残っていて、現代でも調査が可能な楽器は、唯一教会に設置された大型のパイプオルガンであるので、今回の調査では教会オルガンの調査に絞ることにした。

教会のオルガンは一度設置すると容易に音高を変えることができないので、教会で演奏される場合は、他の楽器や声楽がオルガンに合わせて音高を決定するしかない。このように教会オルガンの音高が標準と見なされ、教会の外で演奏されるオーケストラの音高にも影響を与える傾向は西洋では18世紀後半まで続いた。そこでオルガンの周波数を調査することでその地域と時代で使われた音高がある程度予想できる。

中世やバロック時代に作られたオルガンは、後にルネサンス、古典時代になっても、楽器として正しく機能するように、修理や改造が加えられて、使い続けられた。しかしその改変の履歴が必ずしも残されているとは限らない。時代が移ると同じ地域でも音高が違うことがあるのはそのような改変のためであると考えられる。或いはオルガンの移設や、教会の建物自体を改修するために、パイプの長さを縮めなくてはならないこともあったかもしれないし、単に世の中が異なる音高を好むようになったために、パイプの長さが変更されたかもしれない。真実の理由は記録が残されていない以上不明であるが、と

もかく過去の音高を知るため、単に古いオルガンの現在の音高を調べただけでは不十分である、ということになる。最も信頼できる情報は近代以降になって修復されたオルガンから得られることがある。復元の過程における調査で、当初のまま保管されているパイプがいくつか発見されることがあり、それらのパイプによって最終的に全体の音高を元通りに再構築することができる[4]。

いずれにせよ、音高を調査する過程で低い音高だけでなく、440Hzよりも高い音高も多数存在していることが解った。高い音高だと567Hzとなって、現代の平均律に直すとD（587.3Hz）に近いかなり高い音高も存在する。音高がこのように時代とともに変遷していたという事実が、調査によって明らかとなり、本楽曲を作る上で大きな動機となった。

今回選んだ音高は以下の6種類である。

⑴ 1361年, The Halberstadt Cathedral, Halberstadt Germany, 506Hz
⑵ 1511 年 , Arnolt Schlick mentioned the size of pipes in his book, 377Hz
⑶ 1619年, North part of Germany, 567Hz
⑷ 1688年, St Jacob Church in Hamburg, Hamburg Germany, 489Hz
⑸ 1879年, St Jacob Church in Hamburg, Hamburg Germany, 494Hz
⑹ 1750年頃, J.S.Bach's pitch, 480Hz

これらは基本的に周波数の違いが大きく、わかりやすくなるように選んでいる。しかしながら以下に述べるような歴史的理由も個別に考慮に入れている。

1361年にHalberstadt教会に設置されたオルガンは1495年に修理され、1619年にオルガン演奏者・音楽理論家のMichael Praetoriusの著作『Syntagma Musicum』にパイプの長さが記載されており、これにより基準音が506Hzであったと推定される。比較的古くまた音高が非常に高い例であることから（1）に採用した。

Arnolt Schlickが1512年に著した『オルガン職人とオルガン奏者の鑑(Spiegel der Orgelmacher und Organisten)』は、オルガンに関する世界初の論文で、当時から現在まで重要なものとされている。1511年に作成された基準音377Hzのパイプは、Schlickの記述通りだと音がかなり低く[5][6]、珍しい例だったことから採用した。また彼の音律はピタゴラス音律に近い性質を持っているため、他のミーントーン（中全音律）との違いが解りやすくなるのも今回（2）に採用する理由となった。

1619年に作成された基準音567Hzのオルガン（3）は、確認できる資料で一番高い音高である。これはHermann von Helmholtz著『音感覚論（Die Lehre von den Tonempfindungen als physiologische Grundlage für die Theorie der Musik; 心理学的基礎としての音楽理論のための音の感覚の研究）』第4版[5]の英訳者Alexander John Ellisによる付録（p. 504）に記載されている[6]。“North German church pitch, called by Praetorius chamber pitch, taken as a meantone Fourth（503 cents）above Praetorius's ‘suitable pitch’ a’ 424.2”、とあって、場所は明記されていないが「北ドイツ教会のピッチ」、またはMichael Praetoriusによって「室内楽ピッチ」と呼ばれたことがわかる。

St. Jacob Churchのオルガンは当時最も有名なオルガン製造者の一人であったArp Schnitgerによって1688年に作られ、当初は基準音が（4）の489Hzであったが、1879年に（5）の494Hzに作り変えられたことが調査によって判明している[5][8]。

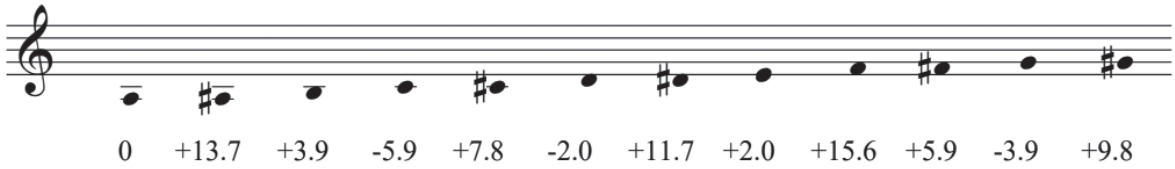
Johann Sebastian Bachが20才の学生だった1705年当時、彼が滞在していたLübeckやHamburgなど北ドイツの教会で彼が聞いたオルガンの基準音は、487Hz、484Hz、481Hzなどであった[4]。このことからBachが用いた基準音は480Hzである、という通説が生まれたものと思われる。（6）の480Hzはこれに基づく。

後にBach が主に活躍したLeipzigやWeimarなどのオルガンは声楽の伴奏に使われたためにChorton（choir pitch、声楽ピッチ）で調律されていた。この基準音はおよそ465Hzであった（ほとんど同じ、462Hzを平均とするコルネットピッチCornet-tonというものもある）。これに対して室内楽には415HzのKammerton（chamber pitch、室内楽ピッチ）が用いられることが多く、このため通常、バロック音楽の基準音は415Hzであると言われている。ほかにもフランスでは392Hzなどのより低い基準音が用いられた。

2.2. 時代によって変遷する音律

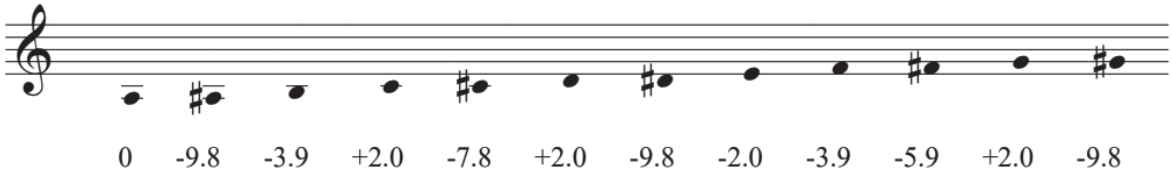
Aの高さが各時代と地域によって違うことが解ったので、次にそれぞれの音律がどの時代のどの地域で使われていたか推測するための調査を行った。その結果、本作では、前節に示した音の高さに図1に示す5つの音律を組み合わせて用いた。

Pythagorean Tuning A = 506Hz, 377Hz

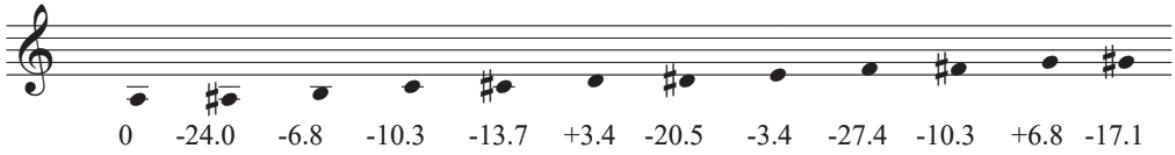


Arnolt Schlick Tuning A = 377Hz

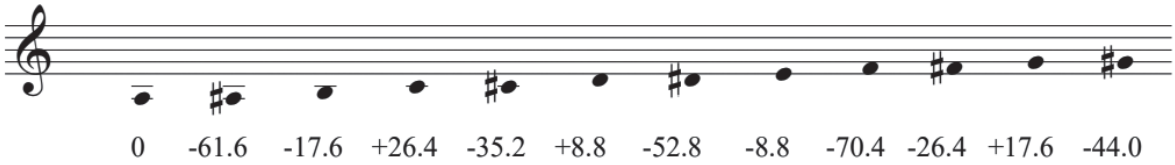
- which comes from a manuscript, Spiegel der Orgelmacher und Organisten, written in 1511



Aaron's 1/4 Meantone Tuning A = 489Hz



1/2 Meantone Tuning A = 494Hz



Werckmeister III Tuning A = 480Hz, 567Hz

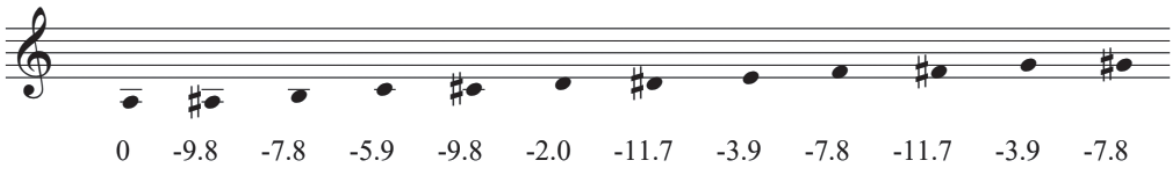


図 1. 本作に用いた音律

2.2.1. *ピタゴラス音律*

ピタゴラス音律の呼称はAD2世紀に数学者で音楽理論家であるNicomachus of Gerasaが『Enchiridion harmonices（音響便覧)』に記した逸話「ピタゴラスの金槌」に由来する。ピタゴラスは鍛冶屋の前を通りすがり、金槌の音に協和音と不協和音があって、その違いが金槌の重さの比率によって生まれていることを発見した。この伝説の真偽はともかく、西洋音楽においてピタゴラス音律が現在確認できる最も古い音律であることは確かであり、中世西洋音楽から初期ルネサンス期まで使われていた。この音律は完全5度（周波数比2:3）と完全4度（周波数比3:4)、そして8度（周波数比1:2）の音程を上下に拡張して作る。グレゴリオ聖歌はもっぱらこれら1:2、2:3、3:4の音程を用いて歌われた。ルネサンス期に入ると西洋音楽はグレゴリオ聖歌のようなモノフォニー（単声音楽）からポリフォニー（多声音楽）へと発展していき、3度（長3度は4:5、短3度は5:6）や6度（長6度は3:5、短6度は5:8）の使用が増え、3度をより純正に近づけるため5度を狭めるミーントーンがピタゴラス音律に代わって使われるようになる。

2.2.2. *ミーントーン音律*

イギリス人作曲家のJohn Dunstaple（1390 - 1453）が3和音を使い始めるまで、長3度は濁った音とされ使われなかった。Dunstaple の音楽はイギリス本島の作曲家だけでなくブルゴーニュ楽派に影響を与え、その後ヨーロッパに広く伝わった。イタリアやスイスでもDunstapleの楽譜が残っている。

Dunstaple以降、4度や5度とともに3度も美しく響かせる音律が必要になる[9]。たとえばドミソの和音でドとソは2:3だが、ドとミは64:81、ミとソは27:32になる。ド−ソの純正度をやや犠牲にして、ド−ミ、ミ−ソの純正さを上げることでドミソ全体の純正さを上げ、和音を美しく響かせる、これがミーントーンの考え方である。ピタゴラス音律や純正律には全音（whole tone）に大全音（major tone, 8:9）と小全音（minor tone, 9:10）の2種類があり、それらの中間の全音を使うのでミーントーン（mean tone、中全音）というのである。

思うに古代、西洋のピタゴラス音律、中国の三分損益法、或いはヴェーダに記録された音律などはいずれも1:2、2:3、3:4の単純な整数比を繰り返して作られたものだが、それ以上微妙な音程で調律することは当時不可能だったのだろう。「「無個性」で「神秘的」で「ただただ美しい」」「音楽の本質的実体をなしているのは実際に耳に聞こえる音響現象という表層ではなく，むしろその響きを根底において律している数的秩序といういわば深層だ」[9]などと言ったカルト的理由でピタゴラス音律が用いられたというより、古代には単に高度な調律方法（うなりの周期を用いるなど）[10]が知られていなかっただけではないか。そしてミーントーンが登場する時代になって初めて、4:5や5:6、或いはもっと込み入った比率でも調律できるようになったのではなかろうか。

ミーントーンについてまず言及したのはFranchinus Gaffurius『Practica musicae』（1496） とPietro Aaron『Thoscanello de la musica』（1523）である。それ以前から局所的にミーントーン音律は使われていたと考えられ、また、この頃から広く一般的に使われるようになったものと思われる。

プロテスタントに対抗してカトリック教会は1545年から1563年にかけてトリエント公会議を開き、典礼における音楽を世俗音楽から分離した。ミーントーンは世俗音楽に始まり教会音楽にも採られるようになったが、当時すでに教会音楽にもポリフォニーが普及しつつあったことから、ミーントーンをやめてピタゴラス音律に戻すなどという時代錯誤な制限は課されなかった。

ミーントーンといってもさまざまな種類があり、歴史的にどの地域、年代によってどのミーントーンが使われていたかは大まかな推測しかできていない。今曲では違いの出やすいもの、即ち1/2と1/4を選んでいる。

2.2.3 *Schlickの音律*

ルネサンス時代にプファルツ選帝侯領Heidelbergの宮廷で活躍した作曲家Arnolt Schlickは前述の彼の著書『Spiegel der Orgelmacher und Organisten』において音律についても言及しており、「Schlick音律」と呼ばれている。それはPietro Aaronの1/4コンマミーントーンとほぼ同時期のもので、そのコンセプトもミーントーンとほぼ同じものである。すべての3度を完全に純正かほとんど純正に調律した結果、ミーントーンやピタゴラス音律同様に、うなりを感じるほどに広い5度（狼の5度）を含む[7]。

2.2.4. *Werckmeisterの音律*

Bachは『平均律クラヴィーア曲集』を作曲したので、彼は平均律を用いていたと思われがちだが、Bachが自ら演奏するときに用いた音律は、時代的に、平均律が現れる直前に Andreas Werckmeister が考案した音律であった可能性が高い。Andreas Werckmeister は Bachより40年ほど前に活躍したオルガン演奏者・音楽理論家である。ピタゴラス音律や純正律が音程に有理数だけを用いるのに対し、ミーントーンやWerckmeister音律は平方根や3乗根、4乗根も用いる。平均律は12乗根を

用いる。これら累乗根の正確な数値は、スコットランドの数学者John Napierが対数を発見するのを待たねば得られなかったはずだ。

Werckmeister音律はミーントーンでも平均律でもなく、Well-temperament（Wohltemperierte, Well-tempered）と呼ばれる。『平均律クラヴィーア』はWell-tempered Clavier、つまり良く調律された鍵盤楽器という意味であって、これを平均律と呼ぶのは誤訳であろう。

Napierが対数を発見したのが1614年、WerckmeisterがWell-temperedな音律を考案したのが1691年、BachがDas Wohltemperirte Clavierを作曲したのは1722年から1742年にかけてである。Napierの対数発見前から対数や平均律についての素朴な概念はあったに違いなく、リュートやギターなどのフレット式弦楽器では古くからフレットを対数間隔で（つまり対数的に等間隔に）配置していたはずだ。一方、鍵盤楽器が平均律で調律されるようになったのはずっと後のことであると考えられる。

3. 作品紹介

3.1. チェロの刻むリズム

前節の2.2では、古い教会オルガンの音高に対して、それが制作された時代に対応する音律を当てはめた。一方で、オルガンが設置された地域を表現するために、地域を表すリズムを当てはめることにした。本曲におけるチェロは地域を表すリズムで基準音の音高を奏でる。チェロのリズムを地域の識別に用いるために、空港コードをモールス信号に変換してリズムとして用いた。今回用いた空港コードは次の6つである。

⑴ Paris Charles de Gaulle Airport -- CDG
⑵ Magdeburg–Cochstedt Airport -- CSO
⑶ Mannheim City Airport -- MHG
⑷ Kiel Airport -- KEL
⑸ Hamburg Airport -- HAM
⑹ Leipzig/Halle Airport -- LEJ

もちろん教会オルガンの場所は、必ずしも空港の位置とは一致しないので厳密に位置を特定するものではない。またBachは各地のさまざまな音高のオルガンを演奏しているため、彼が用いたと思われるWerckmeister音律を、彼が主に活躍したLeipzigの空港に割り当てるなどした。

本曲の音律と基準音と空港の対応関係をまとめると次のようになる。

⑴ 1316年, The Halberstadt Cathedral, Halberstadt Germany, 506Hz --- Pythagorean Tuning -- Paris Charles de Gaulle Airport -- CDG
⑵ 1511年, Arnolt Schlick mentioned the size of pipes in his book, 377Hz --- Arnolt Schlick Tuning, Pythagorean Tuning --- Magdeburg–Cochstedt Airport -- CSO
⑶ 1619年, North part of Germany, 567Hz --- Werckmeister III Tuning --- Mannheim City Airport -- MHG
⑷ 1688年, St Jacob Church in Hamburg, Hamburg Germany, 489Hz --- Aaron's 1/4 Meantone Tuning --- Kiel Airport -- KEL
⑸ 1879年, St Jacob Church in Hamburg, Hamburg Germany, 494Hz --- 1/2 Meantone Tuning --- Hamburg Airport -- HAM
⑹ 1750年頃, J.S.Bach's pitch, 480Hz --- Werckmeister III Tuning -- Leipzig/Halle Airport -- LEJ

ここで、筆者らが今回テーマとした楽曲に先行事例があるかどうか、ということについて考察しておきたい。

ピッチベンドもしくはピッチシフト（ピッチスケーリング）は曲の演奏中に音高を変える操作である。ピッチベンドはピッチホイールなどのデバイスを回転させて音の高さを変える。通常はビブラート奏法を行うための仕組みであるが、一時的に一つの音符のピッチをずらすことにも使える。一方でピッチシフトは演奏の速さは変えずに全体の音高を変更し、一オクターブ上げたり下げたりした音を重ねたり、音の高さがずれている録音をそろえてミキシングする時などに使われる。ペダルを踏んでピッチを変えるなどの奏法もある。いわゆるボイスチェンジャーもピッチシフトの一種である。

本作で用いた手法はピッチシフトに近い。しかしながらピッチシフトを積極的に活用した楽曲は、皆無ではないとしても極めて稀であろう。転調（特に長調から長調、短調から短調の転調）は一種のピッチシフトであって、古典音楽でも頻繁に用いられる手法であるが、平均律の範疇で調を変えるものである。例えば周波数を 5/4倍すればハ長調はヘ長調になり、3/2倍すればト長調になる。しかし音律をピタゴラスからミーントーンやその他の歴史的音律に変えるという曲、またはそれに近い試みを行った曲を筆者らは現時点で発見することができなかった。

モールス信号を利用した楽曲の事例は数多くある。比較的古い例としてはIgor Stravinskyが1953年に作曲を開始したバレエ音楽『Agon』がある[11]。Kraftwerk

が1975年に作曲した『RadioActivity』や、Rushの1981年の曲『YYZ』では非常にわかりやすい形でモールス信号が使われており、また日本のポップスでも1977年に発表されたピンクレディーの『SOS』の冒頭にモールス信号が使われていることはよく知られている。

3.2. 440Hzを知らせるビープ音

本作はAが440Hzである、という前提を崩すために書かれている。従って聴く際に常に440Hzを意識される必要があるため、一番解りやすい通知音であるところのビープ音を440Hzの音高で常時鳴らすことにした。

3.3. ピアノパートにおける音程を聴かせるための音塊

ピアノによって音律の違いを聴かせるためには、12音全てを同時に鳴らす必要があるが、そのままクラスターで鳴らした場合、人間の耳はただのクラスターとして認識するため違いを知覚できない。

そのため、低い音域において長短3度音程を避けた上で、12音が鳴る、かつ様々な音程がきちんと知覚できるようなクラスターとして以下のような分散を設定した。

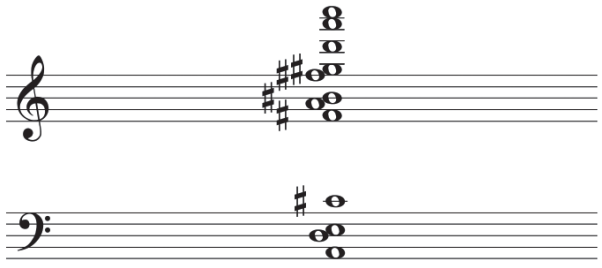


図2. 本作に用いた音塊

まず、図2にある一番下のAから完全音程である4度(D)、5度（E）としたのは、周波数の振動比率が3:4及び2:3と割り切りやすい比率であり、音律の違いによってその振動比率が少しずつずれた時に、人間に知覚しやすくなる、つまり全ての音律の違いが解りやすくなる、という目的のために配置した。その1オクターブ上に、音階の一部と知覚されないように音を配置していった。

例外として一番上の2つの音程は3度であるが、高い音域において、長短3度の響きがあっても、一般的に人間の耳には和音として認識されない。周波数の振動比率として単純化できても、高音になるにつれ実際に周波数が大きくなれば、和音として認識されにくくなるからである（低い音域において和音を鳴らしても、周波数が低すぎて和音として認識されない現象、即ちLow interval limitと呼ばれる現象と同じと言える)。

また、図2のように基音のAが低い位置にある場合、2オクターブを超える音は、もはやジャズで使われる9度、11度のように和音として認識されず、倍音の一部として認識されるため、3度音程があったとしても和音として認識される可能性は低い。

ピアノパートを演奏するに際して、

a. 音塊の形は崩さない
b. 音塊のオクターブ上下は自由（ランダム）
c. 各音の大きさは、すべての音が聞こえる必要があるため、ある程度の幅の制限はあるが基本的に自由(制限付きのランダム）

とした。

ピアノパートはLogic Proで音律とピッチを変更しつつ、MIDIデータをピアノのサンプル音で鳴らし、wavファイルに録音している。

4. ゲームエンジンによる演奏

4.1. シンセサイザーと音律

現代音楽は無調音楽である。それは必ずしも単なる不協和音の羅列ではない。どのような音律を用いるか、1オクターブを何段階にどんな音程に分けるかということと、和音が協和するかどうかということは関係ない。現代音楽を奏でるためのソフトウェアに求められる機能もまた、勝手な周波数で音を鳴らすことができるという、ただそれだけではない。美しく響く音、つまり、音楽理論に基づいたなんらかの規則性を有する音を鳴らす装置が求められているのである。それはつまり新たな音律を定義することと同義である。そのためいわゆる微分音（microtonal music）を積極的に活用することになる。すでにバロック音楽の時代から12平均律の他にFrancisco de Salinasによる19平均律やChristiaan Huygensらによる31平均律[12]、Robert Holford Macdowall Bosanquetによる22平均律[13]、Joseph Sauveurによる43平均律[14]、そのほかありとあらゆる平均律が考案されてきた。

弦楽器は弦を張る強さを変えることで比較的容易に音律を変更することができるが、周波数や波形を自由に形成、改変できる電子機器やソフトウェアのほうが、より正確で複雑な音律を作り出すことができるために、現代音楽で用いられる機会も多い。『Switched On Bach』（1968）で一躍先駆的シンセサイザー奏者となったWendy Carlosもまた、数々の音律を独自に研究してい

たことで知られる[15][16]。

音律を変更し編集できるDAWもすでにいくつかある。Logic Proはプロジェクト設定においてミーントーンやWerckmeisterなど歴史的な音律のほか、Baliや中国など世界各地の音律を一覧から選択して用いることができる。また、それらの一覧に無い音律も自由に編集して、プリセットとして保存することができる。また弦楽器などで行われている適用的チューニングを鍵盤楽器でも自動的に行えるようにするHermode Tuning（HMT）機能も備えている。Cubaseもプロジェクト設定でHermodeチューニングのタイプや深さを設定することができる。Bitwig Studioには図3に示すようにMicro-Pitchという「ノートエフェクト」が用意されており、Logic Proと同じようにピッチを上げ下げしたり、Werckmeister、Euler、Wendy Carlos Optimized、Pythagoreanなどの西洋音楽史的、あるいはArabicやChineseなど世界各地の伝統音楽に基づくもの、またはその他さまざまなプリセットされた音律を選ぶことができるほか、自らその数値を設定して保存することができる。

音律を選んだり自分で編集する機能が無いDAWでも、ピタゴラス音律、ミーントーン音律、その他さまざまな音律や微分音を使えるようにするVSTも多数ある[17][18]。また、ヤマハやカワイ、ローランドなどから音律を選択できる電子ピアノが販売されている。

4.2. 通信プロトコル

本システムはクライアント・サーバー通信による分散環境で動く。サーバーがハブとなって複数のクライアント間で譜面（音楽データとタイムライン）を共有し、音はクライアント側で鳴らす。

まずMIDI通信に使われることが多いUDPやRTPと、Web などで汎用的に使われるHTTPやTCPなどのプロトコルを比較する。本システムでは必ずしもMIDIデータのみ通信することを想定しているわけではないが、楽曲の演奏に用いる以上、その通信データのほとんどがMIDIもしくはその類いのnote（音符）のデータになるのは当然である。また楽曲制作には通常複数の音楽ソフトを使い、それらで楽譜を共有するには事実上MIDI形式を用いるしかない。MIDIの後継フォーマットとしてMIDI 2.0やOSC（OpenSound Control）などが提案されているものの、必ずしも普及しているとは言いがたい。本作でもこのような状況に鑑みて、MIDIを主に使い、必要に応じてMIDI形式を拡張し、もしくはMIDIとは別にデータを補って使う。

4.2.1. *UDP*

従来はMIDIコントローラーをMIDIケーブルで直接パソコンにつないで打ち込んだり演奏した。やがてMIDIケーブルよりもUSBケーブルでつなぐほうが一般的になった。さらにはLANでも接続できるようになった。現時点、MIDIコントローラーどうし、もしくはコントローラーとPCをLAN経由で接続する際に用いられる標準的なプロトコルは RTP-MIDI である[19]。RTP-MIDIを実装したインターフェイス機器としてiConnectivityなどがある。

前述のOSCはUDPを通信プロトコルとして使う。UDPもTCPと同様にインターネット上のクライアント・

3(a) プリセットに登録された音律を選択しているところ。

3(b) ピッチや音程を編集できるだけでなく、LFO（低周波発振器）などのエフェクトで変調することもできる。

図 3. Bitwig Studio 4.1 Micro-Pitch。

サーバー通信に用いられ、下位レイヤーでIP層を共有する。UDPとTCPはいずれもIP層の上位層、トランスポート層に位置する。

TCPは輻輳処理を行い、パケットの到着を保証し、パケットが途中で損失したり遅延した場合には再送する。パケットの到着順も保証される。UDPはパケットの到着を保証しないし、再送もしない。クライアント側から見るとUDPのパケットはいきなり何の前触れもなく送りつけられて、それきりである。パケットが到着する順番が入れ替わることもある。その代わり、TCPよりも速い。このようにTCPは正確であるが速度は必ずしも速くなくてよい通信に、UDPは速さを優先し、必ずしも正確さを必要としない通信に用いられる。

OSCがUDPを使用しているのはおそらく、従来のMIDIケーブルやUSBケーブルによってMIDIコントローラーとPCを接続するような、1対1の一方的な通信を想定しているからだろう。しかし本作では当初から、弦楽四重奏曲のような、複数のクライアント（楽器）どうしで連携するアンサンブル演奏を想定している。自律的機能単位（オブジェクト）どうしが相互に会話して処理するのがオブジェクト指向の計算モデルであるが、筆者らも究極的にはそのようなアンサンブルシステムを作りたいと考えている。

UDPを用いてパケットロスが発生し、鳴らない音がでるのは、実時間性を最優先するならば許容範囲かもしれない。しかしUDPでMIDIデータだけを送った場合には､楽器間で演奏にずれが生じる。もちろんMIDIデータ以外を送信し、到着を確認したり再送したりすることで、開始時刻を合わせたり経過時間を共有することはできるが、そのための余計なデータ量が増え処理も複雑になり、そもそもUDPを用いるメリットが失われる。TCPはパケットが失われた場合は再送される。再送も失敗した場合には接続自体が失われる。UDPにはTCPのようにサーバーとクライアントがハンドシェイクして接続を確立するというプロセスはない。

4.2.2. *RTP*

UDPを下位層とするRTP（Real-Time Transfer Protocol）はUDPヘッダに独自のRTPヘッダを付けたパケットをサーバーからクライアントへ送る。今日IP電話やIP放送、また遠隔会議システムなどにも、もっぱらRTPが使われている。RTP-MIDIもまたそのようなRTPの特性に基づいた拡張プロトコルの一つである。

RTPもUDP同様、接続を確立するとか接続が失われるという概念はない。クライアントはある識別子をもつ相手から送られてくるパケットのRTPヘッダから、その通し番号とそのパケットが作られた時刻等を得るだけである。RTPプロトコルによる通信では、仮に途中でいくつかのパケットが失われても、受け取り側で、正確な順序と時間間隔を復元して、音声や動画を再生することができる。たとえパケットロスが発生しなくても、期待された時刻までに届かないパケットは破棄される。

RTPは音声や動画などの連続的なデータをサーバーから一つまたは複数のクライアントへ伝送するためのシンプルなプロトコルであり、ストリーミング専用のRTSP（Real-Time Streaming Protocol）などもある。またクライアントからサーバーへ、画質調整やセッション参加者の識別情報、ログアウトなどのリクエストを送るには、別にRTCP（Real-Time Control Protocol）を用いる。

RTP-MIDIはたしかにMIDIデータだけを送受信するプロトコルとしては優れた面を持っている。さまざまなインターフェイスも実装され市場に流通しているので、MIDI端末をLANでつなぐには最適である。無線LANやBluetoothで電子ピアノをスマホのアプリから操作する、などと言った用途に適している。

しかしながら筆者らが求めているのは従来の音楽機器の代替ではない。ゲームエンジンをベースとしてそのコンポーネントとして楽器が使えるシステムを作りたい。或いは、DAWなどの音楽機器の中にゲームエンジンを仕込むとしても、実際にやらなくてはならないことは同じことである。現状、オーディオワークステーションとゲームエンジンはほとんど別の用途に使われているが、両者の統合が実現すればさまざまな新しい表現が可能になる。

4.2.3. *TCPとUDPの組み合わせ*

ゲームエンジンの世界を見ると、多くの高品質なゲームは、TCPとUDPを組み合わせ、しかも同時にそれぞれのプロトコルで複数のポートを占有して通信を行っている。各ゲームでどのようにプロトコルやポートが使われているか、詳細は不明だが、おそらく、勝敗の決定に関わるクリティカルな当たり判定、例えばプレイヤーが敵の弾に当たって死ぬかどうかといった、ごく一部の処理には、信頼性を重視し、通信コストをかけてTCPを使い、もしくはUDPを使っていてもより確実な判定をする必要があるときには、やはり計算コストや通信コストをかけてTCP等で補完する、などという手法が採られていると考えられる。RUDP（Reliable UDP）はこのような目的のためにUDPを拡張したプロトコルの一つである[20][21][22][23][24][25]。

一方で、必ずしも正確さを要しない処理に関しては

TCPを使う必要がない。単に背景を描画するといった処理にはUDPが用いられることもありえるし、BGMを流すにはUDPポートを介して RTPが使われているかもしれない。

Webの世界を見ると、TCPを使わずUDPを独自拡張して信頼性を確保しようとするプロトコルに、GoogleがWebアクセスを高速化するために開発しているQUIC[26]などがある。QUICではTCPのハンドシェイクにあたる認証をTLS（Transport Layer Security）で行い、また同時にTLSによって暗号化や改竄の検出を行う。

4.2.4. *REST*

Unreal Engineはマルチプレイヤーのネットワーク対戦ゲームに使われてきた長い歴史と実績があり、多くのユーザーが恩恵を受けている。しかしそのシステム開発には非常に高いハードルがある。レイテンシーが低い（遅延が少ない）通信を実現するためには、TCPとUDPを複雑に組み合わせた独自プロトコルを用いるサーバーをC++でコーディングしなくてはならない。研究段階、もしくは開発の初期段階においていきなり、こうした実用レベルにチューニングされたシステムを実装することは、必ずしも得策ではない。むしろ柔軟性や拡張性を持ったプロトコルを用いていったんプロトタイピングを行ったほうが開発効率を上げることができる。そこで筆者らは通信にREST（REpresentational State Transfer）と呼ばれるWebベースのデータ通信を用い、また開発にはBlueprintというビジュアルスクリプティングツールだけを使うことにした。

インターネットで現状最も汎用的な目的に用いられているプロトコルは、言うまでもなくTCP/IPスタックのアプリケーション層に位置するHTTPである。RESTはHTTPプロトコルによってJSON（Javascript Object Notification）形式のデータをやりとりする。Web則ちHTTPはTCPを使っているので、本作はトランスポート層にTCPを、アプリケーション層にHTTPだけを通信に使っていることになる。

Webにおけるクライアント・サーバー通信はHTMLやCSSはもちろんだが、Javascriptを用いてXMLやRSSなど、テキストベースのデータをやりとりしている。画像や動画などといったバイナリデータも往々にして可読性があるJSONにエンコードして送受信されるのは、それだけJSONというテキストデータ形式が簡易だからであり、また今日ではHTTPでデータを圧縮して送受信するのが一般的になったからでもある。

JSONに近いデータ形式にはXML、RSSなどがあり、それらのいずれが世界標準規格となるか議論されてきたが、標準化とあまり縁の無さそうなJSONが結局生き残った。Twitter APIやGoogleのJavascript APIなどもすべてJSONをRESTでやりとりしている。JSONがここまで使われる理由は、JSONとJavascriptの親和性にある。JSONはJavascriptの連想配列そのものである。JavascriptはCSSとともにHTMLの規格の一部であり、Webそのものと言ってよい。AJAX（Asynchronous JAvascript XML）も、その略称の中にXMLを含むとは言いながら実際にやりとりされるデータ形式はXMLではなくJSONであり、クライアントからGETやPOSTメソッドでサーバーにリクエストを送ってデータを取ってくる。それゆえに誤解を恐れずに言えば、AJAXとはRESTそのもの、JSONそのものである。ページ遷移不要なWeb上のGUIを作ろうと思えばAJAXを使うことになる。今、Webの世界で使われているクライアント・サーバー間の非同期通信とはRESTであると言い切ってもあながち間違いではない。

RESTがWeb業界の事実上の標準であるということはユーザーも開発者もドキュメントも多く、対応するプラットフォームや開発環境も多く、その開発のスピードや頻度も他と比べて格段に速く、淘汰圧も高いことを意味する。GoogleもAppleもMozillaもMicrosoftもしのぎを削って Javascript エンジンを最適化し高速化しているのは、Javascriptを制するものがWebを制するからに他ならない。今まさに皆が使っている技術を使うことには非常な恩恵があり、今後こうした傾向は続いていくと考えられる。Blueprintも当然JSONをサポートしている。

ともかくも、筆者らは誰もが使うWebベースの通信を採用しようと考えてRESTを検討し、実際すぐに適当なツールが揃ったので、ある意味必然的にそれらを使うことになった。

Blueprintは煩雑なC++プログラミングからアーティストやクリエイターを解放するためにEpic Gamesが提供する直感的でわかりやすいツールである。大規模なリリース製品よりは、プロトタイプのマッシュアップ開発に適した環境と言ってもよい。より多くの、必ずしもプログラミングを得意としない人たちを開発に巻き込むために、BlenderやUnityなどでも近年はビジュアルスクリプティングを積極的に採り入れつつある。

筆者らはBlueprintでも利用できるVaREST [27] というREST用のプラグインを用いて通信することにし、クライアント側だけをUE4のBlueprintで作り、サーバー

はPython3のスクリプトで書き、Ubuntu Linux上で動作させることにした。

Pythonには標準でREST用のライブラリが用意されている。またMIDIファイルを読み込むライブラリも使用できる。PythonにはPythonの連想配列があるのだが、Python は自前の連想配列をJSON形式に変換して送受信することができる、というわけである。クライアント側のUE4 VaRESTプラグインと組み合わせることによってきわめて速やかにMIDIデータをやり取りするREST通信をマッシュアップすることができた。

4.3. シンセサイザーの開発環境

単にソフトシンセを設計するには、すでにDAWがあり、VSTがある。DAWやVSTを開発するための開発環境もある。または、MaxやProcessing、TouchDesigner、Super Colliderなど、アーティストやミュージシャン、デザイナーが使いやすいように用意されたツールもある。それらと比べてゲームエンジンは必ずしも「音楽ソフト」として広く認知された存在ではない。インタラクティブアート制作にゲームエンジンを用いる例もあまり見当たらない。近年Unityがその汎用性ゆえにインタラクティブアートに用いられることが増えてきた。Unityはアートやデザインのさまざまな可能性を試すことができる便利なツールだが、残念ながらUnityには現時点でシンセサイザーの機能はほとんど無いに等しい。Unreal Engineは他のゲームエンジンに先駆けて音楽を扱おうとしている。

現時点で、わざわざゲームエンジンでソフトシンセを開発しようという者はごく限られている。しかしゲームエンジンには従来の音楽機器や音楽ソフトとは違って、楽器をもっと多様な用途に広げる可能性がある。MaxやSuper ColliderやDAWやVSTをゲームやCGに転用するより、少なくとも筆者らにとっては、ゲームエンジンに楽器の機能を持たせるアプローチのほうが手っ取り早い。

「テクニックが先にあるんじゃなくて、それにはどういうものを使ったらいいのかというふうに考える」「こういう技術が出来た。これを使わないと時代に遅れちゃうからこれを使って何か出来ないかと考えても何も出てこない」世界的シンセサイザー奏者で立体音響の先駆者でもある冨田勲が芸術科学会誌のインタビューに答えた言葉である[28]。彼が設計した東京ディズニーシー アクアスフィア・テーマミュージックは6チャンネルの立体音響で、昼の部のBGMはロンドンフィルハーモニーが3回に分けて演奏し、1回あたりの演奏に48チャンネル、合計144のソースが使われた。最終的にこれらを 6チャンネルにミックスダウンするのにSteinberg社のNuendo（同社が提供する業界標準的なDAWであるCubaseの上位版で、ポストプロダクション用に特化している）が使われた。「ディズニーシーの音楽もNuendoが出てきたので、やっとできるようになった表現です」と冨田勲は語っている。この楽曲のリリースは2002年。Nuendoが最初に世に現れたのは2000年のことであった。以来Nuendoは音楽業界最高峰のDAWとして君臨し続けている。

新しい技術は常に生まれてくるが、それらの最新技術を使っただけでは、冨田勲が到達した立体音響システムを超えることはできない。だが、「これを使って自分の表現したいものが出来るんじゃないかと思うテクノロジーの情報を集めてやってみる。」たとえばPCをネットワークで組み合わせ、ゲームエンジンでプログラミングすればNuendo以上のDAWを、冨田勲が駆使したシンセサイザー以上のシンセを、あるいは立体音響システムを組めるかもしれない、そう考えるのは単なる夢想ではなく、おそらく冨田の思想を継ぐことになるだろう。

1964年に発表されたMoogシンセサイザー以来、トランジスタや抵抗などの素子で電気的に作られた発信回路や変調回路に鍵盤を組み合わせた装置が楽器として市販されるようになったが、このようなアナログシンセはやがてコンピュータとDA（digital to analog）変換器に置き換えられ、ICチップにまとめられてFM（frequency modulation）シンセサイザーとなり、家庭用ゲーム機の基板にも搭載されるようになった。今日では専用ハードウェアからPCやスマホなどで動くソフトシンセに主流は移りつつあり、DAWのプラグインという形で、有料もしくは無料の、数えきれぬほど多くのシンセサイザーが市場にあふれている。

ほんの数年の間でソフトシンセもDAWも様変わりした。誰もが容易にシンセの開発に手を出し、有象無象のシンセがDAWのVSTという形で巷に洪水のように溢れかえっている[29]。このような状況を誰が予測し得ただろうか。

さらに最近、JUCE [30]のようなDAWやVST専用の開発環境が現れ、また、本稿で採り上げるUE4（Unreal Engine 4）のModular Synthや、次世代のUE5（Unreal Engine 5）のMetasoundのようなゲームエンジン上でプログラム可能なシンセサイザーが利用できるようになり、楽器を演奏する能力は持たなくとも、プログラミング技術とアイデアさえあれば、これまでになかったような音楽表現ができる可能性が出てきた。

ソフトシンセの最先端を走っているのはおそらくNative Instruments（NI）社が提供するReaktorである。ReaktorはNI社のDAWであるKomplete用に開発された単なるプラグインではなく、シンセを自ら音楽家が設計できるようになっている。プラグを抜き差しして、配線によって発信器（オシレーター）や変調器（モジュレーター）、ノイズ発生器などの電気回路と、イコライザーやコンプレッサー、ディストーションなどのエフェクターを組み換え繋ぎ変え組み合わせて音色をデザインするのがいわゆるanalog modular synthesizerであるが、Reaktorではアナログモジュラーシンセのパッチワークを模した、ビジュアルスクリプティングによく似たGUIによって、シンセの構成要素を自在に組み換え、オリジナルの音色、つまりプリセットを作る。多くのソフトシンセとは単なるプリセット、もしくはその寄せ集めにすぎないが、Reaktorはそのプリセットを無限に作ることができる。

またBitwig Studioが提供するGrid（Poly Grid + FX Grid）も、GUI ベースのモジュラーシンセで、Reaktorによく似たシステムであり、やはり、アナログシンセサイザーが物理的に配線をパッチワークして行ってきた音色作りを可能な限りGUI上でできるようにしようとするものである。ソフトウェアとDAコンバーターで任意の波形を生成できるようになっても今なおRobert Moogが考案した電気回路モデルが好まれ続けているのは、音楽家にとって何かしら直感的で、必然性があるからなのだろう。

ReaktorやGridは多機能ではあるが、あくまでもGUIであって、開発環境ではなく、ソフトの中身、つまりソースコードを直接いじることはできない。コーディングと組み合わせることもできない。一方、DAWやソフトシンセ開発専用のJUCE自体はGithub上で管理されるC++の開発環境そのものであり、シンセでもDAWでもない。本稿で用いたUE4もまた開発環境であるが、同時にその環境の中にシンセのコンポーネントModular Synthが埋め込まれており、そのシンセはGUIによって設計することができ、またソースコードと組み合わせて使うこともできる。

JUCEは本格的なプログラミングの技能を必要とするが、UE4 Modular Synthならばビジュアルスクリプティングだけでコーディングできるのではるかに簡易である。さらにUnreal Engine 5（UE5）のMetasoundはそこからさらに一歩を進めた、ReaktorやBitwig Gridに似たGUIを備えた高機能なモジュラーシンセである。UE4 Modular SynthもUE5 Metasoundも、現時点でシンセサイザーとしての完成度はReaktorやGridにはまだまだ及ばない。しかしながらネットワーク通信や情報処理演算、VRシステムやその他さまざまな入出力デバイスと組み合わせることが容易なため、拡張性が高い。

DAWもゲームエンジンも素材（アセット）やプラグインを組み合わせる道具には違いないが、ゲームエンジンは作った音をゲームやリアルタイムレンダリングに利用できる。ゲームとDAWの融合というのみならず、人工知能（AI）と組み合わせた自動演奏、もしくは人とAIのコラボ演奏、或いはインタラクティブアートなども魅力的な発展形として考えられ、今後単なる音楽ソフトという枠を超えてシンセを発展させてくれると期待される。

要約するならば、もし私たちが最先端のDAW + VST環境を用いてシンセサイザーを設計するならばその選択肢としてはReaktor か Grid がある。開発環境からシンセサイザーを作るならば、JUCEかUnreal Engineがある。GUIベースの開発環境としてはおそらく現時点ではUnreal Engineしかない。なによりもUnreal Engineを用いれば高品質なゲームやリアルタイムレンダリングの制作と連携させることができる。筆者らがUnreal Engineを選んだ主な理由はこれである。

4.4. サーバーの仕様

当初筆者らはサーバーもUnreal Engineで開発しようとしたが、実際にはPythonを使ってごく簡単なサーバーを試作するにとどまった。その理由はすでに4.2節で述べたように、オンラインゲームで実際に使われているネットワーク機能をいきなり使うのはあまりにも敷居が高いからである。本システムのサーバーの役割は基本的には今のところ、MIDIファイルを読み込み、それをJSONにエンコードしてクライアントからGETリクエストが来るたびにレスポンスとしてそのJSON化されたMIDIコードを返すだけである。Python の MIDIライブラリには mido [31]を使っている。またMIDIをJSONにエンコードもしくはデコードし、REST通信を行うためのWebパッケージに Flask [32] を使用している（Flask自身にWebサーバー機能があり、JSONをエンコード・デコードし、REST通信を行う機能がある）。

なお、midoは独自にMIDI over TCP/IPというクライアント・サーバー通信機能を実装している[33]が、これは当然、クライアントとサーバーがともにmidoを用いて書かれたPythonスクリプトどうしであることが前提なのであって、やりとりされるデータもMIDIのバイトデータそのままであり、今回のようなPythonとUE4の通信には利用できない。

MIDIファイルに記述された譜面には当然複数の楽器

（パート）が含まれる。これらのパートはそれぞれのクライアントに一対一に割り当てられる仕様となっている。それぞれのクライアントには一意のIDが割り当てられており、IDごとにそれぞれのパートのMIDIコードが順次送られる。

4.5. クライアントの仕様

クライアント側のシンセはすべて共通で今のところ1種類しかなく、前述のようにUE4標準のコンポーネントModular Synthを用いて作られており、複数のプリセット（音色、波形、立ち上がり、減衰、うなりなど）を切り替えられるようになっている。

図4に示すGUIによってクライアントを起動してプリセットを選び、サーバーに準備が完了したというリクエストを送る。これに対してレスポンスが返ってくるとクライアントは演奏開始を待つ状態になり、演奏開始時刻が送られてくるまでリクエストを繰り返し、開始の合図を待つ。

そうしてクライアントを次々に立ち上げていき、最後のクライアントを立ち上げたら、どれか一つのクライアントから演奏開始のキューを送る。REST通信は非同期であって、サーバーとクライアントがリクエストとレスポンスをやり取りするタイミングはクライアント間でばらばらにずれているので、その時間差込みの開始時間をそれぞれのクライアントに送る。これによって（ネットワークの遅延の差を除けば）すべてのクライアントで一斉に演奏が開始する（非同期通信の範囲で開始時刻を可能な限り同期する）。

UE4 VaRESTプラグインはそれ自身がWebクライアントとなってJSONを受け取り、デコードして演奏開始時刻、チューニング変更のタイミング、MIDIコードなどのデータを取り出す。

UE4のModular SynthはMIDIとの相性が良い。例えば一定時間指定の音量（velocity）で、MIDIコードで表された音程の音を鳴らすための関数Note Onがもともと備わっている。現時点ではMIDIコードのうちNote On以外の命令は実装していない（テンポの変更などの機能はできるだけ早く実装する予定である）。

4.6 音符の分割送信

クライアント側はMIDIのNoteOn命令を一つ実行するたびに再びリクエストしてレスポンスを待つ。ごく感覚的に言えば､楽譜の音符を一つリクエストして鳴らし､次の音符が鳴るまでの時間待ってから再びリクエストを

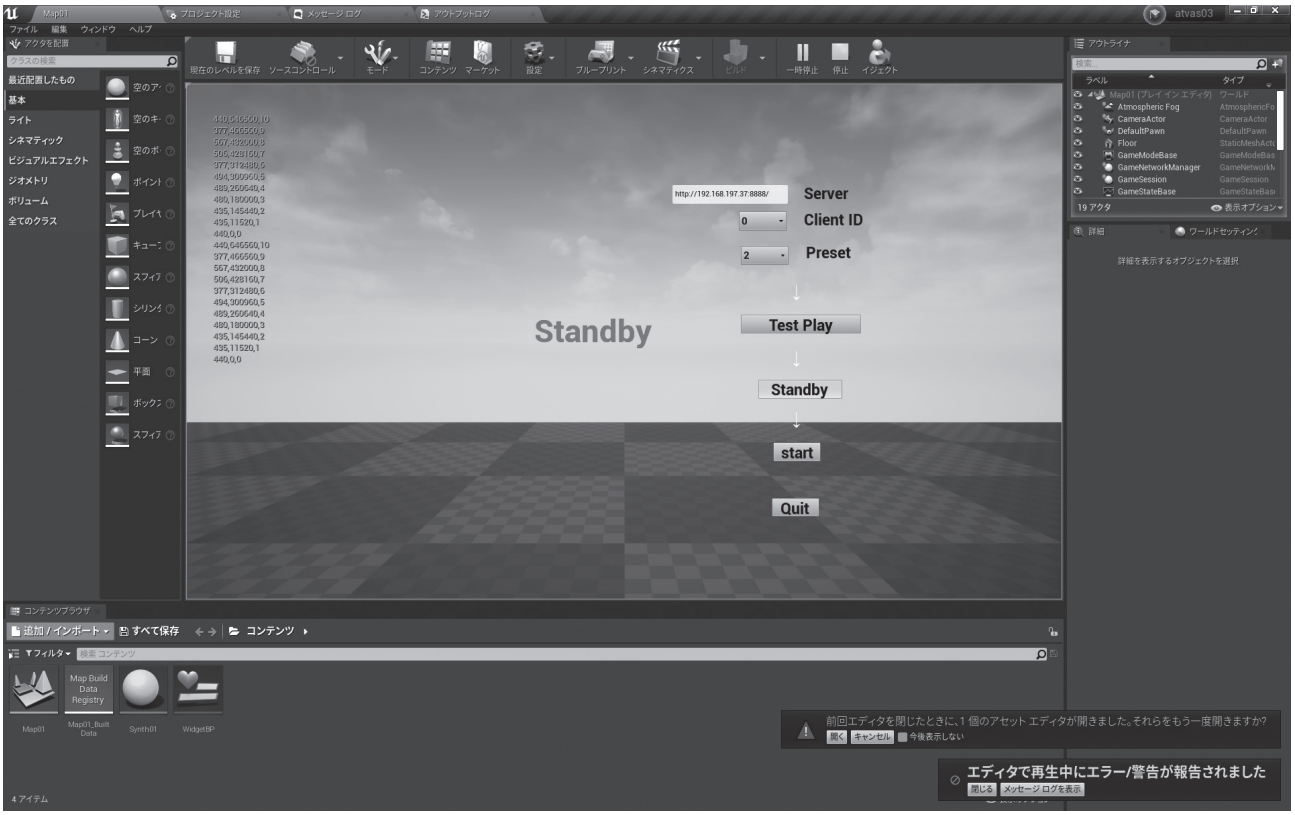


図4. クライアントのインターフェイス（演奏の開始待ち状態）。クライアント側ではサーバーのURL、クライアントIDを入力し、テスト音（ドミソの和音）を鳴らしながら（Test Playボタンを押す）プリセットを選び、Standbyボタンを押す。すべてのクライアントがStandbyになったあと、どれか一つのクライアントでstartボタンを押す。またQuitボタンで終了する。

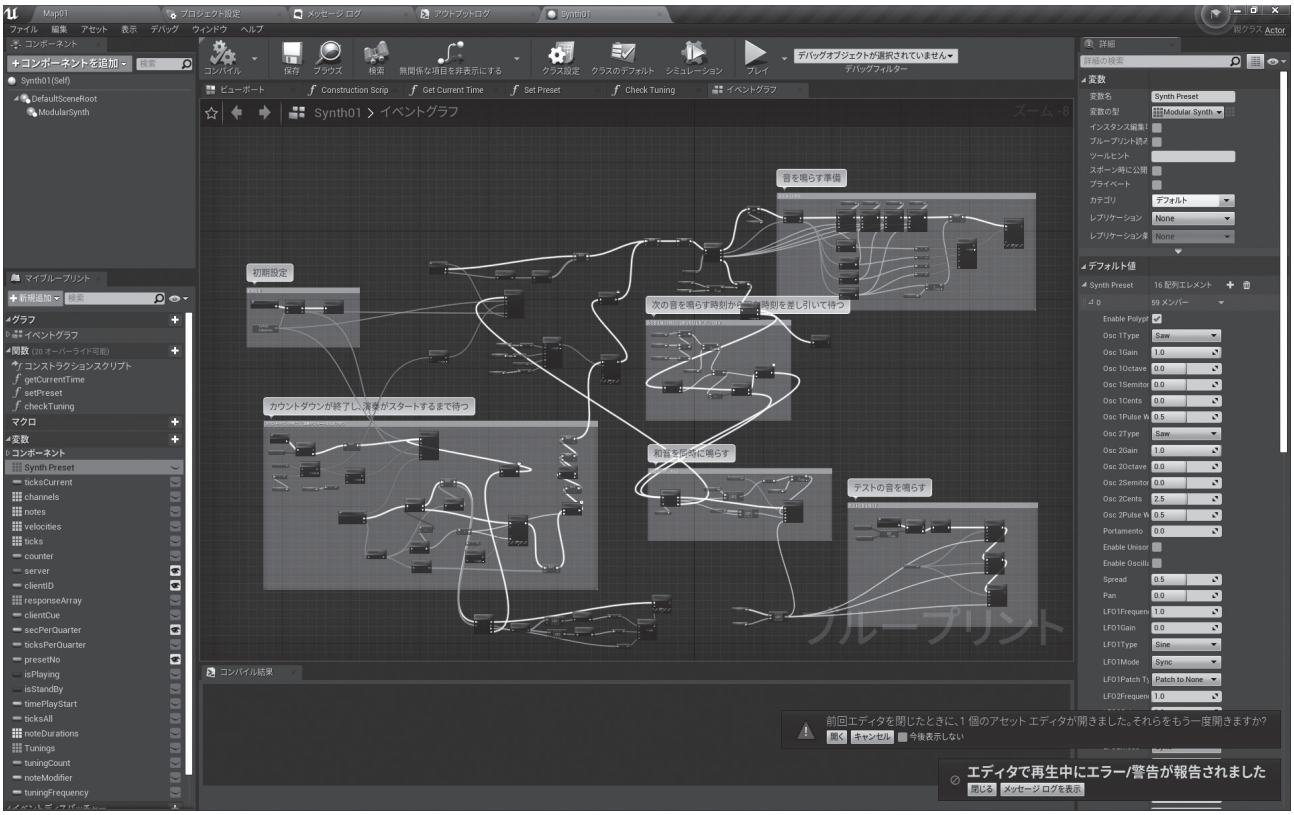


図 5. Blueprint による VaREST と Modular Synth のコーディング（一部）

送るという仕様にしている。

なぜ音符をいちいち一つずつ取りにいくのか、なぜ音符をまとめて何個も取りにいかないのかといえば、それはサーバー側で譜面をリアルタイムで書き換えられるようにするためである。一人の演奏者がサーバーを操作して、まだ演奏されていない後半部分を変更する。或いはあるクライアントからサーバーにキューを送って、すべてのクライアントで一斉に基準音や音律を変更する。そのように、できるだけ任意のタイミングでキューを出して演奏を途中で変更できるよう､クライアント･サーバー通信の仕様は一回当たりの通信データサイズを極力小さくして、つまり「粒度」を小さくして、小回りが利くようにしてある。

図5は、クライアント側がスタンバイ状態から演奏状態に移行し、ループの中でサーバーからデータを受け取り、指定された経過時間の後に音を鳴らすスクリプトの一部である。この図ではわかりにくいので、図6にリクエストとレスポンスについて簡潔に図示した。

Standbyリクエストでサーバーに送るのはクライアントIDである。クライアントIDは楽譜のパートに対応している。このリクエストに対して、サーバーは他のクライアントがすべてStandby状態になっていれば開始キューを送るとともに、チューニングの変更タイミングのリストと、開始までの時間を送る。

音符リクエストに対してサーバーは音符（MIDIのNote On命令）を一つずつ送り返す。それぞれのクライアントはMIDIコードの tick 数の積算値を保持しており、この値から次に音を鳴らすタイミングを計算する。

5. 実験と考察

5.1. 原曲の本来の演奏方法

原曲はビープ音、チェロ、ピアノの三つのパートからなる。曲は11のセクションで構成されそれぞれのセクションに基準音Aが割り当てられる。

チェロは演奏者が実際の楽器を演奏途中にペグを回して弦の張力を変えることでチューニングを変更する。コンサートでは生音をそのまま観客に聴かせるのではなく、一旦マイクで音を拾い、ミキサーで音量を調整してスピーカーで再生する。

ピアノはLogic Proで一旦録音した音声ファイルをスピーカーから再生している。

ビープ音もまた録音したものをスピーカーで再生している。

ピアノは、地域（つまりモールス信号の種類）の数だけ、聴衆の回りにスピーカーを配置する。チェロ奏者は聴衆の正面に位置し、その音源はスピーカー一つだけで

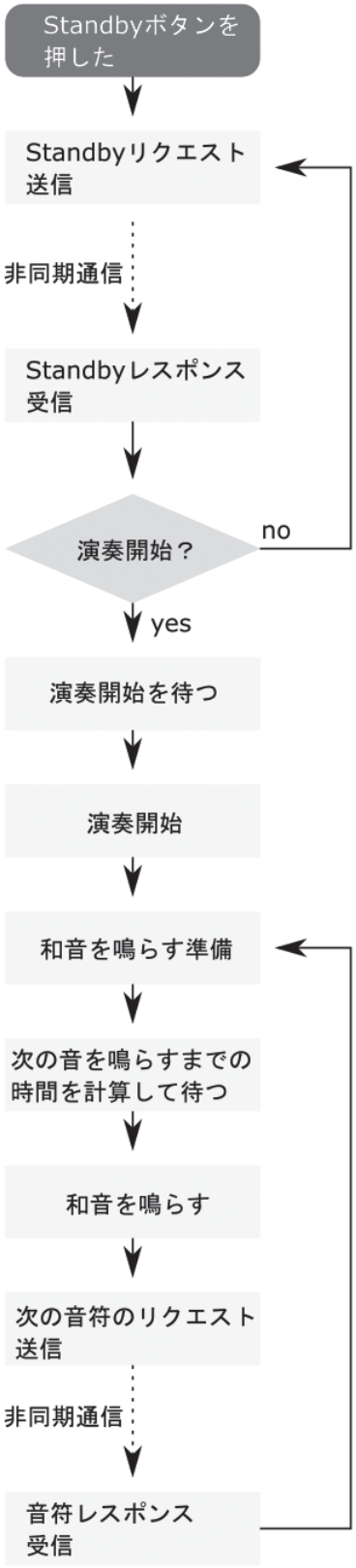


図 6.　クライアント・サーバー通信（図 5 の一部に相当）

ある。ビープ音も一つだけである。

原曲のピアノパートは即興演奏的な性格のものであり、具体的な譜面は無い。

5.2.　本実験の演奏方法

ピアノパートは3.3.節に定められた規則に従い、Python mido で自動生成することとした。すなわちMIDI番号で言うと、45、50、52、61、66、69、72、78、80、86、93、96番の音のベロシティを 1から127までランダムに決め（MIDIで音量は0から127までの整数）、またやはりランダムに MIDI番号に+12、0、または -12を加えてオクターブを上か下にずらして（MIDI番号が12増えれば1オクターブ上がる）、12音を同時に鳴らす。こうすることによってランダムな自動演奏でもある程度統一感を持たせることができる。音を鳴らすタイミングはセクションごとに適当にパターンを変えている。今回は実験を容易にするためと音を聞き取りやすくするため、ピアノを一つのクライアントで済ませた。

ビープ音は一定間隔で単純な矩形波を短く鳴らす。これもPython midoで自動生成した。

チェロパートには譜面が存在するのでこれをMusescore [34]で採譜し、MIDI データを書き出して、ビープ音パートとピアノパートに合わせた。

今回、原曲を完全に再現したMIDIデータが得られたのではなく、実験のためのダミーのデータであると考えていただきたい。図7はサーバーで読み込んだ実際のMIDIデータを表している。ピアノパートは最初音を鳴らすタイミングだけが手打ちのパートに書かれていて、そのタイミングでPythonスクリプトによって12個の和音に展開する。実際にクライアントに配布して鳴らされるのは和音のパートだけである。またピッチシフトのタイミングだけを記したパートもあり、これも音として鳴らすことはない。

ピアノとチェロの音はModular Synthでノコギリ波や矩形波のオシレーターから得られる音を加工して作っているが、現時点では仮のものであり、将来的にはサンプル音源を使うかも知れず、詳細は省略する。

今回、何度か予備的な実験を行い、最終的に 3台のPCをクライアントとして使った（ピアノ、チェロ、ビープ音にそれぞれ1台ずつ）。もっと多くのクライアントを使って、有線もしくは無線LANを用いて自由にそれらを配置できるはずだが、音がちゃんと同期して鳴っているかどうかを聴いて確認するために、3台のクライアントPCとそのスピーカーは左右に比較的近距離に並べて配置した。

この実験の主たる目的は、REST通信を用いて原曲の演奏が支障なく再現できているかどうかを確認することである。遅延が感じられるかどうかを実際に筆者らが耳で聞いてみて、さらに、この曲について事前知識を持たない第三者数名に被験者として聞いてもらい、率直な感想を求めた。これにより、クライアント間に概ね音のずれはなく、正常に動作していることを確認した。

最初、原則として音符を一つ一つサーバーからクライアントに送ってみたが、和音は完全に同時に鳴らないでごく小さな遅延が感じられた（極めて速いアルペジオのようになる。それはそれで面白いエフェクトだが意図したものではない）。そのため和音は同時にまとめて送って、同時に鳴らすようにした。和音の場合、MIDIコードでは最初の音だけに長さの情報があり、他の音の長さは0になっているので、これによってサーバーはどこまでの音がひとまとまりの和音になっているかわかる仕掛けになっている。

5.3. 考察

音符を一つずつ送れば通信の頻度が多くなりレイテンシーの原因になりかねない。現在のところ、LANでデータを送受信する遅延は通常1ms以下であり、（ネットワークに障害でもない限り）10msを越えることはほとんどない。

たとえばテンポ♪=120（アレグロくらい）、すなわち1分間に4分音符が120個だとすると、4分音符一つの長さは500ms（0.5秒）。16分音符でも125msあり、1msはもちろんのこと、10msのレイテンシーでも耳で聞いてわかるほどではないと考えている。

もちろんround robin（往復）による遅延に関しては次の音の待ち時間に繰り込んであるので、これが影響することはない。現状この有線LANの上に構築したシステムでは、音符と音符の時間間隔は十分に広く、その合間に各種のキューを多少挟んだ程度で支障はなく、長時間同じ曲をループさせても音のずれが蓄積されるようなことはない（つまり、クライアント間で演奏がずれているとしたら演奏開始のタイミングだけで、あとはtick数で経過時間を同期させている）。

このシステムでは（ピアノなどの）一つの楽器が和音を鳴らすケースには既に対応しているが、複数の楽器（クライアント）でかなり緻密な和音を構成しなくてはならないような場合に（たとえば第一バイオリンが速い旋律を、第二バイオリンがゆっくりした旋律を、ハーモニーを奏でながら弾くような場合。ピアノ奏者が両手で弾くような場合に相当）、やはり一種の違和感を覚えることがある。これが実際の微妙なずれによるものなのか、或いは感覚的なものなのか、原因ははっきりしていない。人が演奏する場合も完全に譜面の通り、遅延無く弾けるわけではない。今後複数の（もっと遅延がわかりやすい）曲を演奏してみて聞き比べ、また遅延を実際に計測するなどして明らかにしていきたい。

チューニングを変更するタイミングは最初にまとめてサーバーからクライアントに送っている。MIDIで表される譜面の最小時間単位はtickである。つまり、演奏開始から何tick目でセクションが変わって、チューニングをどのくらい変更するかという配列を最初にまとめて送っている。最初にあらかじめ送るデータと途中随時送るデータはそれぞれ意味合いや役割が違うので、ほんとうは、MIDIのNote On命令と同じように（MIDIプロトコルを拡張する形で）チューニングを変更する時点で随時その命令を送りたかったのだが、今回は実装を見送った。

MIDIのtickは4分音符の長さに比例する。通常は480tickで一つの4分音符に相当する。4分音符の長さはテンポで変わるので当然テンポが変わればtickの長さも変わる。テンポが変わる楽曲ではtickを絶対時間として使えないので、今後はtickではなく実時間でクライアント間のタイミングをそろえるようにする予定である。

最後に今回実現できたこととできなかったことをまとめる。

サーバーをハブとしてクライアントどうしでアンサンブル演奏することはできた。LAN環境で遅延はほぼ感じられない。

演奏途中にピッチシフトすることは、最初にピッチシフトのタイミング情報をすべてのクライアントに配布することによって実現した。今後は事前配布するデータによらず、任意のタイミングでキューを送ってピッチシフトできるようにしたい。

ピタゴラス、ミーントーン、Werckmeisterなどの音律を切り替えること、つまりピッチをセント単位で各音階ごとにずらすこと、これは未実装である。今は単にすべて平均律を用いている。音律変更は、音階と周波数の対応表をそれぞれの音律ごとに作れば良いだけなので、原理的にはBlueprintで簡単に記述可能である。音高と音律を任意のタイミングで変えられる奏法が可能になれば、西洋音楽だけでなく、世界各地の民族音楽に、それぞれ音律が何種類もあるので、それらを調査し、組み合わせて曲を作ることも不可能ではない。今後はたとえば

インドの22分割音律（Śruti）[35]や、31平均律のFokkerオルガン[12][36]などの楽曲を採譜するGUIの開発や、演奏にも挑戦したい。

任意のタイミングでテンポを変更すること、これも未実装である。ピッチベンドなどのMIDI命令も未実装である。現時点ではMIDI命令はNoteOnしか実装していない。難しいことではないので順次対応していきたい。

ピアノやチェロなどのサンプル音を鳴らすこと。これはUE4 Modular Synthではできない。つまり現時点で実現は難しいが、将来UE5が正式リリースされればその標準コンポーネントであるMetasoundにサンプラーの機能があるので、それを使ってできるようになる予定である。ゲームエンジンの進歩は非常にめまぐるしいので、現在できないことでも数年後には当たり前になっているかもしれない。

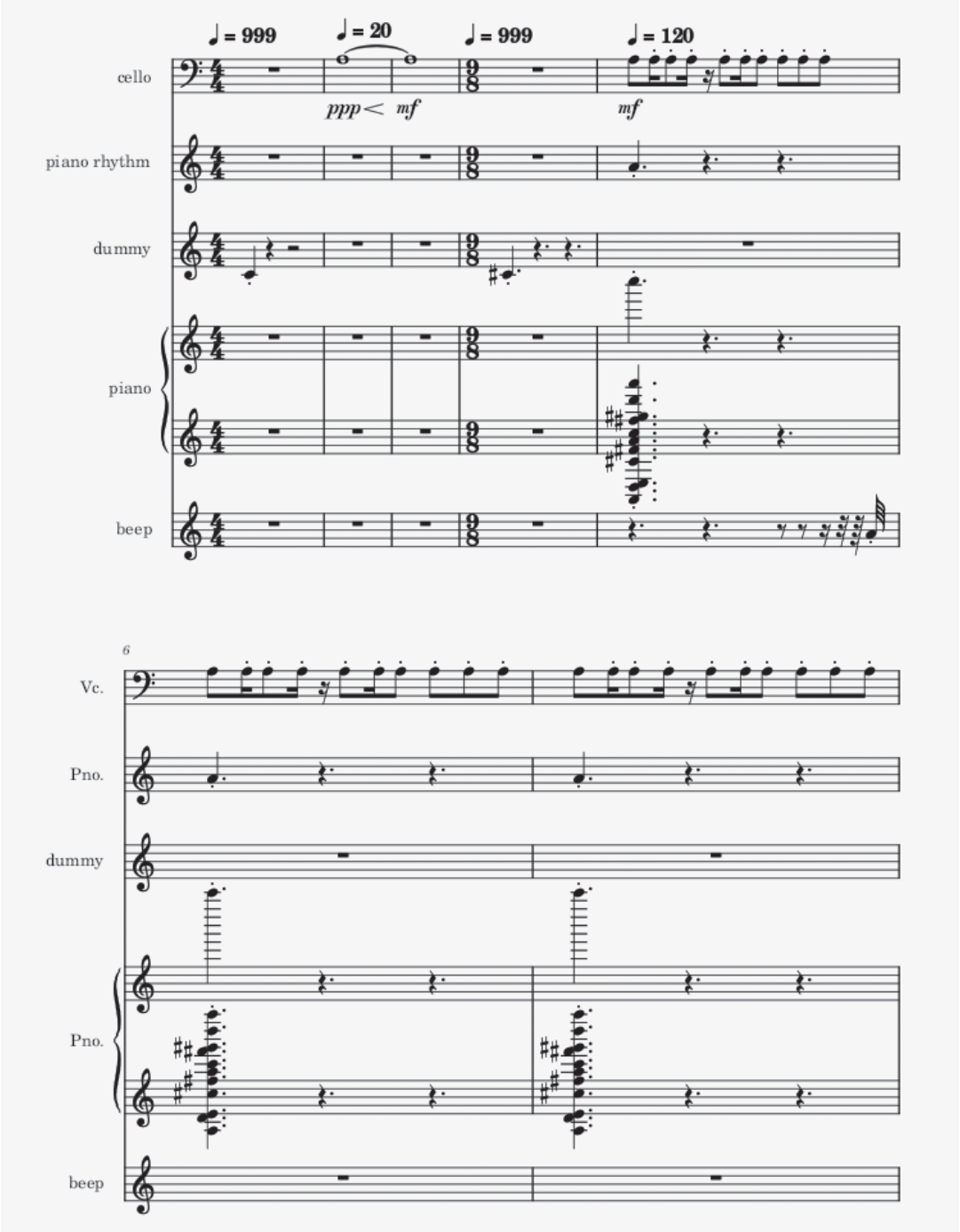


図7. 自動生成した譜面（冒頭部分。piano rhythmはピアノを鳴らすタイミングを表し、dummyはチューニングを変更するタイミングとして使われ、実際には演奏されない）。

6. おわりに

本研究は当初MMORPGやネットワーク対戦ゲームと同等のシステムを用いて、ネットワーク分散シンセサイザーを構築することを目的の一つとしていた。しかしながら今回筆者らはいまだにゲームエンジンのネットワーク機能を活かしきれてはいないし、サーバーとクライアントの間でやりとりするデータ形式もMIDIを多少拡張した程度であり、かつまた、Unreal Engineのシンセサイザー機能も現在のバージョンではさほど充実してはいないので、単純な音を鳴らすことしかできない。

今回、本作はプロトタイプレベルの試作に留まった。筆者らは、本システムを構築する過程で、これから解決していかねばならないいくつかの課題を見出した。それと同時に、新しい音律、新しいインターフェイス、新しい作曲の可能性を多く発見した。

DAWやVSTはすさまじい勢いで進化を続けているが、微分音的楽曲を作成するためのソフトウェアやインターフェイスは今もほとんど手つかずの状態である。現代音楽の今後は、新たな作曲支援ソフトの開発にかかっている。新しいツールが供給されれば新しい方法論が生まれ、視野が開ければさらなるツールの需要を生む。商業ベースでは採算が取れないが個人だと敷居が高すぎて作れない。そういった分野にこそ研究者は挑戦していかなくてはならないし、ゲームエンジンはこの分野におけるブレイクスルーのための有効なツールになってくれるだろう。

現時点でもある程度遅延のない演奏は可能だが、さらに正確さと速さを兼ね備えるにはTCPとUDP、或いはRUDPやQUIC、場合によってはRTPを組み合わせねばならない。そのためには Unreal Engine のネットワーク機能を完全に活用し、同時に既存のネットワーク対戦ゲームで具体的にTCPやUDPがどのように使われているかをサーベイする必要がある。また、さまざまな楽曲に対応するには、MIDIをベースにしつつMIDIを拡張したプロトコルを作らなくてはならない。

ゲームエンジンを使えばクライアントのソフトシンセどうしが（たとえばジャズの即興演奏のように）通信しながら自律的に演奏するというような楽曲も可能になる。しかしそのためにはまずクライアントにNPC（non-playable character）と同じようなAIを実装しなくてはならない。NPCとは則ち敵のボットや味方のボット、村人のように、プレイヤーが操作するのではなく、ソフトウェア的に勝手に動くキャラクターのことである。

ジャズの即興演奏にしてもまったくデタラメに弾いているわけではなく、ある法則性に従って他の奏者と協調して演奏しているのであり、その法則性をコーディングするなり学習させれば、それはもはや即興演奏ではなくて作曲された楽曲と言って良いものになる。ゲームに使われるAI（たとえば、ビヘイビアツリーなど）を利用することも可能かも知れない。このようなことは本研究の遠い目標であって今一足飛びにそこまで実現できるわけではない。しかしながらこれらのことはすべてゲームエンジンでは現在すでに普通に行われていることである。

最後に多少筆者らの感想も述べさせてもらいたい。

実際に楽器を配置していろいろな楽曲を鳴らしてみて、私たち（筆者、および被験者ら）は大いなる違和感を覚えた。例えば弦楽四重奏曲をクライアント･サーバー環境で鳴らすことを考えてみる。部屋の四隅に第一バイオリン、第二バイオリン、ビオラ、チェロを配置して部屋の真ん中でそれらの音を聴いた場合にそれが果たして一つの楽曲として知覚されるだろうか。ただ単にばらばらに楽器が演奏されているように感じないだろうか。

主旋律を担当する楽器が前方右と左でときどき交代し、副旋律や伴奏が背後から聞こえる程度であればまだ一つの合奏として知覚できるかもしれない。しかし主旋律が前から後ろへ移動したり、また前へ移ったりしたとき、私たちの知覚は大いに混乱して、それらの個々の演奏が一つの合奏を構成していると考えることが困難なのではなかろうか。

私たちはふだん二つのスピーカーから聞こえてくるステレオ音声にあまりにも慣れ親しんでいる。5.1chや7.1chなどのサラウンド音響にしても、結局は前方二つのスピーカーがメインであり、さらにその両スピーカーの間に映像ディスプレイが置かれていれば、人はその映像に注意を奪われ、聴覚は視覚に支配されて、音像は安定して空間に定位する。しかしながら映像もなく、音も常に予期しない方向から聞こえてくるとなると、それはたとえて言えばジャングルの中で不意に襲ってくる野獣に常に警戒している原始人のような心境であって、私たちは落ち着いて音楽を鑑賞できないだろう。

石原が2012年に、観客の前でこの曲を上演したときには、コンサートホールの客席の周囲を囲むようにピアノパート用の6chのスピーカーを配置し、チェロ奏者がチェロ用のスピーカーとともに正面に位置した。観客は自然と目の前のチェロ奏者とチェロの音に意識を集中することになり、一方でピアノの音は周囲から副旋律として聞こえてくる仕掛けになっていた。演奏者がいて、主旋律と副旋律の役割分担が明確なために、一般的なコン

サートと同じ心境で聴くことができた。しかし演奏者がおらず、ただ単にスピーカーだけが並べられて音が鳴っていたとしたら、果たして同じ効果が得られたであろうか。そもそも観客は、奏者がいなければいつ演奏が始まり、いつ終わるか、どっちを向いて何を聴き、何をすれば良いかもわからないだろう。

当たり前のことではあるが、音楽は音だけではない。パフォーマンスや視覚的効果などの演出、コンサート会場におけるありとあらゆる設定、調整、雰囲気などの要素が大事なのだ。今回、分散演奏システムを構築して改めて気づかされた。

私たちは無意識のうちに演奏する人を注視して楽器を演奏する手指などを見ながら音楽を聴いている。映像の収録となれば、コンサート会場にも複数のカメラが置かれ、多くの場合は単に会場を撮りっぱなしにするのではなく、演奏の進行に合わせて効果的にカメラを切り替えている。それらが映像や音響のプロフェッショナルらによって自然に演出されているために私たちは違和感を覚えないだけなのである。

映像がない場合にはその視覚を補うために、主要な楽器ごとにマイクで音を拾い、レコーディングエンジニアは主旋律の音量をミキサーで随時上げ下げしている。そのようなレコーディングやマスタリングの手間ひまがかかっていない、生の演奏を、映像も無しに、楽器が並ぶ中に自分も入り込んで聞くとき、私たちは困惑するだろう。聞き覚えのある､古典的で有名な曲ならばともかく、初めて聞く新曲で、音色もシンセサイザーによる合成音であり、しかも前衛的な曲であれば、人はその楽曲を容易に一つの音楽とは認識しない、おそらくは単なる環境音とみなすに違いない。では環境音と音楽の違いは何かと言えば、これにも厳密な定義があるわけではない。

そこで私たちは、音像をこれまでにないやり方で空間に配置するのであれば、それに合わせた曲を作り、それに合わせたハーモニーなりメロディーなりリズムを考案しなくてはならないのであり、ただ単に旧来の楽曲を、そのまま立体音響システムを用いて演奏するだけでは意味がなく、なんらかの編曲手法が新たに必要になるということになる。かつてルネッサンス期に遠近法が発明されたときに、或いは近代その画法が日本にもたらされたときに、その画法に適した画題や構図が必要になった状況と同じであるといえよう。

参考文献

[1] ISO 16:1975 Acoustics — Standard tuning frequency (Standard musical pitch) https://www.iso.org/standard/3601.html

[2] Gaudeamus Muziekweek https://gaudeamus.nl/

[3] H. Feldmann: “Die Geschichte der Stimmgabel - Die Erfindung der Stimmgabel, ihr Weg in der Musik und den Naturwissenschaften”. Laryngorhinootologie 76(2): pp. 116–122, 1997.

[4] Bruce Haynes: “*A History of Performing Pitch: The Story of 'A'*”, Scarecrow Press 2002.

[5] Hermann von Helmholtz. “*On the Sensations of Tone as a Physiological Basis for the Theory of Music*,” 4th edition, translated by Alexander John Ellis, 1912.

[6] Dr. Brian Blood: music theory online : pitch, temperament & timbre
https://www.dolmetsch.com/musictheory27.htm

[7] Douglas Earl Bush, Richard Kassel: *The Organ: An Encyclopedia*, Routledge, pp. 534, 2014.

[8] Pipe Organs: “Pitch: 495,45 Hz at 18° Celsius, 493,85 Hz at 16° Celsius”,
http://mypipeorganhobby.blogspot.com/2008/12/st-jacobi-kirche-hamburg.html

[9] 馬場良始 :「ピタゴラス音律から新しい音律へ – 小学校専門科目「数学」での実践 –」, 数学教育研究 42, pp. 37 – 59, 2013.

[10] 馬場良始 :「音律の探求 (16 世紀以降) – 小学校専門科目「数学」での実践 –」, 数学教育研究 42, pp. 61 – 85, 2013..

[11] Y Guerra: Igor Stravinsky: “Hidden Rhythmic Patterns and Codes in his Orchestral Music,” A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Music, escholarship.org 2021.

[12] Carlton Gamer: “Some Combinational Resources of Equal-Tempered Systems,” Journal of Music Theory, Vol. 11, No. 1, pp. 32 – 59, 1967.

[13] Paul Erlich: “*Tuning, Tonality, and Twenty-Two-Tone Temperament*,” originally published in Xenharmonikôn 17, Spring 1998; revised 2002.

[14] 藤井守、中村桂子 :『TALK 音の響きにいのちのつながりを聴く』生命誌 56、2008.

[15] Wendy Carlos: “Three Asymmetric Divisions of the Octave,” 1989,
http://www.wendycarlos.com/resources/pitch.html

[16] Wendy Carlos: “Tuning: At the Crossroads,” Computer Music Journal, 11(1), pp. 29–43, 1987.

[17] Kite Giedraitis: Alt-Tuner,
http://www.tallkite.com/alt-tuner.html

[18] Biptunia: Microtonal Synth,
https://biptunia.com/

[19] Tobias Erichsen: rtpMIDI:
https://www.tobias-erichsen.de/software/rtpmidi.html

[20] 床子琢郎、加藤朗 :「立体音響を用いる小型サラウンドスピーカー環境の研究」、慶應義塾大学修士論文、2018.

[21] 寺田直美 :「ストリーミング配信サーバにおける管理支援に関する研究」NAIST-IS-MT0151067, 奈良先端科学技術大学院大学修士論文、2013.

[22] 岸田崇志、前田香織、河野栄太郎、近藤徹、相原玲二 :「多様な遠隔コラボレーションを実現する音声伝送システム」、情報処理学会論文誌 vol. 45, no. 2, pp. 517 – 525, 2004.

[23] 丸山裕貴、鈴木秀和、内藤克浩、渡邊晃 :「偽サーバ利用型のゲー

ムハッキングを防止する方式の提案」、電気・電子・情報関係学会東海支部連合大会講演論文集 2019 (L5-5) 1-1 2019.
[24] Jun-Ho Huh: “Reliable User Datagram Protocol as a Solution to Latencies in Network Games,” MDPI Electronics 2018, 7, 295.
[25] Sou Takabayashi,Yoshihiro Ito: “A Study on QoE Improvement of Online Games with UDP Multipathization by SDN,” SHS Web of Conferences 77, 04001, 2020
[26] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi: “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” ACM SIGCOMM '17, Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 183 -- 196, 2017.
[27] VaRest; REST API plugin for Unreal Engine 4 https://github.com/ufna/VaRest
[28]「常識的な音の感覚 冨田勲インタビュー 音響・音楽へのこだわりと発想の原点」芸術科学会誌 DiVA 2 号、pp. 56–59, 2001.
[29] た と え ば Plugins4Free – Free Instrument Audio Plugins Archive https://plugins4free.com/
[30] JUCE
https://juce.com/
[31] mido; MIDI Objects for Python
https://github.com/mido/mido
[32] Flask
https://palletsprojects.com/p/flask/
[33] Mido Docs: Socket Ports - MIDI over TCP/IP https://mido.readthedocs.io/en/latest/socket_ports.html
[34] Musescore
https://musescore.org/
[33] Lewis Rowell: “Music and Musical Thought in Early India,” 2015.
[34] Huygens-Fokker Foundation,
https://www.huygens-fokker.org/
[35] Lewis Rowell ; “*Music and Musical Thought in Early India*,” University of Chicago Press, 2015.
[36] Huygens-Fokker Foundation,
https://www.huygens-fokker.org/

補足

youtubeで実験動画を公開しています。
Altered Tunings in A (An experimental play using a dummy MIDI data)
https://www.youtube.com/watch?v=s2lEiJl7fNA