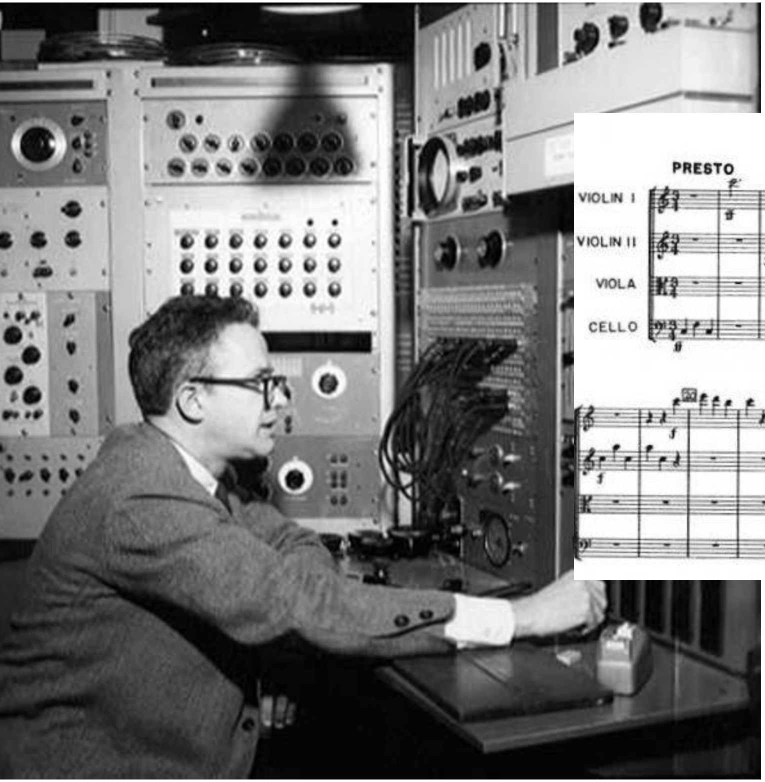
Rule-Based Composition

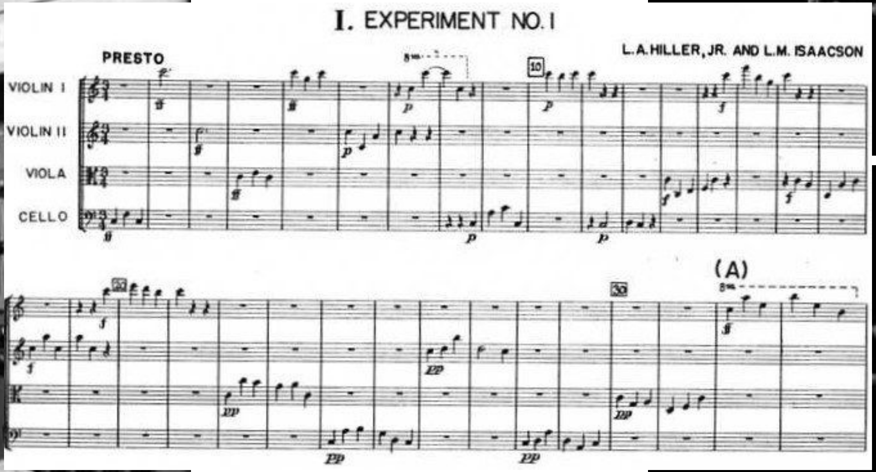
Prof. Michael Casey

MUS14, COSC 89/189 Week 3a

Hiller and Isaacson, *Illiac Suite* (1956)

https://www.musicainformatica.org/topics/illiac-suite.php



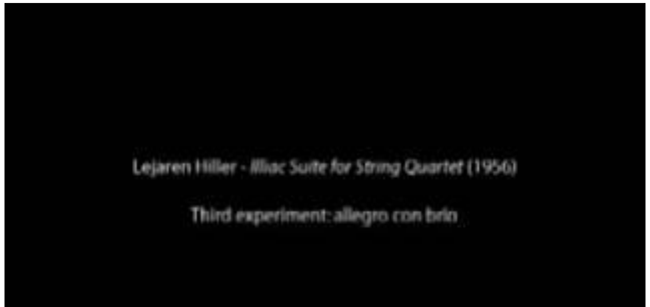


Lejaren Hiller - *Illiac Suite for String Quartet* (1956)

Fi... eriment: presto, andante, allegro

- *Illiac Suite for String Quartet* (1956)

riment: adagio, ma non troppo lento



Lejaren Hiller - *Illiac Suite for String Quartet* (1956)

Third experiment: allegro con brio

Tech
Closeup

FROM DARKNESS, *LIGHT*

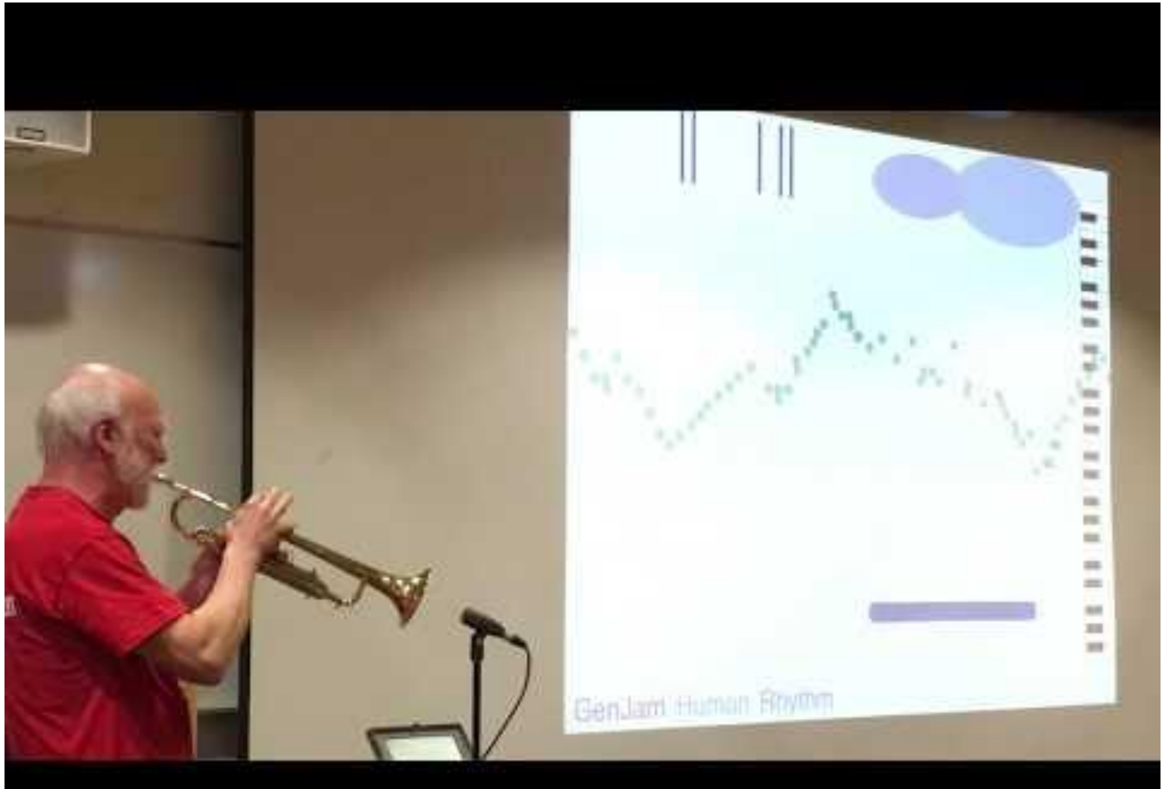
Emily Howell

```
(defvar *illegal-verticals* '(0 1 2 5 6 10 11 13 14 17 18 22 23 25 26 29 30 34 35 -1 -2
-3 -4 -5 -6 -7 -8) "Illegal verticals")
(defvar *illegal-parallel-motions* '((7 7)(12 12)(19 19)(24 24)) "Illegal parallel
motions")
(defvar *illegal-double-skips* '((3 3)(3 4)(3 -3)(3 -4)(-3 -3)(-3 -4)(-3 3)(-3 4)
                                 (4 3)(4 4)(4 -3)(4 -4)(-4 -3)(-4 -4)(-4 3)(-4 4))
"Illegal double skips")
(defvar *direct-fifths-and-octaves* '((9 7)(8 7)(21 19)(20 19)) "Direct fifths and
octaves")
(defvar *solution* () "Where the initial solution is stored")
(defvar *counterpoint* () "The resulting counterpoint is stored here.")
(defvar *save-voices* () "Pitches only of the resultant counterpoint are stored here.")
(defvar *rules* () "Where the program stores its rules")
(defvar *seed-note* 60 "The note which produces the possibilities for the opening note
")
(defvar *seed-notes* '(64 62 59 57 55 60) "Must start with a reasonable seed note for
the program to work")
(defvar *backtrack* () "Sets the backtrack printout routine.")
(defvar *cantus-firmus* '(69 71 72 76 74 72 74 72 71 69) "The cantus firmus")
(defvar *new-line* () "Shere the new line is stored")
(defVar *rs* (make-random-state t) "Variable for storing the current random state.")
(defvar *save-rules* () "Variable for storing rules during the inference process.")
(defvar *print-state* t "Variable for setting whether or not the Listener window prints
the various steps of composition.")
(defvar *auto-goals* () "Variable for having the program create its own goals.")
(defvar *models* () "Where the models for the goals are stored.")
(defvar *saved-templates* () "Where the templates for getting successful seeds are
stored.")
(setq c1 36 d1 38 e1 40 f1 41 g1 43 a1 45 b1 47 c2 48 d2 50 e2 52 f2 53 g2 55 a2 57 b2
```

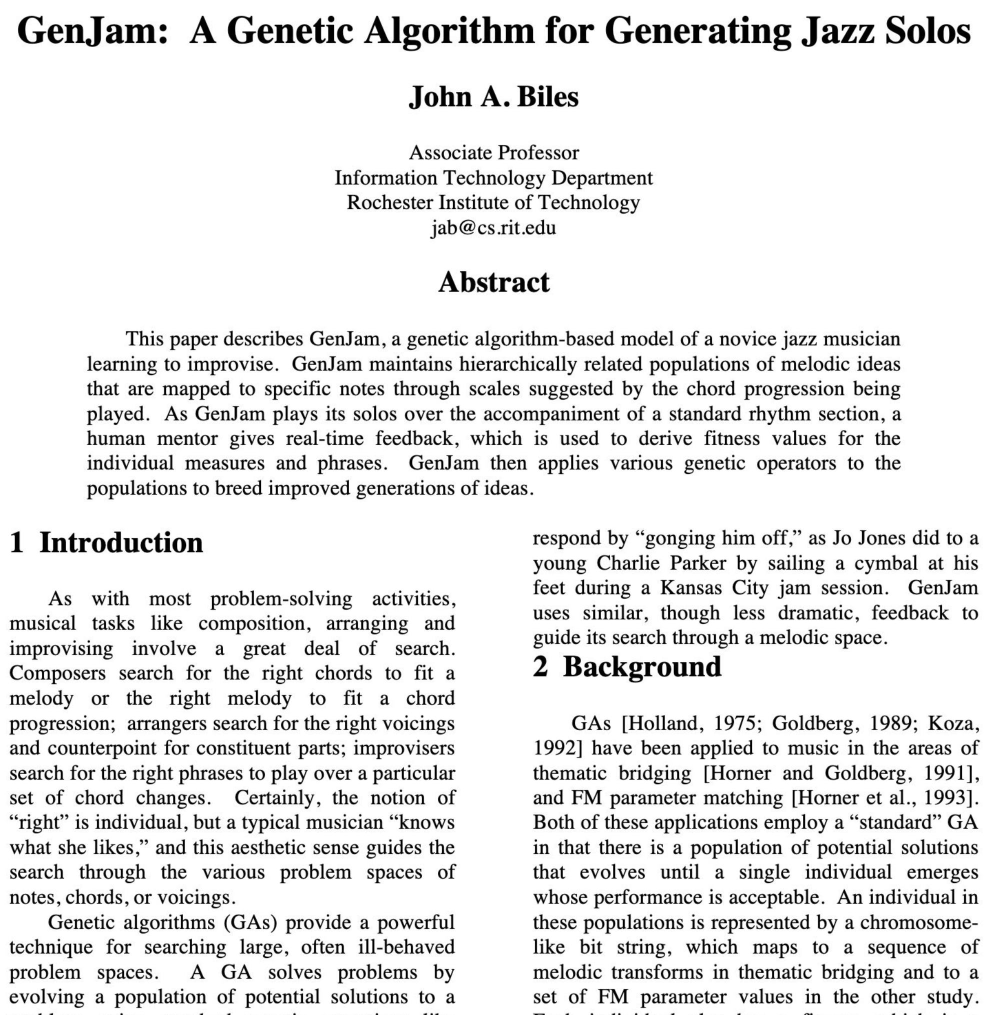
Band-in-a-Box

BAND IN A BOX®
What is Band-in-a-Box®?
Available for Windows® and Mac®
www.pgmusic.com

Genetic Algorithms - *GenJam,* by Al Biles



https://genjamorg.files.wordpress.com/2019/07/bilesicmc94.pdf



GenJam: A Genetic Algorithm for Generating Jazz Solos

John A. Biles

Associate Professor
Information Technology Department
Rochester Institute of Technology
jab@cs.rit.edu

Abstract

This paper describes GenJam, a genetic algorithm-based model of a novice jazz musician learning to improvise. GenJam maintains hierarchically related populations of melodic ideas that are mapped to specific notes through scales suggested by the chord progression being played. As GenJam plays its solos over the accompaniment of a standard rhythm section, a human mentor gives real-time feedback, which is used to derive fitness values for the individual measures and phrases. GenJam then applies various genetic operators to the populations to breed improved generations of ideas.

1 Introduction

As with most problem-solving activities, musical tasks like composition, arranging and improvising involve a great deal of search. Composers search for the right chords to fit a melody or the right melody to fit a chord progression; arrangers search for the right voicings and counterpoint for constituent parts; improvisers search for the right phrases to play over a particular set of chord changes. Certainly, the notion of "right" is individual, but a typical musician "knows what she likes," and this aesthetic sense guides the search through the various problem spaces of notes, chords, or voicings.

Genetic algorithms (GAs) provide a powerful technique for searching large, often ill-behaved problem spaces. A GA solves problems by evolving a population of potential solutions to a

respond by "gonging him off," as Jo Jones did to a young Charlie Parker by sailing a cymbal at his feet during a Kansas City jam session. GenJam uses similar, though less dramatic, feedback to guide its search through a melodic space.

2 Background

GAs [Holland, 1975; Goldberg, 1989; Koza, 1992] have been applied to music in the areas of thematic bridging [Horner and Goldberg, 1991], and FM parameter matching [Horner et al., 1993]. Both of these applications employ a "standard" GA in that there is a population of potential solutions that evolves until a single individual emerges whose performance is acceptable. An individual in these populations is represented by a chromosome-like bit string, which maps to a sequence of melodic transforms in thematic bridging and to a set of FM parameter values in the other study.

Flow Machines

(F. Pachet)

Daddy’s Car

Daddy’s Car

At SONY CSL Research Lab François Pachet, together with the musician Benoit Carré and thanks to the support of the European Research Council, created the first-ever entire songs composed by Artificial Intelligence: “Daddy’s Car”.”Daddy’s Car” is the result of FlowMachines, a system that learns music styles from a huge database of songs. Exploiting unique combinations of style transfer, optimization and interaction techniques, FlowMachines composes novel songs in many styles. “Daddy’s Car” is composed in the style of The Beatles.

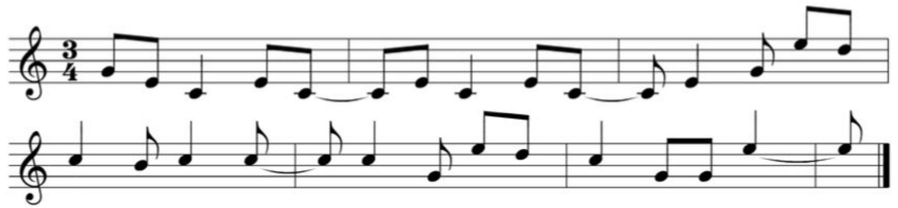
French co [redacted] he two songs are excerpts of albums composed by Artificial Intelligence to be released in 2017. The research leading to these results has received funding from the European Research Council under the European Union’s Seventy Framework Programme (EP7/2007-2013 Grant Agreement no.291156). The research leading to these results has been conducted by Sony Computer Science Laboratories Paris (Sony CSL Paris) and Pierre and Marie Curie University (UPMC), under the leadership of François Pachet.

The “Continuator” (2011) Francois Pachet (Spotify)

Markov-Model
Learning from Music

Figure 4: The transition probabilities for the pitches of the melody of Figure 3

	C_3	E_3	$F_3^{\#}$	G_3	A_3	B_3	C_4	D_4	E_4
C_3	0	1	0	0	0	0	0	0	0
E_3	1/2	0	1/4	1/4	0	0	0	0	0
$F_3^{\#}$	0	0	0	1	0	0	0	0	0
G_3	0	1/3	0	1/3	0	0	1/6	0	1/6
A_3	0	0	0	0	0	1	0	0	0
B_3	0	0	0	0	1/2	0	1/2	0	0
C_4	0	1/5	0	1/5	0	1/5	1/5	0	1/5
D_4	0	0	0	0	0	0	1	0	0
E_4	0	0	0	0	0	0	0	1	0



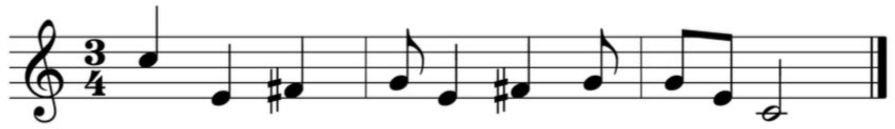


Figure 7: A 3-bar melody generated from the Markov model of Star Spangled Banner with a **`meter`** constraint and a cardinality constraint stating that two $F_3^{\#}$ must appear in the melody.

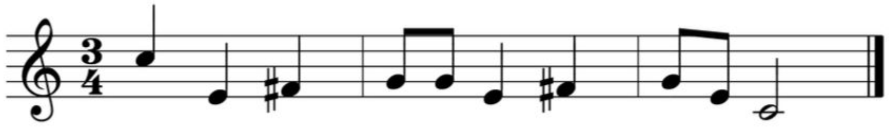
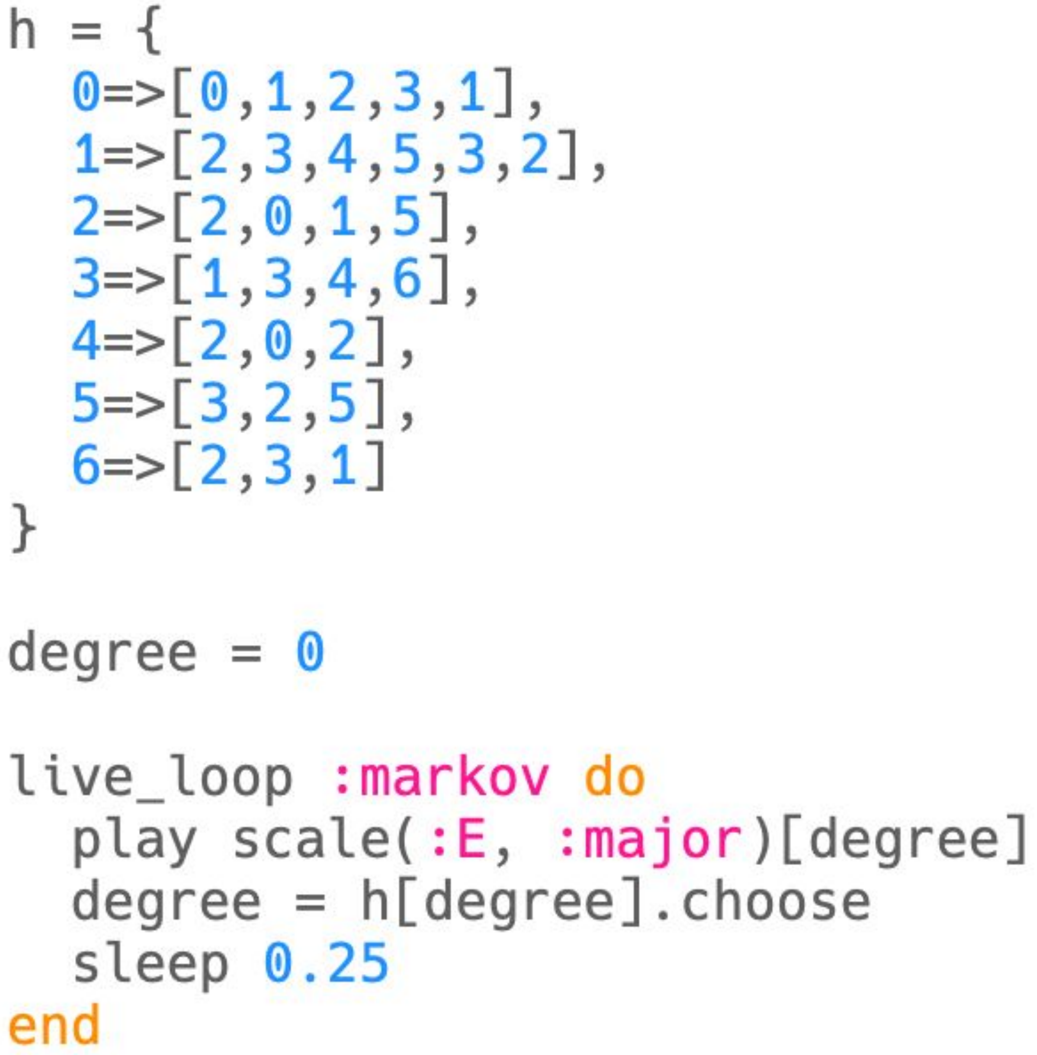


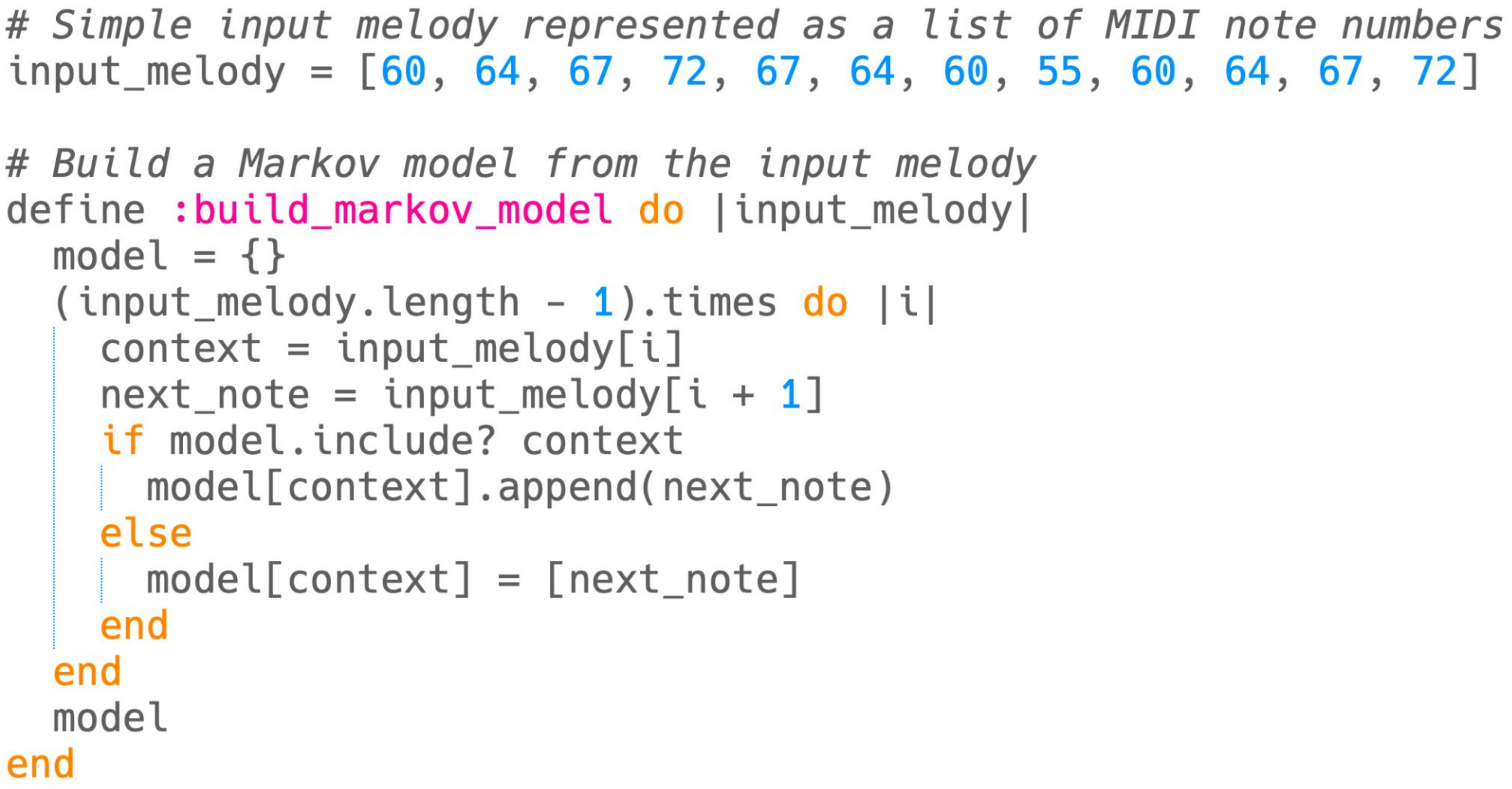
Figure 8: Another solution (see Figure 7). This solution does not have the syncopated rhythm of Figure 7.

Markov chain (melody)

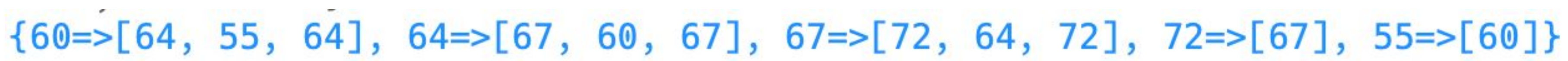


```
h = {
  0=>[0,1,2,3,1],
  1=>[2,3,4,5,3,2],
  2=>[2,0,1,5],
  3=>[1,3,4,6],
  4=>[2,0,2],
  5=>[3,2,5],
  6=>[2,3,1]
}

degree = 0

live_loop :markov do
  play scale(:E, :major)[degree]
  degree = h[degree].choose
  sleep 0.25
end
```

Markov Model Learning

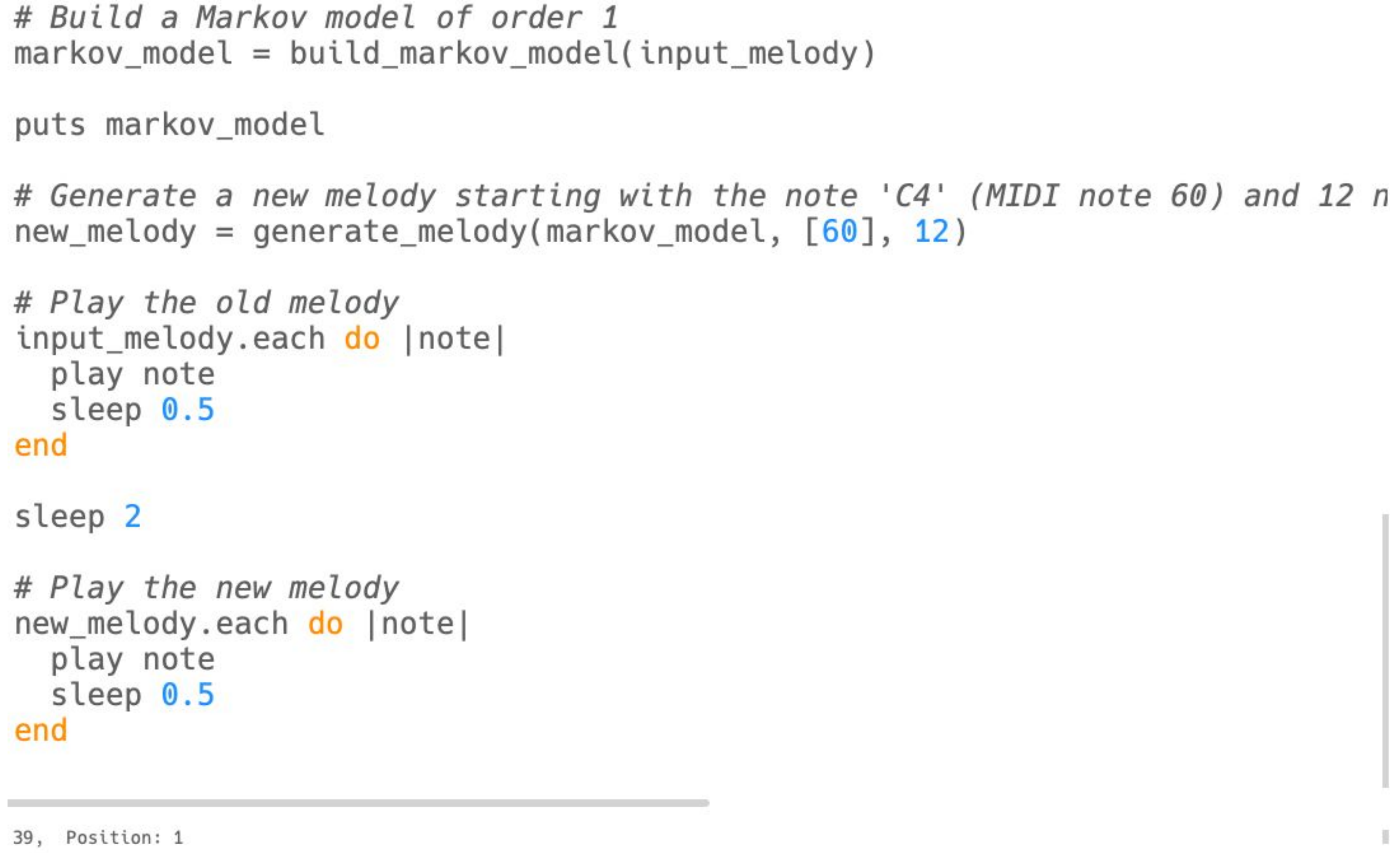


```
# Simple input melody represented as a list of MIDI note numbers
input_melody = [60, 64, 67, 72, 67, 64, 60, 55, 60, 64, 67, 72]

# Build a Markov model from the input melody
define :build_markov_model do |input_melody|
  model = {}
  (input_melody.length - 1).times do |i|
    context = input_melody[i]
    next_note = input_melody[i + 1]
    if model.include? context
      model[context].append(next_note)
    else
      model[context] = [next_note]
    end
  end
  model
end
```



```
{60=>[64, 55, 64], 64=>[67, 60, 67], 67=>[72, 64, 72], 72=>[67], 55=>[60]}
```

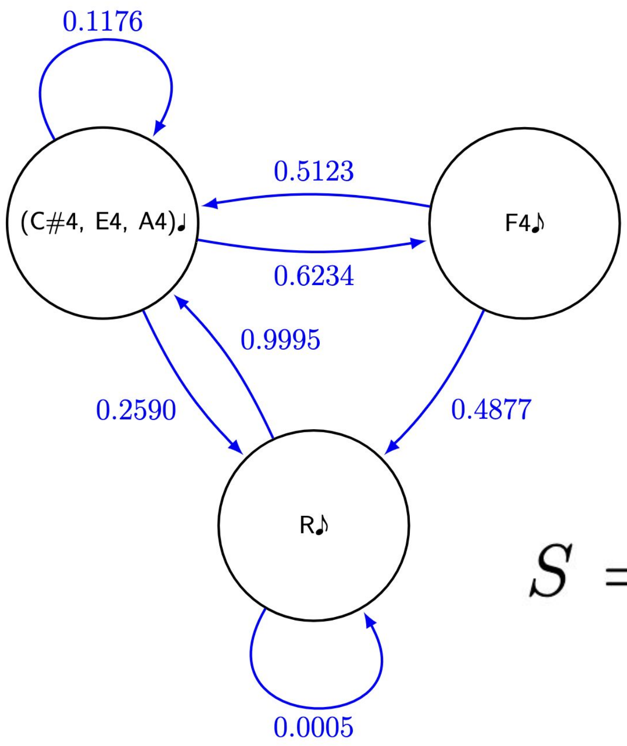
```
# Generate a new melody using the Markov model
define :generate_melody do |model, start_note, length|
  melody = start_note.dup
  (length - 1).times do
    context = melody[-1]
    if model.include? context
      next_note = model[context].choose
      melody.append(next_note)
    else
      break
    end
  end
  melody
end
```



```
# Build a Markov model of order 1
markov_model = build_markov_model(input_melody)

puts markov_model

# Generate a new melody starting with the note 'C4' (MIDI note 60) and 12 n
new_melody = generate_melody(markov_model, [60], 12)

# Play the old melody
input_melody.each do |note|
  play note
  sleep 0.5
end

sleep 2

# Play the new melody
new_melody.each do |note|
  play note
  sleep 0.5
end
```

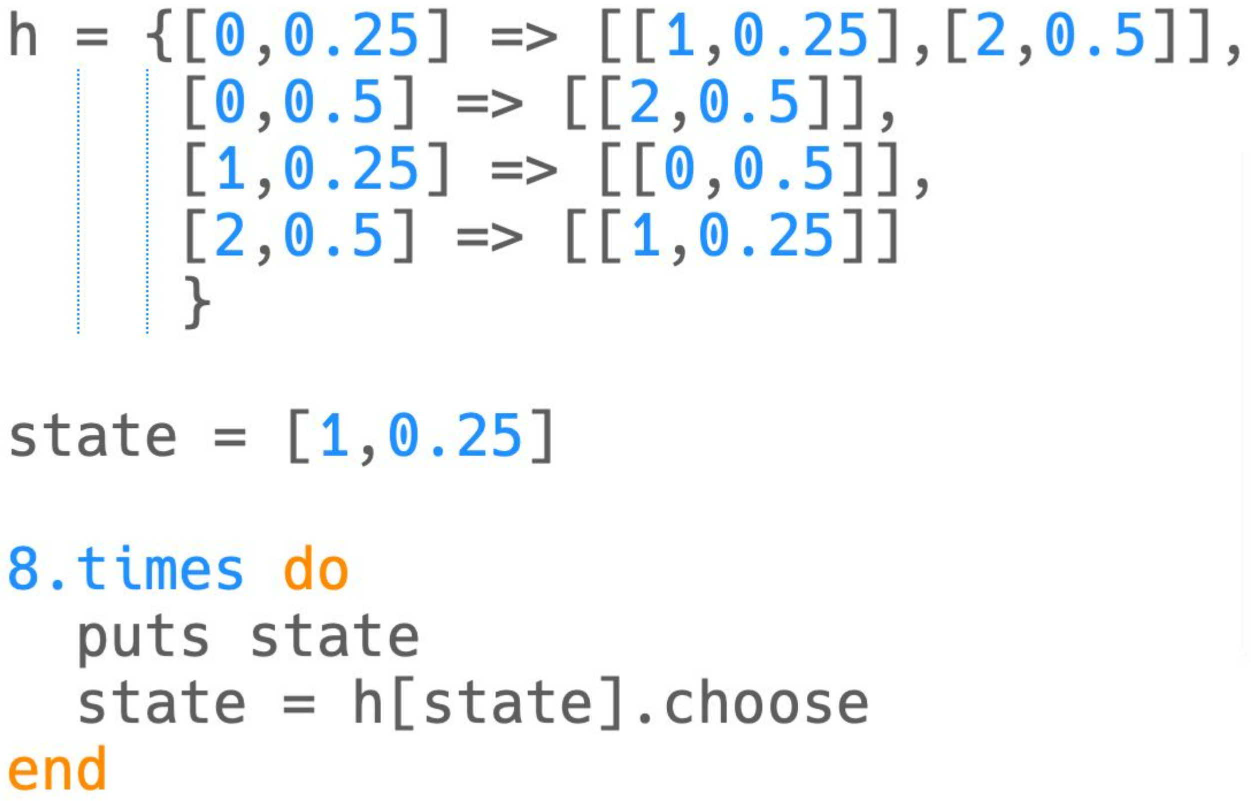
39, Position: 1

Markov Pitch, Rhythm, Note Sets:



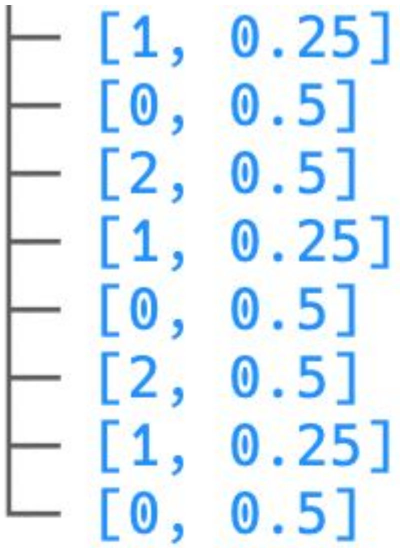
$$S = \{(\text{C\#4, E4, A4})♩,\ \text{F4}♪,\ \text{R}♩\}$$

Multiple viewpoints models



```ruby
h = {[0,0.25] => [[1,0.25],[2,0.5]],
     [0,0.5] => [[2,0.5]],
     [1,0.25] => [[0,0.5]],
     [2,0.5] => [[1,0.25]]
     }

state = [1,0.25]

8.times do
  puts state
  state = h[state].choose
end
```

Output



```
├─ [1, 0.25]
├─ [0, 0.5]
├─ [2, 0.5]
├─ [1, 0.25]
├─ [0, 0.5]
├─ [2, 0.5]
├─ [1, 0.25]
└─ [0, 0.5]
```

Exercises (for ideas, look in Sonic PI, Google, ChatGPT)

Practice generating (composing!):

a melody from a chord sequence

a chord sequence using a markov chain

a melody using the markov chain chord sequence

accompaniment patterns for the melody

melodic variations on a generated main theme (theme and variations)

    return to the main theme after a few variations