Chapter 9
Structures

Most Important Information

Structures are a subclass of collections that allow customized ordering of the execution of their morphs. This ordering, or behavior, is controlled by a user controlled function which is set in the structure using **PUT.BEHAVIOR:** (as in collections). In addition, structures maintain a *tendency grid* that can contain data used by a behavior to determine the next morph to execute. Individual cells within the tendency grid can be edited using **PUT.TENDENCY:** .

Structures

Structures are a more powerful subclass of collections. Structures differ from collections in that they contain a square grid of *link tendencies*. Each component morph of a structure has a set of tendencies linking it to all other component morphs of that structure, typically used for determining likelihood of going from that component morph to any other. However, these numbers may be used by behaviors to perform a wide variety of other functions, and of course may be accessed by any other intelligent morph (productions, actions).

See the file HP:DEMO_STRUCTURE

A cosmological analogy is that the weights of collections (inherited by structures, productions, and so on) are the "mass" of a morph, and these link tendencies are a sort of “specific attraction”. That is, you can specify using weights the absolute attractiveness of a morph, and with link tendencies the specific inclination of one morph towards another morph.

A cognitive analogy is that weights represent "strengths" or "activation levels" of mental or proprioceptive units, like neurological processing entities or memory engrams, and link tendencies represent "strengths of associations" or "strengths of connections" among such units.

In a typical behavior of a structure these two values may be used in combination to determine the musical behavior of a large hierarchy.

In the case of structures, we use the word *source* to indicate the morph which the tendency is from, and *target* to indicate the morph which the tendency is to.

Markov Chains

In a simple transition matrix, the probability of executing a given child is found by reading the row of the one that last played. The probability is the value in a given column divided by the sum of all the values in that row.

```
P(i>j) is the probability that j follows i
T(i>j) is the matrix value at row i, column j
P(i>j) = T(i>j) / SUM( T(i>k) ) for k = 0..n-1
```

If all but one column has zeros then that column will always be next. For example, here is a matrix where the order of execution would be 0,2,1,3,0,2,1,3 and so on:

```
     0   1   2   3
    ---------------
0    0   0  10   0   ( 2 follows 0 )
1    0   0   0  10   ( 3 follows 1 )
2    0  10   0   0
3   10   0   0   0
```

Here is an example of a more complex chain:

```
     0   1   2   3
    ---------------
0   0   0  10   0   ( 2 always follows 0 )
1   4   4   4   4   ( any could follow 1 )
2   0  10   0   5   ( usually 1 but sometimes 3)
3   0   0   0  10   ( only 3 follows 3!)
```

Structures add another twist to this process. Each child morph has a *weight* that is normally set to 1. The probability of execution is actually the product of the matrix value and the weight for each morph divided by the sum of those products. Thus the execution of a child can be made twice as likely by setting its weight to 2.

```
W(j) is the weight of child j
for k = 0..n-1
P(i>j) = T(i>j)*W(j) / SUM(T(i>k)*W(k))
```

These *Markov Chains* are used for high level decisions, eg. which major section of a piece to play next. A simple Markov Class is available that can be used for low level decisions, such as what note to play next. The Markov tables described here are *first order*. They only consider one previous state. Higher order tables can be created that consider several previous states. A *second order* table would be NxNxN and could be used to determine the next note from the TWO previous notes.

NY-composer/guitarist Nick Didkovsky has used Markov Chains in real time performance in a piece called *Meta Music/Meta Text* for the *Downtown Ensemble*. He uses HMSL to analyse an Amiga SMUS file containing compositional excerpts from each member of the ensemble, and generates material from this analysis. Nick is a very active HMSL composer in New York who has also done a lot of work with phoneme processing, Lisp to HMSL interfaces, and HMSL/Gamelan pieces. He has also generated HMSL-based algorithmic compositions for his band *Dr. Nerve*.

Structure Methods

Method	Stack diagram
}STUFF:	(0 morph-1 morph-2 morph-3 ... morph-n -- ,) (does NEW: and adds morphs to structure)
ADD:	(addr-morph -- , add to collection)
CHAIN:	(index -- index' , Markov chain)
DEFAULT:	(-- , set defaults in the structure)
EXTEND:	(n -- , extends structure and NEW:s grid)
FILL.TENDENCIES:	(value -- , set all tendencies T[s->t] to value)
GET:	(index -- morph)
GET.LAST:	(-- lastchoice , return last one chosen)
GET.TENDENCY:	(s t -- T[s->t])
GET.TGRID:	(-- tendency-grid)
PRINT:	(-- , print structure)
PUT:	(morph index --)
PUT.LAST:	(lastchoice -- , set starting point for chain)
PUT.REPEAT:	(n -- , set repeat count)
PUT.TENDENCY:	(T[s->t] s t --)
RESET:	(-- , reset control flags and pointers)
SCAN.ROW:	(value elmnt# -- t , look for bucket }
SUM.ROW:	(elmnt# -- sum , sum of tendencies times weights)

Table 9-1. Structure Methods

OB.STRUCTURE subclass of OB.COLLECTION

Several of the following methods are redefined for structures from the parent class definition (OB.COLLECTION), and several of them are new to the Structure class. "s" and "t" stand for source and target (morph) respectively.

CHAIN: (index -- index' , Markov chain)

Select next item in chain based on weights and tendencies of components.

EXTEND: (n -- , extends structure and NEW:s grid)

The square grid associated with the structure cannot be extended so it is reallocated and is cleared.

FILL.TENDENCIES: (value -- , set all tendencies T[s->t] to value)

Tendency grids are initialized to 0 when a structure is defined.

GET.TENDENCY: (s t -- T[s->t])

Returns the tendency from morph s ("source") to morph t ("target"). Can be used by a behavior. s and t are the indices of morphs in the structure.

GET.TGRID: (-- address-of-TENDENCY-GRID)

This is mostly used for advanced applications where large scale transformations are wanted on the tendency grid. This grid is of class OB.SHAPE so any of that class's methods could be used directly on the grid.

NEW: (number_of_collections -- , allocate room)

Allocates room for the given number of component morphs. Uses the NEW: of morphs. Also allocates a square array to contain the tendencies.

PRINT: (-- , prints)

Prints repeat-count, weight, component morphs, and the tendencies.

PUT.TENDENCY: (T[s->t] s t --)

Defines the tendency from morph s to morph t (T[s->t]). Basic method for editing a cell in the structure's tendency grid.

GET.LAST: (-- lastchoice , return last one chosen)

PUT.LAST: (lastchoice -- , set starting point for chain)

SUM.ROW: (index -- sum , sum of tendencies * weights)

Returns sum of products of the weights of component morphs and the tendencies toward those morphs for a given row. Primarily for *internal* use.

SCAN.ROW: (value index -- index' , look for bucket)

Structure Internals

Below we give the definition of **CHAIN:** and **CHOOSE:** because they demonstrate the use of **SUM.ROW:** and **SCAN.ROW:** .

```
:M CHAIN:  ( index -- index' , Markov chain )
    dup sum.row: self
    choose
    swap scan.row: self
    dup iv=> iv-st-last
;M

:M CHOOSE: ( -- index , next based on last one )
    iv-st-last chain: self
```

```
;M
```

There is a predefined behavior that uses these methods. You can use it for Markov Chaining. It is the default behavior for OB.STRUCTURE.

```
: STR.BHV.MARKOV ( structure -- index 1 )
   choose: [] 1
;
```