

「Scratch3.0」のマルチスレッド機能を用いた音楽演奏アプリケーションの開発

Development of Music-player Application Software
Using Multithread Functions in “Scratch 3.0”

茂出木敏雄

Modegi Toshio

概要

米国 MIT により開発されたビジュアル・プログラミング開発環境「Scratch」は、単一のスプライトや複数のスプライト上でマルチスレッド機能を簡便に実装できる。本稿では、「Scratch3.0」の拡張機能として実装されている「音楽(MIDI 機能)」および「ペン(2次元グラフィックス)」を用いて、2種の音楽演奏アプリケーションを試作した事例を紹介する。第1の試作例は、「Scratch3.0」の拡張機能「音楽」を基に MIDI シーケンサを試作し、「クローン」と呼ばれるコピー・スプライトを並列実行させて、和音を含む楽曲を4声でカノン(輪唱)演奏させることを可能にしたものである。第2の試作例は、「Scratch3.0」の拡張機能「ペン」を基に、21 鍵のソフトウェア鍵盤を表示させ、画面上で選択したキーを根音として、3個のスプライトを並列実行させて、長三和音または短三和音を自動演奏させることを可能にしたものである。

Abstract

The visual programming environment “Scratch” developed by MIT in US, it makes implementation of multithread functions easy, in a single “sprite” or multiple “sprites.” In this report, we present two kinds of prototypes of music-player applications, using the extended functions of “music (MIDI player)” and “pen (2D-graphics).” The first application is a MIDI sequencer, which can play four-polyphonic canon-chorus of given work of music including harmonies, using “clone sprites” in “Scratch 3.0.” The second application is a 21-key software keyboard, which can generate a major or minor triad chord automatically by composing a selected key as a root note, using three “sprites” in “Scratch 3.0.”

キーワード： スクラッチ, マルチスレッド, MIDI シーケンサ, カノン演奏, ソフトウェア鍵盤, 三和音付け

Keywords: Scratch, multithread, MIDI sequencer, canon chorus, software keyboard, triad-chord generation

1. はじめに

2020 年度より国内の小学校で、2021 年度より中学校でプログラミング授業が必修化され、文献¹⁾のように子供向けのプログラミング教科書が数多く出版されるとともに、巷に子供向けのプログラミング教室が数多く開設されている。これらの教室を含めて「Scratch」⁴⁾⁵⁾などのビジュアル・プログラミン

開発環境が多く、多くの学校で採用されている。米国 MIT により開発されたビジュアル・プログラミング開発環境「Scratch」は、1972 年にアラン・ケイにより開発されたオブジェクト指向言語「SmallTalk」に端を発しており、その開発環境「Squeak」で開発された「Squeak Etoys」が原点になっている¹⁾。この「Squeak Etoys」は、GUI 画面上でジグソーパズルのようにブロックを連結させることによりビジュアルにプログラミングが可能である。このようなビジュアル・プログラミング手法は、子供向けのプログラミングだけでなく、画像処理のアプリケーション開発⁶⁾や、Max/Msp⁷⁾といった音響信号合成のユーザインタフェース⁸⁾にも活用されている。

「Scratch」は、スプライト(妖精)と呼ばれるコスチュームと音を備えたキャラクターの動作をスクリプト(台本)としてプログラミングするもので、開発環境はいわば劇場である。「Scratch」は、Web ブラウザ上で動作し、プラットフォームに依存しない構成になっており、英語・日本語に限らず数多くの言語に対応している(2025 年 9 月現在で 80 言語)。日本語版は通常の漢字を混用した「日本語」モードと、子供向けにひらがなのみを使用した「にほんご」モードに切り替えることができる。Windows、Mac のパソコン向けにダウンロード版も提供されているが、Web ブラウザ版はクラウドで動作する API (Application Programming Interface)が提供され、多言語翻訳など高度な機能を活用したアプリケーションを簡単に開発することができる。ただし、現状の「Scratch3.0」では、メール送信などネットワークのアプリケーションを開発する API は用意されていない。

一般に「Scratch」は、子供向けにアニメーションやゲームのアプリケーションを開発するツールとして知られているが、多くの API が満載され、「micro:bit」と呼ばれる小型ボードコンピュータを制御するような組み込みソフトの開発を含め、大人でも十分手ごたえを感じる汎用的なソフトウェア開発ツールになっている。「Scratch」本体は JavaScript で実装されており、子供では理解が困難な配列操作や三角関数など高度な算術関数も使用できるため、統計計算やグラフ表示などのデータサイエンス分野の高度なアプリケーション開発も可能である。また、JavaScript の関数に相当する、独自にブロックを定義して構造化プログラミングを行う機能を備えているため、大規模なアプリケーションを開発することができる。

特に、「Scratch」が従来の C 言語や JavaScript 言語に比べ優れている機能が、マルチスレッドである。ゲームアプリケーション向けに前述のスプライト(主役、敵、味方、環境など)を複数個定義することができ、各スプライトに独立したスクリプトを定義できるので、複数のスクリプト(スレッド)が同時に実行される。この時、全てのスプライト内のコードで参照できるように変数を共有させることや、スプライトごとに独立したローカル変数を定義することもできる。また、マルチスレッドは、音楽でいえば合奏である。例えば、複数の音楽ファイルを再生する際、各々個別に再生プログラムを記述し、あたかも指揮者が合図を送るように、個々のプログラムにメッセージを送るだけで、複数の再生プログラムを同時に実行させる、あるいは一部の再生プログラムを指定時間だけ遅らせて実行させるといった制御が簡単に行える。

マルチメディア機能として、現状の「Scratch3.0」には動画再生や3次元 CG の機能は実装されていないが、波形オーディオ、MIDI、音声合成(音声認識は不可)といった音響処理機能は実装されている。ファイル入出力機能がサポートされていないため、音響信号処理は実現できないが、2 次元グラフィックスを用いた波形の表示や、音声、音楽、効果音などの合成や再生の制御は行える。

先報告¹⁰⁾にて、「Scratch3.0」を用いたプログラミング手法の基本と、波形オーディオと MIDI を含む音響分野のアプリケーション開発を行うための基本機能とプログラミング事例について紹介した。本稿では、「Scratch3.0」のマルチスレッド機能を活用し、MIDI にフォーカスした 2 種の音楽演奏アプ

リケーションとして、カノン演奏機能をもたせた MIDI シーケンサ(正確には、MIDI シーケンサに類似した音楽プレーヤ)と、三和音の自動演奏機能をもたせた 21 鍵のソフトウェア鍵盤の試作事例を紹介する。

2. 「Scratch3.0」を用いたマルチスレッド・プログラミングの基本

「Scratch3.0」でマルチスレッドを実現するには、複数のスプライトを用いて、各スプライトに変数やコードを定義する手法が基本となるが、単一のスプライトでも簡便にマルチスレッドを実現することができる。図 1 に示すように、スプライトの形態に基づき3通りのマルチスレッド手法を実現できる。



図 1 「Scratch3.0」における 3 種のマルチスレッド実現形態

2.1. 単一スプライトでのマルチスレッドの実現

図 1-(A)に示すように、単一のスプライト上に複数のコードを配置し、各コードの先頭にイベント・ブロックの「(旗)が押されたとき」や「(メッセージ)を受け取ったとき」を配置すれば、これらのコードが同時に起動される。この時、各コードは、単一の CPU コアにより時分割で切り替わりながら並列実行される。後述する図 2-(B)に示す方法と同様に、スプライト 1 内にローカル変数を定義できるが、スプライト 1 内の各コードにおいてはグローバル共有変数と同等に動作するので、「コード 1」と「コード 2」内で重複するローカル変数を使用する場合は、留意する必要がある、変数名を変更するといった対策が必要になる。

CPU 演算を伴わない、拡張機能「音楽」ブロック群の「()の音符を()拍鳴らす」のような音楽演奏ブロックであれば、並列実行による遅れは殆ど生じない。また、CPU 演算を伴う処理であっても、以下理由により並列実行による遅れは顕著でない。

スプライトにて独自のブロックを定義する際、図 2-(A)に示すように、「画面を再描画せず実行する」というオプション設定がある。デフォルト設定では、スプライトの各ブロックは、ブロックの処理が完了しても画面更新のタイミングまで待ち状態となり、各ブロックの処理にブレーキがかかっている。例えば、先報告¹⁰⁾の「4.3.節 波形合成・うなり(拡張機能「ペン」使用)」にて 3 つのサインカーブの描画を実行させると、描画が終了するまで数 10 秒程度待たされるが、図 2-(A)に示すように、独自定義ブロック「波形表示」のオプション設定「画面を再描画せず実行する」を ON に設定すると、瞬時(1 秒未満)に 3 つのサインカーブの描画が終了する。即ち、デフォルト設定では、CPU が空く待ち時間が結構あるため、複数のコードを時分割に並列実行させる余裕がある。

ただし、複数の楽器音で構成されている楽曲をマルチスレッドでアンサンブル演奏させる場合、単一のスプライトで異なる楽器音を設定できないため、後述する図 1-(B)(C)に示すように、複数のスプライトやクローン・スプライトを作成して、各トラックを異なるスプライトに割り当てる必要がある。

(A) ブロック作成時の高速実行指定



(B) 変数作成時のローカル変数指定



図 2 ブロック作成時および変数作成時のオプション設定

2.2. 複数のスプライトを用いたマルチスレッドの実現

図 1-(B)に示すように、複数のスプライトを定義し、スプライトごとに、個別に変数やコードを配置し、前節と同様に、各コードの先頭にイベント・ブロック群の「(旗)が押されたとき」や「(メッセージ)を受け取ったとき」を配置すれば、これらのコードが同時に起動される。あるスプライトで送信されたメッセージは全スプライトに送信されるが、変数はスコープが全スプライトで共有されるグローバル共有変数と、スコープが各スプライトに限定されるローカル変数を区別して定義できる。図 2-(B)に示すように、新しい変数を作る際、オプション設定で「すべてのスプライト用(デフォルト)」を選択すれば前者、「このスプライトのみ」を選択すれば後者になり、後者の場合は変数名にスプライト属性が付加され、スプライト間で変数名が重複しても互いに独立した値が設定される。

コードは、変数と異なり、全てのスプライトで共有されるコードを定義できず、スプライトごとに個別に、場合により重複して定義する必要がある。他のスプライト上のコードを流用する場合は、後述する図 14 のように先頭ブロックのドラッグ操作によりコピーできる。この時、コードがローカル変数を参照している場合は、ローカル変数も自動的にコピーされ、ローカル変数のスプライト属性はコピー元のままであるが、コピー先のスプライト属性をもつローカル変数として動作する。

この方法では、スプライトごとに CPU コアが割り当てられるので、マルチコア CPU のマシンで実行させれば、並列実行により処理速度が向上し、この方法がスクラッチにおけるマルチスレッドの基本である。

2.3. クローン・スプライトを用いたマルチスレッドの実現

マルチスレッドのアプリケーションでは、殆ど同一のコードを複数のスプライトで並列実行させる場面が多い。即ち、図 1-(B)において、「ローカル変数 1」と「ローカル変数 2」および「コード 1」と「コード 2」が殆ど同一である場合が多い。その際、2.2.節のように複数のスプライトを定義し、各スプライトにコードをコピーする操作を繰り返し、同一名のローカル変数を繰り返し定義するのは面倒である。そこで、「Scratch 3.0」には、「クローン」と呼ばれるバーチャルなコピー・スプライトを生成する機能が用意されている。

図 1-(C)に示すように、単一のスプライト上で、変数とコードを定義し、制御ブロック群の「(自分自身)のクローンを作る」を配置すれば、クローン・スプライトが作成される。この時、変数とコードは自動的にコピーされ実行されるので、クローン・スプライトにおいて独立した変数として動作させたい場合は、2.2.節で述べたローカル変数の設定を行っておく必要がある。図 1-(C)に示す「ローカル変数 1」と「ローカル変数 1C」は、変数名は同一だが、各々異なるスプライトに属する独立した変数として、実行時には互いに異なる値が設定されることがある。

クローン・スプライトが作成された時に、スプライトに定義されている、どのコードを起動させるかは、制御ブロック群の「クローンされたとき」を配置する際に指定できる。例えば、図 1-(C)において、「スプライト 1」では「コード 1A」を実行させ、「スプライト 1 のクローン」では「コード 1B」を実行させるようにしている。また、クローン・スプライトは複数個作成できるが、各クローン・スプライトに ID を設定し、各クローン・スプライトの ID に基づいて、異なるコードを実行させるように分岐させることもできる。

2.2.節で述べた複数のスプライトを定義する方法との違いは、2.1.節と同様に、単一のスプライトしか定義していないため、クローン・スプライトごとにコード、コスチューム、音の定義は変更できず、同一コードで異なる動作をさせるには一工夫が必要となる点である。

3. MIDI シーケンサの試作とマルチスレッドによるカノン演奏の実現

本節では、先報告¹⁰⁾の 3.1.節で紹介した「Scratch3.0」の拡張機能「音楽」を用いて、和音を含む楽曲データを打ち込めば MIDI 演奏を実行できる、MIDI シーケンサ(後述する通り、正確には、MIDI シーケンサに類似した音楽プレーヤ)を試作した事例を紹介する。更に、サンプル楽曲として、ドイツ民謡『かえるの合唱』を用い、入力した楽曲データを複数の声部に割り当て、クローン・スプライトを用いたマルチスレッド機能により、カノン演奏(輪唱)を実現する手法を紹介する。

3.1. 拡張機能「音楽」を用いた MIDI シーケンサの試作

「Scratch3.0」の拡張機能「音楽」を用いて、楽曲を演奏させるための API(Application Programming Interface)としては、「(ノートナンバー)の音符を()拍鳴らす」という単音を演奏させるブロックしか提供されていない。和音を含む一般的な楽曲データを演奏させるにあたり、問題点が2つある。まず、単音を鳴らすには音符の音高をノートナンバーで数値指定しなければならないこと。また、和音を鳴らすための汎用的な API が提供されていないこと。本稿ではこれらの問題点に対する解決策を提案するものである。

3.1.1. 「音名-MIDI ノートナンバー変換」独自定義ブロックの開発

拡張機能「音楽」に標準搭載の音符を鳴らすブロック「(ノートナンバー)の音符を()拍鳴らす」にて、音符のノートナンバーを直接指示する方法をとっている。これに対し、表1のように DTM でよく用いられている、英語の音名「C(ド)D(レ)E(ミ)F(ファ)G(ソ)A(ラ)B(シ)」にオクターブ番号を付加した音名表記で指示できるようにすると、音符データを打ち込む際の操作性が向上する。この時、ノートナンバー「60」を音名で表現する方法として、表1のように「C3」(ヤマハ式)以外に「C4」と表記する手法も国内外で使用されるが、本稿では表1のようにヤマハ式で表記する。

表 1 音名－ノートナンバー対応表

			ド		レ		ミ	ファ		ソ		ラ		シ	ド		
音名	C-2	--	C3	C#3	D3	D#3	E3	F3	F#3	G3	G#3	A3	A#3	B3	C4	--	G8
				Db3		Eb3			Gb3		Ab3		Bb3				
ノート ナンバー	0	--	60	61	62	63	64	65	66	67	68	69	70	71	72	--	127

表1に基づいて、音名「C-2」～「G8」の 128 音を MIDI ノートナンバー0～127 に変換する独自定義ブロック「ノートナンバー変換」を実装した事例を図 3 に示す。同ブロックの引数「note」に音名を文字列で入力すると、変数「NN」に変換されたノートナンバーが算出される。本ノートナンバー変換処理は、図 3-(A),(C),(D)に示す3段階で構成される。

図 3-(A) の定義ブロック「ノートナンバー変換」では、文字列「note」の1文字目の英大文字を図 3-(B)で示される7要素の配列「音名」={C, D, E, F, G, A, B} (スクラッチではリストと呼ばれる)に合致する要素と、配列「ナンバー」={0, 2, 4, 5, 7, 9, 11}において対応する要素に基づいて基本ノートナンバーを決定する。

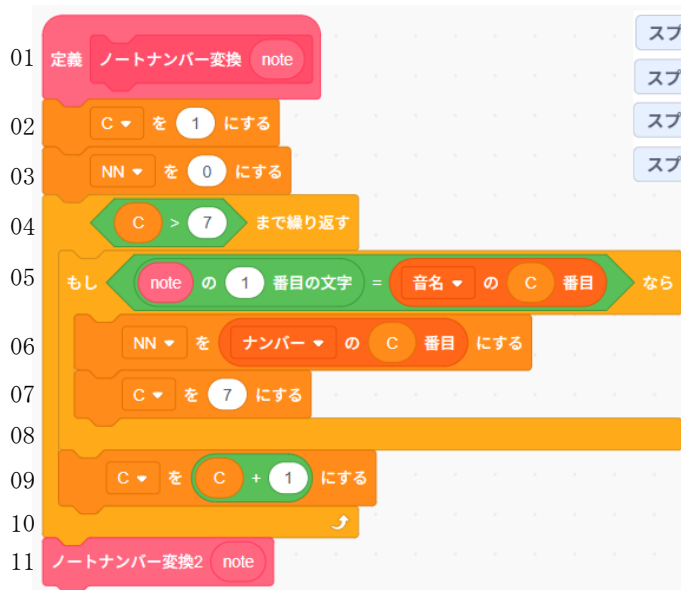
図 3-(C)の定義ブロック「ノートナンバー変換 2」では、引数「note」内に、音名文字列に後続して、半音補正記号の「#」または「b」が付加されている場合、先に決定された基本ノートナンバーに「+1」または「-1」半音の補正を加える。尚、本稿ではダブルシャープ・ダブルフラットはサポートしない。

図 3-(D)の定義ブロック「ノートナンバー変換 3」では、引数「note」内に、音名文字列および半音補正記号に後続して付加されている「-2」～「8」のオクターブ番号を抽出して変数「oct」に設定し、先に決定および補正された基本ノートナンバーに、(oct+2)×12 なるオクターブ補正値を加算して、最終的なノートナンバーを決定する。

図 3-(A)(C)(D)で使用する 4 つの変数「C」「NN」「oct」「P」は、2.2.節で述べた、デフォルト設定のグローバル変数でも動作する。ただし、後述する 4 個のクローン・スプライトを生成して 4 声の輪唱をマルチスレッドにより実行できるようにするため、図 3-(B)に示されるように、全て「スプライト 1」のみで参照されるローカル変数に設定しておく(図 2-(B)に示すように、変数作成時にオプション設定で「このスプライトのみ」を選択)。

また、図 3-(B)に示されるローカル変数「NN2」「長さ」は、後述する和音演奏で使用する変数で、図 3-(B)に示されるローカル変数「楽器」は、後述するカノン演奏(輪唱)を実行する際、4 つの各声部で異なる楽器音を指定できるように追加したものである。これらの変数は、本節のノートナンバー変換処理では参照しないが、同様に「スプライト 1」のみで参照されるローカル変数に設定しておく。

(A) ノートナンバー変換：基本ノート処理

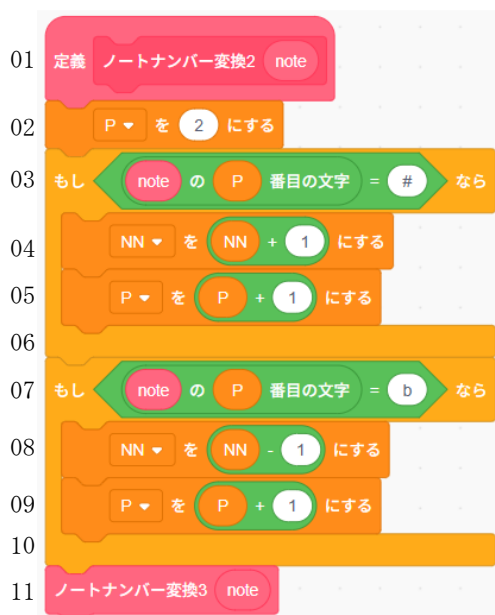


(B) 参照する変数と配列

スプライト1: C	8	スプライト1: NN2	-1
スプライト1: NN	60	スプライト1: 長さ	1.0
スプライト1: oct	3	スプライト1: 楽器	1
スプライト1: P	2		

音名	ナンバー
1 C	1 0
2 D	2 2
3 E	3 4
4 F	4 5
5 G	5 7
6 A	6 9
7 B	7 11
+ 長さ 7 =	+ 長さ 7 =

(C) ノートナンバー変換2：半音処理



(D) ノートナンバー変換3：オクターブ処理

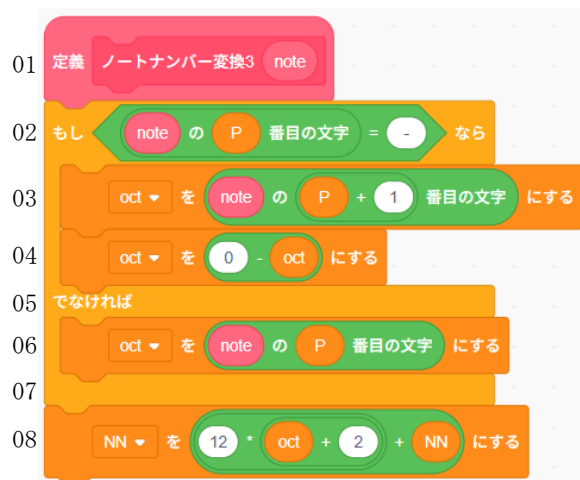


図 3 音名・MIDI ノートナンバー変換処理の実装例 (Scratch3_MIDIseq_F3_1/2/3/C4.sb3)

3.1.2. 和音演奏に関する独自定義ブロックの開発

前節で述べた図 3-(A) の独自定義ブロック「ノートナンバー変換」を、拡張機能「音楽」に標準搭載のブロック「(ノートナンバー)の音符を(長さ)拍鳴らす」を実行させる前に挿入することにより、図 4-(A)のように、音名で指示可能な独自定義ブロック「ノート単音」を構成できる。このブロックを演奏時刻 t にて実行させると、MIDI シーケンサの演奏時刻が $t + (\text{長さ})$ にずれるため、演奏時刻 t にて複数の音を和音演奏させるには工夫が必要である。

例えば、音名「N1」「N2」「N3」の 3 音で構成される三和音を演奏させるには、図 4-(B)に示される

独自定義ブロック「三和音」のように、各音名を図 4-(D)に示される「NN1」「NN2」「NN3」というノートナンバー変数に変換し、引数の「長さ」をローカル変数「長さ」にコピーした上で、演奏時刻 t にて、同一スプライト内で「メッセージ 3C」を送信する。そして、図 4-(C)において、この「メッセージ 3C」を受信すると、拡張機能「音楽」の演奏ブロック「(ノートナンバー)の音符を(長さ)拍鳴らす」に変換された 3 種のノートナンバーを指定して、演奏時刻 t にて、マルチスレッドで同時に演奏される。この時、MIDI シーケンサの演奏時刻は時刻 t の状態に維持されているので、独自定義ブロック「三和音」を起動後に、別の単音や和音を同時に演奏させることもできる。また、和音の演奏時間「長さ」だけずらしたい場合は、拡張機能「音楽」に標準搭載されているブロック「(長さ)拍休む」を用いて、MIDI シーケンサの演奏時刻を $t+(\text{長さ})$ に変更できる。

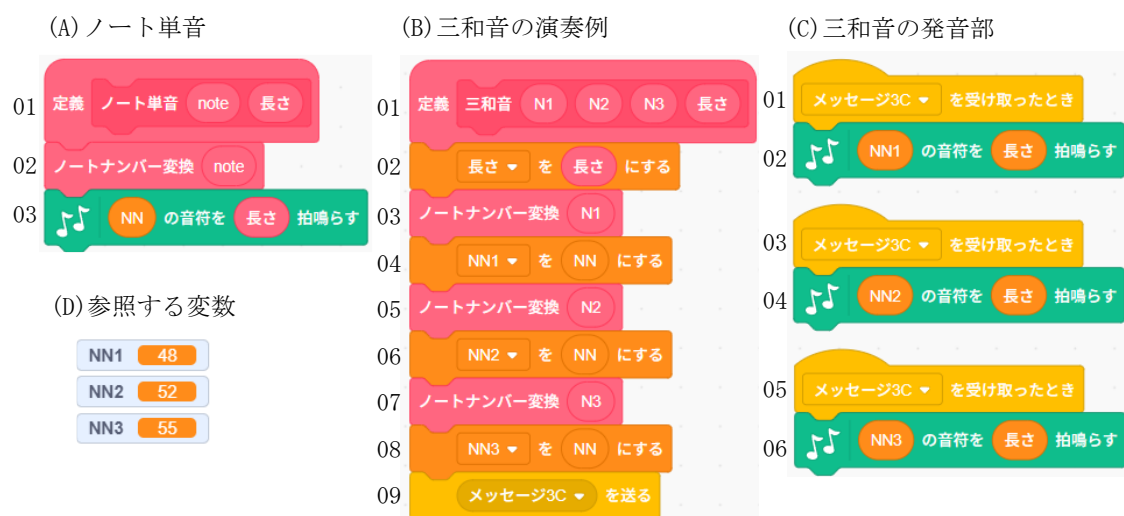


図 4 MIDI シーケンサを構成する基本演奏ブロックの設計 (Scratch3_MIDIseq_F3_1.sb3)

図 4-(B)(C)の方法では既定の数の構成音の和音しか演奏できないため、任意の数で構成される和音を演奏させるには、図 4-(A)の独自定義ブロック「ノート単音」と同様に、単一の音符を演奏させ、かつ、図 4-(B)の独自定義ブロック「三和音」と同様に、MIDI シーケンサの演奏時刻を発音開始時刻 t の状態に維持させるようにすれば良い。これらの機能を実現させるため、図 5-(A)(B)に示されるように、2 種類の手法により独自定義ブロック「ノートと音」を試作した。これらの独自定義ブロック「ノートと音」を複数個並べると、図 5-(C)に示される三和音をはじめ、任意の数で構成される和音を演奏させることができる。

図 5-(A)は、一時的にクローン・スプライトを生成し、クローン・スプライト内で指示された音名「note」をノートナンバーに変換したノートナンバー「NN」の単音を演奏させるようにしたものである。演奏終了後には、生成されたクローン・スプライトは削除される。図 5-(B)は、図 4-(B)(C)の方法と同様に、指示された音名「note」をノートナンバーに変換した上で、単一スプライト上で「メッセージ H」を送信して、変換したノートナンバー「NN」の単音を別のスレッドで演奏させるようにしたものである。

そうすると、図 4-(B)(C)に示される独自定義ブロック「三和音」は、図 5-(C)のように、図 5-(A)または図 5-(B)に示される独自定義ブロック「ノートと音」を 3 回続けて実行させる方法で実装できる。図 5-(C)の独自定義ブロックを演奏時刻 t にて実行させると、音名「N1」「N2」「N3」の 3 音の演奏が順番に開始される。図 5-(B)の 3 段目の独自定義ブロック「ノートナンバー変換」での処理待ちが殆ど無け

れば、これら 3 音はマルチスレッドで殆ど同時に演奏され、MIDI シーケンサの演奏時刻は時刻 t の状態に維持される。

図 5-(B)の実装では、複数のスプライトやクローン・スプライトで並列実行している場合、図 5-(B)の 5 段目で送信している「メッセージ H」が全てのスプライトに送信され、6 段目以降の処理が、1～5 段目の処理を実行していない他のスプライトでも同時に実行されてしまうことに留意する必要がある。

この対策として、本稿では、ローカル変数「NN2」を追加し、初期状態では値「-1」に設定しておく。4 段目で、「ノートナンバー変換」が行われた「NN」をローカル変数「NN2」にコピーした上で、5 段目で「メッセージ H」を送信するようにする。ローカル変数「NN2」は通常「-1」に設定されているので、6 段目で「メッセージ H」を受信したスプライトの中で、ローカル変数「NN2」に 0 以上の値が設定されているスプライトのみ有効で、8 段目で音符の演奏がノートナンバー「NN2」で実行される。この時、演奏される時間は、2 段目により引数「長さ」の値を基にコピーされたローカル変数「長さ」となる。演奏が実行されると、9 段目で、ローカル変数「NN2」は値「-1」に戻される。このようにして、「メッセージ H」を受信した他のスプライトの演奏処理が、同時に実行されることを阻止することができる。

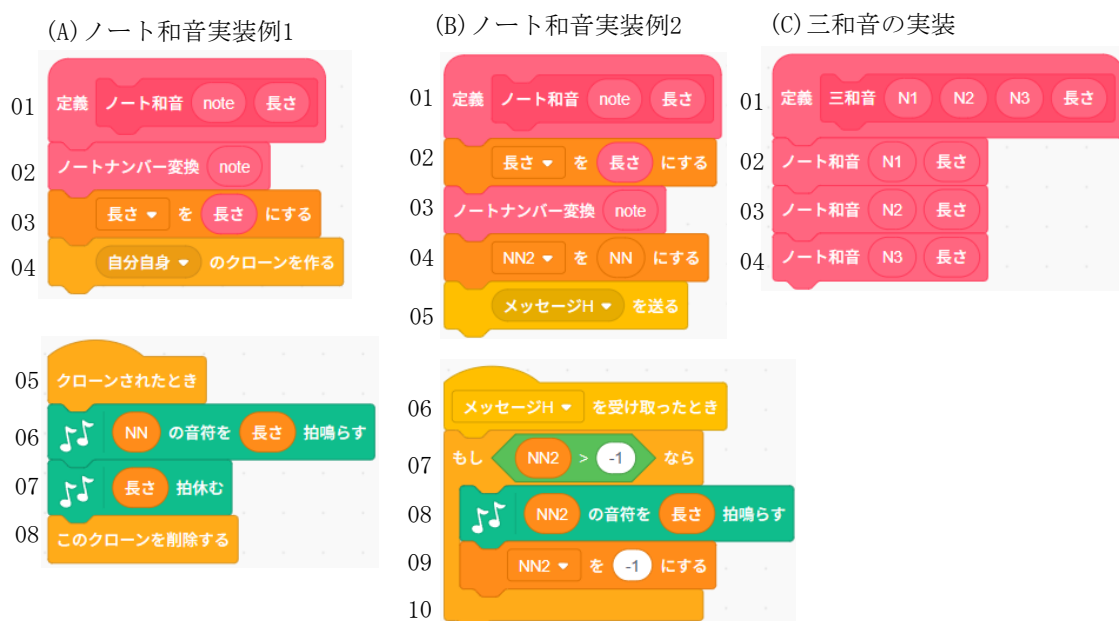


図 5 MIDI シーケンサを構成する和音演奏ブロックの設計
(Scratch3_MIDIseq_F3_2/3/C4.sb3)

3.1.3. 和音演奏に関する独自定義ブロックの評価

前節で述べた図 5-(A)(B) で示される 2 種の「和音ノート」独自定義ブロックにつき、後述する図 6 に示されるサンプル楽曲を用いて、後半 5～8 小節で使用される C2-E2-G2 の三和音の演奏品質にフォーカスして以下 3 通りの手法で比較評価を行った。

第 1 の手法は、前半 1～4 小節で単一音の和音を加える(メロディーと重なり二和音になる)箇所では、図 5-(A)を使用し、後半 5～8 小節で三和音を加える(メロディーと重なり四和音になる)箇所では、図 4-(B)(C)を使用した(Scratch3_MIDIseq_F3_1.sb3)。

第 2 の手法は、前半 1～4 小節で単一音の和音を加える(メロディーと重なり二和音になる)箇所では、

は、図 5-(A)を使用し、後半 5～8 小節で三和音を加える(メロディーと重なり四和音になる)箇所では、図 5-(A)(C)を使用した(Scratch3_MIDIseq_F3_2.sb3)。

第 3 の手法は、前半 1～4 小節で単一音の和音を加える(メロディーと重なり二和音になる)箇所では、図 5-(B)を使用し、後半 5～8 小節で三和音を加える(メロディーと重なり四和音になる)箇所では、図 5-(B)(C)を使用した(Scratch3_MIDIseq_F3_3.sb3)。

第 1 の手法が最も優れているが、三和音に限定され汎用性がないという欠点がある。第 2 の手法は第 1 の手法および第 3 の手法に比べ、特に図 5-(C)において「N2」音と「N3」音の発音の遅れが目立ち、三和音が常にアルペジオのような演奏になってしまう問題が発覚した。また、第 2 の手法では、後述するように、複数の声部で輪唱させる際に、クローン・スプライトを用いたマルチスレッド実行ができないという問題がある。一方、第 3 の手法では、第 2 の手法と同様に、「N2」音と「N3」音の発音の遅れが発生する可能性はあるが、評価した結果では、第 2 の手法ほど顕著な遅れは発生せず、和音の演奏品質にそれほど影響を与えないことを確認した。そこで、図 5-(A)の方法は不採用とし、本稿では以降、第 3 の手法で使った図 5-(B)の方法で進める方針とした。(Scratch3_MIDIseq_F3_C4.sb3)。

3.1.4. 試作した MIDI シーケンサの仕様と使用方法

本稿で紹介する MIDI シーケンサは、「Scratch3.0」の拡張機能「音楽」API に基づいて実装した音楽プレーヤーで、厳密には MIDI 仕様に基づいたシーケンサではない。('Scratch3.0'の拡張機能「音楽」API 自体は JavaScript の MIDI-API に基づいて実装されている)。そのため、以下の通り MIDI 仕様と異なる部分があり、MIDI 仕様のピッチベンドなどのコントロールはサポートしない。また、一般的な MIDI シーケンサに実装されている、MIDI デバイスの制御、リアルタイム MIDI 収録、SMF (Standard MIDI File) ファイルの読み書きなどもサポートしない。

MIDI 仕様では、デルタタイムと呼ばれる直前のイベントとの時間差を各イベントの実行時刻として指示しながら、各音符を「ノートオン(発音開始)」と「ノートオフ(発音終了)」という 2 つのイベント対で入力する。これに対し、本稿で試作した MIDI シーケンサでは、各音符を独自定義ブロック「ノート単音」または「ノート和音」のいずれかを用いて入力する。MIDI 仕様では無音区間(休符)は「ノートオン(発音開始)」時刻のデルタタイムで制御可能なため入力不要だが、本稿で試作した MIDI シーケンサでは、休符について明示的に指示する必要があり、拡張機能「音楽」に標準搭載のブロック「()拍休む」を用いて入力する。

MIDI 仕様では、「ノートオン(発音開始)」イベントから「ノートオフ(発音終了)」イベントまでの実行時刻の差に基づいて、音符の音長を指示するが、本稿で試作した MIDI シーケンサでは、拡張機能「音楽」に標準搭載のブロック「()の音符を()拍鳴らす」のように、拍数で音長を指示するため、事前に、拡張機能「音楽」に標準搭載のブロック「テンポを(BPM)にする」を用いて 1 分あたりの拍数を設定しておく必要がある。例えば、テンポを 120[BPM]に設定すれば、1 分あたり四分音符で 120 拍となり、1 拍の長さは 0.5 秒となる。

独自定義ブロック「ノート単音」を実行すると、シーケンサの時刻が指定した拍数だけずれるが、独自定義ブロック「ノート和音」を実行する場合は、シーケンサの時刻が変化しない。そのため、例えば、三和音を入力する場合は、2 つの和音構成音を「ノート和音」を用いて入力し、残りの 1 つの和音構成音を「ノート単音」を用いて入力すれば良い。あるいは、3 つの和音構成音全てを「ノート和音」を用いて入力した後に、拡張機能「音楽」に標準搭載のブロック「()拍休む」を用いて、シーケンサの時刻を

指定した拍数だけずらすようにしてもよい。

MIDI 仕様では、「ノートオン(発音開始)」イベントにて音の強弱(ベロシティ)を指定できるが、本稿で試作した MIDI シーケンサでは、基本機能「音」に標準搭載されているブロック「音量を()%にする」を用いて、同様な指定は行える。ただし、和音構成音ごとに音量の指定はできない。また、MIDI 仕様のプログラム(楽器音)に相当する設定は、拡張機能「音楽」に標準搭載のブロック「楽器を()にする」を用いて、同様に選択し設定することができる。ただし、選択可能な楽器音は 21 種で、GM (General MIDI) 標準の 128 種のプログラム・コードとは異なるため、留意する必要がある。ちなみに、MIDI 仕様では、音高情報をもたないパーカッションの楽器音は、第 10 チャンネル(Ch.9)上にノートイベントを定義し、ノートナンバーで楽器音を指定する方法をとっているが、Scratch では、拡張機能「音楽」に標準搭載のブロック「(打楽器コード)のドラムを()拍鳴らす」を用いて、18 種の打楽器より指示する方法をとる。

これら音量や楽器音はスプライト単位に設定することができるため、複数の楽器音で構成されるマルチトラックの楽曲を入力する場合は、各トラック(MIDI 仕様ではチャンネル)を異なるスプライトに割り当てれば良い。この時、各トラックは演奏データが異なるだけで、演奏制御のプログラムのコードは殆ど同じになるため、クローン・スプライトを用いても良い。後述する 3.3 節では、4 種のクローン・スプライトを生成して、各々異なる楽器音を設定している。

3.2. サンプル楽曲「かえるの合唱」の音符データの打ち込み

前節で述べた MIDI シーケンサを構成する図 4-(A)の独自定義ブロック「ノート単音」と図 5-(B)の独自定義ブロック「ノートと和音」を用いて、図 6 に示すサンプル楽曲『かえるの合唱』(ドイツ民謡)を打ち込んだ結果を図 7 に示す。楽曲は全 8 小節で構成され、先頭に、独自定義ブロック「初期設定」を配置して、音量、テンポ、楽器音を指定するようにした。



図 6 サンプル楽譜:ドイツ民謡『かえるの合唱』

音量は、後述する 4 つの声部でカノン演奏(輪唱)させる際は、100%では合奏音が歪む可能性があるがあるので、図 7 に示すように、「音量を(40)%にする」と設定した。テンポは四分音符=120[BPM]に設定することにより、四分音符の長さは 1 拍(=0.5 秒)とした。尚、楽器音は後述する 4 つの声部でカノン演奏(輪唱)させる際、声部ごとに異なる楽器音を指定できるように、ローカル変数「楽器」で楽器音コードを設定できるようにした。第 6 小節は第 5 小節の繰り返しであるため、第 6 小節を除く全 7 小

節の楽曲データを小節ごとに独自定義ブロックで入力した。

第 5～8 小節で使用される「C2」「E2」「G2」の三和音は、図 7 下段中央に示すように、新たに独自定義ブロック「和音 C」を作成し、図 5-(C)の独自定義ブロック「三和音」と同様に、図 5-(B)に示される独自定義ブロック「ノートと和音」を 3 回続けて実行させる方法で実装した。また、第 7 小節は各拍が八分音符の同一音で構成されるため、図 7 下段中央に示すように、「二連音」という独自定義ブロックを新たに作成して入力するようにした。

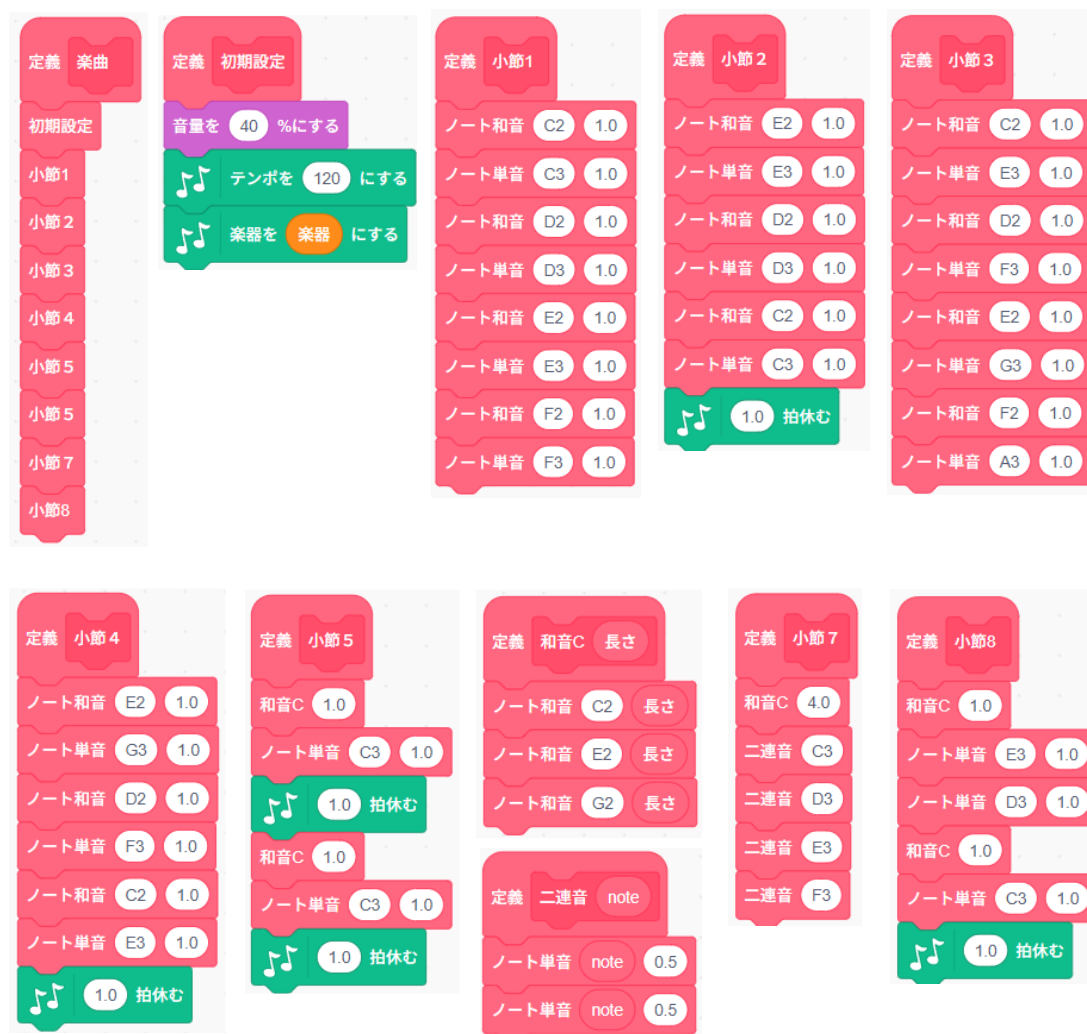


図 7 サンプル楽曲の試作した MIDI シーケンサへの打ち込み例
(Scratch3_MIDIseq_F3_1/2/3/C4.sb3)

3.3. クローン・スプライトを用いたカノン演奏の実現

単一のスプライト上に、図 3-(A)～(D)に示すコード、図 4-(A)(D)に示すコード、図 5-(B)(C)に示すコード、および図 7 に示すコードを配置し、起動用のメインプログラムとして、図 8-(A)に示すコードを追加して、楽器音としてピアノ音(No.1)を指定して実行させれば、図 6 に示すサンプル楽曲が各小節 4.0 拍(2 秒)のテンポでピアノ演奏される。尚、図 8-(A)の 3 行目は、前述した独自定義ブロック「ノートと和音」で、他のスプライトに対する排他処理に使用される変数「NN2」の初期化処理である。

続いて、図 6 に示すサンプル楽曲を 2 小節ずつ遅らせながら、同一楽曲を 4 つの声部でカノン演奏(輪唱)させるようにしたのが、図 8-(B)に示すコードである。4 つの声部に対応させて 4 個のクローン・スプライトを直前に生成したクローン・スプライトより 2 小節分(8 拍、4 秒)だけ遅らせながら生成し、マルチスレッドで実行させている。また、各声部を識別しやすくするため、各クローンを生成する際、ローカル変数「楽器」に、4 種の楽器音{1: ピアノ, 3: オルガン, 8: チェロ, 12: フルート}を順次指定し、各声部に異なる楽器音を割り当てるようにした。音量は 3.2 節で述べたように、一律に 40% に設定した。尚、図 8-(C)の 2 行目は図 8-(A)の 3 行目と同様に、前述した独自定義ブロック「ノートと和音」で、他のスプライトに対する排他処理に使用される変数「NN2」の初期化処理である。

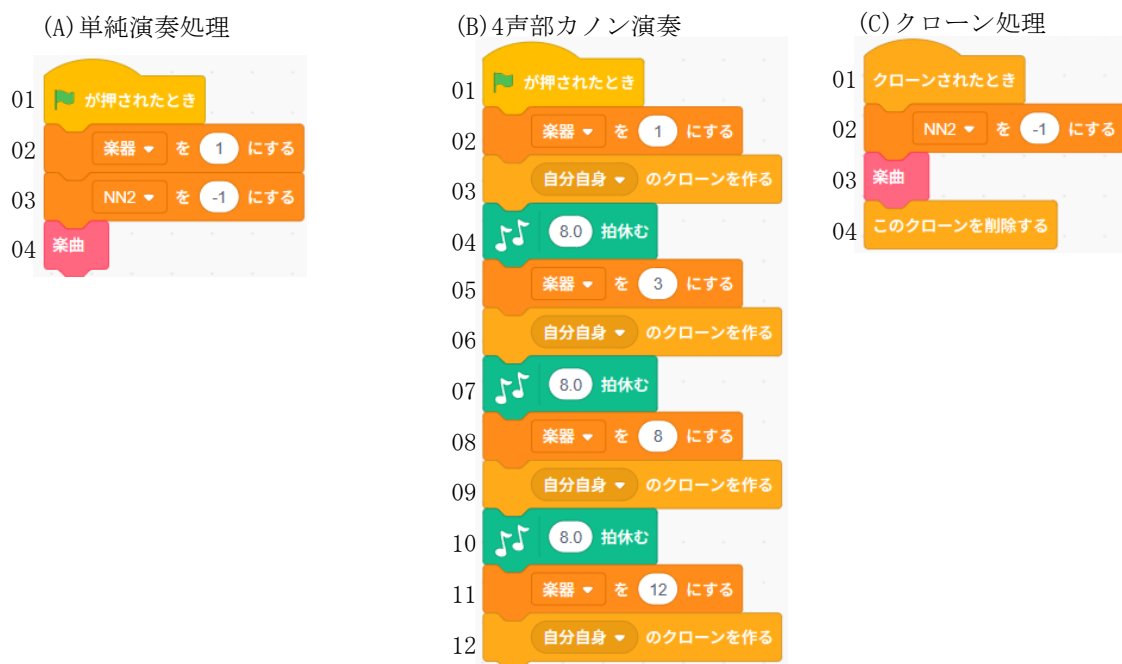


図 8 サンプル楽譜の演奏起動処理と 4 声部カノン演奏例
(Scratch3_MIDIseq_F3_1/2/3/C4.sb3)

4. 三和音自動演奏機能付き 21 鍵ソフトウェア鍵盤の試作

先報告¹⁰⁾の 4.4.節にて、「Scratch3.0」の拡張機能「音楽」「ペン」を用いて、1 オクターブ強 14 鍵のソフトウェア・ピアノ鍵盤を試作した事例を紹介した。本稿では、これを拡張し、ユーザが打鍵したキーをルート音として、三度音、五度音を加えた、長三和音や短三和音を自動演奏させるアプリケーションを実装する。猫・犬・蝶のキャラクターからなる 3 つのスプライトを設定し、根音をスプライト 1 (猫キャラクター)、三度音をスプライト 2 (犬キャラクター)、五度音をスプライト 3 (蝶キャラクター)に割り当てて、3 つのスプライトを用いたマルチスレッドで和音演奏を実行させるようにした。

4.1. 起動処理と鍵盤表示(スプライト 1 のみ)

先報告¹⁰⁾では、1 オクターブ強 14 鍵のソフトウェア・ピアノ鍵盤をグラフィック表示させていたが、本稿では 1 オクターブ内で指示されたキーをルートとして、三度音、五度音を加えた三和音を演奏させるため、鍵盤全体を 21 鍵に拡張させる。

図 9-(A)が本アプリケーションの起動処理で、先報告¹⁰⁾の図 11-(A)に、演奏和音のモード指定を追加している。図 9-(A)の 2～3 段目のブロックにて、変数「mode」には、長三和音の場合は 1、短三和音の場合は 2、打鍵したキーを単音でそのまま演奏する場合は 0 を設定する。図 9-(B)が本アプリケーションの実行例で、mode=1 で、C3 キーを打鍵した場合の長三和音の演奏例を示す。

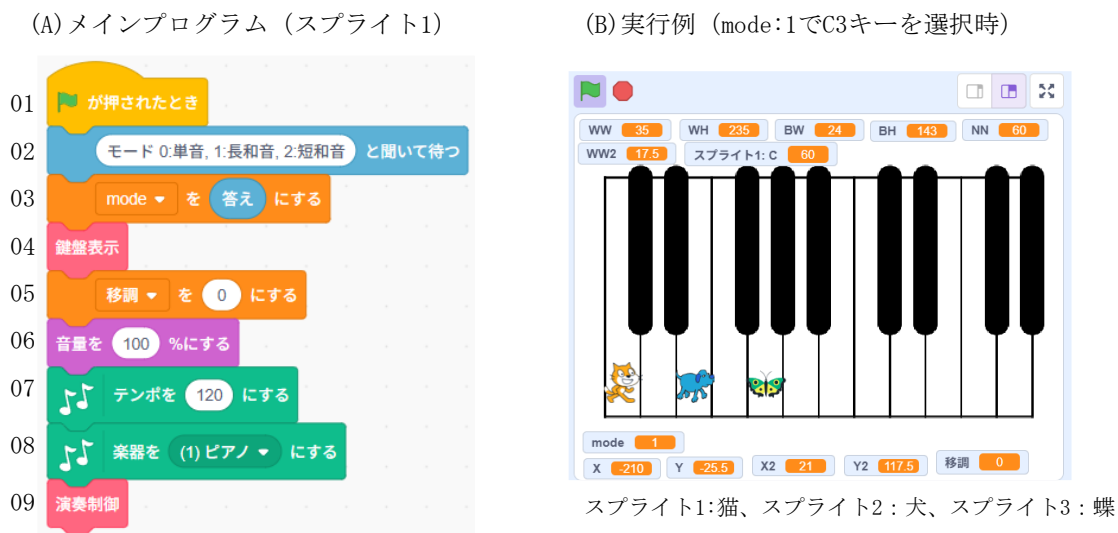


図 9 21 鍵ソフトウェア鍵盤の起動処理と実行例(Scratch3_piano_F9_3.sb3 スプライト 1)

図 9-(A)の 4 段目のブロックで図 10-(A)に示す「鍵盤表示」独自定義ブロックを起動している。図 10-(A)の 2 段目のブロックでピアノ鍵盤を押下する操作を行うデフォルトの猫キャラクターのサイズを 40%に指定している。尚、スプライト 2 の犬キャラクターおよびスプライト 3 の蝶キャラクターのサイズは、各スプライトの初期設定画面(図 14 の右下部参照)にて、各々 30%に設定している。後述するように、表示されているピアノ鍵盤に対するマウス操作による対話処理を行う、図 11 に示す、「演奏制御」独自定義ブロックは、スプライト 1 のみに実装されているため、猫キャラクターのみ、マウスに連動して鍵盤の上を動くカーソルの役目をする。

図 10-(A)の 3～6 段目のブロックでは、白鍵と黒鍵の縦横サイズを画素単位で指定しており、変数「WW」は白鍵の幅、変数「WH」は白鍵の縦方向の長さ、変数「BW」は黒鍵の幅、変数「BH」は黒鍵の縦方向の長さである。図 10-(A)の 7～9 段目のブロックでは、画面をクリアにした後に、後続の 10 段目のブロックで実行する、図 10-(B)に示す「白鍵」独自定義ブロックにおけるペンの色と太さを指定している。白鍵を表示した後に、図 10-(A)の 11 段目のブロックで図 10-(C)に示す「黒鍵」独自定義ブロックを実行することにより、図 9-(B)に示すピアノ鍵盤の表示が完成する。

図 10-(B)に示す「白鍵」独自定義ブロックでは、まず 2～7 段目のブロックにて、図 9-(B) に示すピアノ鍵盤全体の下端の水平線(6 段目のブロック)と上端の水平線(7 段目のブロック)を描画している。水平線を描画するにあたり、図 10-(B)の 2～5 段目のブロックにより、 $X=-WW \times 4$ 、 $Y=-WH/2$ 、 $X2=WW \times 4$ 、 $Y2=WH/2$ に設定し、6,7 段目のブロックでは、図 10-(D)に示す「直線描画」独自定義ブロックを実行させている。

図 10-(B)の「白鍵」独自定義ブロックの 8～13 段目のブロックでは、変数「C」を 0 から 12 まで変えながら、ピアノ鍵盤全体の 13 本の垂直線を変数「WW」間隔で描画している。垂直線を描画するにあ

たり、図 10-(B)の 11 段目のブロックでは、同様に図 10-(D)に示す「直線描画」独自定義ブロックを 13 回実行させている。



図 10 鍵盤表示の実装例(Scratch3_piano_F9_3.sb3 スプライト 1)

図 10-(C)に示す「黒鍵」独自定義ブロックでは、2 段目のブロックでペンの太さを変数「BW」に変更している。図 10-(C)の 3～12 段目のブロックでは、変数「C」を 1 から 12 まで変えながら、ピアノ鍵盤の黒鍵部に太さ「BW」の幅をもつ短い垂直線を変数「WW」間隔で描画している。垂直線を描画するにあたり、7 段目のブロックでは同様に図 10-(D)に示す「直線描画」独自定義ブロックを実行させている。9～11 段目の条件判断ブロックでは、 $C=3$ 「E#3」と $C=7$ 「B#3」および $C=10$ 「E#4」の場合は白鍵になるため、垂直線の描画をスキップする処理を行っている。従って、変数「C」を 1 から 12 まで変えながら、9 本の垂直線を黒鍵として描画している。

以上のようにして、図 9-(B)に示すピアノ鍵盤を得ることができる。図 9-(A)、図 10-(A)～(D)に示す全てのコードはスプライト 1 のみに配置する。

4.2. 演奏制御と打鍵位置の認識(スプライト 1 のみ)

先報告¹⁰⁾の 4.4.2.節にて、前節で作成したソフトウェア・ピアノ鍵盤に対して、マウス操作により鍵盤をクリックして対話形式に楽器音を鳴らす手動演奏機能の試作例を紹介した。本稿では、21 鍵に拡張したピアノ鍵盤に対して同様な機能を実現する。



図 11 演奏制御・マウス指示による打鍵位置の認識 (Scratch3_piano_F9_3.sb3 スプライト 1)

話を図 9 に戻すと、図 9-(A)の 5 段目のブロックで、変数「移調」に MIDI ノートナンバーへのオフセット値を設定できるようにしている。表示するピアノ鍵盤の左端の白鍵は、初期状態では、ピアノの中央の「ド(C3, MIDI ノートナンバー60)」に設定されているが、変数「移調」により演奏音の音高を上下させることができる。図 9-(A)の 6～8 段目のブロックは、図 7 の「初期設定」独自定義ブロックと同一である。図 9-(A)の 6 段目のブロックで、演奏する楽器音の音量を%で指定する。本来、ピアノ鍵盤

では、楽器音を鳴らす長さ(デュレーション)を、鍵盤を押してから放すまでの時間で指示するが、マウス操作では難しいため、本アプリケーションでは固定の長さだけ鳴らすようにする。そのため、図 9-(A)の 7 段目のブロックでテンポを四分音符=120[BPM]に指定して、鍵盤をクリックした際に、楽器音を鳴らす長さ(デュレーション)を、先報告¹⁰⁾と同様に拍単位で固定の長さ(1.0 拍=0.5 秒)で演奏するようにしている。図 9-(A)の 8 段目のブロックで、演奏する楽器音の楽器(「ピアノ」など)を指定する。

図 9-(A)の 9 段目のブロックで図 11 に示す「演奏制御」独自定義ブロックを起動している。

図 11 は、2~15 段目のブロックでスペースキーが押されるまで無限に反復するループを構成しており、マウスがクリックされると、3~14 段目のブロックが実行され、「マウスの x 座標」と「マウスの y 座標」を基に、MIDI ノートナンバーを算出して変数「C」に格納し、12 段目のブロックで「メッセージ 1」を送って楽器音を鳴らすように制御している。この時、本稿では、後述するように 3 個のスプライトに「メッセージ 1」を送って三和音を鳴らすように制御している。

まず、図 11 の 4 段目のブロックで「マウスの x 座標」と「マウスの y 座標」を読み取り、5 段目のブロックで、変数「X2」に $X2 = \text{「マウスの x 座標」} - X$ の値を設定し、鍵盤位置を算出するのに使用する。変数「Y」には黒鍵の下端の Y 座標が収納されているので、6 段目のブロックで、「マウスの y 座標」>Y ならば、黒鍵を指示したものと判断し、7 段目のブロックで図 12-(B)に示す「黒鍵算出」独自定義ブロックを起動する。そうでなければ、8 段目のブロックで白鍵を指示したものと判断し、9 段目のブロックで図 12-(A)に示す「白鍵算出」独自定義ブロックを起動する。

図 12-(A)の「白鍵算出」独自定義ブロックでは、2,3 段目のブロックで、白鍵の鍵盤位置を、変数「X2」を「WW」で割り算することにより求め、小数点以下切り捨てにより変数「C」に格納する。左端の白鍵の MIDI ノートナンバーを 60 (ピアノ中央の「ド」)とすると、

C<3 の場合は、5 段目のブロックにより $C = C \times 2 + 60$ にて MIDI ノートナンバーが得られ、
3≤C<7 の場合は、8 段目のブロックにより $C = C \times 2 + 59$ にて MIDI ノートナンバーが得られ、
7≤C<10 の場合は、11 段目のブロックにより $C = C \times 2 + 58$ にて MIDI ノートナンバーが得られ、
10≤C の場合は、13 段目のブロックにより $C = C \times 2 + 57$ にて MIDI ノートナンバーが得られる。

図 12-(B)の「黒鍵算出」独自定義ブロックでは、2,3 段目のブロックで、黒鍵の鍵盤位置を、変数「X2-WW/2」を「WW」で割り算することにより求め、小数点以下切り捨てにより変数「C」に格納する。

C<3 の場合は、5 段目のブロックにより $C = C \times 2 + 61$ にて MIDI ノートナンバーが得られ、
3≤C<7 の場合は、8 段目のブロックにより $C = C \times 2 + 60$ にて MIDI ノートナンバーが得られ、
7≤C<10 の場合は、11 段目のブロックにより $C = C \times 2 + 59$ にて MIDI ノートナンバーが得られ、
10≤C の場合は、13 段目のブロックにより $C = C \times 2 + 58$ にて MIDI ノートナンバーが得られる。

このようにして、算出された MIDI ノートナンバー「C」に対して、図 11 の 11 段目のブロックにて「移調」する補正を行い、演奏するノートナンバー「NN」を決定し、12 段目のブロックで「メッセージ 1」を送って楽器音を鳴らすように制御している。この時、図 11 の 12 段目のブロックで、図 13-(A)の 15 段目で使用しているブロック「(NN)の音符を(1.0)拍鳴らす」を直接実行させることもできるが、後続の音をクリックして鳴らすまでに 0.5 秒以上待たされて、0.5 秒未満の高速な連打が行えなくなる。そこで、マルチスレッド手法を適用し、MIDI サウンドを演奏する処理を別のスレッドに分離し、図 11 の 12 段目のブロックでは「メッセージ 1」を送って、別のスレッドで演奏させるようにする。これにより、前の音の演奏が終了しなくても、後続の音を鳴らすことが可能になる。また、本稿では、後述するように 3 個のスプライトに「メッセージ 1」を送って三和音を鳴らすように制御している。

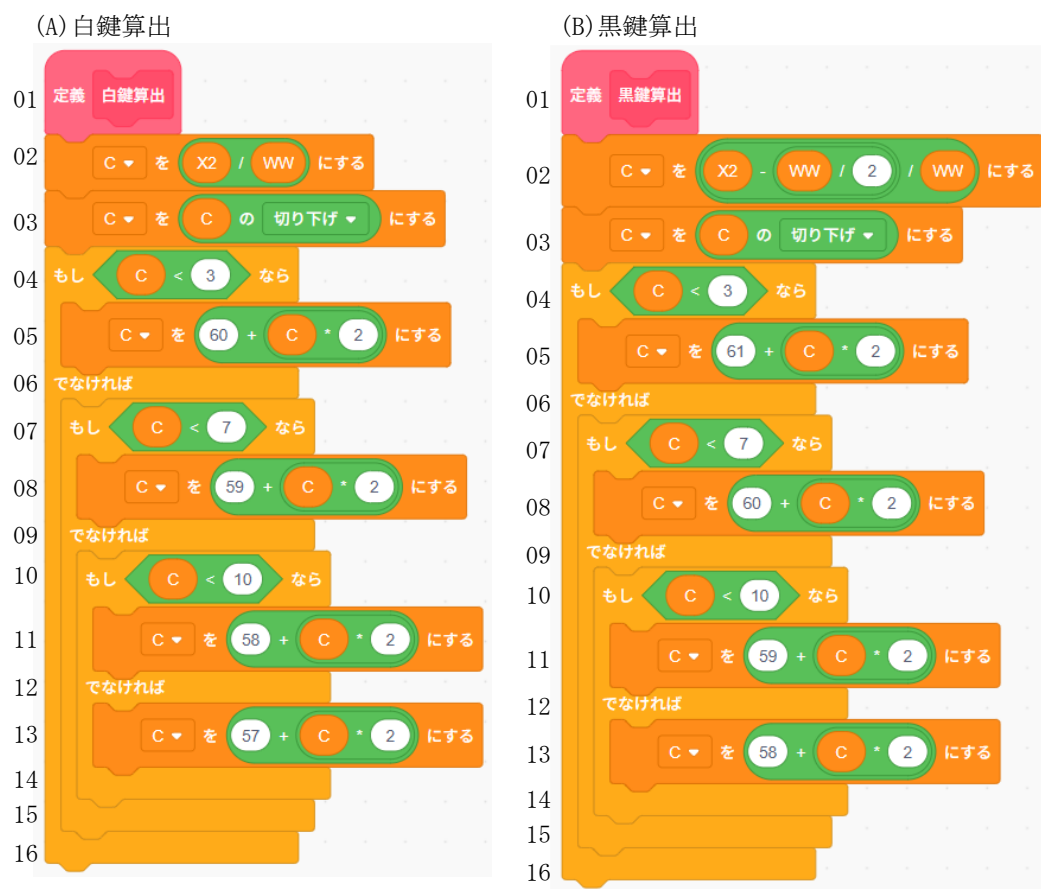


図 12 打鍵位置認識:黒鍵・白鍵算出 (Scratch3_piano_F9_3.sb3 スプライト 1)

4.3. 演奏処理と打鍵位置の表示(全スプライト共通)

本稿のアプリケーションでは、4.1.節で作成したピアノ鍵盤に対して打鍵を行い、4.2.節で認識した打鍵音を根音として、長三和音または短三和音を構成する2音につき、鍵盤を自動的にクリックさせて楽器音を同時に鳴らす自動演奏機能を付加する。そのため、本節では、長三和音または短三和音を構成する3音の各々ノートナンバーに基づいて、鍵盤位置を算出してマウスカーソルを移動させて、指定されたMIDIノートナンバーの楽器音を鳴らす処理を作成する。

図13-(A)の「演奏」独自定義ブロックでは、引数として与えられた「音高」パラメータと「音長」パラメータを用いて、15段目のブロックで演奏させるのが主であるが、併せて、14段目のブロックで図13-(B)に示す「座標計算」独自定義ブロックを起動してピアノ鍵盤上の3種のスプライトのキャラクターの位置を「音高」に対応する鍵盤の位置に移動させる制御を行っている。

図13-(B)に示す「座標計算」独自定義ブロックを起動するにあたり、図13-(A)の2から13段目のブロックで、「音高」パラメータを基に、鍵盤位置を指示する変数「C」を算出する。これは、Cの値が1(半音)上がったときに、白鍵から白鍵に移動する箇所、Cの値に1を加える補正を行う。具体的には、2段目のブロックで $C = \text{「音高」}$ として、3,4段目のブロックで $C > 76$ ならば、 $C = C + 3$ に補正し、6,7段目のブロックで $C > 71$ ならば、 $C = C + 2$ に補正し、9,10段目のブロックで $64 < C \leq 71$ ならば、 $C = C + 1$ に補正する。

図13-(B)の「座標計算」独自定義ブロックでは、変数「X」には白鍵の左端のX座標が収納されて

いる。そこで、2 段目のブロックで $WW2=WW/2$ として、3 段目のブロックでキャラクターの X 座標を、 $(C-60) \times WW2 + X + WW2$ で与える。



図 13 自動打鍵演奏処理 (Scratch3_piano_F9_3.sb3 3 スプライト共通)

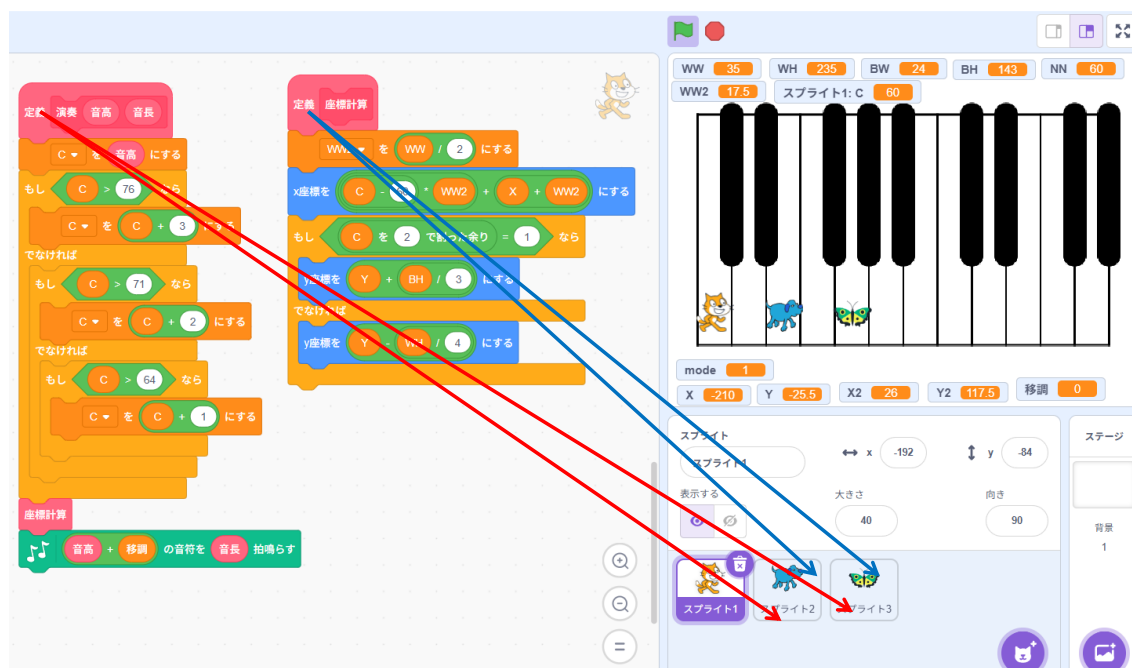


図 14 スプライト間のコードのコピー操作 (Scratch3_piano_F8_3.sb3 スプライト 1)

一方、変数「Y」には黒鍵の下端の Y 座標が収納されている。そこで、4 段目のブロックで C が奇数(C を 2 で割った余りが 1)の場合は黒鍵であるから、5 段目のブロックでキャラクターの Y 座標を、 $Y+BH/3$ で与え、C が偶数の場合は白鍵であるから、7 段目のブロックでキャラクターの Y 座標を、 $Y-WH/4$ で与える。

図 13-(A)(B)に示すコードはスプライト 1 だけでなく、スプライト 2 およびスプライト 3 でも実行させる必要があるため、スプライト 1 に記述された図 13-(A)の「演奏」独自定義ブロックおよび図 13-(B)の「座標計算」独自定義ブロックのコード一式をスプライト 2 およびスプライト 3 にコピーする必要がある。コピー操作は、図 14 に示すように、「演奏」独自定義ブロックおよび「座標計算」独自定義ブロックの各々先頭の定義ブロックを、スプライト 2 およびスプライト 3 のアイコンにドラッグ&ドロップ操作することにより実行できる。

4.4. 三和音構成音の打鍵(3 スプライト別並列処理)

4.2.節で述べた図 11 の「演奏制御」独自定義ブロックの 12 段目で、「メッセージ 1」が送信されたら、3 種のスプライトで同メッセージを同時に受信し、4.3.節で述べた図 13-(A)の「演奏」独自定義ブロックをスプライトごとに変数「mode」の値に基づいて実行する。変数「mode」には、長三和音の場合は 1、短三和音の場合は 2、打鍵したキーを単音で演奏する場合は 0 があらかじめ設定されている。

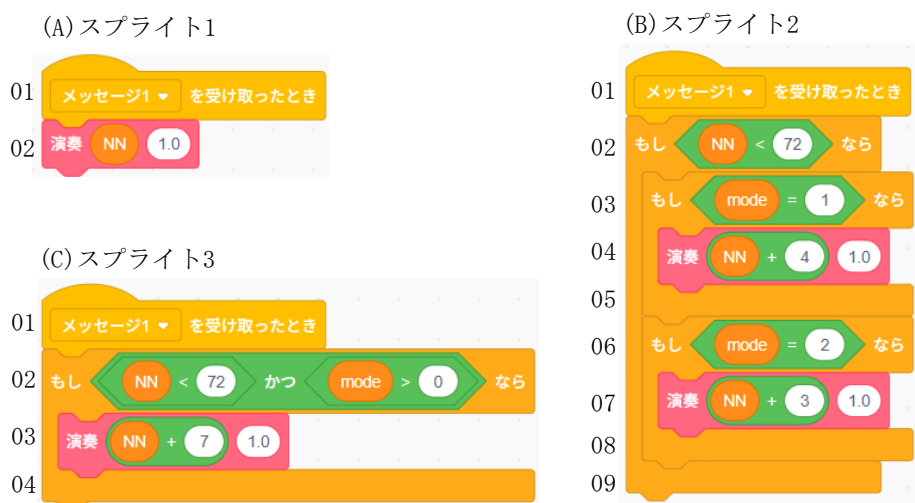


図 15 3 スプライトでのアンサンブル演奏 (Scratch3_piano_F9_3.sb3 各スプライト)

図 15-(A)はスプライト 1 での動作を示し、打鍵されたノートナンバー「NN」を図 13-(A)の「演奏」独自定義ブロックにそのまま渡して演奏させる。音長のパラメータは本稿では一律 1.0 拍(0.5 秒)に設定している。

図 15-(B)はスプライト 2 での動作を示し、打鍵されたノートナンバー「NN」が 72 未満で、mode=1 であれば、長三度音「NN+4」を、mode=2 であれば、短三度音「NN+3」を、図 13-(A)の「演奏」独自定義ブロックに渡して演奏させる。

図 15-(C)はスプライト 3 での動作を示し、打鍵されたノートナンバー「NN」が 72 未満で、mode>0 であれば、五度音「NN+7」を、図 13-(A)の「演奏」独自定義ブロックに渡して演奏させる。

5. まとめ

以上、本稿では、先報告¹⁰⁾で紹介した「Scratch3.0」の拡張機能「音楽」を用いて、和音を含む楽曲データを打ち込めば MIDI 演奏を実行できる、MIDI シーケンサを試作した事例を紹介した。更に、入力した楽曲データを 4 つの声部に割り当て、4 つのクローン・スプライトを用いたマルチスレッド機能により、カノン演奏(輪唱)を実現する手法を紹介した。「Scratch3.0」の制約から、MIDI 仕様のピッチベンドなどのコントロールはサポートできないが、楽器音の指定、ベロシティ(音量)の指定はサポートした。

3.1.3.節で述べたように、本稿で提案した和音演奏に関する独自定義ブロックでは、和音構成音の演奏を時系列に実行させているため、和音構成音の数が多くなると、演奏遅れが発生し、アルペジオ風の演奏になってしまうという問題がある。「Scratch3.0」の制約により本問題をどこまで解消できるか不明だが、今後の検討課題としたい。

また、先報告¹⁰⁾にて「Scratch3.0」の拡張機能「音楽」「ペン」を用いて試作した、ソフトウェア・ピアノ鍵盤の鍵数を、本稿では 21 鍵に拡張し、ユーザが打鍵したキーをルート音として、長三和音や短三和音を自動演奏させるアプリケーションを、3 つのスプライトを用いたマルチスレッドにより試作した事例を紹介した。今後はソフトウェア鍵盤の鍵数を拡張し、自動演奏可能な和音のバリエーションの拡充を検討したい。試作例では、音長を 1 拍(0.5)秒固定にしているが、これを可変にする方法についても併せて検討したい。

本稿で紹介したプログラミング例題の全てのデータおよび、その基になった、筆者が担当している本学・情報系科目向けの Scratch プログラミング演習教材(テキストとプログラミング例題の全データ)については、以下サイトにて一式を公開しているので、教育・研究・その他にご活用ください。

[独自制作 Scratch 演習教材の公開サイト(2025.9 現在、2025 年 9 月 16 日版を公開)]

以下サイトにて、本稿に掲載されている全ての例題のプログラムデータが収録されているファイル[Scratch_kiyou2509.zip]をダウンロードできます。

(各図に明記されている”Scratch3_MIDIseq_F3_1.sb3”などのファイル名で各データが収録されています)。

また、基本操作と例題[Scratch_manual_v3.pdf]と、高度なプログラミング事例その 1(グラフィックスと音響)[Scratch_advance_v3.pdf]、および高度なプログラミング事例その 2(描画の高速化)[Scratch_advanceh_v3.pdf]の全 3 巻からなる独自制作 Scratch 演習教材、および掲載プログラムの全データが収録されているファイル[Scratch_example.zip]を以下サイトにてダウンロードできます。

筆者のホームページ:<http://www.bekkoame.ne.jp/~modegi/>

または <https://sites.google.com/view/hptoshiomodegi>

筆者の連絡先:modegi@bekkoame.ne.jp

参考文献

- 1) 松林弘治『子どもを億万長者にしたければプログラミングの基礎を教えなさい』、(株)KADOKAWA、初版第1刷、February 20.2015、136-148 頁。
- 2) 石嶋洋平著、安藤昇監修『子どもの才能を引き出す最高の学び プログラミング教育』、(株)あさ出版、第1刷 July 8.2018、78-87 頁。

- 3) 石井英男, 吉岡直人, 赤石昭宏, 相川いずみ, 森谷健一『小学生からはじめるプログラミングの本』、日経パソコン編、日経 BP、電子書籍初版 February 12.2021, 43-98 頁。
- 4) Scratch Web 版:<https://scratch.mit.edu/>、Scratch ダウンロード版:
<https://scratch.mit.edu/download/> (2025 年 9 月アクセス)。
- 5) 鈴木喬裕『はじめての人も、挫折した人も「Scratch で今からはじめるプログラミング」Scratch だ
からスラスラわかる』、日経 BP、電子書籍初版 March 29.2021, 23-56 頁。
- 6) Ali Moghiseh and Andreas Jablonski, “ToolIP, Visual Programming for Image
Processing”, http://www.ipol.im/event/2012_imlib/slides/toolip.pdf (2025 年 9 月ア
クセス)。
- 7) Max/Msp:<https://akihikomatsumoto.com/maxmsp/max.html> (2025 年 9 月アクセス)。
- 8) 山内卓哉『「音とコンピュータ」そのプログラミング、サウンドデザインそしてメディアアート』、
(株)インプレス R&D、電子書籍 Ver.1.2 版 December 7.2016, 7-47 頁。
- 9) Unity のリアルタイム開発プラットフォーム | 3D/2D, VR/AR のエンジン:
<https://unity.com/ja> (2025 年 9 月アクセス)。
- 10) 茂出木敏雄『ビジュアル・プログラミング環境「Scratch3.0」を用いた音響アプリケーションの開発』
尚美学園大学芸術情報研究 第 34 号、2021 年 12 月、45-66 頁。
(<https://shobi-u.repo.nii.ac.jp/records/763> 2025 年 9 月アクセス)。