

AI x MUSIC

目指せAIミュージシャン！

人工知能作曲基礎講座

【VERSUSイノベーションスクール】

GoogleのAI音楽ライブラリーMagentaで体験する未来の音楽制作

第3回



AI（機械学習）で作曲できるキーボード！

AWS DeepComposer が発表

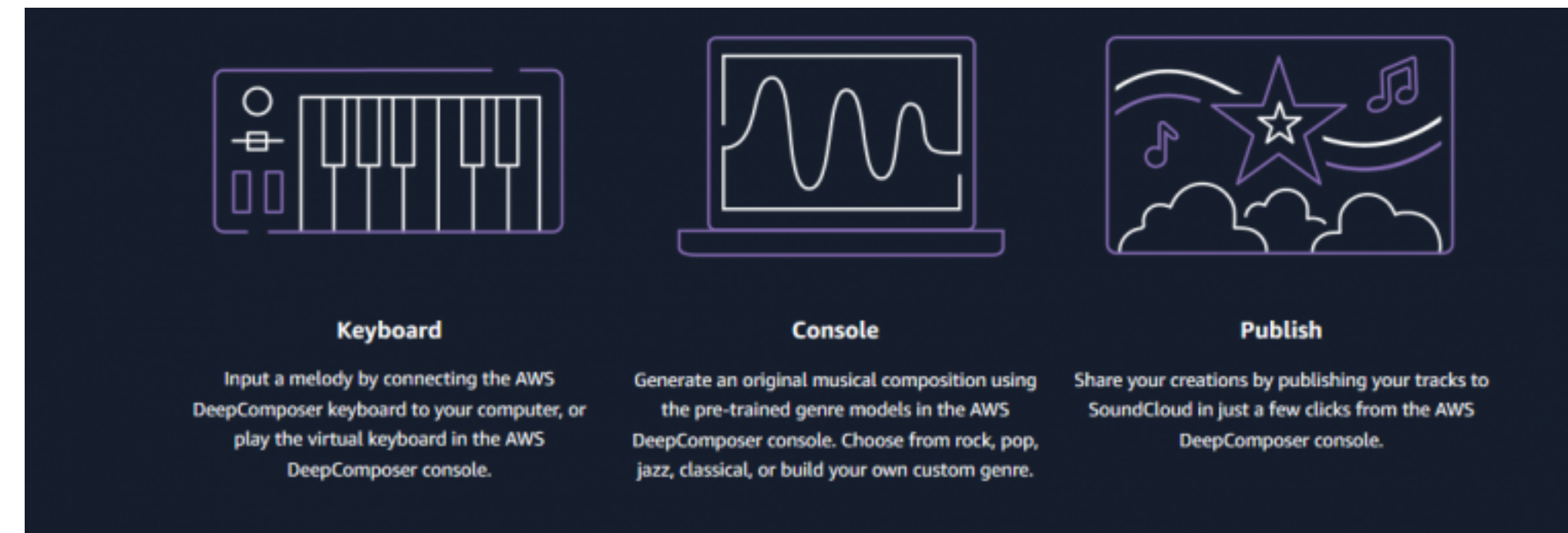
AI（機械学習）で作曲できるキーボード！ AWS DeepComposer

AWS（アマゾンウェブサービス）が世界初の機械学習で音楽生成できるキーボードを年次イベントAWS re:Invent 2019で発表しました。

<https://aws.amazon.com/jp/deepcomposer/>



AWS DeepComposerでの生成の流れ



- 1・AWSにログインしDeepComposerコンソールへ（バージニア北部リージョンなので注意）。
 - 2・DeepComposerキーボードから演奏しメロディー入力。またはコンソール画面にも仮想キーボードがあるのでPCでの入力も可能な様です。
 - 3・生成モデルの選択。Pre Trained（学習済み）モデルを使用し、ロック、ポップ、ジャズ、クラシックから希望のジャンルを選択。
 - 4・生成。複数の楽器演奏アレンジの曲が生成されます。
 - 5・生成曲はコンソールで視聴可能。
 - 6・SoundCloud経由でダイレクトにシェアできます。
- 3のモデルの選択は、自分で学習データを用意し、独自モデルを作る事も可能。

MusicVae

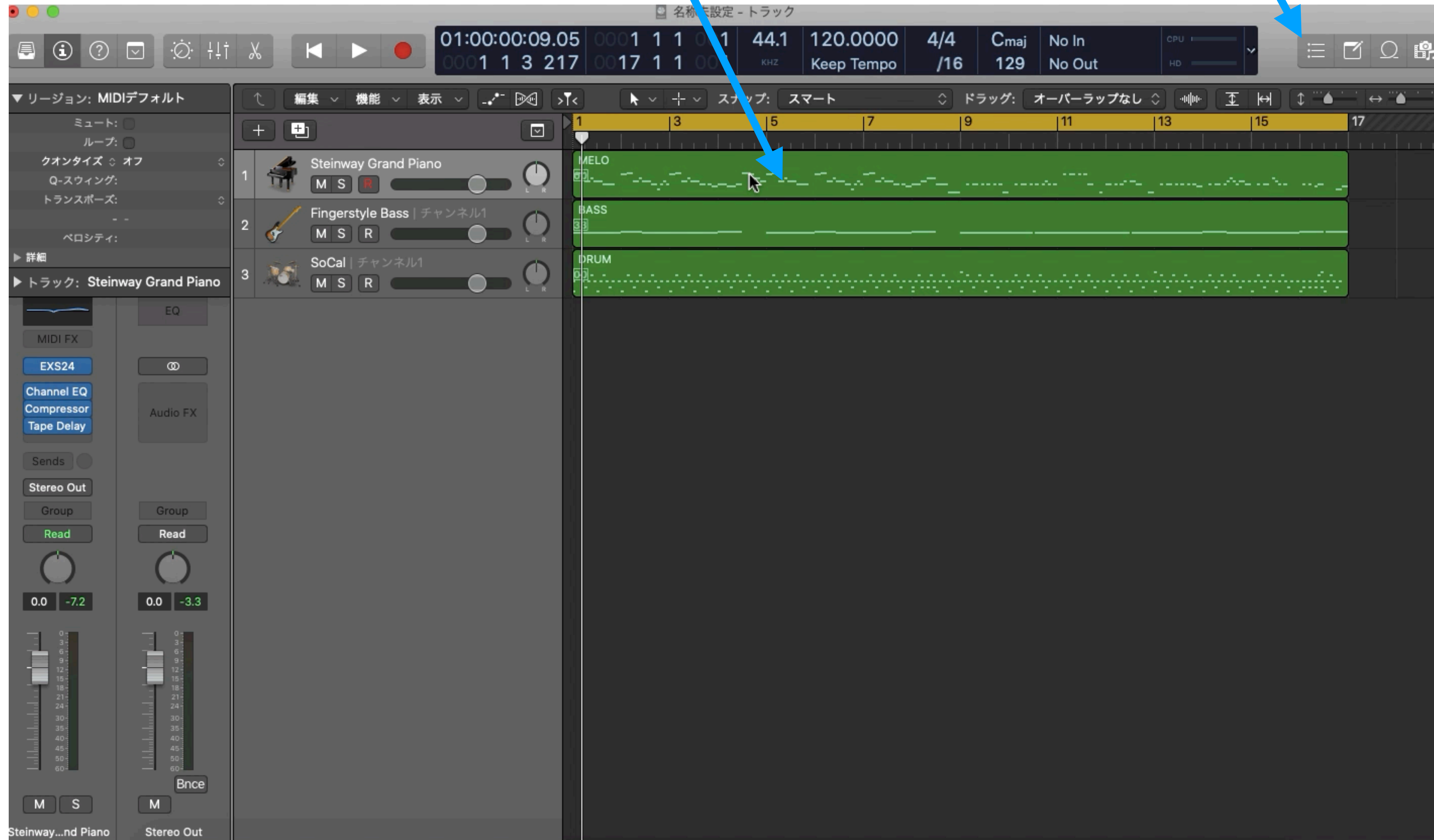
Interpolateモードの補足

- ・使用するMIDIファイルの長さは問いませんが最適な結果を得るためにはモデルの長さに合わせてください。
- ・ただし最低モデルの長さ以上のファイルが必要です。
- ・マルチトラックのファイルでドラムを含むモデルの場合、ドラムトラックをch10にする事
- ・その他のパート（メロディー、ベース）は同一チャンネルでも良いですが、ドラムチャンネルも合わせそれぞれにプログラムチェンジを入れる事

Logic Pro でのMIDIチャンネル変更

トラック選択

リストエディター



Magentaでできる音楽生成（本講座内で解説予定）

- 単音のシンプルなメロディー生成
- ドラムトラックの生成
- 3パート演奏（メロディー、ベース、ドラム）の楽曲生成
- **コード進行に沿ったアドリブメロディー生成**
- 単音メロディーにハーモニーを生成
- 表現力豊かなピアノ楽曲の生成
- Ableton Live（または他のDAW）での音楽生成プラグイン活用

ImprovRNN

Improv RNNとはどんなモデルか？

Improv RNNはMelody RNNをベースにした単音のメロディー生成モデルです。

Melody RNNとの違いは、任意に指定されたコード進行に従って生成される事。

これによりコード進行に合わせたアドリブメロディーの生成を可能にします。

improvはImprovisation（即興演奏）の略です。

Improv RNNは通常MIDIファイルの代わりにMusic XMLファイルのリードシートで学習されています。

学習済みファイルのダウンロード（公式GitHubのImprov RNNページ中央）

https://github.com/tensorflow/magenta/tree/master/magenta/models/improv_rnn

First, set up your [Magenta environment](#). Next, you can either use a pre-trained model or train your own.

Pre-trained

If you want to get started right away, you can use a model that we've pre-trained on thousands of MIDI files:

- [chord_pitches_improv](#)

こちらからダウンロード

Generate a melody over chords

```
BUNDLE_PATH=<absolute path of .mag file>  
CONFIG=<one of 'basic_improv', 'attention_improv' or 'chord_pitches_improv', matching the bundle>
```

Basic Improv

Melody RNNの Basicに似たメロディ生成を行います。

違いは任意の入力されたコード進行に従ってメロディー生成する点です。

コードはOne-hot 表現によるベクトルで48のトライアドコードに対応しています。

C, C#, D, D#, E, F, F#, G, G#, A, A#, Bの各トライアド

Major, Minor, Augmented, Diminished

に対応

One-Hot表現のベクトルによるコードの表現

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0.....]

例えばこれはDの指定されたトライアドを表すなど

MacでのBasic Improv音楽生成コード例

```
improv_rnn_generate \  
--config=basic_improv \  
--bundle_file=/ご自身のパス/chord_pitches_improv.mag \  
--output_dir=/ご自身のパス \  
--num_outputs=5 \  
--qpm=120 \  
--primer_melody="[60]" \  
--backing_chords="C G Am F C G Am F" \  
--render_chords
```

improv_rnn_generateを実行

設定の指定

学習済みデータ指定

テンポの指定

生成用のノートナンバー指定

コード進行の演奏データも生成

コード進行8小節

configとbundle_fileを合わせる必要はありません

Windows でのBasic Improv音楽生成コード例

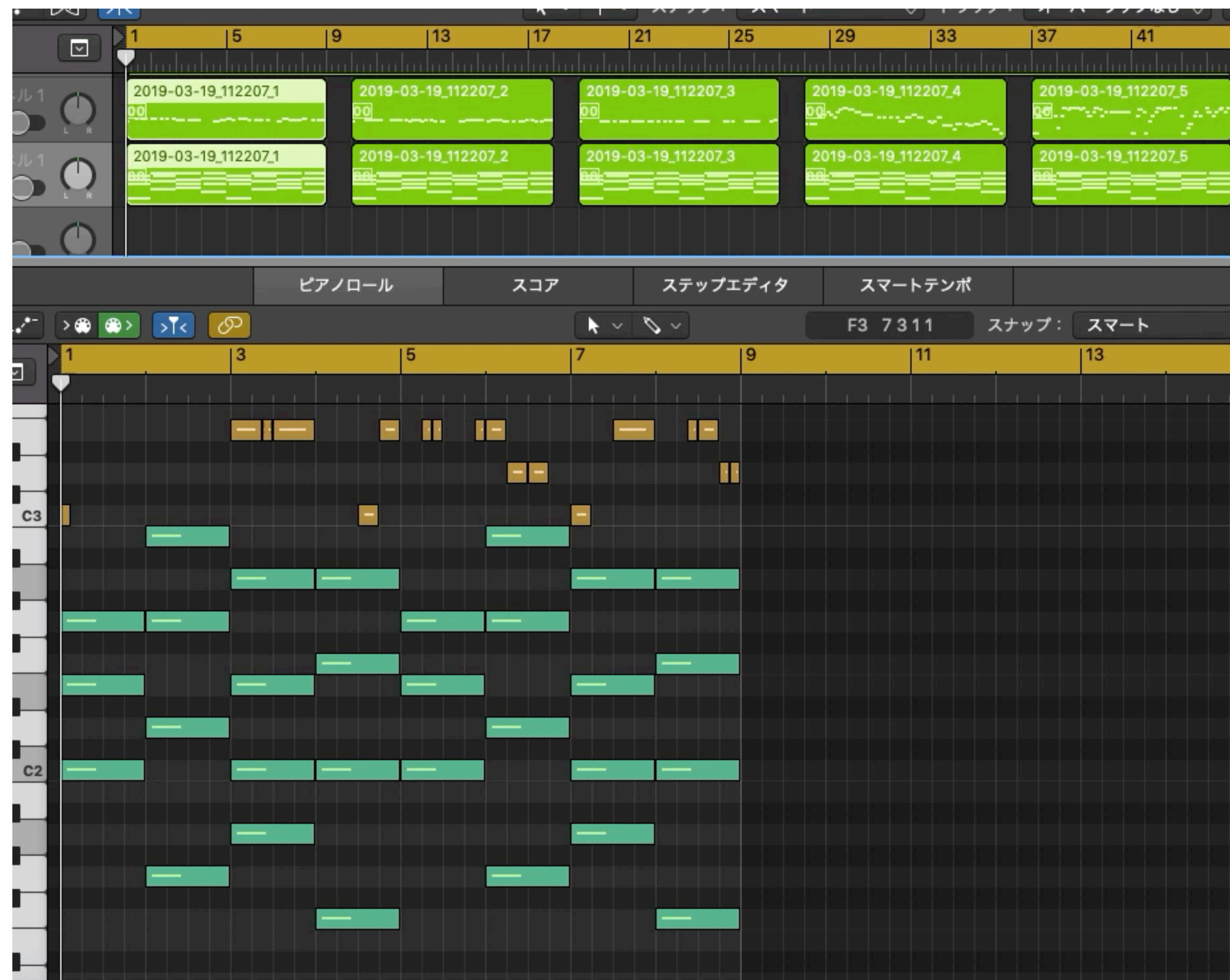
```
improv_rnn_generate ^ ← improv_rnn_generateを実行
--config=basic_improv ^ ← 設定の指定
--bundle_file=¥ご自身のパス¥chord_pitches_improv.mag ^
--output_dir=¥ご自身のパス ^
--num_outputs=5 ^
--qpm=120 ^ ← テンポの指定
--primer_melody="[60]" ^ ← 生成用のノートナンバー指定
--backing_chords="C G Am F C G Am F" ^
--render_chords ← コード進行の演奏データも生成
```

学習済みデータ指定

コード進行8小節

configとbundle_fileを合わせる必要はありません

Improv RNN Basic Improvでの音楽生成



※資料では動画は再生できません

Attention Improv

Melody RNNの Attentionに似たメロディ生成を行います。

より時間軸に整合性を保ったメロディー生成を可能にするモデルです。

Basic同様に任意の入力されたコード進行に従ってメロディー生成、

コードはOne-hot 表現によるベクトルで48のトライアドコードに対応しています。

C, C#, D, D#, E, F, F#, G, G#, A, A#, Bの各トライアド

Major, Minor, Augmented, Diminished

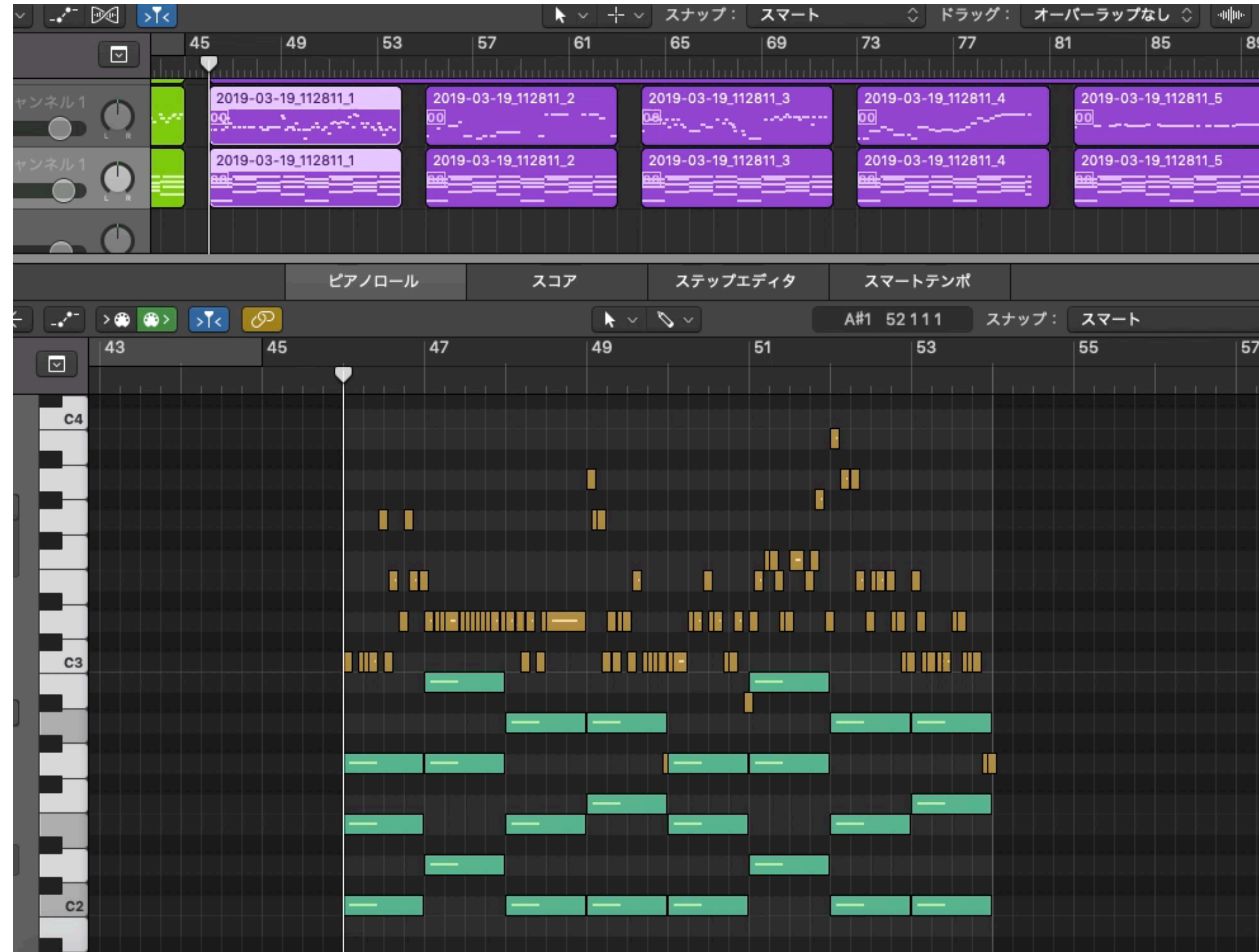
に対応

One-Hot表現のベクトルによるコードの表現

```
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0.....]
```

例えばこれはDの指定されたトライアドを表すなど

Improv RNN Attention Improvでの音楽生成



※資料では動画は再生できません

Chord Pitches Improv

Basic Improvに似たメロディ生成を行います。コードの指定が違います。より複雑なコード、およびコード進行の指定が可能です。

One-hot表現でルート音を指定します。

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

これはCを表します

バイナリーのベクトルによってコードのピッチクラス（使用音階のセット）を表現します

```
[1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0]
```

例えばこれはC7#9を表す。赤字が#9のテンションノート

One-hot表現でコードのベース音を指定します。

```
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
```

これはDを表します

これにより分数コードの指定も可能です。例：C7/Dなど

improv_rnn_generateを実行

Chord Pitches Improv

```
improv_rnn_generate\  
--config=chord-pitches_improv \  
--bundle_file=/ご自身のパス/chord_pitches_improv.mag \  
--output_dir=/ご自身のパス/  
--num_outputs=5 \  
--primer_melody="[60]" \  
--backing_chords="C G Am F C G Am F" \  
--render_chords
```

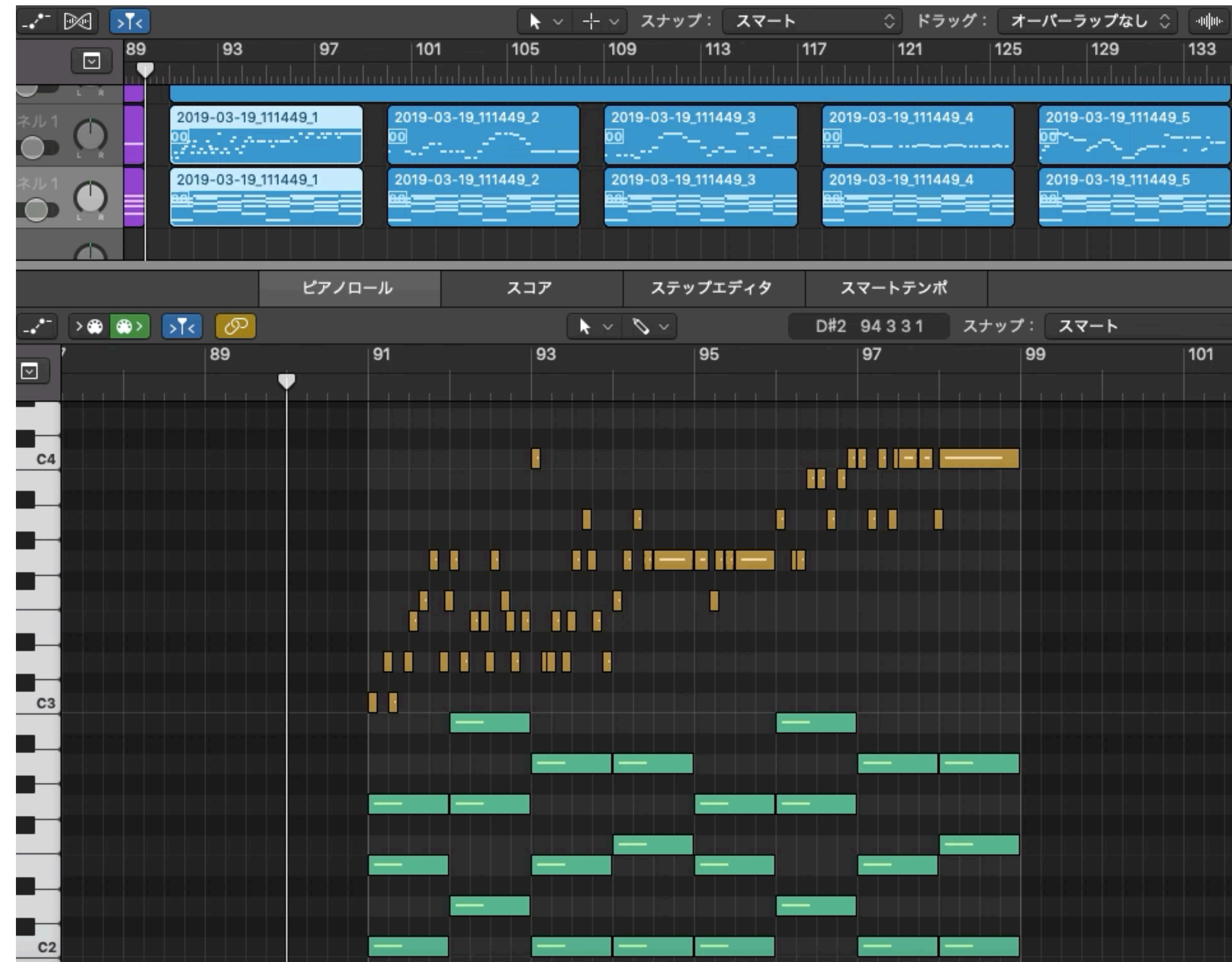
モデルの指定

magファイルの指定

生成用のノートナンバー指定

コード進行8小節

Improv RNN Chord Pitches Improvでの音楽生成



※資料では動画は再生できません

任意のMIDIファイルでの生成も可能

Chord Pitches Improv

```
improv_rnn_generate \  
--config=chord_pitches_improv \  
--bundle_file=/Volumes/Transcend/magenta-model/chord_pitches_improv.mag \  
--output_dir=/任意のパス \  
--num_outputs=5 \  
--primer_midi=任意のパス/primer.mid \  
--backing_chords="C G Am F C G Am F" \  
--render_chords
```

improv_rnn_generateを実行

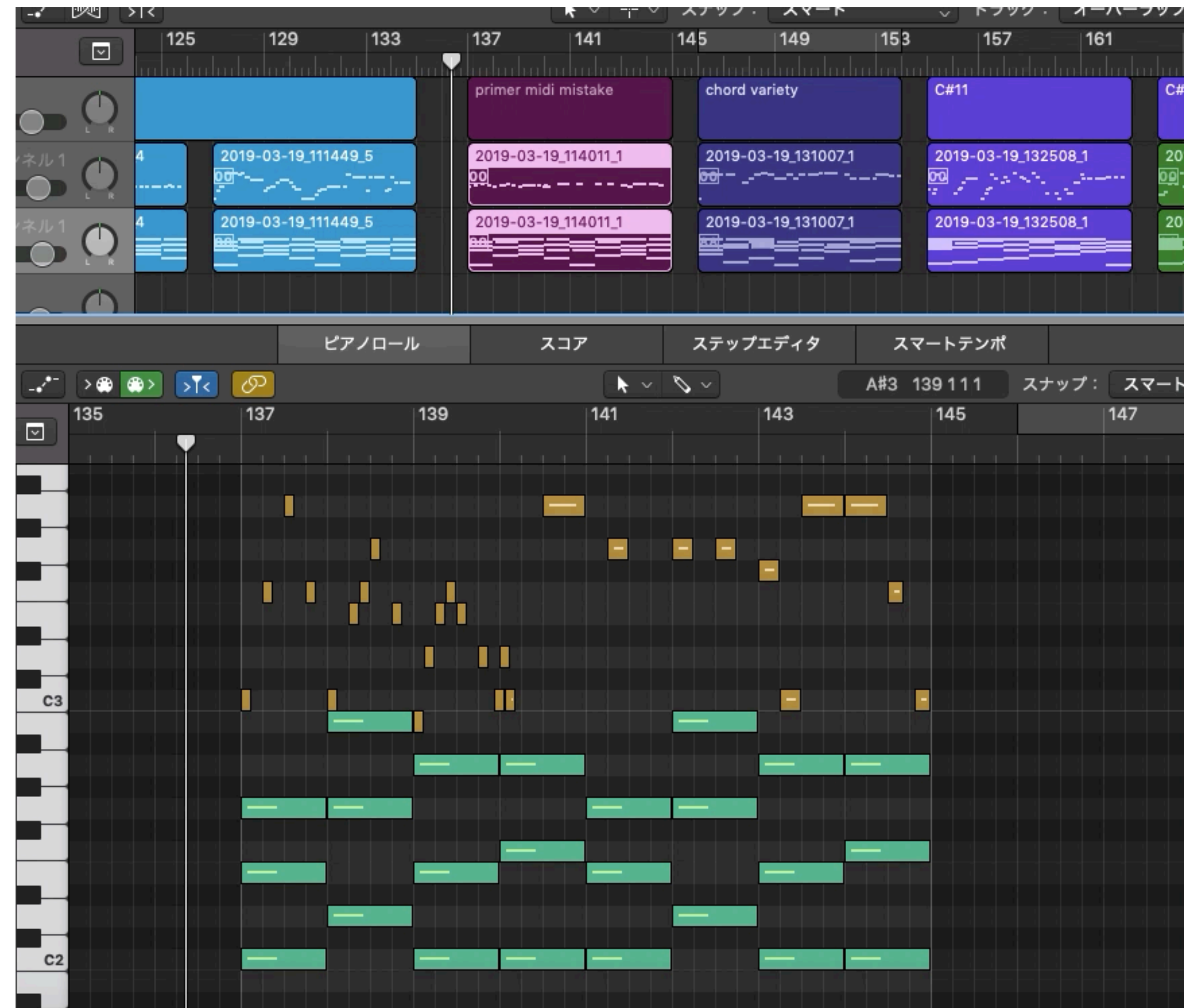
モデルの指定

magファイルの指定

生成用のMIDIファイル指定

コード進行8小節

Improv RNN Chord Pitches Improvでの音楽生成



※資料では動画は再生できません

コード進行のバリエーション

Chord Pitches Improv

```
improv_rnn_generate \  
-config=chord-pitches_improv \  
--bundle_file=/ご自身のパス/chord_pitches_improv.mag \  
-output_dir=/ご自身のパス/  
--num_outputs=5 \  
-primer_melody="[57]" \  
-backing_chords="Am Dm G7/F C F Bm7b5 Eb E"\  
--render_chords
```

improv_rnn_generateを実行

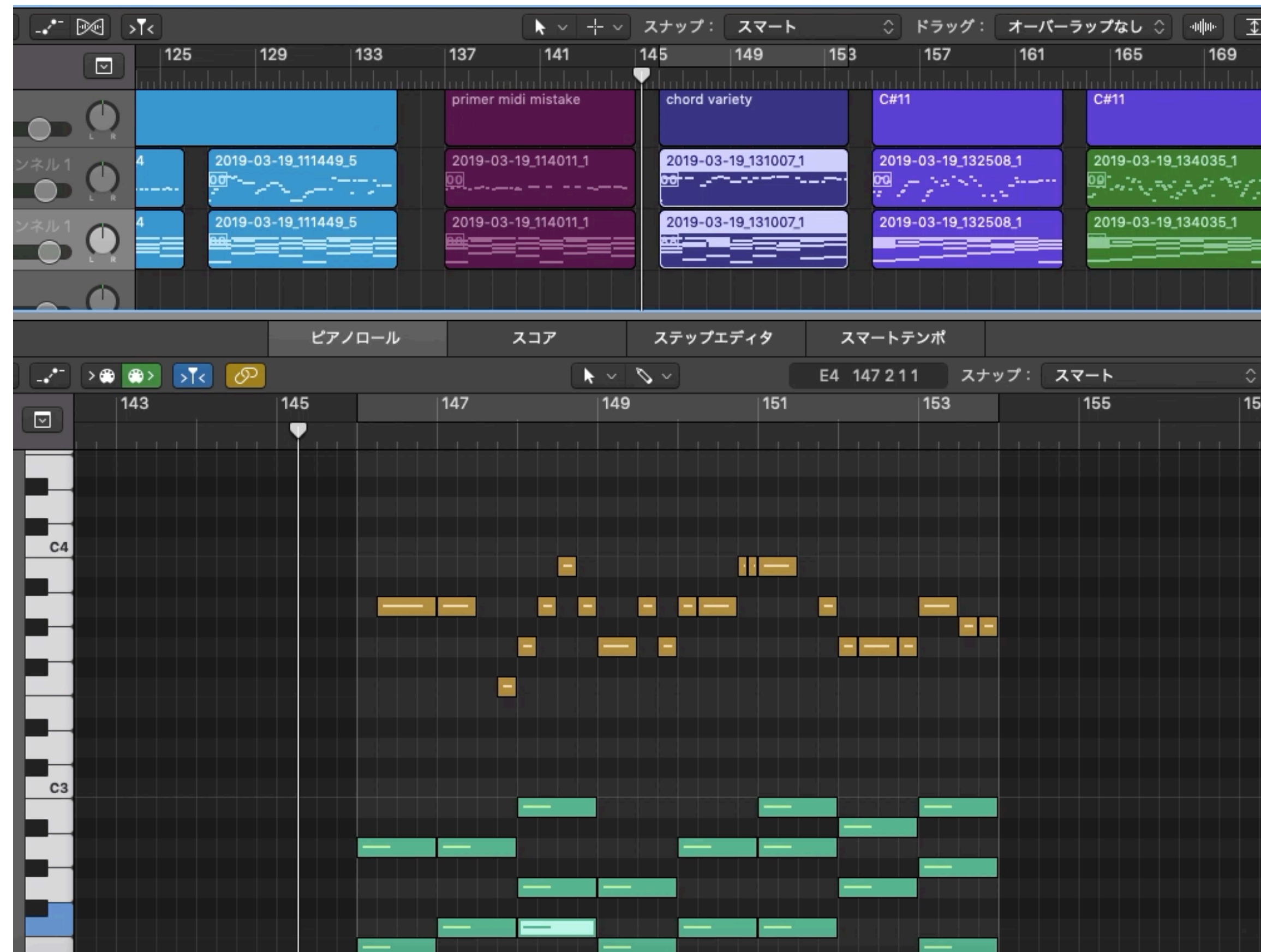
モデルの指定

magファイルの指定

生成用のノートナンバー指定

コード進行8小節
分数コードも可能

Improv RNN Chord Pitches Improvでの音楽生成



※資料では動画は再生できません

Improv RNNのモデル

4和音以上や分数コードの使用も可能です

Chord Pitches Improv

```
python magenta/models/improv_rnn/improv_rnn_generate.py \  
--config=chord-pitches_improv \  
--bundle_file=/ご自身のパス/chord_pitches_improv.mag \  
--output_dir=/ご自身のパス/  
--num_outputs=5 \  
--primer_melody="[57]" \  
--backing_chords="C#m11 D E F G A B C" \  
--render_chords
```

improv_rnn_generate.pyを実行

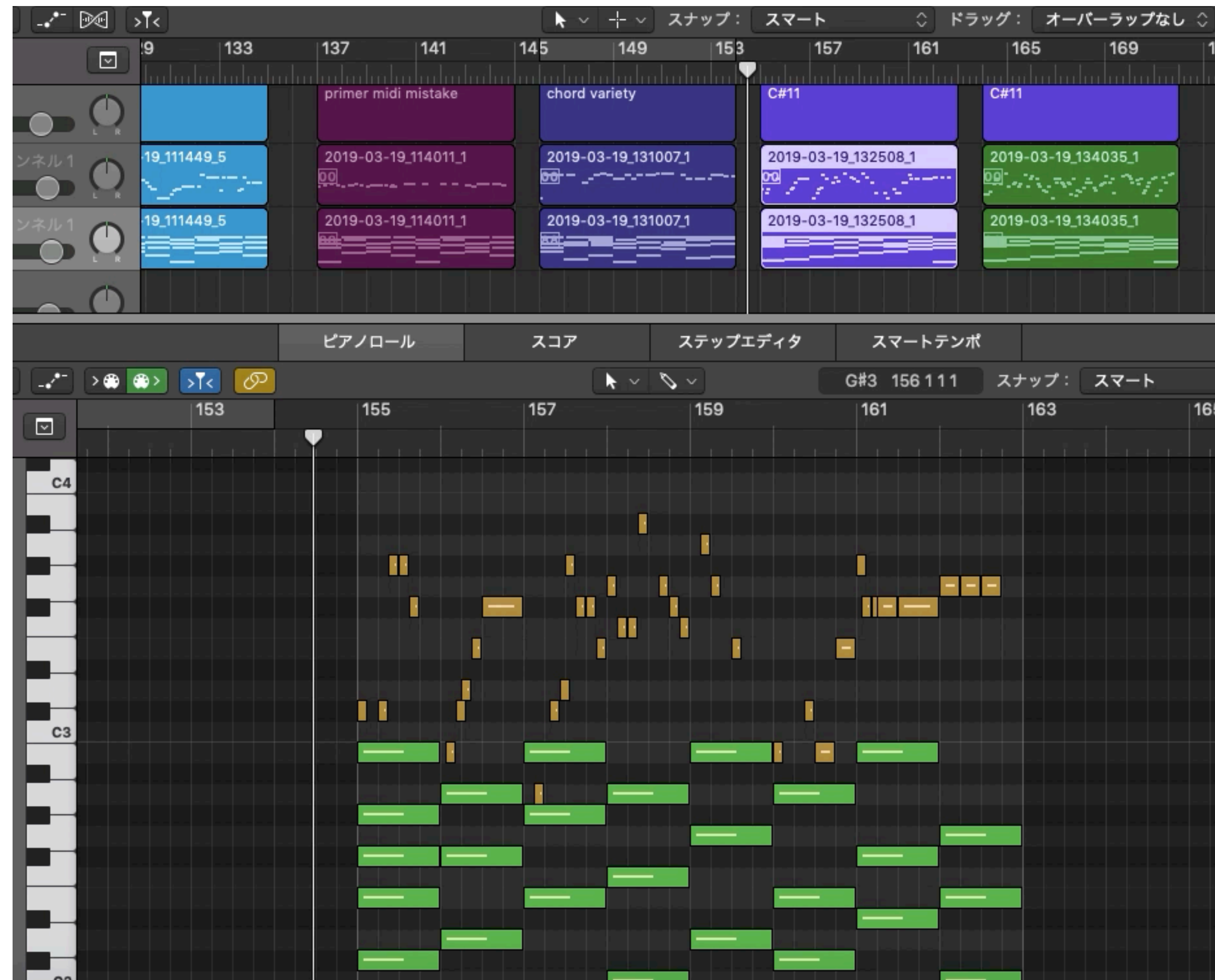
モデルの指定

magファイルの指定

生成用のノートナンバー指定

コード進行8小節
テンションコードも可能

Improv RNN Chord Pitches Improvでの音楽生成（動画）



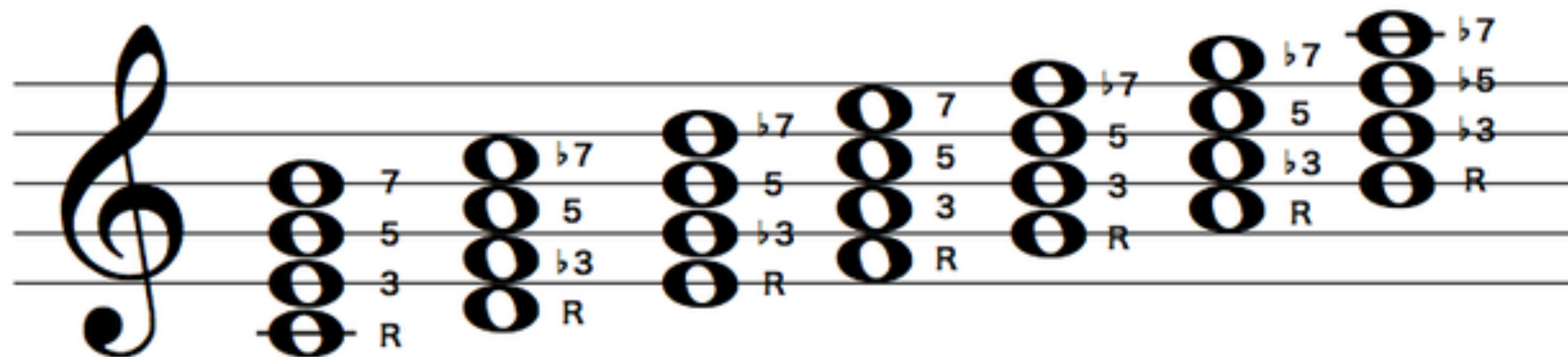
※資料では動画は再生できません

C#11 Improv

ImprovRNN コードの指定方法

コード

メジャーダイアトニックコード



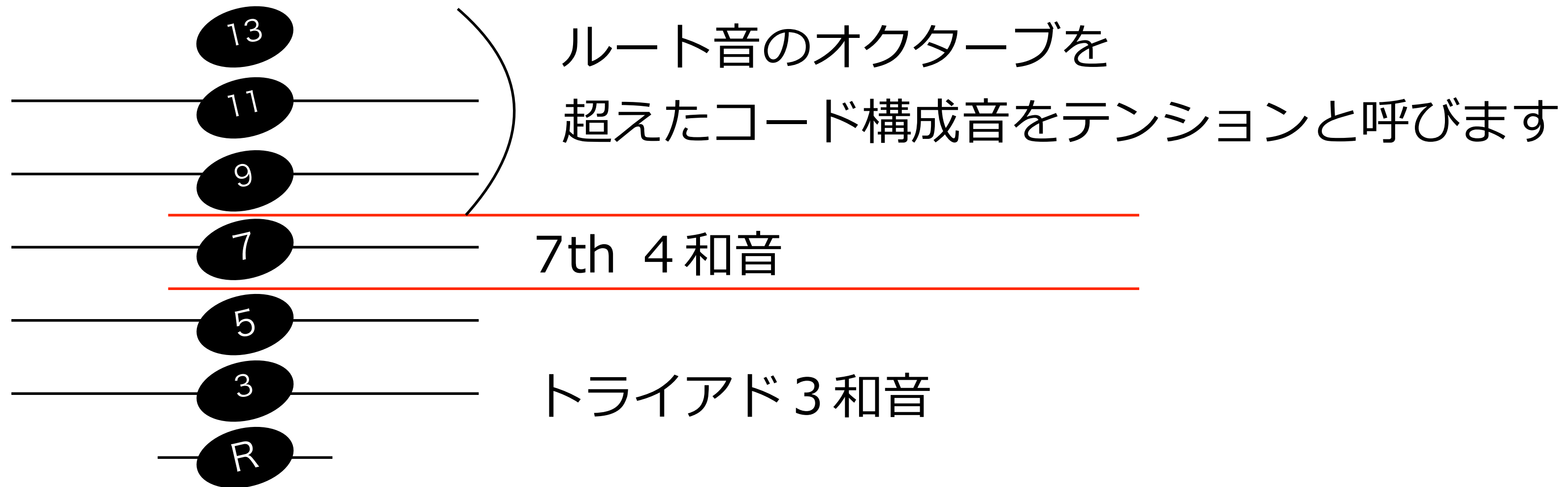
コード・ネーム	CΔ7	Dm7	Em7	FΔ7	G7	Am7	Bm7(♭5)
ディグリー・ネーム	IΔ7	IIIm7	IIIIm7	IVΔ7	V7	VIIm7	VIIIm7(♭5)

マイナーダイアトニックコード

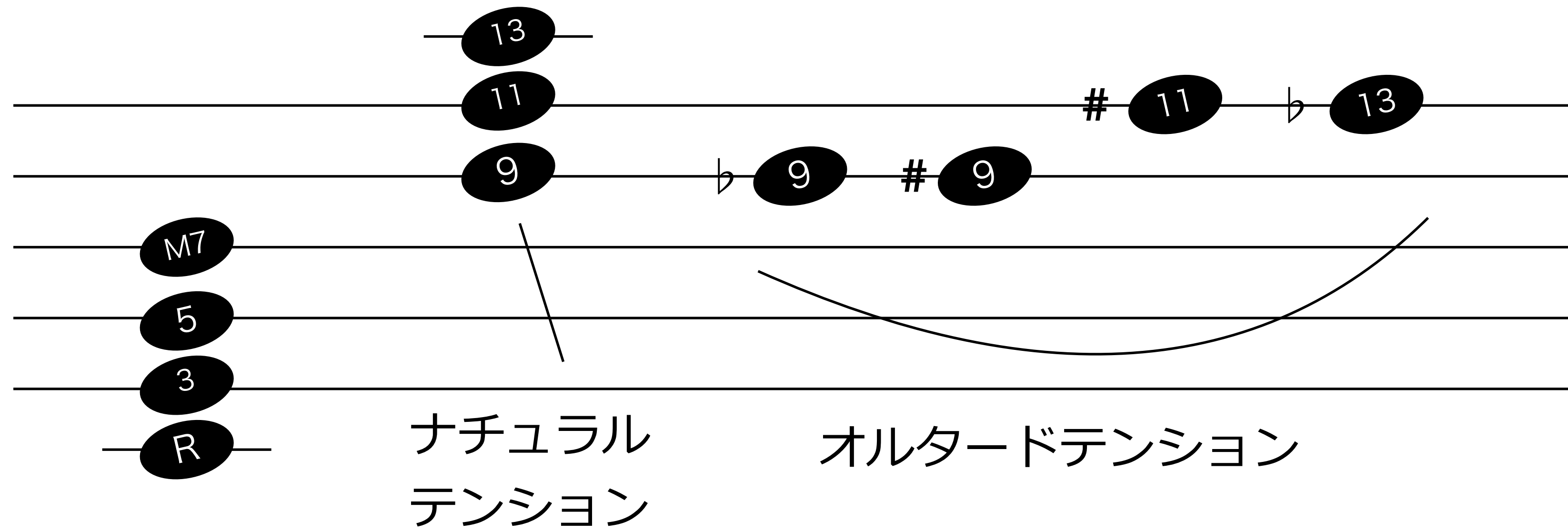
The image displays seven minor diatonic chords in C minor on a treble clef staff. Each chord is represented by a vertical stack of notes with a flat sign to the left. The notes are labeled with their Roman numeral and quality symbols to the right. Below the staff, the chord names and Roman numeral symbols are listed.

Cm7	Dm7^(b5)	E^bΔ7	Fm7	Gm7	A^bΔ7	B^b7
I^m7	II^m7^(b5)	^bIII^Δ7	IV^m7	V^m7	^bVI^Δ7	^bVII7

コードというのは3度堆積を基本とします



テンションコードの把握の仕方 9th



9のみ#とbがある

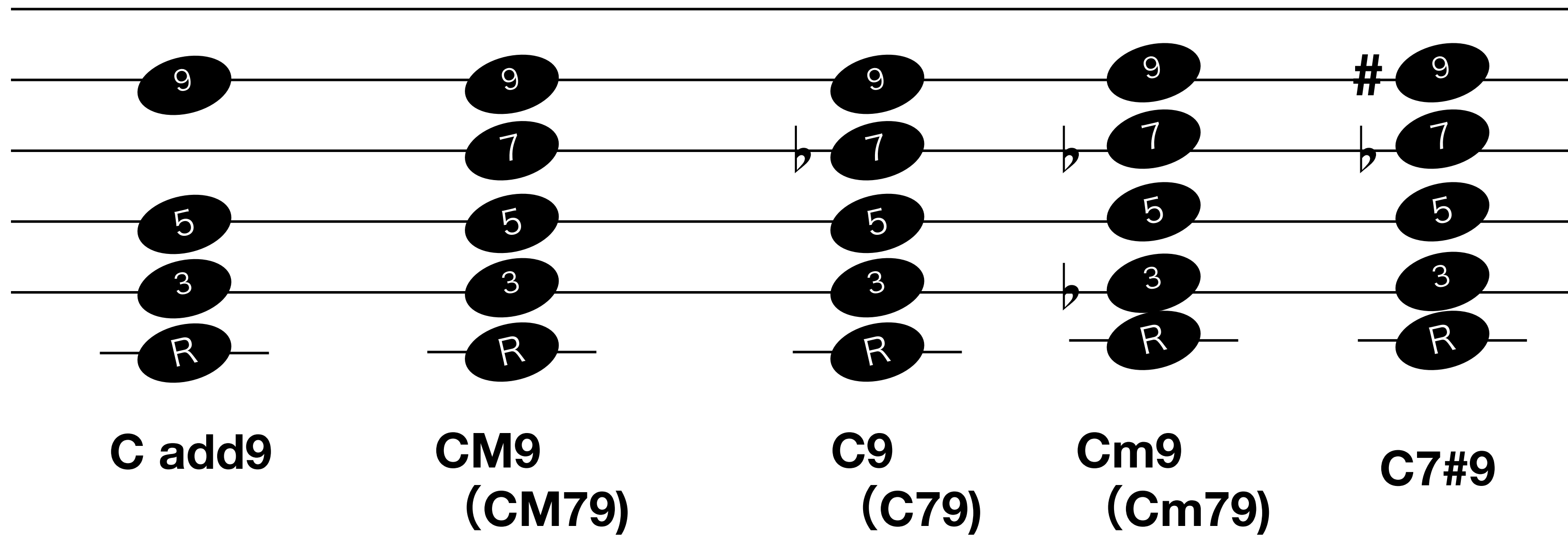
11は#のみ (bは3度の音)

13はbのみ (#は7th)

7thを含まない9はadd9 (11はadd11)

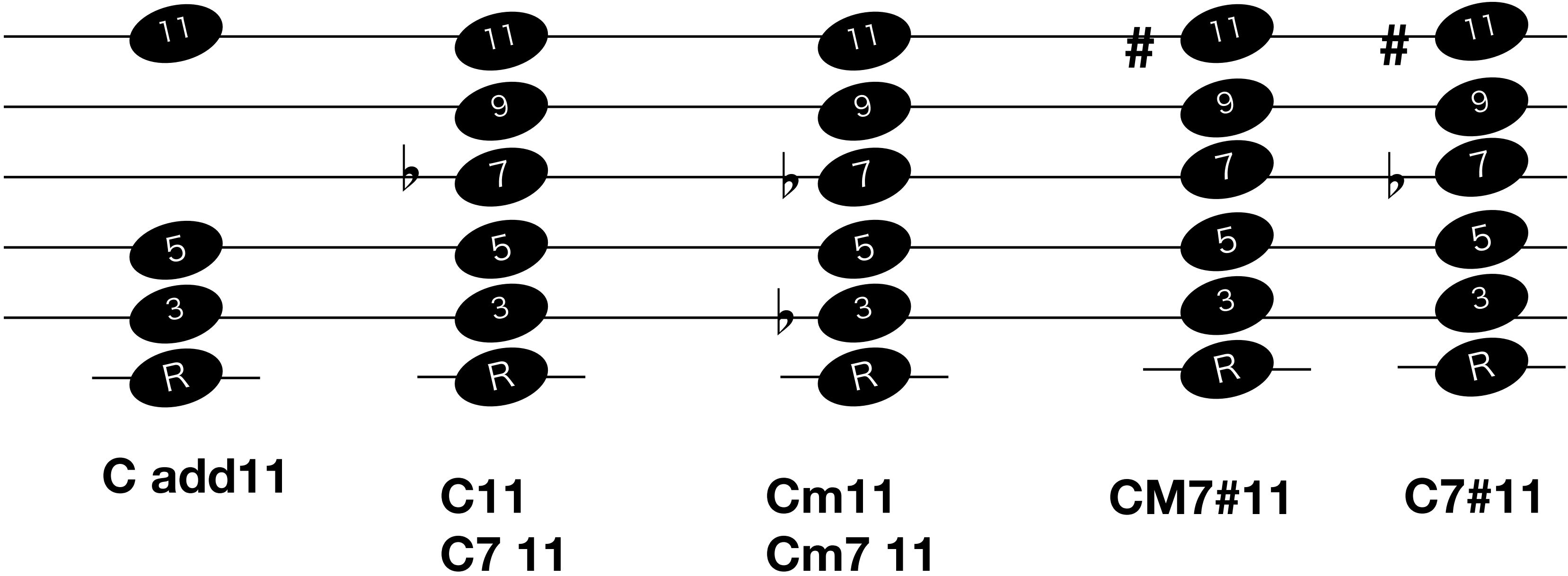
7thを含まない13は6

テンションコードの把握の仕方 9th



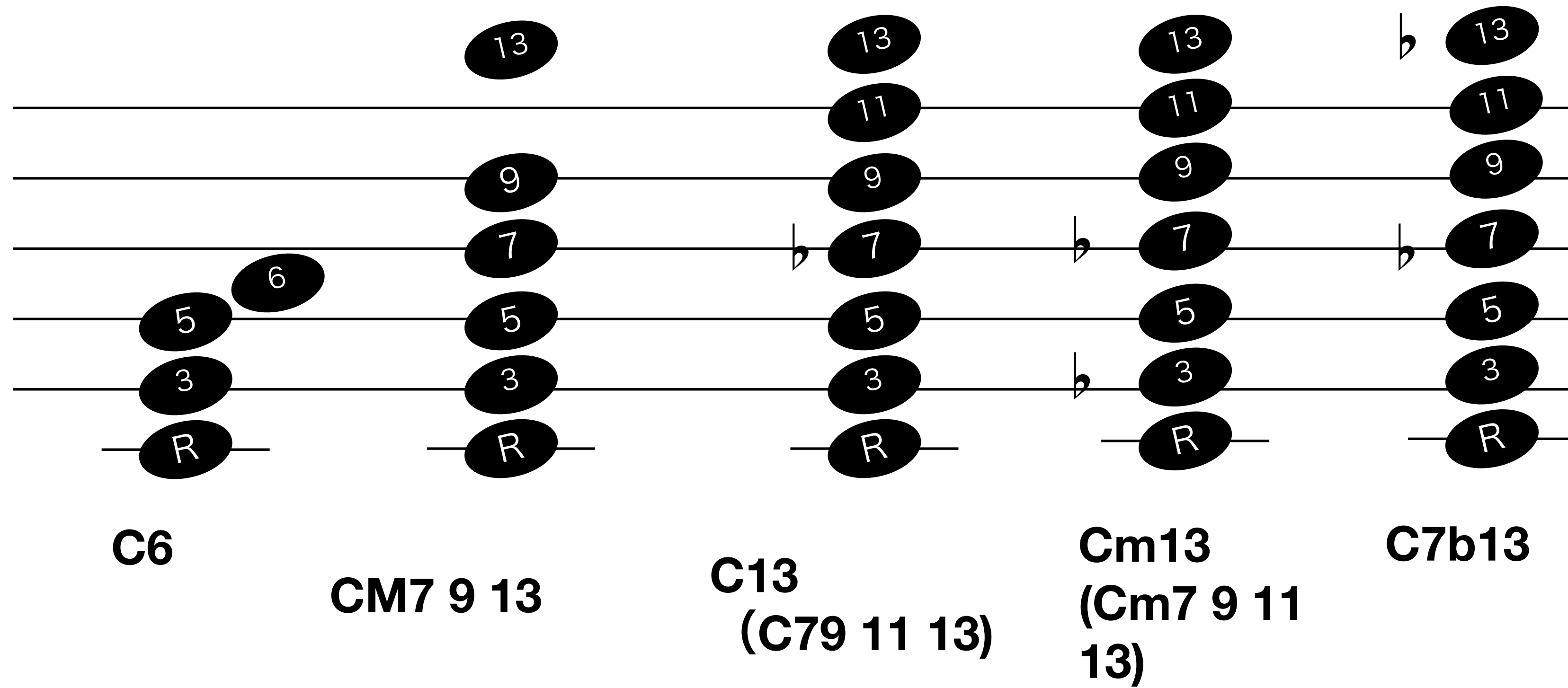
9のみ#とbがある

テンションコードの把握の仕方 11th



11は#のみある

テンションコードの把握の仕方 13th



13はbのみある

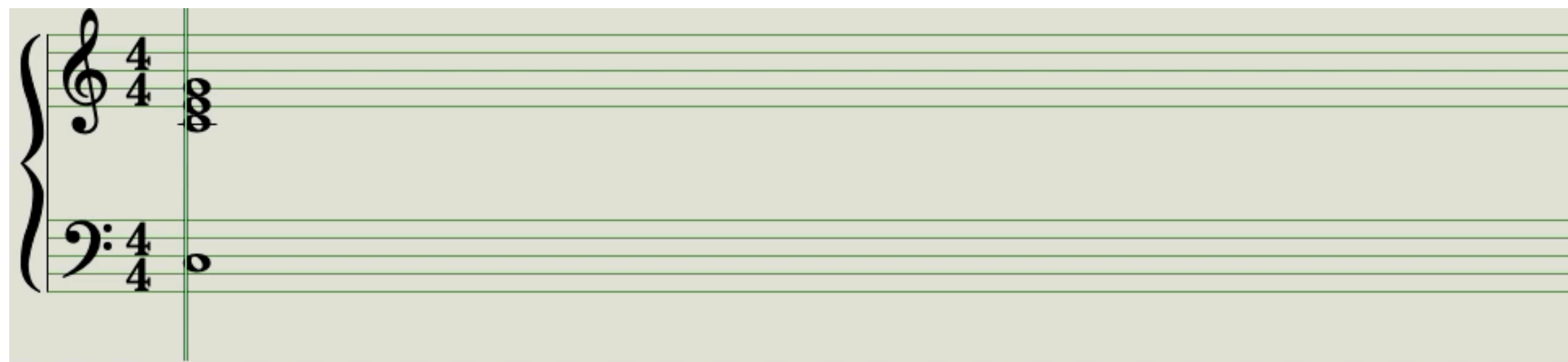
分数コード

分数コード

C/G あるいはC on Gなどと表記されますがどちらも同じです。
分母がベース音（ルート音）でその上の分子に構成音が重なります。

C

C
ベース音がC



Musical notation for a C chord in 4/4 time. The treble clef shows a C4 chord (C-E-G) in the right hand, and the bass clef shows a C2 note in the left hand. The time signature is 4/4.

C/G

C/G
ベース音がG



Musical notation for a C/G chord in 4/4 time. The treble clef shows a C4 chord (C-E-G) in the right hand, and the bass clef shows a G2 note in the left hand. The time signature is 4/4.

※資料では動画の再生はできません

**分数コードには
2種類あります**

分数コードには2種類あります

C/G

C/G
ベース音がG
でコード構成音に
含まれる

ベース音がコード構成音に含まれる場合

Cの転回形と捉える

C/D

C/D
ベース音がD
でコード構成音に
含まれない

ベースがコード構成音に含まれない場合

Dの転回形と捉える

※資料では動画の再生はできません

ImprovRNN コードの指定方法

improv_rnnのコードの指定方法

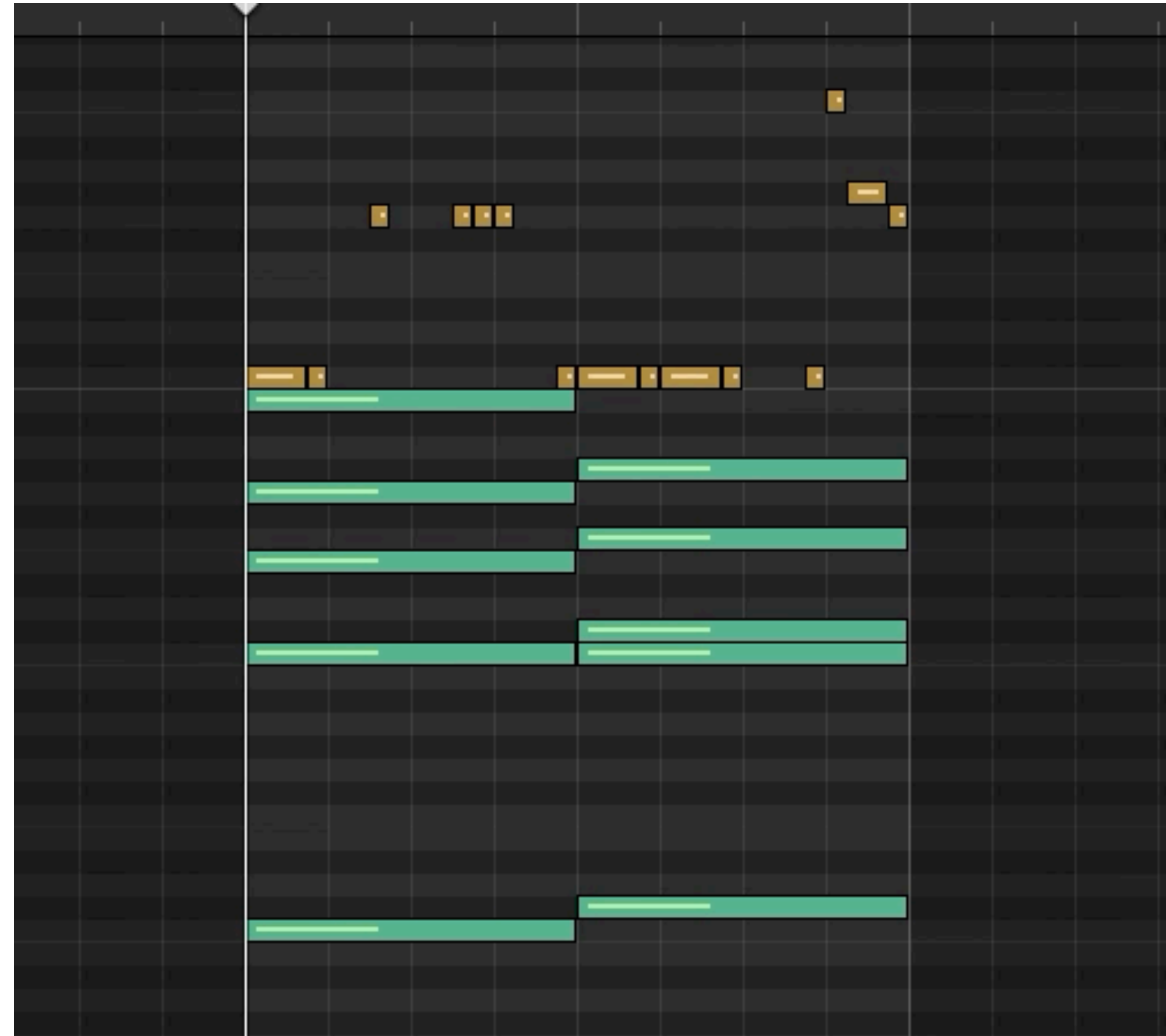
メジャー7thコードの指定方法

```
improv_rnn_generate \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="Cmaj7 C#maj7" \  
--steps_per_chord=16 \  
--render_chords
```

メジャー7は
maj7
#はノート名の後に指定

1ステップが16分音符
16ステップは1小節

メジャー 7thコードの指定方法



※資料では動画は再生できません

Cmaj7-C#maj7

improv_rnnのコードの指定方法

マイナー7thコードの指定方法

```
improv_rnn_generate \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="Cmaj7 C#maj7 Dm7 D#m7" \  
--steps_per_chord=8 \  
--render_chords
```

1ステップが16分音符
8ステップは1/2小節
1小節に2コード

メジャー7は
maj7
#はノート名の後に指定
マイナー7は
m7

マイナー 7thコードの指定方法



※資料では動画は再生できません

Cmaj7-C#maj7-Dm7-D#m7

improv_rnnのコードの指定方法

その他バリエーションコードの指定方法

```
python magenta/models/improv_rnn/improv_rnn_generate.py \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="Cmaj7 C#maj7 Cm7 Cbm6 Cmaj9 Cmaj#9 Cmaj#11 Cmajb13" \  
--steps_per_chord=8 \  
--render_chords
```

1 ステップが 1/6 分音符
8 ステップは 1/2 小節
1 小節に 2 コード

bはb (フラット)

6

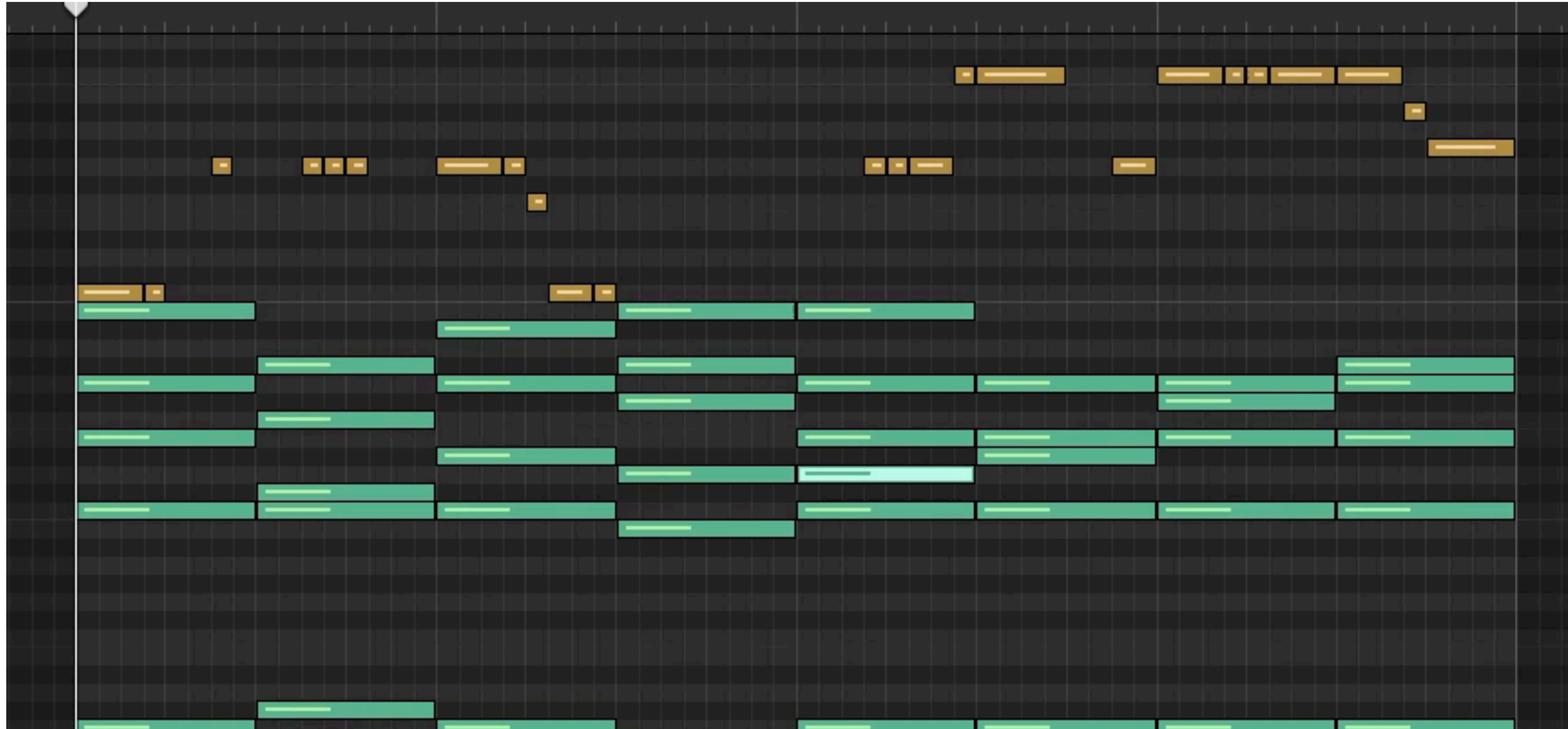
maj9

maj#9はC+#9th

maj#11はC+#11th

majb13はC+b13th

その他バリエーションコードの指定方法



※資料では動画は再生できません

Cmaj7-C#maj7-Dm7-D#m7

improv_rnnのコードの指定方法

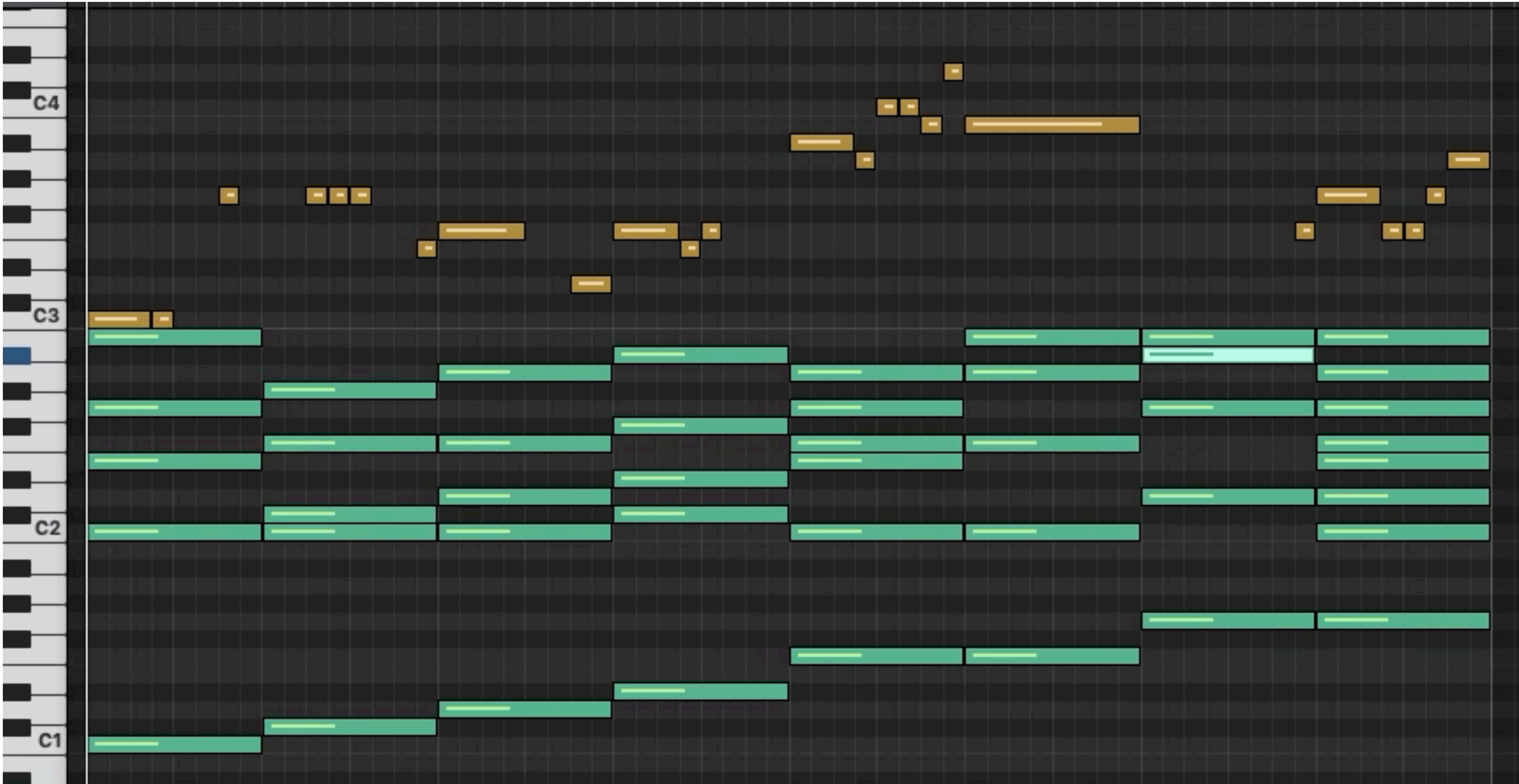
その他バリエーションのコード進行で生成

```
improv_rnn_generate \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="Cmaj7 C#maj7 Dm7 D#m7 Fmaj9 Fmaj#11 Gmaj#9 G13" \  
--steps_per_chord=8 \  
--render_chords
```

1ステップが16分音符
8ステップは1/2小節
1小節に2コード

maj9
maj#9はC+#9th
maj#11はC+#11th
13はC13th

その他バリエーションコード進行で生成



※資料では動画は再生できません

variation

improv_rnnのコードの指定方法

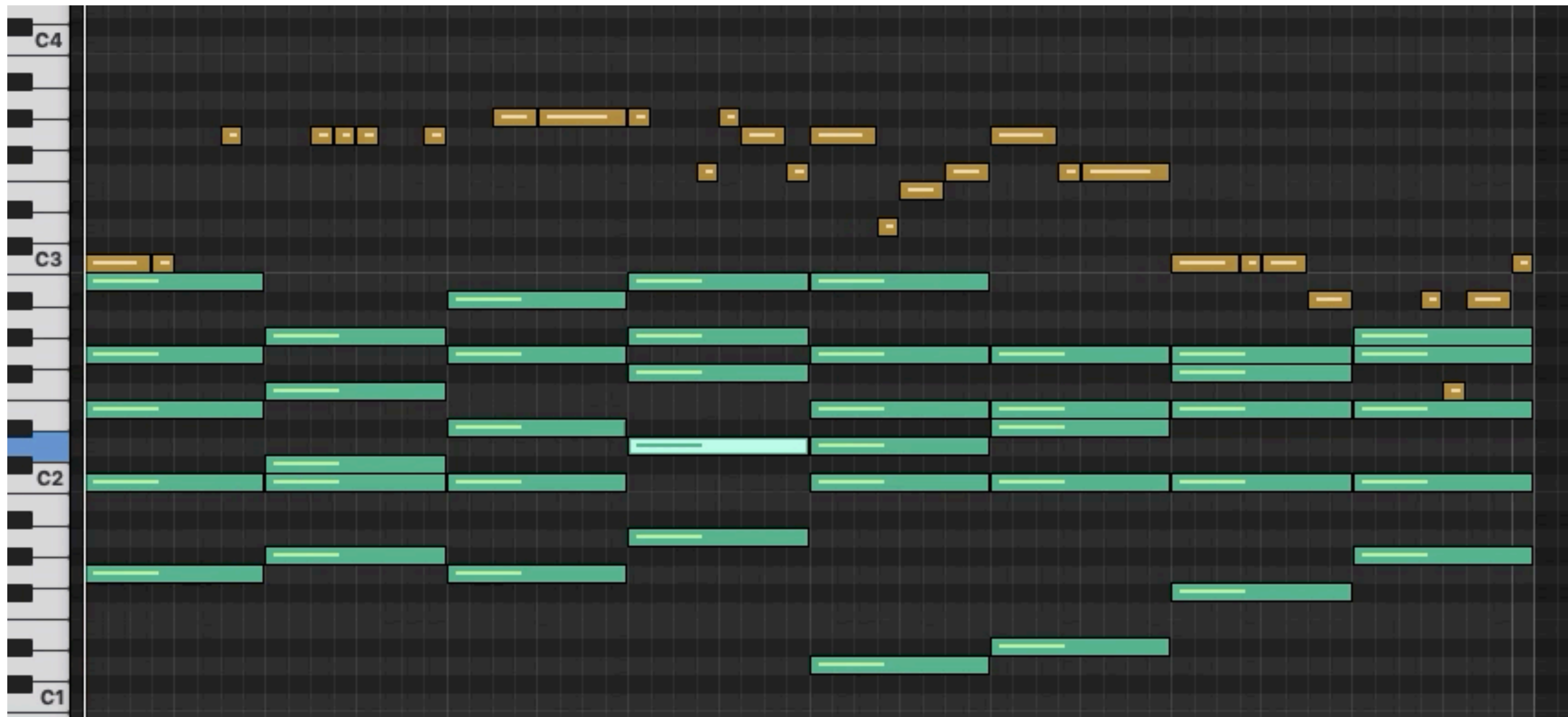
分数コードの指定

```
python magenta/models/improv_rnn/improv_rnn_generate.py \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="Cmaj7/G C#maj7/G# Cm7/G Cbm6/A Cmaj9/D Cmaj#9/D# Cmaj#11/F# Cmajb13/Ab" \  
--steps_per_chord=8 \  
--render_chords
```

1ステップが16分音符
8ステップは1/2小節
1小節に2コード

分母（ベース音）は単音指定
マイナーやテンション指定はしない

分数コード進行で生成（動画）



※資料では動画は再生できません

compound chord

improv_rnnのコードの指定方法

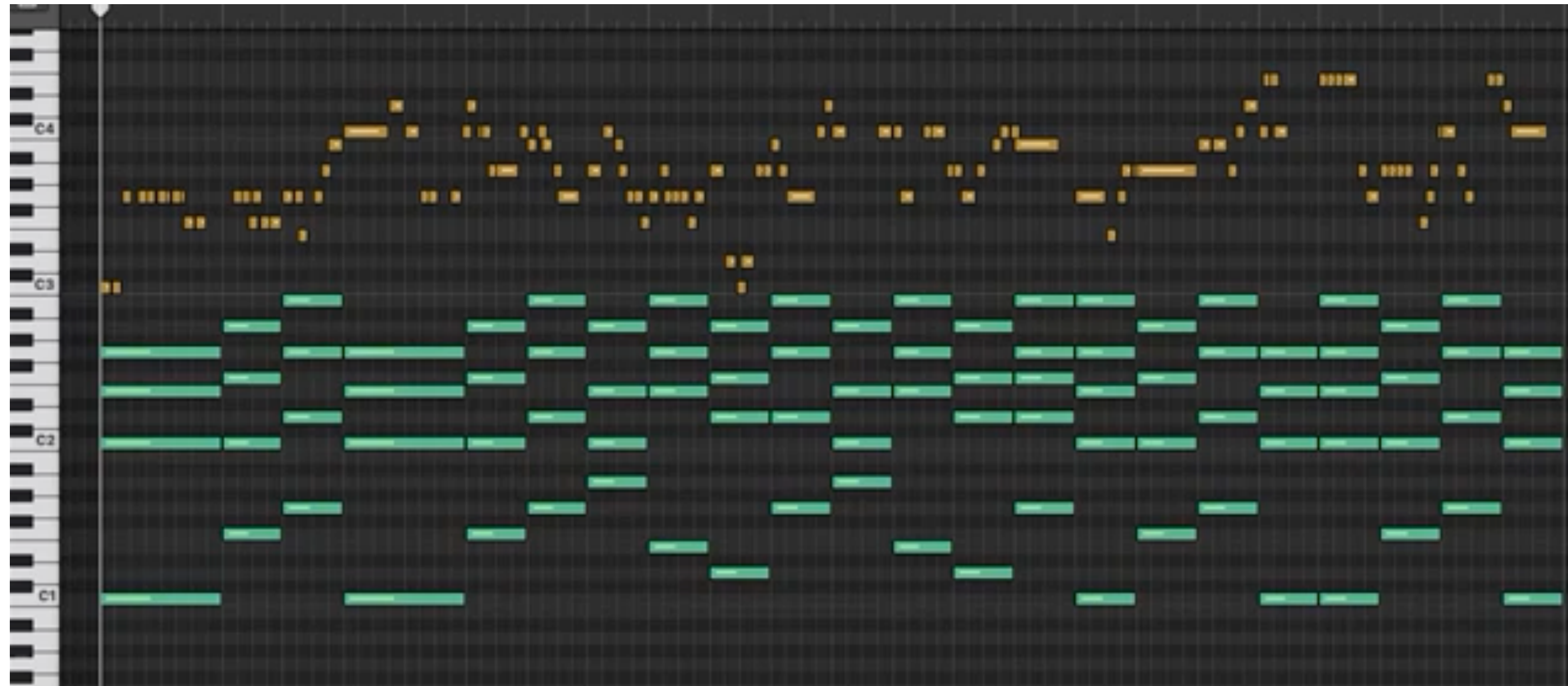
長い楽曲の生成も可能です

```
python magenta/models/improv_rnn/improv_rnn_generate.py \  
--config=chord_pitches_improv \  
--bundle_file=任意のディレクトリ/chord_pitches_improv.mag \  
--output_dir=任意のディレクトリ\  
--num_outputs=生成曲数 \  
--primer_melody="[60, -2, -2, 60, -1, -2, 67, -1, -2, -2, 67, 67, 67, -1, -2, -2]" \  
--backing_chords="C C F G C C F G Am Em Dm G Am Em Dm G7 Cmaj7 F G C Cmaj7 F G C" \  
--steps_per_chord=16 \  
--render_chords
```

↑
1ステップが16分音符
16ステップは1小節

↑
長い楽曲もコード進行数を増やす事で
可能です
(連続したコードは繋がります)

フルコーラス楽曲（24小節）生成（動画）



※資料では動画は再生できません

improv-song

Magentaでできる音楽生成（本講座内で解説予定）

- 単音のシンプルなメロディー生成
- ドラムトラックの生成
- 3パート演奏（メロディー、ベース、ドラム）の楽曲生成
- コード進行に沿ったアドリブメロディー生成
- 単音メロディーにハーモニーを生成
- 表現力豊かなピアノ楽曲の生成
- Ableton Live（または他のDAW）での音楽生成プラグイン活用

Polyphony RNN

Polyphony RNN

Polyphony RNNは、シングルトラックのポリフォニック演奏を可能にした音楽生成モデル。
LSTM(Long-Short Term Memory)を使用している。

Bach Bot (バッハの合唱曲を元にした自動音楽生成アルゴリズム) にインスパイアされ合唱曲的な音楽生成を実現する。

(BachBot論文はこちら)

https://ismir2017.smcnus.org/wp-content/uploads/2017/10/156_Paper.pdf

Polyphony RNNのトレーニング済みモデルはこちらよりダウンロード

https://github.com/tensorflow/magenta/tree/master/magenta/models/polyphony_rnn

Polyphony RNN

```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=生成曲数 \  
--num_steps=ステップ数 16ステップで1小節 \  
--primer_pitches="[67,64,60]" \  
--condition_on_primer=true \  
--inject_primer_during_generation=false
```

上記およびその他コマンドの詳細補足

- primer_pitches は1/4拍ごとに使用するコードをMIDI準拠の数値の構成音で指定
- primer_melody は生成に使用する最初のメロディーを数値で指定 (-2 = イベントなし, -1 = 音を止める, values 0 through 127 = MIDIに準じた音名)
- primer_midi は生成に使用するメロディーをMIDIファイルで指定できる
- condition_on_primer は、主にprimer_pitchesを使用の際true、primer_melody(およびmidi)の場合はfalse。trueにより入力値を和音と認識して指定のコードでの生成を可能にする
- inject_primer_during_generation はtrueにすると生成曲の最初にprimerで指定したメロディーを含む

Polyphony RNN で音楽生成

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含まない

```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=5 \  
--num_steps=128 \  
--primer_pitches="[67,64,60]" \  
--condition_on_primer=true \  
--inject_primer_during_generation=false
```

生成曲

- 128ステップ（8小節）
- primer_pitches : コード指定はCメジャー 67=G 64=E 60=C
- condition_on_primer : true primer_pitchesの入力値を和音として指定のコードでの生成をする（Cメジャー）
- inject_primer_during_generation false 生成部分にはprimerで指定した音を含まない

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含まない

The screenshot displays a music software interface with a piano roll editor. The piano roll shows a complex polyphonic texture with multiple voices across a 9-measure span. The interface includes a piano roll editor, a score editor, and a step editor. The piano roll shows a dense arrangement of notes, with a prominent bass line and a complex upper register. The score editor shows a 4/4 time signature and a tempo of F1 4 4 2 1. The step editor shows a sequence of notes for the first measure.

※資料では動画の再生はできません

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含まない

The screenshot displays a music software interface with a piano roll. The piano roll shows a sequence of notes across 9 measures. A blue oval highlights the first two notes in the first measure, with a blue arrow pointing to it from the text '入力値を和音と認識' (Recognize input values as chords). Another blue arrow points to the first note of the first measure from the text 'primerを楽曲に含まない' (Do not include primer in the music). The interface includes a timeline at the top with markers at 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, and 49. Below the piano roll, there are tabs for 'ピアノロール', 'スコア', 'ステップエディタ', and 'スマートテンポ'. The piano roll has a keyboard on the left with labels C5, C4, C3, C2, and C1. The piano roll itself has a grid with measures 1 through 9 and notes represented by horizontal bars.

入力値を和音と認識

primerを楽曲に含まない

通常の和音を与えた生成はこちら

※資料では動画の再生はできません

Polyphony RNN

入力値を和音と認識せずに楽曲生成でprimerを楽曲に含めない

```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=5 \  
--num_steps=128 \  
--primer_pitches="[67,64,60]" \  
--condition_on_primer=false \  
--inject_primer_during_generation=false
```

生成曲

- 128ステップ（8小節）
- primer_pitches : コード指定はCメジャー 67=G 64=E 60=C
- condition_on_primer : **false** primer_pitchesの入力値を和音としては認識せずに生成
- inject_primer_during_generation false 生成部分にはprimerで指定した音を含めない

Polyphony RNN

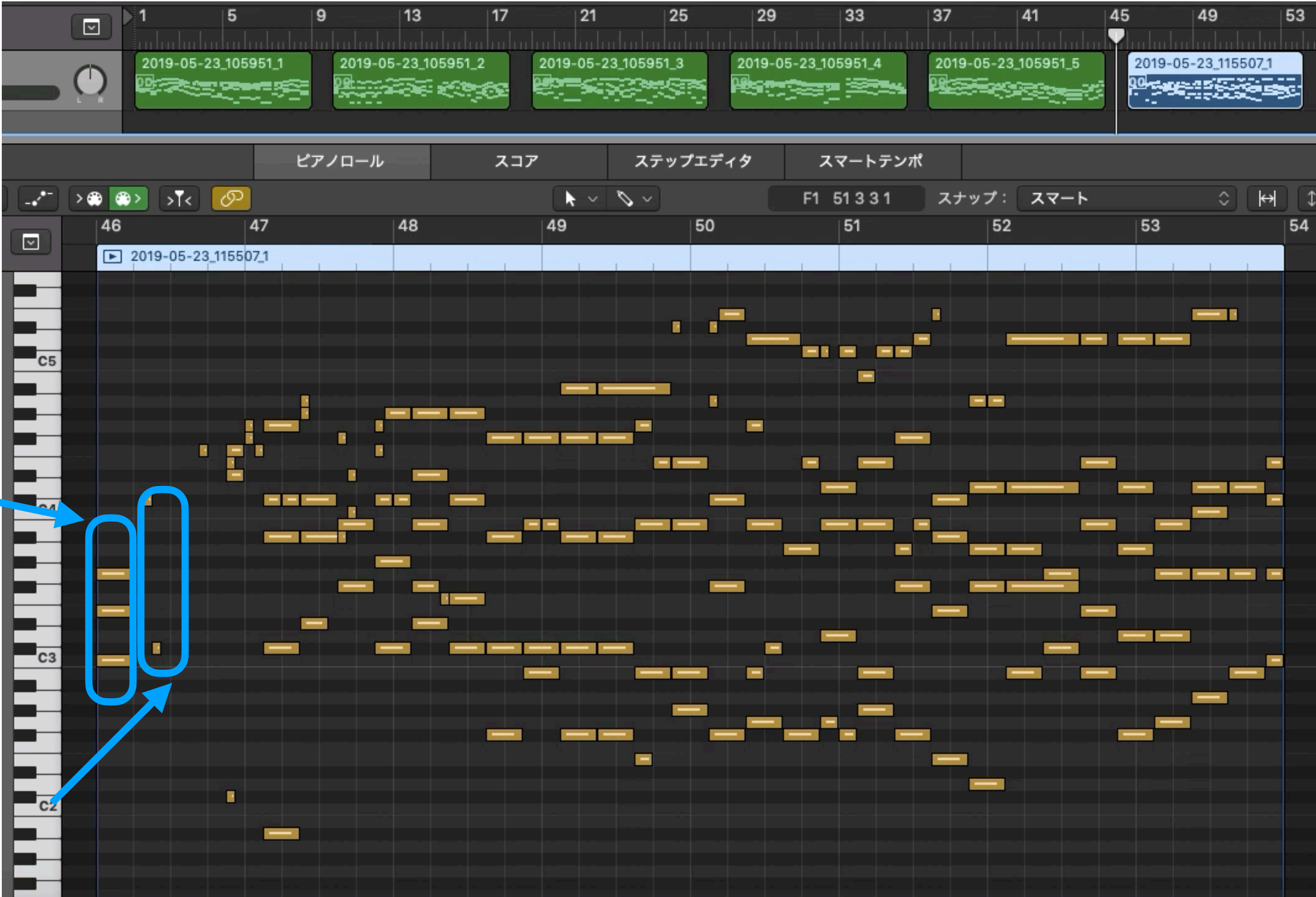
入力値を和音と認識せずに楽曲生成でprimerを楽曲に含まない

The screenshot displays a music software interface, likely Ableton Live, showing a piano roll. The top section features a timeline with markers at 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, and 53. Below the timeline, there are several audio waveforms labeled with IDs: 2019-05-23_105951_1 through 2019-05-23_105951_5, and 2019-05-23_115507_1. The main piano roll area shows a complex polyphonic arrangement of notes across multiple staves, with a keyboard view on the left. The notes are represented by yellow bars on a dark grid. The piano roll is currently focused on the time range from 46 to 54. The interface includes various controls such as a volume knob, a solo button, and a mute button. The piano roll is labeled with notes C2, C3, C4, and C5. The software interface also shows a piano roll, score, step editor, and smart tempo tabs. The piano roll is currently selected. The piano roll shows a complex polyphonic arrangement of notes across multiple staves, with a keyboard view on the left. The notes are represented by yellow bars on a dark grid. The piano roll is currently focused on the time range from 46 to 54. The software interface also shows a piano roll, score, step editor, and smart tempo tabs. The piano roll is currently selected.

※資料では動画の再生はできません

Polyphony RNN

入力値を和音と認識せずに楽曲生成でprimerを楽曲に含まない



入力値を和音と認識しない

primerを楽曲に含まない

The screenshot shows a music software interface with a piano roll. The piano roll displays a sequence of notes and chords over time, with a keyboard layout on the left. Annotations include a blue arrow pointing to a specific note and a blue oval highlighting a region of the piano roll.

通常用途ではない

※資料では動画の再生はできません

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含む

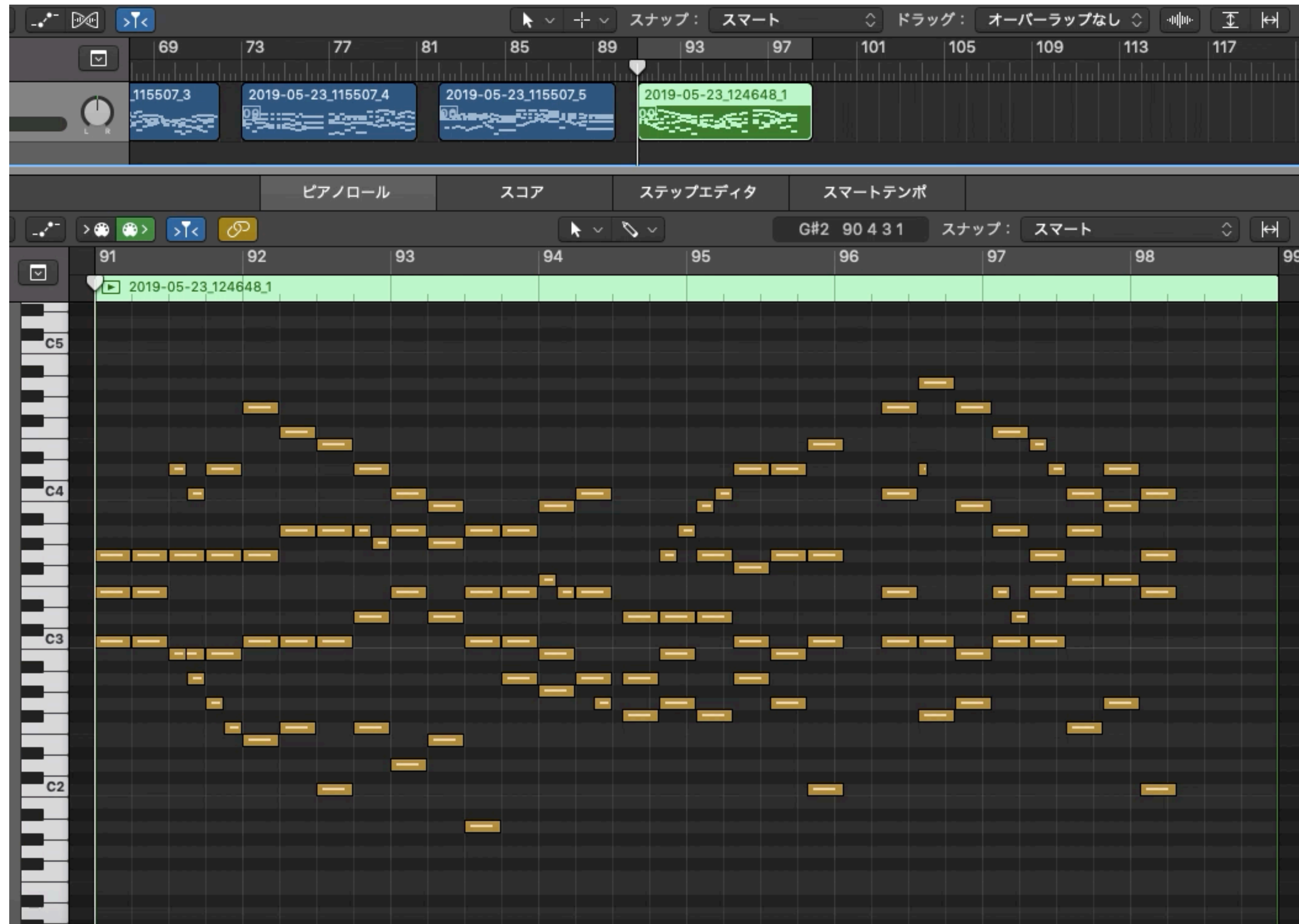
```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=5 \  
--num_steps=128 \  
--primer_pitches="[67,64,60]" \  
--condition_on_primer=true \  
-inject_primer_during_generation=true
```

生成曲

- 128ステップ（8小節）
- primer_pitches : コード指定はCメジャー 67=G 64=E 60=C
- condition_on_primer : true primer_pitchesの入力値を和音として指定のコードでの生成をする（Cメジャー）
- inject_primer_during_generation **true** 生成部分にはprimerで指定した音を含む

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含む



※資料では動画の再生はできません

Polyphony RNN

入力値を和音と認識した楽曲生成でprimerを楽曲に含む

The screenshot shows a music software interface with a piano roll. The piano roll has a vertical axis for pitch (C2 to C5) and a horizontal axis for time (measures 91 to 99). A track labeled '2019-05-23_124648_1' is highlighted in green. Annotations include a blue circle around the first few notes and arrows pointing to them from the text '入力値を和音と認識' and 'primerを楽曲に含む'.

入力値を和音と認識

primerを楽曲に含む

通常の和音を与えた生成では使用しない。含まずに生成

※資料では動画の再生はできません

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含む

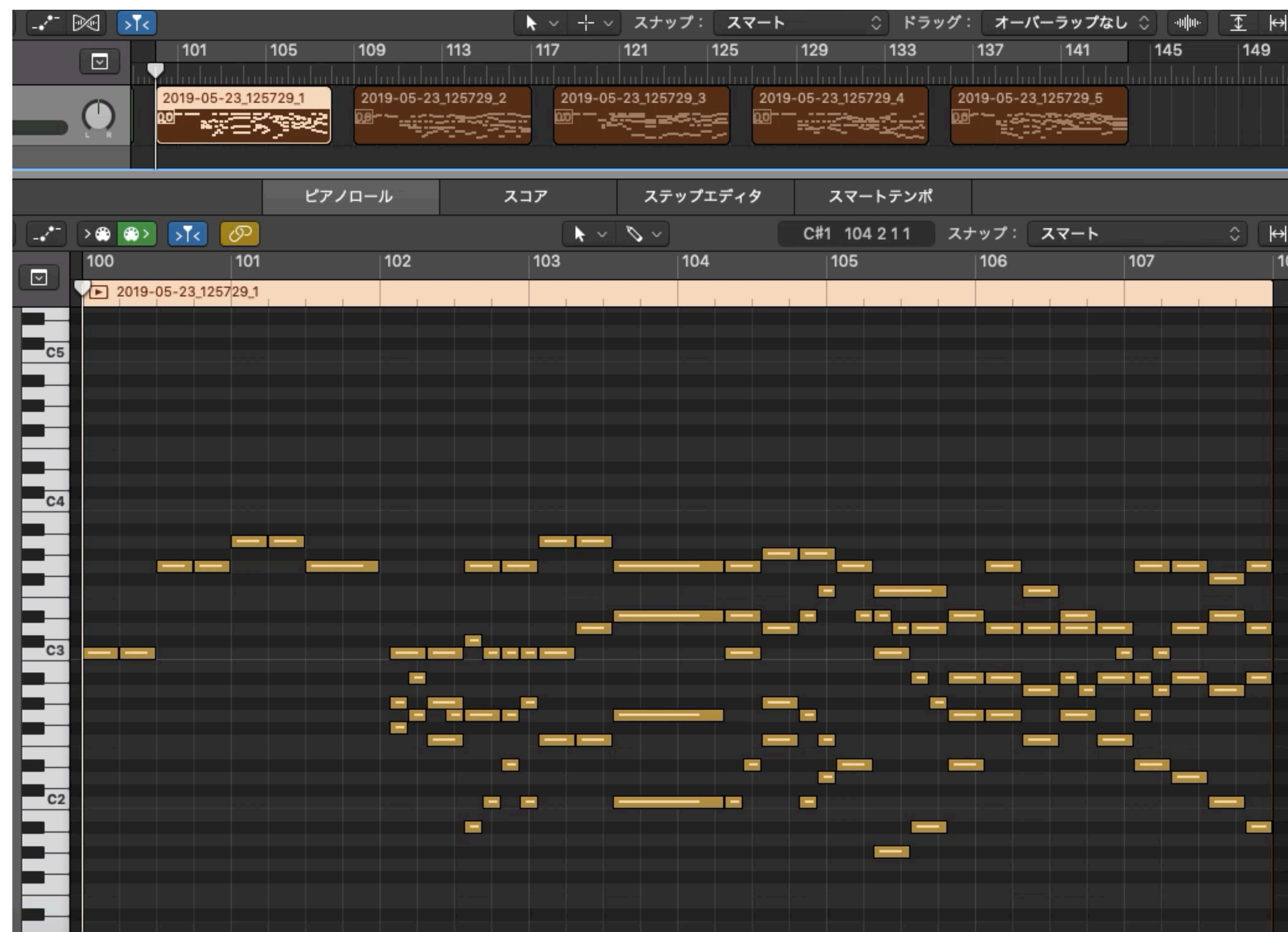
```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=5 \  
--num_steps=128 \  
--primer_melody="[60, -2, -2, -2, 60, -2, -2, -2, "\\  
"67, -2, -2, -2, 67, -2, -2, -2, 69, -2, -2, -2, "\\  
"69, -2, -2, -2, 67, -2, -2, -2, -2, -2, -2, -2]" \  
--condition_on_primer=false \  
--inject_primer_during_generation=true
```

生成曲

- 128ステップ（8小節）
- primer_melody : メロディーはきらきら星
- condition_on_primer : true primer_pitchesの入力値を和音として認識せずメロディーとして指定
- inject_primer_during_generation true 生成部分にはprimerで指定した音を含む

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含む



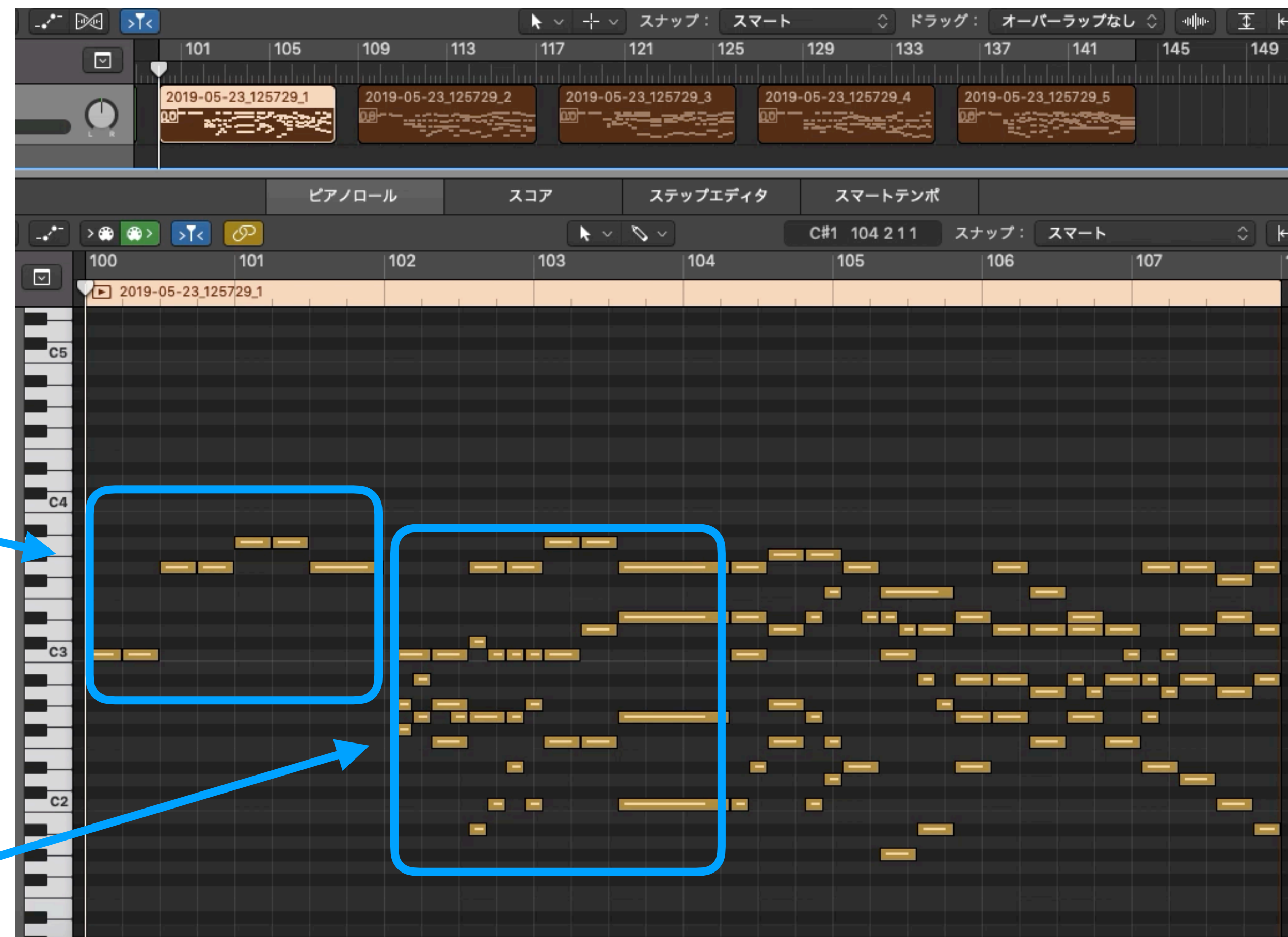
※資料では動画の再生はできません

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含む

入力値を和音と認識
しない、メロディー
で認識

primerを楽曲に含む
= 入力曲にハーモニーを



入力曲にハーモニーを加え生成する場合この方法

※資料では動画の再生はできません

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含めない

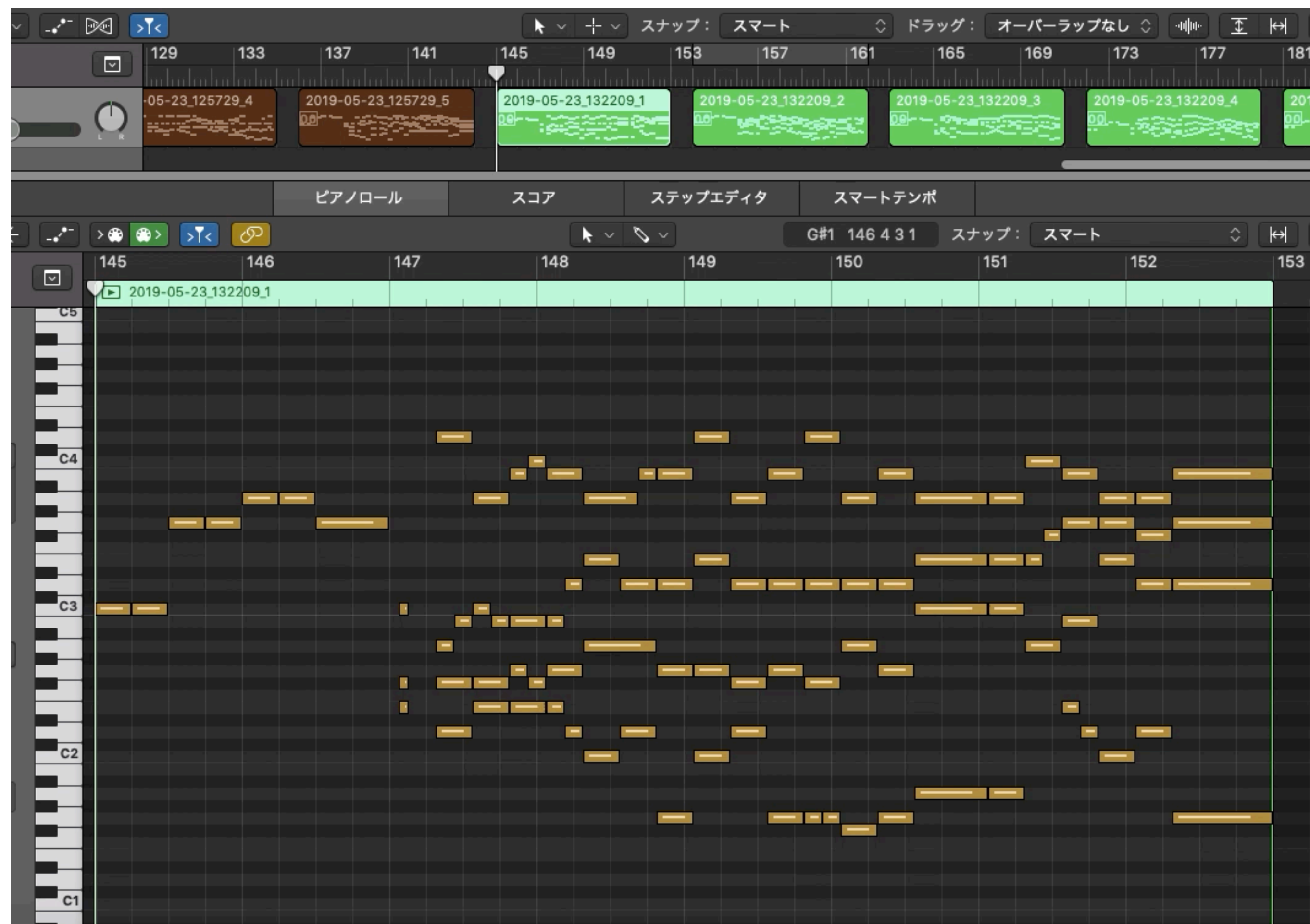
```
polyphony_rnn_generate \  
--bundle_file= polyphony_rnn.magへの絶対パス \  
--output_dir=任意の出力ディレクトリー\  
--num_outputs=5 \  
--num_steps=128 \  
--primer_melody="[60, -2, -2, -2, 60, -2, -2, -2, "\\  
"67, -2, -2, -2, 67, -2, -2, -2, 69, -2, -2, -2, "\\  
"69, -2, -2, -2, 67, -2, -2, -2, -2, -2, -2, -2]" \  
--condition_on_primer=false \  
-inject_primer_during_generation=false
```

生成曲

- 128ステップ（8小節）
- primer_melody : メロディーはきらきら星
- condition_on_primer : true primer_pitchesの入力値を和音として認識せずメロディーとして指定
- inject_primer_during_generation **false** 生成部分にはprimerで指定した音を含めない

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含まない



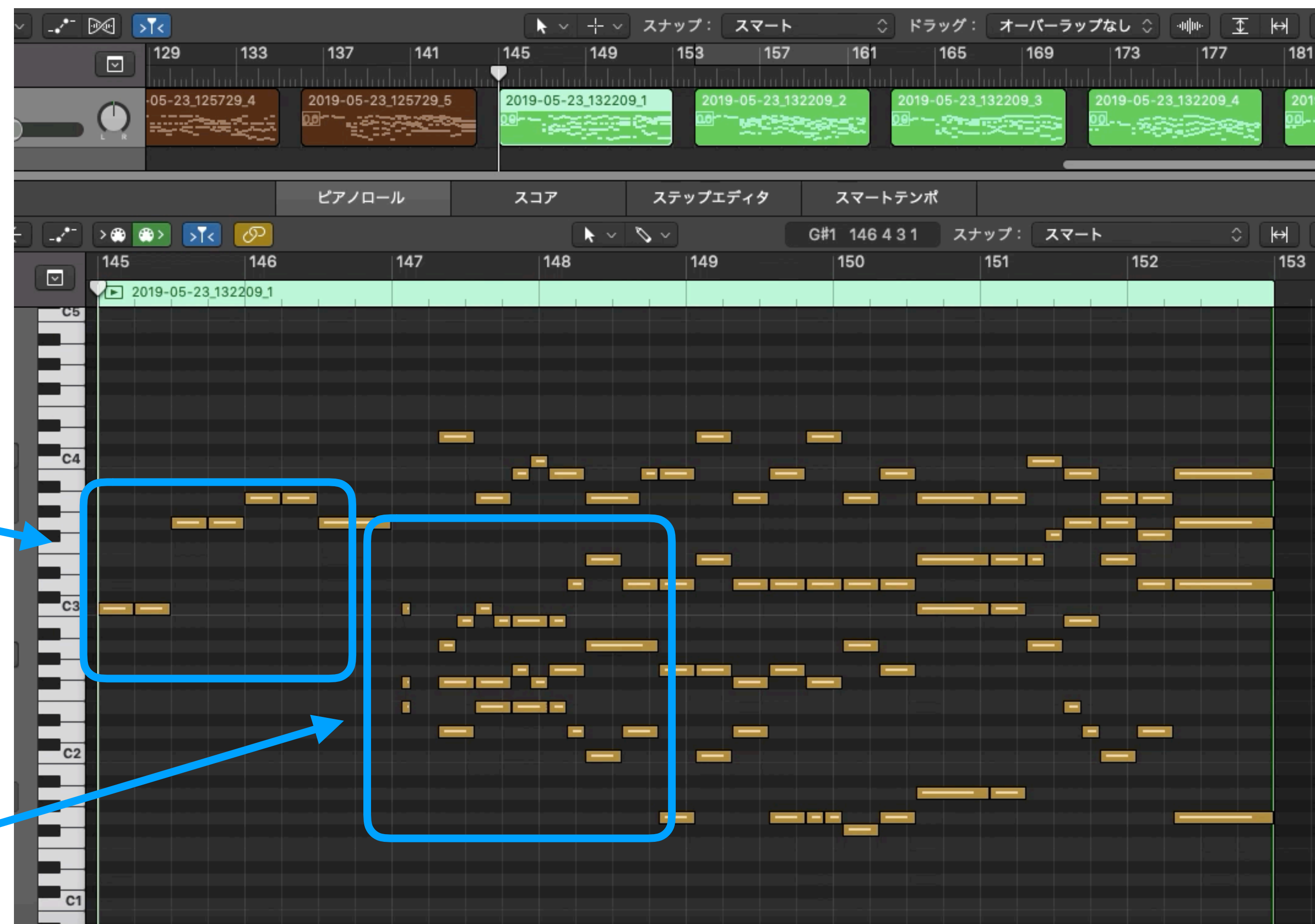
※資料では動画の再生はできません

Polyphony RNN

メロディーによる楽曲生成でprimerを楽曲に含まない

入力値を和音と認識
しない、メロディー
で認識

primerを楽曲に含まない



入力曲を元に新たなハーモニー曲の生成

※資料では動画の再生はできません

Pianoroll RNN

Pianoroll RNN

Pianoroll RNNは、シングルトラックのポリフォニック演奏を可能にした音楽生成モデル。
LSTM(Long-Short Term Memory)を使用している。

Polyphony RNNと同様、同時に和音進行する合唱曲の様な音楽生成ができる。
NADE (Neural Autoregressive Distribution Estimator) と呼ばれるモデルを使用。
(ニューラル自己回帰分布推定)

公式ページにモデルがあるのでダウンロードの上使用

https://github.com/tensorflow/magenta/tree/master/magenta/models/pianoroll_rnn_nade

[pianoroll_rnn_nade](#) ウェブからスクレーピングしたバリエーションのあるピアノ曲

[pianoroll_rnn_nade-bach](#): バッハの合唱曲データセット

Pianoroll RNN

```
pianoroll_rnn_nade_generate \  
--bundle_file= 任意のディレクトリーのpianoroll_rnn_nade.mag \  
--output_dir=任意のディレクトリー \  
--qpm=90 \ テンポ  
--num_outputs=5 \ 生成曲数  
--num_steps=128 \ ステップ数 16ステップで1小節 128ステップは8小節  
--primer_pianoroll="[(67,64,60,), (62,), (64,), (65,), (69,67,64,), (), (70,),  
(65, 62,)]"
```

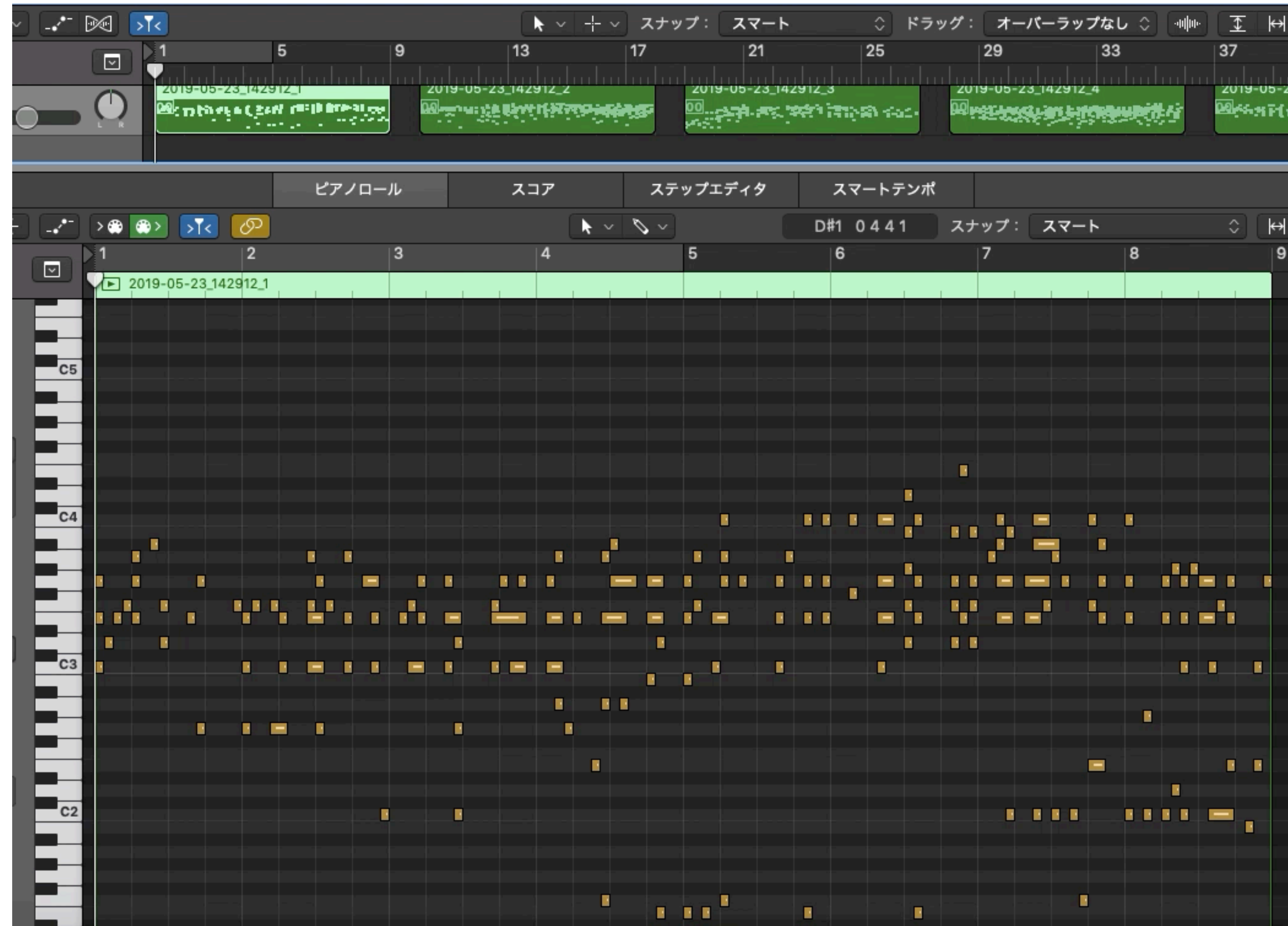
その他のコマンドや上記コマンドの詳細補足

- qpm はテンポ (BPM)
- primer_pianoroll は最初の各ステップで使用する音階 (和音可能) をPythonのリストとタプルで指定できる

例: "[(55,), (54,), (55, 53), (50,), (62, 52), (), (63, 55)]"

1ステップが16分音符。タプル内は和音。空のタプルは休符。

Pianoroll RNN



※資料では動画の再生はできません

Magentaでできる音楽生成（本講座内で解説予定）

- 単音のシンプルなメロディー生成
- ドラムトラックの生成
- 3パート演奏（メロディー、ベース、ドラム）の楽曲生成
- コード進行に沿ったアドリブメロディー生成
- 単音メロディーにハーモニーを生成
- 表現力豊かなピアノ楽曲の生成
- Ableton Live（または他のDAW）での音楽生成プラグイン活用

Performance RNN

Performance RNN

Performance RNN はポリフォニック演奏を可能にした音楽生成モデルです。

Dynamics（ベロシティによる抑揚）やタイミングの微妙な変化までを再現した音楽生成ができます。

他のモデルと違い以下の様な特徴を持ちます

- ノートオンイベント(*pitch*): start a note at *pitch*
- ノートオフイベント(*pitch*): stop a note at *pitch*
- タイムシフト(*amount*): advance time by *amount*
- ベロシティ: change current velocity to *value*

Performance RNN

- ノートオンイベント(*pitch*): start a note at *pitch*
- ノートオフイベント(*pitch*): stop a note at *pitch*

通常は発音イベントが発音時間とともに生成されますが、ノートオンとノートオフのイベントが別々に生成されます。

その際、ノートオンのないノートオフイベント、またノートオフのないノートオンイベントは無視されます、

- タイムシフト(*amount*): advance time by *amount*
- ベロシティ: change current velocity to *value*

表現力豊かなタイミングをサポートするために、最大1秒まで10ミリ秒単位で進むタイムシフトイベントでクロックを制御します。(BPMに応じて拍が決まるモデルではありません)

ベロシティは、1～127までのMIDI準拠に変換されます

Performance RNN

学習済みモデル

モデルに使用している学習データはミネソタ大学で開催されたPiano-e-Competitionの参加者の演奏をMIDI化したものを使用しています。

GithubのPerformance RNNのページからダウンロードできます。

https://github.com/tensorflow/magenta/tree/master/magenta/models/performance_rnn

- [performance](#)
- [performance_with_dynamics](#)
- [performance_with_dynamics_and_modulo_encoding](#)
- [density_conditioned_performance_with_dynamics](#)
- [pitch_conditioned_performance_with_dynamics](#)
- [multiconditioned_performance_with_dynamics](#)

Performance RNN

各モデルにはそれぞれ少しずつ違いがあり特徴つけられています。

- [performance](#)
ベロシティの変化を含みまないモデルです。
- [performance_with_dynamics](#)
ベロシティの変化を含むモデルです。
- [performance_with_dynamics_and_modulo_encoding](#)
ベロシティの変化を含むモデルです。modulo encodingというエンコード方法が用いられています。
- [density_conditioned_performance_with_dynamics](#)
ベロシティの変化を含むモデルです。note densityの制御ができます。
- [pitch_conditioned_performance_with_dynamics](#)
ベロシティの変化を含むモデルです。pitch conditionの制御ができます。
- [multiconditioned_performance_with_dynamics](#)
ベロシティの変化を含むモデルです。note densityとpitch conditionの制御ができます。

Performance RNNで音楽生成

```
performance_rnn_generate \  
--config=<one of 'performance', 'performance_with_dynamics', etc., matching the bundle>\   
--bundle_file=<absolute path of .mag file> \  
--output_dir=/tmp/performance_rnn/generated \  
--num_outputs=10 \  
--num_steps=3000 \  
--primer_melody="[60,62,64,65,67,69,71,72]"
```

生成オプションコマンド

primer_pitches: 生成元となる音を和音としてベクトルで指定 例: "[60, 64, 67]"

primer_melody: 生成元となるメロディーをベクトルで指定 0~127 ノートナンバー -1ノートオフ -2ノーイベント

例: "[60, -2, 60, -2, 67, -2, 67, -2]"

primer_midi: 生成元となる音をMIDIファイルで指定できる

notes_per_second: 1秒内の音数

その他はMelody RNNなどと同様なので参照 (qpmなどはありません)

Performance RNNで音楽生成

```
performance_rnn_generate \  
--config=pitch_conditioned_performance_with_dynamics \  
--bundle_file=/ご自身の環境に合わせパス指定/pitch_conditioned_performance_with_dynamics.mag \  
--output_dir=/ご自身の環境に合わせパス指定/ \  
--num_outputs=1 \  
--num_steps=3000 \  
--notes_per_second=1 \  
--pitch_class_histogram="[2, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1]"
```

- pitch conditionのパラメーターのあるモデルを選択する事 例:pitch_conditioned_performance_with_dynamics

pitch_class_histogram

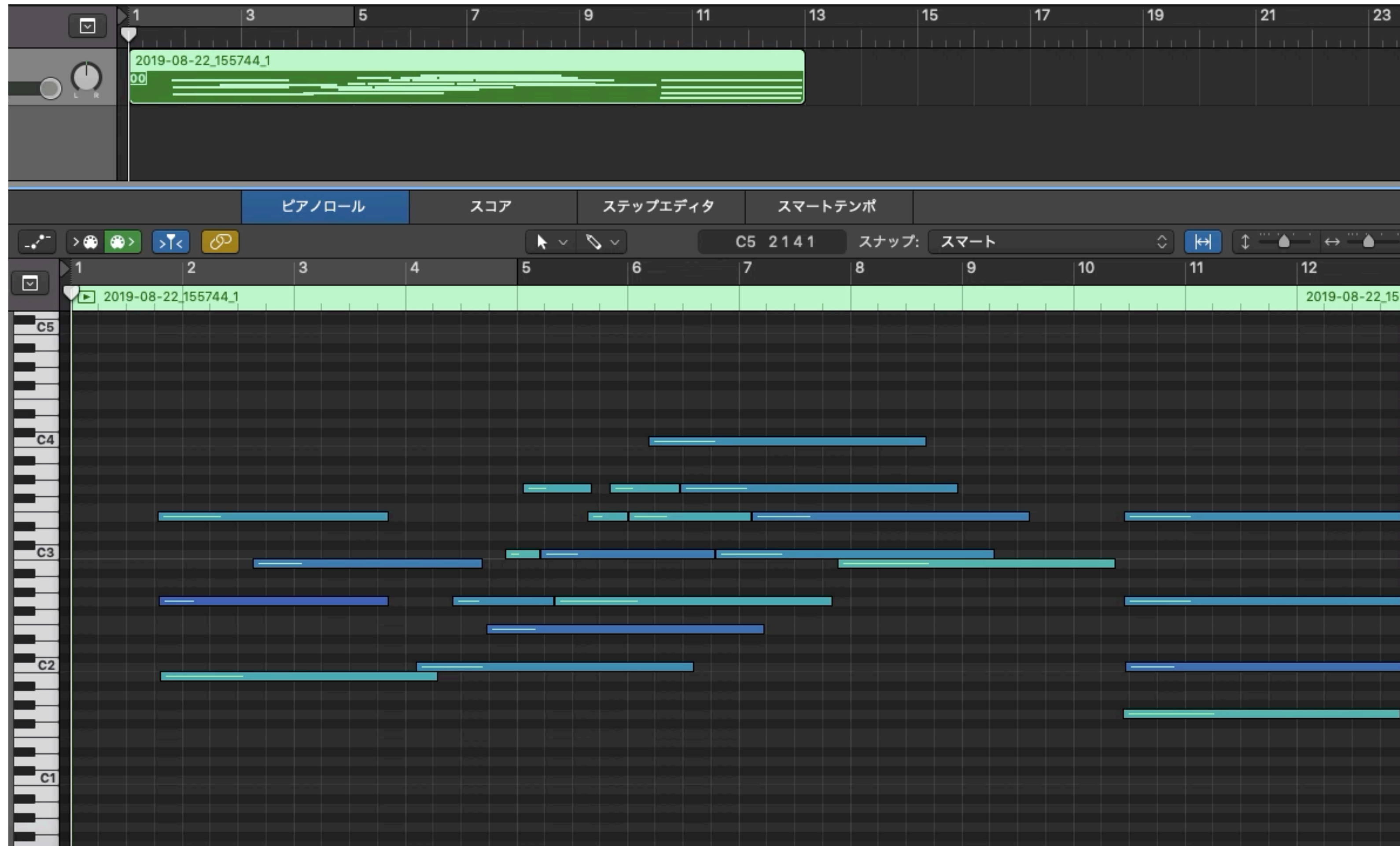
は使用する音を指定することができるコマンド

12音それぞれ個別に重みを数値で指定。

上記例はCM7 (c, e, g, b) でcの重みのみ2。

notes_per_secondを1に指定。

Performance RNNで音楽生成



※資料では動画は再生できません

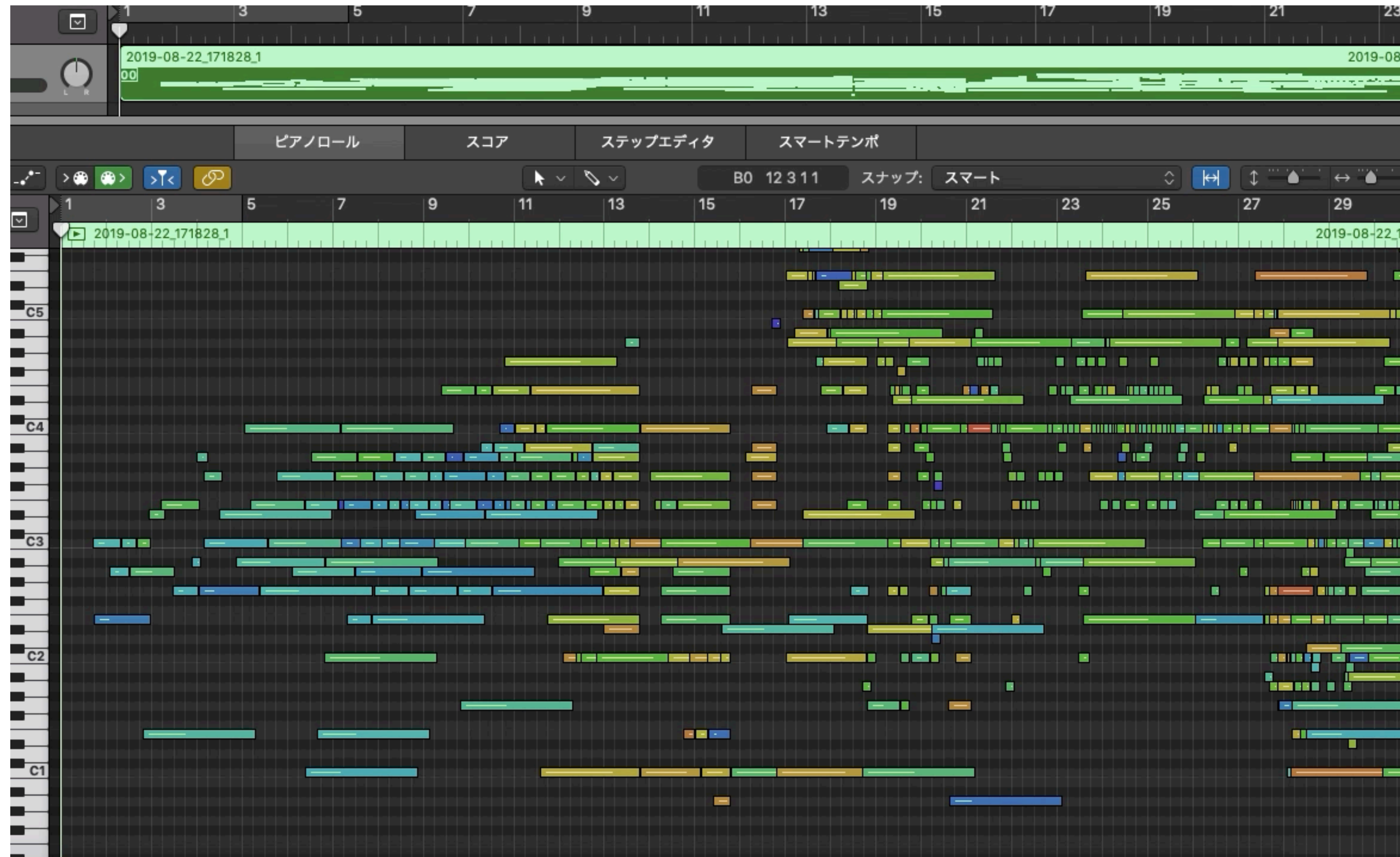
CM7 (c, e, g, b) でnotes_per_secondを1に

Performance RNNで音楽生成

```
python magenta/models/performance_rnn/performance_rnn_generate.py \  
--config=pitch_conditioned_performance_with_dynamics \  
--bundle_file=/ご自身の環境に合わせてパス指定/pitch_conditioned_performance_with_dynamics.mag \  
--output_dir=/ご自身の環境に合わせてパス指定/ \  
--num_outputs=1 \  
--num_steps=9000 \  
--notes_per_second=64 \  
--pitch_class_histogram="[5, 0, 1, 1, 4, 0, 1, 3, 0, 2, 1, 1]"
```

ブルーノートのスケールでnotes_per_secondを64に

Performance RNNで音楽生成



※資料では動画は再生できません

ブルーノートのスケールで生成

AI音楽の苦手な事

- **終止を理解していません。（ドミナント進行など）**
- **分節の理解が弱い**
- **構成ができない**
- **盛り上がりや抑揚の理解ができない**
- **音程の理解をしていない（確率のみの選択）**