P2

Overview

- Git repo: https://github.com/pycsham/BluesSoloGenerator (the BluesScale branch, instead of master branch)
- This report serves as a continuation of P1 since the implementation of the music generator is based off of the terminal symbol generator(following Keller's CFG) that I have implemented by P1. The first few sections are descriptions from P1 with some modifications.
- This phase of implementation is heavily based upon basic jazz theory and subjective music taste. I would describe the discovery process and explain the final implementation in the following report. I have decided to describe the discovery in chronological order since it would give the necessary background for the jazz theory involved and decisions that have been made throughout the implementation. I think the discovery process is important in algorithmic composition, since every decision made is an attempt to incorporate more/less parameters into the algorithm.
- A lot of the jazz music theory that I have learnt and discovered is through my sister, Melody since she is a piano major and is knowledge accumulated through years of listening to jazz music and playing multiple instruments. But I will resources to some fundamental jazz theory that I have referenced.

Purpose

To develop a blues solo improviser over a chorus
The type of blues I would like to generate:

- I would like to develop a solo improvisor i.e. a melodic line (called a "lick" in jazz)
- The blues I am referring to here is the classical blues format: 12 bars per chorus with a specific chord progression (1111-4411-5511). The time signature would be 4/4. This is the type of jazz that emerged in the early history of jazz (during the 20-30s). Since it is an earlier jazz form, it has less musical complexity and is therefore easier to codify.
- The key would be in C major (instead of E major stated in P1). This is because of the simplicity of C major in terms of music theory (but C major is actually one of the hardest keys to play in as an instrumentalist, especially if you're a pianist, so this key is chosen for theoretical simplicity and not practical simplicity).
- Since the nature of blues and jazz in general has a mixture of music theory and creativity, I would like to explore means to encode both in the improviser: i.e. both the algorithmic nature of music theory(be it classical or jazz) and randomness that is necessary in any blues/jazz performance.
- If possible, I would like to simulate a jazz performance, in which there is interaction between jazz musicians ("call and response" between different soloists - this is explained more below)

Tools

- Python3
  - Using the random library (pseudorandom number generators) in incorporate randomness in the algorithm
- Music 21
  - Description:
    - A python library that is created for musicology
    - It is being used to study datasets of music, to edit musical notation and composition
    - One of the basic building blocks of music21 is the Notes class
      - The Notes object represents a single note (pitch, duration etc)
      - The pitch and duration could either be set through parameters to the Notes object, or, if you need more control over the pitch and duration, you could create pitch and

duration objects.
        - On operating systems with MIDI playback support, you could directly play the midi of a note, or you could see the XML of the music score if you have installed a Music-XML reader
        - One thing to note is that, Apple removed MIDI support in newer versions of QuickTime, so I have been using garage band to play the MIDI files generated
    - The Stream object in music21 is similar to a list in Python. You could use the Stream object to group together notes and write the entire stream to a MIDI file. You could use Python list functionalities (e.g. append) on Streams. Making it quite convenient to work with if you have basic Python knowledge
    - There are a lot of other features in music21, but I did not use those features. Most of their features focus on the analysis of music and composition, instead of algorithmic composition.
    - Overall, I have found music21 well-documented and intuitive to use. Though it has a huge emphasis on music analysis instead of algorithmic composition.
- Garage Band
    - Creation of a backing track
        - Utilizing Music21 and garage band, I have created a backing track (using the C major key, chord progression of the classical blues) that consists of: a guitar and piano playing syncopated chords and a drums track I downloaded from: http://www.musicez.com/bluesrp.html
        - See the function: generateBackingTrack in BluesMusicGeneration.py for how the simple backing track was created
        - The backing track is saved as a garage band file in the folder: results/Backing Track/
        - The backing track is a good reference to see the rhythmic pattern of the generated track (swing and syncopation) and as a chord reference. This is also to simulate a simple jazz band, to hear how the generated track would perform with other instruments.
    - Playing around with different instrumentation
        - I also used the Garage band to modify the instrumentation of the generated track, just to hear how it sounds on different instruments (mostly the common instruments in a jazz band: acoustic guitar, brass instruments, certain woodwind instruments)
    - In order to evaluate the generated tracks independent from key, scale and backing track, the same backing track and key and chord progression is used for all generated tracks

P1 background and Implementation (still used in P2)

- An introduction to the formal grammar proposed
    - The set of terminal symbols of the grammar consists of both the rhythm and tone of a note. For example, a terminal symbol in this grammar would be "A4", where "A" denotes an approach tone, while "4" denotes a quarter note. The definition of the different tones will be explained further below.

    - The grammar consists of two layers: there is a layer that takes in n as a parameter, where n denotes the number of beats you wish to generate. And the grammar is simply a recursive set of production rules that gives you a set of non-terminals, that are going to be expanded by the next layer of the CFG. The following is the set of production rules for the skeleton of the melody to be produced, where each number inside the square brackets are the corresponding probabilities:

1) P(0) → empty [1]
2) P(1) → Q1 [1]
3) P(2) → Q2 [1]
4) P(3) → Q2 Q1 [1]

5) P(n) → Q2 P(n – 2) [.25]
6) P(n) → Q4 P(n – 4) [.75]

- As 4 beats denote one bar, for the purpose of generating a 12-bars blues solo, the input to this would be 48 beats. One feature in jazz and blues is syncopation: instead of stressing on the strong beats, jazz musicians would create "swing" in the rhythm by stressing on the weaker beats. This could be created by allowing half-notes to straddle the mid-point of a measure (Keller). This is incorporated in the above structure seen in rule 5. Where Q2 denotes half a measure.

- Moving onto the next layer of production rules. The first part of a terminal symbol denotes a set of tones: e.g. C denotes a chord tone (musical notes in the current chord), L denotes a color tone (which are tones that are not in the current chord, but is harmonious to the current chord), R denotes a rest. The second part of a terminal symbol denotes the duration of the note. For example 1 is a whole-note. (more explanation on the jazz theory in the following sections)

- The probabilities in the next layer of grammar doesn't necessarily add up to 1 for a particular non-terminal. Therefore, some normalization needs to be performed on the proposed CFG.

- Since the generated sequence of non-terminals simply denotes a proposed tone, and not the actual note, some post-processing needs to be done to map the terminal symbol to an appropriate note. The note itself would be selected from a set of notes that is built according to the key and chord sequence that the user specifies.

• Implementation of the generation of terminal symbols from the CFG
  - See the following git repo for the current implementation of the generation of terminal symbols: https://github.com/pycsham/BluesSoloGenerator/CFG.py (in the BluesScale branch instead of master)

  - The CFG class:
    - Consists of two fields as of now, since the non-terminal symbols aren't really used in the generation of terminal sequences, it is currently commented out.
    - The constructor function maps each rule to a set of integers within the range 1-100, according to the probabilities specified in the paper by Keller. This is so that, if I want to adjust the probabilities, the function automatically assigns appropriate integers to each rule. These numbers are used in the generation process, where we will need to randomly select a production rule to substitute.
    - The two main functions in the class are: generate and generate skeleton
    - Generate recursively applies the production rules on each non-terminal from left to right, by randomly selecting a production rule that has the non-terminal on the left hand side, according to the specified probabilities.
    - GenerateSkeleton recursively applies the production rules for generating the high level skeleton/structure of the music piece. n is the number beats. Currently defaults to 48 for a 12 bars blues.
    - I have fixed an issue with the GenerateSkeleton function: I forgot to add the n=0 base case during P1
  - The Main.py file:
    - contains the main function, that creates a CFG instance, supplying the class with the set of terminals and production rules. Some normalization was done by hand in the CFG as mentioned in the section above. I have modified some probabilities for production rules.

The reasoning would be explained in sections that follows.

- I have decided to generate all 48 beats at once, instead of generating one segment at a time according to chord sequences, since there is a correlation between the melodic sequence in each bar. Most jazz musicians who improvise on-the-fly, creates new changes in the melodic line based on the music that has been played so far and their knowledge on the underlying chord sequence. Therefore, to imitate that process, I am going to generate all 48 beats at once, independent of the chord sequence, then map the terminals to the correct notes according to the scale and chords.

P2 experimentation process

- Most of the description for the implementation below are in the BluesMusicGeneration.py file unless specified otherwise (the main changes in implementation for P2)
- **Step 1: Jazz music theory and Keller's CFG**
    - The step 1's implementation is in the master branch. This is because, despite the failure of the first attempt, it illustrates the experimentation process and I wanted to keep this branch for future experimentation.
    - Upon the successful implementation of Keller's probability CFG, the sequence of terminal symbols consists of two things: the tone and the rhythm
        - The rhythm:
            - follows standard musical notation.
            - Denoted by a number (e.g. 8/3 is a note in an 8th triplet).
            - One challenge I faced, was that to create "swing", some notes stride across bar lines, making it difficult to group the terminal symbols together into bars. At this stage of implementation, due to the use of floating pointer numbers in self.rhythmMap, the grouping of notes into bar was unsuccessful (fixed in step 2)
        - A really good resource for learning basic jazz theory (in 20 minutes): https://www.youtube.com/watch?v=RpObAWZ0SKM
        - The tone: could be of the following categories:
            - Chord Tones:
                - Tones in the chords
                - blues chords are generally dominant 7th chords. Which are basically chords that consists of the major chord triads (1 3 5 in a major scale) and a 7th note. The 7th note is what gives blues chord a "bluesy sound"
            - Color Tones:
                - The notes that, in addition to the chord tones, form the "extended jazz chords". the 9th, 11th and 13th notes of a major scale.
                - A good reference for how to find the extended chords: https://www.youtube.com/watch?v=6u4Qmfc54nw
            - Approach Tones:
                - Defined as tones that "approaches" the chord tone and color tones
                - Instead of approaching both chord tones and color tones, I have modified the definition to just approaching chord tones. This is to reduce the level of chromaticism in the generated track.
                - There are many different types of approach tones. You could approach a color/chord tone above/below and diatonic or chromatic (diatonic is simply the note above or below the target tone on the major scale, chromatic is a half note above or below the target tone)
                - I have chosen to use the chromatic below approach tone to create more chromaticism. But this makes the music quite atonal, so I later tried different approach tones (mostly diatonic).
                - An approach tone would force the next tone to be a chord tone

        - This blog contains details on the various approach tone techniques: https://terencewrightguitar.com/approach-note-series-1-root-and-third/
    - Scale tones:
        - Defined as a tone in the scale that sounds harmonious with the current chord
        - I was not sure what are the "scale tones". So I simply used the classical blues scale tones
        - I originally thought that the classical blues scale tones changes according to the chord. But later discovered that, in jazz theory, the blues scale is chosen based on the underlying key. The version on git currently contains this error.
    - Helpful tone:
        - Could be one of a chord tone, color tone or approach tone

- Results were unsuccessful:
    - Music generated was very atonal. Possibly due to a combination of the random choice of octave, the random choice of notes from a set of notes, leading to too much chromaticism. On top of that, since the definition of the different tones were not well defined in the research paper (in general, jazz music theory does not have a lot of "standard" terminology), so it was difficult to understand what were the proper set of notes that each category of tones should map to.
    - Another error that could have caused the atonality is the wrong use of scale tones
        - But even after switching the scale tones to only use chord tones and switching the approach tones to use diatonic below, the music still sounds relatively atonal
        - I suspect it's due to the frequent use of "color tones" and the random choice of octaves
        - Another issue is, licks are usually runs. Which means, a long section in the lick is supposed to go in "one direction" on the keyboard. The choice of the next note should be dependent on the choice of the previous note.
    - But the music generated clearly showed syncopation and "swung" notes
    - I also learnt a lot more about jazz theory after this stage
    - Due to time constraint, I was not sure how much longer would I need to play around with all of the different parameters to generate decent sounding blues solo licks. So I have decided to simply use the rhythmic structure generated by Keller's CFG, and map the pitches according to the standard blues scales (described in details below). This would greatly reduce the randomness in pitch and would most likely generate decent sounding tracks.

- Step 2: more jazz music theory and modification of implementation (see the branch BluesScale on git repo)

    - Switched to use key C (major) since it is the simplest and most basic key
    - Chords are still dominant 7th chords (as described above)
    - One major change in this stage, is instead of using the suggested tones in the CFG, the generated terminal symbols are mapped to pitches on the blues scales instead.
    - The blues scale:
        - a good reference for how soloing over the blues scale works: https://www.youtube.com/watch?v=wXGfjzPtuLQ
        - A pentatonic scale that is commonly used in blues improvisation
        - For the key C major, there are two that are sonorous to the 3 dominant 7th chords used: The C and A blues scales (see self.bluesScale in code for the exact pitches)
        - The idea is to play notes on the blues scale, either in upwards or downwards directions
        - Since the blues scales is circular in nature, I have used the modular operator when

selecting the next index on the list of notes in a blues scale

- As mentioned above a few of the problems are: random octave selection, and notes are selected too randomly that they don't form "runs"
    - To mitigate theses issues, the generated track would go in a single direction on the selected blues scale until the direction is switched
    - When direction is switched, the octave that is chosen is either one above or one below the current octave
    - Also at this stage, I have limited the octaves to 2-6
    - The starting point of the entire piece is not random: it starts from the note C and the C blues scale at octave 4
- Randomness:
    - Whenever we are "switching directions", the new blues scale and new note is also chosen at random
- Direction is switched after the following notes/conditions:
    - A rest: A rest is usually when jazz musicians (especially brass/woodwind soloists) "catch their breath" and create a new "lick".
    - Bounds of octave: once we've reached the upper or lower bound of octave (2 and 6)
    - Chord change: as mentioned above, the blues scale isn't chosen based on the chords but the underlying key. But a change in chord gives a change in the atmosphere and color. Therefore it makes musical sense to switch direction of the lick after a chord change
    - After a long note (a full note or a half note): This follows a similar logic as the rest.
- I have also fixed the issue of grouping notes into bars mentioned above. The grouping is performed at the constructor of the BluesMusicGeneration class. This grouping takes into consideration notes that strides along bars and triplets, to avoid the use of floating pointer numbers (as much as possible).
- Result:
    - The generated tracks sound decent over backing track
    - But I noticed that tracks that has more triplets sounds more "swung" and "groovy"
    - The sounds aren't too interesting (quite predictable).
    - The tracks from experimentation could be found in the folder /results/Discovery (the mp3 files)
    - But the results at this stage is a significant improvement from stage 1

<u>Step 3: experimentation with the various perimeters</u>

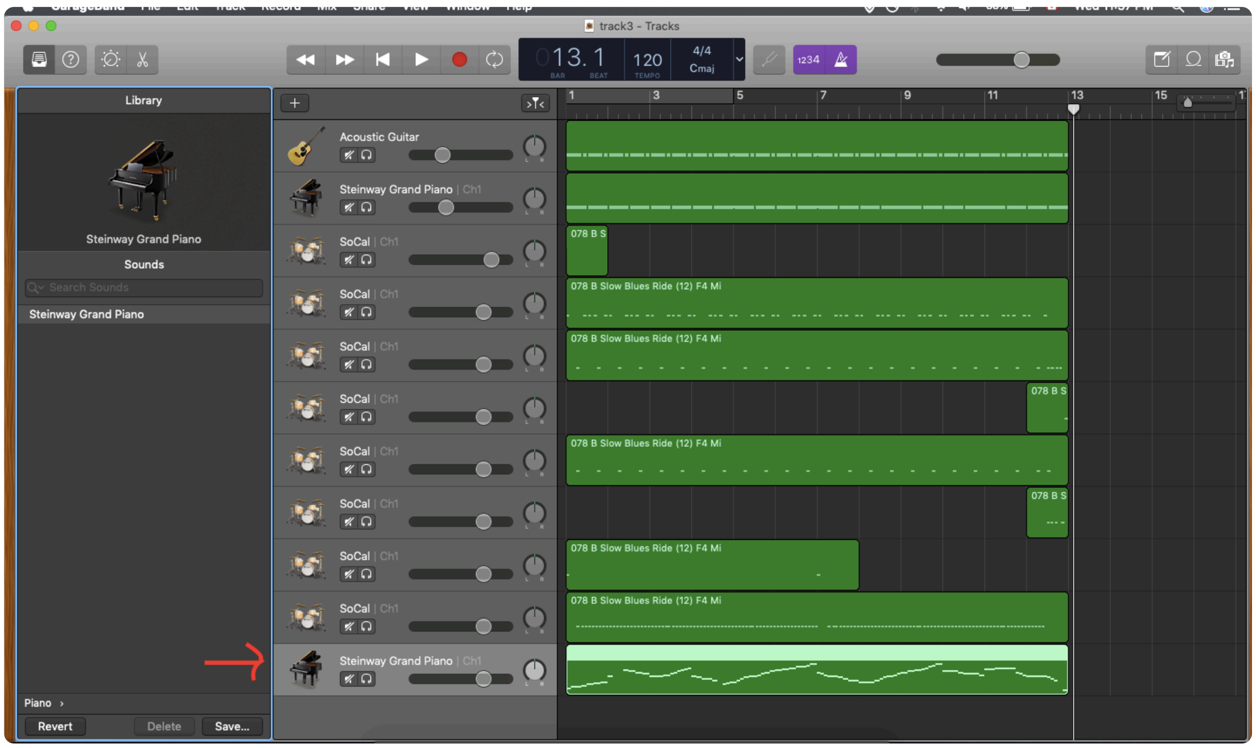
- Modified the probabilities in the CFG to generate more triplets. This gives the overall rhythmic structure more "groove". It was surprising that increasing the probability of quarter triplets gave the rhythmic structure more "swing" than increasing the portability of eight note triplets. I experimented with the probabilities a little to find a balance between increasing the number of quarter triplets and eight note triplets.
- Introduced more randomness:
    - The starting point of the entire track is now randomized: the choice of octave, note and blues scale is random
    - "Switching directions" more frequently: also switching direction after a dotted quarter note since it is a "long" note
    - Adding the use of the "A+C" blues scale (which is a mixture of notes from the A and C blues scale) to generate more complex textures
    - Increasing the bounds of octave to 1-7
    - I tried to make the track "skip notes" on the blues scale. So instead of playing the next note on the blues scale, it could play two notes above/below the blues scale. But this created too much chromaticism and sounded unpleasant. So I removed this from the algorithm.

- I also copied pieced together two generated tracks and used different instrumentation to imitate the "call and response" aspect of jazz performances. Due to the use of limited blues scales, the tracks sounded quite nice and realistic.
    - "call and response" in jazz is the interaction between instrumentalists (mostly between soloists) where one soloist improvises as a "response" to the previous soloist's improvisation
    - This was not possible from Step 1's implementation. Apart from it being a failed attempt, the lack of correlation between the generated tracks makes it impossible to create this "call and response" interaction between tracks.

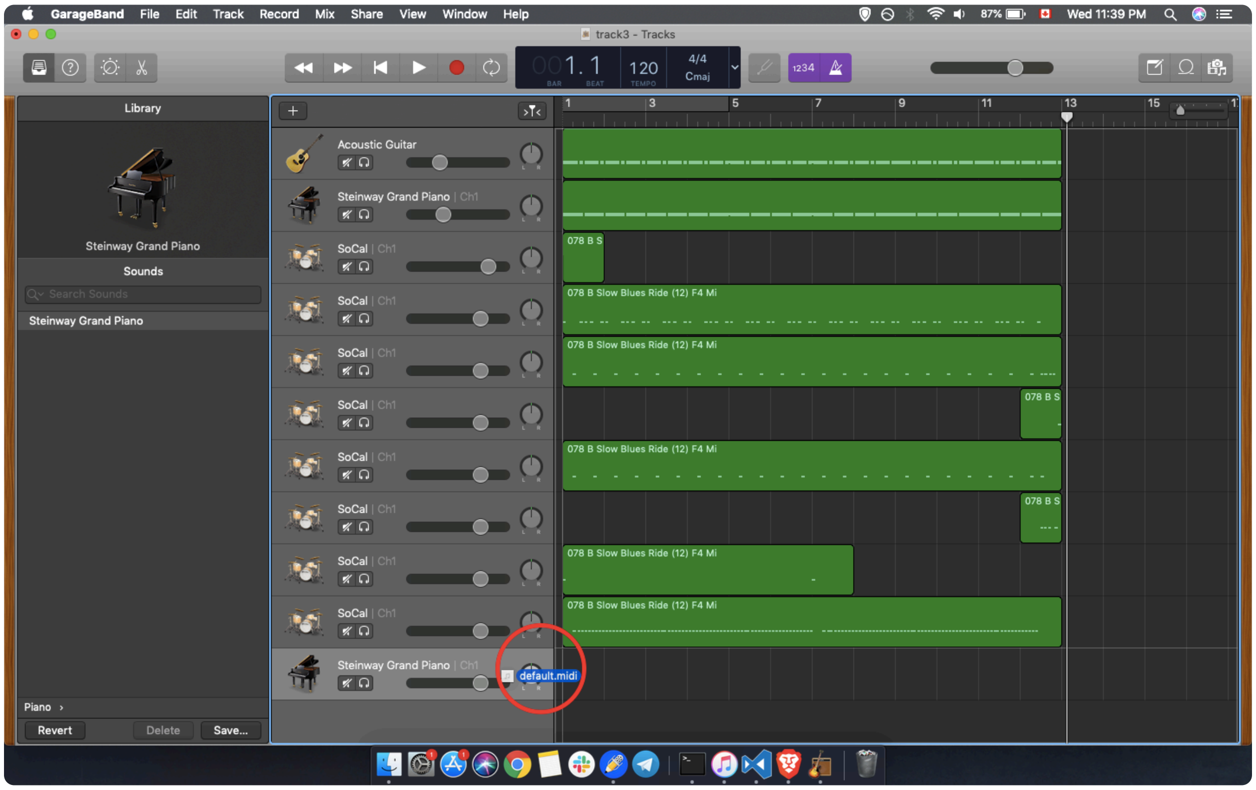
<u>Results and how to create your own track</u>

- The result tracks generated are in the /results/Final Results Tracks folder.
    - I did not exclude any tracks generated after the algorithm is more or less finalized
    - The callResp tracks are simply tracks that are comprised of two generated tracks
- I find the finalized algorithm generate tracks that are a good balance between unpredictability and sounding decent (subjective opinion)
- My favourite is the second lick (piano) in the callResp2 track. It is a lick that I would personally play if I were improvising on the piano.
- How to create your own track
    - You simply need to run Python3 Main.py on the command line
    - This generates a midi filed called "default.midi" in the results folder
    - Then open the backing track on Garage Band (track 3 in the backing track folder)
    - You could delete the bottom most track, and drag and drop the default.midi file as follows:

- You could use different instrumentation for the track (I would recommend using the jazz organ or



brass instruments, and adjusting increasing the "low" configuration of these instruments on garage band)



- To use different instrumentation you could also adjust the octaves in the code, since most instruments have a more limited range compared to the piano
- Some future modifications that could be made to the generator:
    - Customizable key and chord progressions: I believe the current algorithm would work for various keys and common jazz progressions, as long as the blues scales is adjusted accordingly. This could be done pretty easily by writing a function that maps keys and chords to blues scales used.
    - Customizable octave range: to adjust to different instruments. Some instruments have limited range, so the track can't be played properly if you're using a different instrument.
    - The introduction of more commonly used "scales" in jazz improvisation to add more texture
    - I would also like to go back to work on the master branch's implementation, and see if reconfiguring some parameters could get the mapping of the suggested tones in the CFG to the corresponding pitches to work.

Lessons Learnt

- It is really difficult to codify jazz music theory. Jazz theory has some foundation in classical music theory, but is more flexible. The degree of flexibility is highly subjective. Finding a balance of generating "interesting" (i.e. more random melodies) and generating something that is pleasant to the ear took quite a long time.
- Since there are so many factors that contributes to a good jazz performance, I have learnt through the first failed attempt that isolating out components (e.g. rhythm, pitch, instrumentation), diving deep into each component and making them work individually first, then combining the components work a lot better than attempting to codify both pitch and rhythm in the first go.
- The CFG was surprisingly good at capturing the rhythmic structure of jazz music. There is

definitely a lot of swing and groove in the generated tracks. The rhythmic aspect of jazz is one of the hardest things to grasp for soloists, due to the polyrhythm nature of jazz (due to it's origin in african folk music and spiritual music). A lot of jazz musicians learn the "groove" by ear and there is not formal way to be able to "capture the swing". But a simple CFG is capable of generating quite complex yet pleasant rhythms. I could see how the Impro-visor (the tool built by the team who developed the CFG) could be a great aid for beginner jazz musicians.

- I underestimated the difficulty in mapping the tones in the terminal symbol to various pitches. In the first implementation, I introduced too much randomness in the algorithm, which generates a lot of atonal tracks. Chromaticism is the "soul" of blues, it gives tracks a "bluesy" feeling, but adding too much of chromaticism makes the music jarring to listen to. So once again, it took a long time to find a balance between classical music tonality and chromaticism. Which is one of the main skills that jazz musicians have to learn. The use of the 3 blues scales introduces chromaticism, but limits the degree of randomness of the pitches, which helps in generating nice sounding blues tracks.

References

"Blues Midi Rhythm Patterns." *Midi Rhythm Tracks, Midi Loops, Drum Loops*, www.musicez.com/bluesrp.html.

JAZZ TUTORIAL. "Jazz theory explained in 20 minutes(chords, scales, extended harmony)". YouTube. https://www.youtube.com/watch?v=RpObAWZ0SKM

Keller, R. M., & Morrison, D. R. (2007). A grammatical approach to automatic improvisation. In Proceedings of the Sound and Music Computing Conference, pp. 330–337.

MangoldProject. "Soloing Over The Blues Scale". Youtube. https://www.youtube.com/watch?v=wXGfjzPtuLQ

Piano Groove. "Jazz Piano Chords - Extensions 9ths, 11ths & 13ths". Youtube. https://www.youtube.com/watch?v=6u4Qmfc54nw

Terencewrightguitar. "Approach Note Technique: Terence Wright Guitar." *Approach Note Technique*
| *Terence Wright Guitar*, 2 June 2017, terencewrightguitar.com/approach-note-series-1-root-and-third/.

"What Is music21?" *What Is music21? - music21 Documentation*, web.mit.edu/music21/doc/about/what.html.