Project Description

Description of a research/programming project for AI2.

Jacob Peck

CSC 466

Professor Craig Graci

Spring 2011

Artificial Life Music – Algorithmic melody Composition with coupled Lindenmayer Systems and Cellular Automata

Algorithmic composition – the process of composing music with the assistance of well-defined processes – can benefit from certain subsets of artificial life. One of these, Lindenmayer Systems (henceforth L-systems) are systems that model organic growth. They are a variation of Chomsky's context free grammars, where the set of terminal symbols is the same as the set of non-terminal symbols, there is typically only one choice for each rewrite, and such rewrites happen as frequently as is desired. For an example, take the alphabet `{A, B}`, with the rules `{A->AB, B->A}`. Given the seed of "`A`", one sees a progression from `A` to `AB`, to `ABA`, to `ABAAB`, to `ABAABABA`, etc. As you can see, these are very self-similar, yet non-symmetrical, making them rather suitable for melody or rhythm generation.

Cellular Automata, on the other hand, are simulations of various natural phenomena. Perhaps the most famous example, Conway's Game of Life, simulates bacterium growth. In a CA system, the state of each cell in the next generation is determined by the states of its neighboring cells in the current generation. The rules for Conway's Life, for example, state that a live cell remains alive in the next generation if has exactly 2 or 3 live neighbors in the current generation, otherwise it dies (either of isolation or overpopulation). A dead cell comes alive in the next generation if it has exactly 3 live neighbors in the current generation, simulating reproduction. These are very useful as selection grids, if each cell is weighted according to some value or a history of its previous states.

This project will consist of a cellular automata selection grid (with weighted values based upon individual cell history) selecting various L-systems, one per cell. The sheer amount of L-systems this would require would be enormous without some doubling or tripling of assignments, and very tedious to input by hand. So to that end, I intend to allow the program to simply take in one single L-system, and mutate it as it will. There will be a fitness function to determine the fitness of these mutations, and then a user-definable amount of generations to allow these systems to settle into something fit for composition. After this takes place, the user is asked to provide a seed for the composition. After this value is taken in, the program will interactively step through generations, producing a piece of a desired length, the end result of which will (hopefully) be a MIDI file.

A potential demo is as follows (user interactions are in **bold**):

```
[]>(alm)

Welcome to ALM!
Please input an alphabet to use as a list of symbols:

(C D E F G A B)

Please input a rewrite rule for the symbol C:
```

(D F)

Please input a rewrite rule for the symbol D:

(A)

<…snip…>

Please input a seed as a single symbol:

F

Alright, we're ready to get started. Just to review, here is the information you have provided me:
Alphabet: C D E F G A B
Rules:
C > D F
D > A
E > G A D
F > B B
G > B D
A > C F C
B > E G E
Seed: F

Mutating system…
Evolving system…
Assigning systems to cells…

<…snip: list all of the sets of rules, assigning them numbers…>

Here is the world:

```
Weights:  0  1  0  1  0  0  0  1  1  0  1
States:   .  X  .  X  .  .  .  X  X  .  X
Rulesets: 0  1  2  3  4  5  6  7  8  9  A  (hex values to save space)
```

Generation 0: F
Iterate? **t**

Iterating the world…
Here is the world:

```
Weights:  0  2  0  1  0  1  1  2  1  0  1
States:   .  X  .  .  .  X  X  X  .  .  .
Rulesets: 0  1  2  3  4  5  6  7  8  9  A
```

Choosing rule 7 (weight 2)…

Rewriting…

```
Generation 1: B D
Iterate? t

Iterating the world…
Here is the world:

Weights:  0  3  1  2  0  1  2  2  1  0  1
States:   .  X  X  X  .  .  X  .  .  .  .
Rulesets: 0  1  2  3  4  5  6  7  8  9  A

Choosing rule 1 (weight 3)…

Rewriting…

Generation 2: E G C B
Iterate? nil

Enter a file name:
example1.midi

Saving file…

Bye!

[]>(bye)
```

And because sometimes, pictures are better than words, here is a flowchart: