A Simple Phrase-Structure-Based Music Generator Implemented in Processing

Adam Meyers
New York University
June, 2008
Outline

- Overview
- How to Change the Grammar
- Understanding the Code
- Suggested Code Revisions
- Summary
Overview of the Program

• Setup
  – Sets up all variables including the grammar
    • Sample Grammar is based on “Mary Had a Little Lamb”
    – Uses Grammar to Generate Sequence of “terminals”
    – Loads this sequence into an array of strings
      • The default value is the string “Done”

• Draw
  – Unless the “terminal” == “Done”
    • Randomly generate MIDI + Duration pair from the next “terminal” in the array, using the Grammar
    • Plays that MIDI + Duration pair
Understanding the Grammar

• Rules are in Backus–Naur Form (BNF)
  – Nonterminal → possibility1 OR possibility2 OR ...

• Uses arrays and objects
  – Objects are like arrays, but slots are named & unordered
    • Array: NotePair[0], NotePair[1]
    • Objects: NotePair.freq, NotePair.relTime
    • Slots are MIDI number & length of note (assumed to be floats)
      – Note class == MIDI%12
      – 0=C, 1=C#, 2=D, 3=Eb, 4=E, 5=F, 6=F#, 7=G, 8=Ab, 9=A, 10=Bb, 11=B
      – 60 = C4
  – Object slots need not be the same type
    • TerminalRuleset.NonTerminal – symbol on left side (string)
    • TerminalRuleset.value – an array of NotePair (ending in “Done”)
    • TerminalRuleSet.numberOfVals – int (number of right hand sides)
Setting the Grammar

• Initialize in setup
  – numberOfTerminals, numberOfNonTerminals
  – Build pieces of grammar and assemble it (bottom up)

• Use current grammar as a model to build grammar
  – Grammar → nonTerminalRuleSet + TerminalRuleSet
  – TerminalRuleSet → TerminalRule1 TerminalRule2, etc.
  – TerminalRule → NonTerminal + array of NotePair
  – etc,
Terminal Rules

• Terminal Rules map NonTerminal to NotePairs
• An extended system could allow NonTerminals map to
  – Rests, Snippets, NonMidi Frequencies, etc.
• Functions would be sensitive to NonTerminal Strings
  – Different Strings (Rest, Snippet, etc.) would have different
    playback rules
NonTerminal Rules

• Current Grammar is somewhat nonsensical
• But you could use phrase structure rules to
  – Put pieces in ABA form
  – Make sonatas, rondos, etc.
• Note that there are shades of gray between a metacomposition and a composition
FillTerminalNodeArray called in setup

- Starts with StartSymbol (set to “Song”)
  - Recursively expands all phrase structure rules
  - “recordTerminal” inserts each “terminal” strings into terminalNodeArray
  - These “terminals” are sort of “nonterminals”
    - Except terminal rules always map to a single symbol
    - The “terminal” strings abstract away from possible different types of terminal nodes (notePair versus snippet)
Draw Function

- Calls `playSymbol` once per call until the next item in `terminalNodeArray` is "Done"
- `playSymbol` calls `playTerminal`, which calls `playNotePair`
- Other types of Terminals could be played differently
Other Ideas

- Alternatives to current signal (a triangle wave)
  - Multiple signals (noise, chords, etc.)
  - Different signals (sounds, musical instruments, etc.)
- Interaction with graphics
- Rules could be invoked with weighted probabilities
- Other elements could be made random
  - Measure $\rightarrow$ random 4 quarter notes
  - Measure $\rightarrow$ random 4 quarter notes with constraint
- Rules can be generated from other rules via transformations
  - For example, given a rule $X \rightarrow AB$, you could add the rule $X \rightarrow ABBA$ (the right hand side and its mirror image)
Summary

• An advantage of any formalization like this
  – one can quickly plug in different sets of instructions

• The particular formalization used was phrase structure rules in BNF

• Objects and Arrays are used to represent the rules and record the notes to be played

• Notes (terminals) are played one at a time by the Draw function (a loop)