PART FOUR

CHAPTER 8 — *Phrase Garden*

8.1. Introduction

The note and numeric matrices used in the compositional style detailed in the previous chapter both contain a significant element of inherent growth that may be viewed as having similarities to organic or biological growth. The note matrices, through the use of transposition, present new notes in each series transposed from the original, much as genetic variations in biological evolution present new biological forms whilst retaining a dominant genetic blueprint. This genetic blueprint may be viewed as analogous to the retention of an interval pattern within the different transpositions of an original note series. As the transpositions unfold, the initial five or six notes of the original note series 'mutate' to cover the total chromatic. The numeric series matrices are developed with an additive process that is similar to that of the Fibonacci series: consecutive numerics from the original numeric series are added to one another to produce new numerics for a second series. These additions in effect represent a growth process that, like the Fibonacci series, moves toward infinity.

Bearing in mind the inherent growth elements of the note and numeric matrices, along with the techniques used in the compositional style, a science- or mathematics-based model was sought for use in the development of a program suited to automation of techniques used in the compositional style. This program would primarily provide a real-time environment for the development of musical materials of the compositional style, a real-time environment allowing the auditioning of generated musical materials, and a monitoring of the growth of those materials. In the search for a model for the program, numerous areas were explored, including developments in grammar with Chomsky's Universal Grammars (Cook 1988); however, the most viable model was found in the field of Artificial Life. The first part of this chapter provides a brief overview of the field of Artificial Life, focusing on a small portion of research undertaken in the field, and including literature citations from varied references in the field. Following this, Artificial Life systems described in the overview are shown to be incompatible with the compositional style, and a further Artificial Life system that is compatible with the compositional style is discussed.

The second part of the chapter details the *Phrase Garden* program, showing how an Artificial Life model for the program is implemented and the manner in which the program relates to the compositional style.

8.2. Artificial Life

8.2.1. Introduction

Artificial Life, or A-Life, is devoted to:

the creation and study of life-like organisms and systems built by humans. The stuff of this life is non-organic matter, and its essence is information: computers are the kilns from which these new organisms emerge. Just as medical scientists have managed to tinker with life's organisms *in vitro*, the biologists and computer scientists of A-Life hope to create life *in silico* (Levy 1992, p.5).

A leading figure in the field of A-Life is Christopher Langton who describes it as:

the study of man-made systems that exhibit behaviours characteristic of natural living systems. It compliments the traditional biological sciences concerned with the *analysis* of living organisms by attempting to *synthesise* life-like behaviours within computers and other artificial media (Langton 1989, p.1).

Aside from pre-20th century mechanical automatons, the history of A-Life essentially began in the early 20th century with the abstract formulation of procedure through the formal application of logic to the mechanical processes of arithmetic, primarily in the work of mathematicians such as Turing and Gödel. These mathematicians:

formalised the notion of a logical sequence of steps, leading to the realisation that the essence of a mechanical process — the "thing" responsible for its dynamic behaviour — is not a thing at all but an abstract control structure, or "program" — a sequence of simple actions selected from a finite repertoire (Langton 1987, p.10).

This formalisation of logical sequences led to a shift from a focus on the mechanics of life toward a focus on the logic of life. In the 1940s the Hungarian mathematician John von Neumann, building on work carried out by Turing, developed through "thought experiments" his automata theory in which he proved that machine self-replication could be achieved. He determined that any such method of self-replication:

must make use of the information contained in the description of the machine in two fundamentally different ways:

- INTERPRETED, as instructions to be executed in the construction of the offspring.
- UNINTERPRETED, as passive data to be duplicated to form the description given to the offspring

Of course, when Watson and Crick unravelled the mystery of DNA, they discovered that the information contained therein was used in precisely these two ways in the processes of transcription/translation and replication (Langton 1987, p.15).

8.2.2. Cellular automata

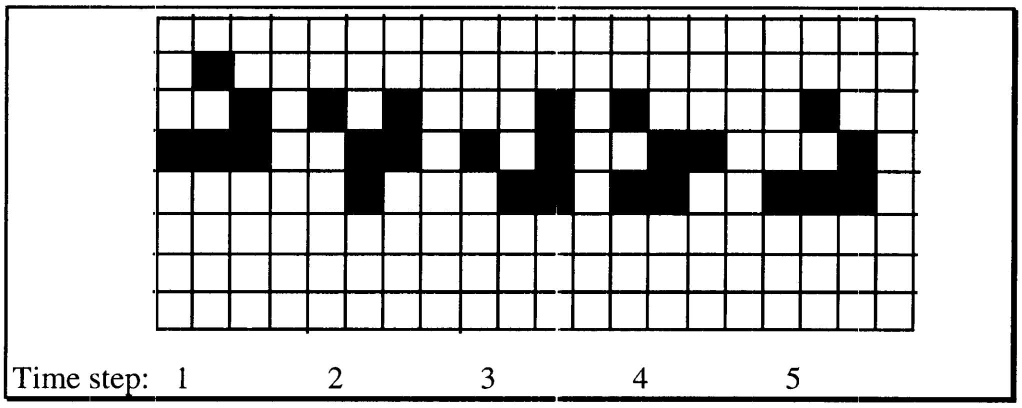
The automaton von Neumann designed was 'a two-dimensional cellular automaton, (CA) whose squares could assume some 29 different states, one of them being earmarked as the "empty" state' (Sigmund 1993, p.22). The automaton environment was an horizonless grid of square cells, each cell subject to a rule table according to which of the 29 states that cell was in. The automaton itself, in a square shape with a tail, occupied some 200 000 of these cells:

The configuration of the grid would change as discrete time steps ticked off. Every cell would hold information that would be known as its state, and at each time step it would look to the cells around it and consult the rule table to determine its state in the next tick (Levy 1992, p.43).

As time steps tick over, the automaton extends outward into empty cells of the environment, eventually forming a replica of itself.

Following von Neumann's death, work on CAs continued, notably in the work of Cambridge University mathematician John Conway. In the late 1960s Conway developed his game of "Life", in which the number of possible cell states in a CA was reduced to two, either filled or empty, and a simple rule set: if a cell is empty, it remains empty at the next time step unless three neighbouring cells are occupied in which case the cell will be filled at the next time step. If a cell is filled it remains filled at the next time step whenever two or three of its neighbours are filled. If only one neighbouring cell is filled the cell will become empty at the next time step. Figure 8.1 shows the self-propagation of a pattern labelled by Conway as a 'glider'. After four time steps the pattern is reproduced but occupies a space one cell lower in the environment.

Figure 8.1 John Conway, "Life", glider self-propagation



The glider is one instance of the general class of propagating structures in CA. These propagating information structures are effectively simple machines — *virtual machines* — which crawl around the lattice like so many ants, interacting with other such machines and with the more passive structures in the array. Their behaviour is reminiscent of the actions of biomolecules — especially enzymes — in their capacity for recognising and altering other structures they encounter in their wanderings, including other *propagating* structures (Langton 1987, p.20).

Further work has been undertaken with CAs, notably in the work of Langton who implemented self-reproducing CAs based on von Neumann's theory on a computer, and also Stephen Wolfram who pioneered 'four qualitative classes of CA behaviour (referred to as Wolfram classes), with analogs in the fields of dynamical systems' (Sipper 1996a). Aside from being used as a modelling framework for A-Life simulations (Gutowitz 1995), CAs have had numerous real-life applications, for example in computer image processing and pathology (Levy 1992, p.73), and 'have been applied to the study of general phenomenological aspects of the world, including communication, computation, construction, growth, reproduction, competition and evolution' (Sipper 1996a). CAs have been implemented in a music program called *Cellular Automata Music* by Dale Millen. This program uses various types of cellular automata including Conway's 'Life' in which cells of a lattice are mapped to MIDI note values (Millen 1994).

8.2.3. Evolutionary algorithms

An important area of A-Life research is concerned with evolutionary algorithms. Evolutionary algorithms are iterative algorithms, each iteration being referred to as a "generation":

> The basic evolutionary algorithm begins with a population of randomly chosen individuals. In each generation, the individuals "compete" among themselves to solve a posed problem. Individuals which perform relatively well are likely to "survive" into the next generation. Those surviving to the next generation may be subject to small, random modifications. If the algorithm is correctly set up, and the problem is indeed one subject to solution in this manner, then as the iteration proceeds the population will contain solutions of increasing quality (Gutowitz 1995).

The most widely used evolutionary algorithm is the *genetic algorithm* (GA) pioneered by John Holland in the mid-1970s. The GA:

> hewed to one of John von Neumann's important lessons: in both biological and artificial systems, the information central to the organism had to be regarded in two manners — both as genetic information to be duplicated and as instructions to be executed. (Levy 1992, p.161).

The information central to an organism is known as the *genotype*. The execution of this genetic information results in the physical organism, known as the *phenotype.*

To apply the genetic algorithm:

> one must define
> 1) a mapping of the set of parameter values into the set (0-1) bit strings, and
> 2) a mapping from bit strings into the reals, the so-called fitness function. A set of randomly-chosen bit strings constitutes the initial population. In the genetic algorithm, a cycle is repeated in which
>
> 1. The fitness of each individual in the population is evaluated.
> 2. Copies of individuals are made in proportion to their fitness.
> 3. Individuals in the population of copies are altered by mutations and recombinations between pairs of individuals (Gutowitz 1995).

The mutations in Step 3 of this cycle usually involve "bit-flipping" wherein a small percentage of bits within bit strings are inverted from 1 to 0 or from 0 to 1. Recombinations are a result of a "crossover" between two bit strings, for example the first half of a bit-string *x* is replaced with the second half of bit string *y* and vice-versa.

An experiment using the genetic algorithm was carried out with artificial "ants", constructed by University of California researcher David Jefferson. Jefferson's ants were strings of 450 binary bits that, like a cell in a cellular automaton, were subject to rules according to a current state within a sequence of 200 time-steps. The ants were to follow a trail of 89 steps on a toroidal grid of 32 by 32 squares:

> Inspired by actual pheromone trails used by ants to aid each other in foraging, the trail twisted and turned, and became increasingly difficult to follow as it progressed. It suffered gaps at several points, and, by the last segment of the trail, there were more missing squares than actual "scented" ones (Levy 1992, p 165).

After determining whether the cell directly in front of the ant was belonging to the trail or not (on or off) an ant would move forward a square, turn left or right without moving, or do nothing.

An initial population of 65 536 ants with randomly selected genotypes could manage no more than four steps in the trail, however the most successful 10% of these were selected for reproduction and copied to return the population to its former size. After 20 generations a small percentage of ants could traverse more than 60 steps while the average ant could traverse 30. At 70 generations the average ant had evolved to traverse the full 89 steps.

Aside from GAs, examples of further evolutionary algorithms include genetic programming in which LISP expressions form the genotype, allowing evolution to occur with computer programs as opposed to bit-strings, and Lindenmayer systems (L-systems). The simplest L-systems use context-free rule sets such as those used in *Symbolic Composer's* `gen-rewrite` algorithm. The rule sets form the genotype within an L-system, the output is the phenotype. More complex L-systems incorporate left and right branching of symbols and context-sensitive rule sets in which there may be more than one symbol on the left side of the rule. Branching and context-sensitive rule sets allow extensive modelling of the growth and development of biological organisms with L-systems. Evolutionary algorithms have been:

> successfully applied to numerous problems from different domains, including optimisation, automatic programming, machine learning, economics, operations research, ecology, population genetics, studies of evolution and learning, and social systems (Sipper 1996b).

The genetic algorithm has been applied to music in various areas. John Biles program *GenJam* uses a genetic algorithm for generating four-measure jazz phrases (Biles 1994), while Gary Lee Nelson has used genetic algorithms to produce rhythmic materials (Nelson 1993).

8.2.4. *Tierra*

Research in the field of A-Life has produced systems in which computer "organisms" may either serve to model actual biological systems, or may be used to study broader facets of biological life. In the former, 'populations of data structures in a computer program are used to represent populations of biological entities (predators and prey, ants, cells, and the like)' (Gutowitz 1995). An example is a system by Craig Reynolds in which autonomous but interacting objects called "Boids" are used to model flocking behaviour in birds (Langton 1989, pp.30-1). In A-Life systems used to study broader facets of biological life, populations of data structures do 'not explicitly represent any living organism or process, but rather obey artificial laws abstractly related to the natural laws governing living processes' (Gutowitz 1995).

An example of the latter is Tom Ray's *Tierra*, an open-ended evolutionary system in which organisms are machine language computer programs or instruction sets that are not coded for any problem solving task, but simply evolve (with minor random mutations) in their computer environment. An original organism or "ancestor" will usually contain 80 instructions, the number of instructions Ray used in his initial work with the program. As *Tierra* runs, the machine language organisms reproduce and compete for computer processing time. Instruction sets eventually evolve that contain fewer instructions and take less and less processing time, making such organisms more fit for survival in the *Tierra* environment. "Parasite" instruction sets also evolve that do not contain instructions for their own self-replication and are, as such, considerably shorter. These parasite sets "borrow" instructions for self-replication from longer instructions sets and, as they are short and take a smaller amount of processing time, they are extremely fit for survival in the *Tierra* environment. Systems such as *Tierra*, with their ability to produce multitudes of generations of organisms in a short space of time, allow a study of broad facets of evolution such as evolutionary 'arms races' between parasites and hosts, studies that are not possible in biological life due to the slow rate of evolutionary change.

Research carried out in the field of A-Life is diverse and provides models for many facets of biological life. Such modelling also provides non-biological fields with computer-based systems and programming methodologies suited to development in those fields. Examples include the use of the genetic algorithm in fields as diverse as economics and ecology (Sipper 1996a), and in computer animation (Levy 1992, p.212), and the use of cellular automata in the 'study of general phenomenological aspects of the world, including communication, computation, construction, growth, reproduction, competition and evolution' (Sipper 1996b).

8.2.5. A-Life systems as models for the Compositional style

Cellular automata, genetic algorithms and Tom Ray's *Tierra* were all examined as possible models for a music program that automated techniques of the compositional style. Each model, however, was discarded due to incompatibilities between the model and the compositional style.

A simple example is in regard to pitch material produced by a cellular automata model. As in the *Cellular Automata Music* program of Dale Millen, cells in a CA lattice may be mapped to pitch. Using Conway's 'Life' as an example, the rules of the game require that multiple cells are active at any given time-step for active cells to remain active in the next time-step. If individual cells of the lattice are mapped to notes, at any given time-step vertical note aggregates must occur. In the compositional style, the primary focus with pitch is on the linear statement of notes from the note matrices. Vertical aggregates are intuitively formed from components of linear pitch material, and are primarily used to punctuate linear pitch material. The use of notes mapped to lattice cells in a CA model suggests a focus on vertical aggregates of notes that is thus incompatible with the compositional style.

Another example is provided by Tom Ray's *Tierra*. Individual instructions within *Tierra* instruction sets may be mapped to individual pitches, and the instruction sets may then provide musical phrases. When *Tierra* is run, long periods of stasis occur wherein a single instruction set, or minor variants of an instruction set, dominate the environment. If an instruction set represents a phrase of music, the same phrase would occur repeatedly. As previously discussed, such repetition is not desired in the compositional style. If the rate of evolution could be controlled or increased within the *Tierra* environment, the program could become a viable model for automating the techniques of the compositional style. However, a primary characteristic of *Tierra* is that it is an open-ended evolution system in which fitness for survival is contained within the program itself, and not determined by outside criteria, such as a problem to be solved by a genetic algorithm. This open-ended nature of *Tierra* allows the study of different areas of evolution such as the emergence of parasites, and evolutionary 'arms-races'. Alteration of the program to suit the compositional style would, in effect, defeat a primary purpose of *Tierra*, this being the production of instruction sets that provide an ability to study facets of open-ended evolution.

The non-open-ended nature of the genetic algorithm provides a more viable model for the automation of techniques in the compositional style. This is primarily due to an ability to control rates of evolution according to desired outcomes. As an example, an eight note series is mapped to bit strings with three bits per note. The choice of eight notes is convenient as, with three-bit strings, there are eight possible combinations of bits, 0 and 1 (i.e. 2^n where n is the number of bits in a string). A possible mapping to notes is shown at the top of Figure 8.2. Melodic phrases can be generated through the random generation of bit strings containing multiples of three bits. As shown in the second part of Figure 8.2, a phrase of four notes can be derived by combining four three-bit strings into a bit string of 12 bits.

Figure 8.2 Genetic algorithm, note mapping, output and breeding

```
Bit string to note mapping:
000 001 010 011 100 101 110 111
 C   D   E   F   G   A   B   C

Four-note phrases:
001 011 000 010          110 100 101 010
 D   F   C   E            B   G   A   E

Breeding with crossover method
D F C  E - Parent 1

B G A  E - Parent 2
   Δ
(Crossover point)

D F A E -  Child 1 (Desired outcome)

B G C E - Child 2
```

A primary use of the genetic algorithm lies in the ability it provides to search quickly through large numbers of possible solutions to a problem. As an example, a small random population consisting of two or more bit strings with 12 bits may be produced, these two strings representing two of the 4096 possible combinations of 12-bit strings. In a musical context, a phrase consisting of the notes D, F, A and E may be a desired outcome. If the two phrases in Figure 8.2 are considered as an initial population and compared with the desired outcome, then neither of the two matches the desired outcome. To match the desired outcome, the two phrases must either be subject to mutation or combined, using a crossover method, to produce new phrases.

As shown in the lower part of Figure 8.2, the use of a single crossover point will result in two 'child' bit strings, one of which matches the desired outcome. The two phrases in the random population of the example both have some resemblance to the desired outcome, the first half of the Parent 1 phrase matches, as does the second half of the Parent 2 phrase. With two of the notes in each phrase matching two of the four notes in the desired phrase, these two phrases can be classified as having a fitness value of 50%. If a phrase of C D A B was produced in the random population it would have a fitness value of 25%, with only one of the four notes matching the desired outcome. Phrases within a random population of two or more phrases may have fitness values of 0%, 25%, 50%, 75% or, as the desired outcome is one of the numerous possibilities, 100%. Using a pattern matching algorithm, phrases can be compared to the desired outcome, and those with fitness values less than 50% deleted from the population. In the 'breeding' phase, phrases in the initial random population that have high fitness values are selected to become parents.

If all phrases in an initial population present fitness values of 0% or 25% only, a mutation function can be used to provide a second generation that might contain phrases with higher fitness values. Mutation in genetic algorithms is generally achieved through minor bit-flipping, from 0 to 1 or vice-versa. The phrase of C D A B, with its 25% fitness value could be subject to mutation in which the first bit (1) of the bit string for a B note (1 1 0) is flipped to a 0, resulting in the bit string for an E note (0 1 0). The resulting phrase would then be C D A E, the A and E providing a fitness value of 50%.

The genetic algorithm presents a powerful tool for searching the possibilities of phrases available with notes from the compositional style note matrices. Possibilities with interval weightings of the compositional style may also be searched using the genetic algorithm. Interval weightings may be derived by assigning a fitness value to sequences of notes. If a weighting of the interval class 3 is desired, sequences of notes that produce that interval may be assigned a fitness value of 100%. In an initial random population there will be no weighting on the desired interval, however, using mutation and crossover to produce further generations, intervals with weightings other than interval class 3 gradually become less common. Such a process could be applied not only to two-note sequences, but also to longer note sequences and phrases. Similarly, rhythmic cells of the compositional style could be mapped to bit strings, allowing a search through the possibilities of various rhythmic cell combinations arising through crossover and mutation.

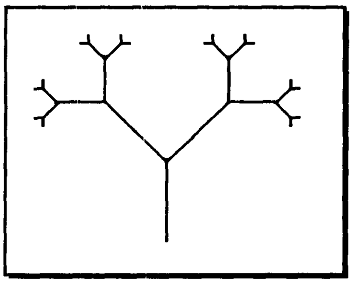
The gradual process of evolving desired outcomes is the primary reason for the dismissal of the genetic algorithm as a model for a real-time program in which techniques of the compositional style are automated. An example of the number of generations required to attain materials that closely represent desired outcomes is provided by John Biles in his discussion of his *GenJam* program. In the initial *GenJam* generations of jazz style phrases 'fitnesses are almost negative... Typically, at around the tenth generation, a "golden" generation occurs where almost all the new phrases sound reasonable' (Biles 1994). Whilst the genetic algorithm provides a useful tool for searching the possibilities of phrases, interval weightings and rhythmic combinations common in the compositional style, the number of generations that must be produced before the algorithm provides relevant musical materials similar to the compositional style are prohibitive in a real-time environment.

8.2.6. *Blind Watchmaker*

In 1987 Christopher Langton organised the first A-Life conference in Los Alamos, the conference bringing together 'computer scientists, biologists, physicists, anthropologists and other assorted "-ists", all of whom shared a common interest in the simulation and synthesis of living systems' (Langton 1989, p.xv). At this conference Richard Dawkins, an Oxford University zoologist, presented a paper on a computer program he had developed to illustrate concepts in his book *The Blind Watchmaker*, the book itself explaining how 'evolution, proceeding by subtle gradations, could achieve the dazzling order and complexity of contemporary life-forms' (Levy 1992, p.172). The Dawkins program, also called *Blind Watchmaker*, generates two-dimensional images on the computer screen, a genotype specified by the user controlling the way lines are drawn to form images.

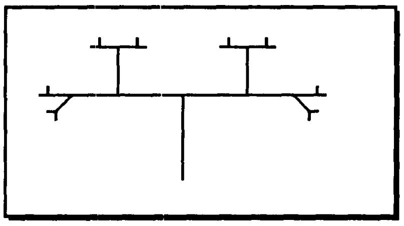
Dawkins' program is based on an artificial, archetypal, *embryology* in which numerical values in the program (genes) influence the development of two-dimensional drawings (phenotypes) (Dawkins 1989, p.202). Figure 8.3 shows the archetypal drawing or phenotype produced by Dawkins' artificial embryology. A recursive branching rule controls the number of branches a phenotype will have. In Figure 8.3 the number of branches, or depth of recursion, for the phenotype is four.

Figure 8.3 *Blind Watchmaker,* Archetypal phenotype



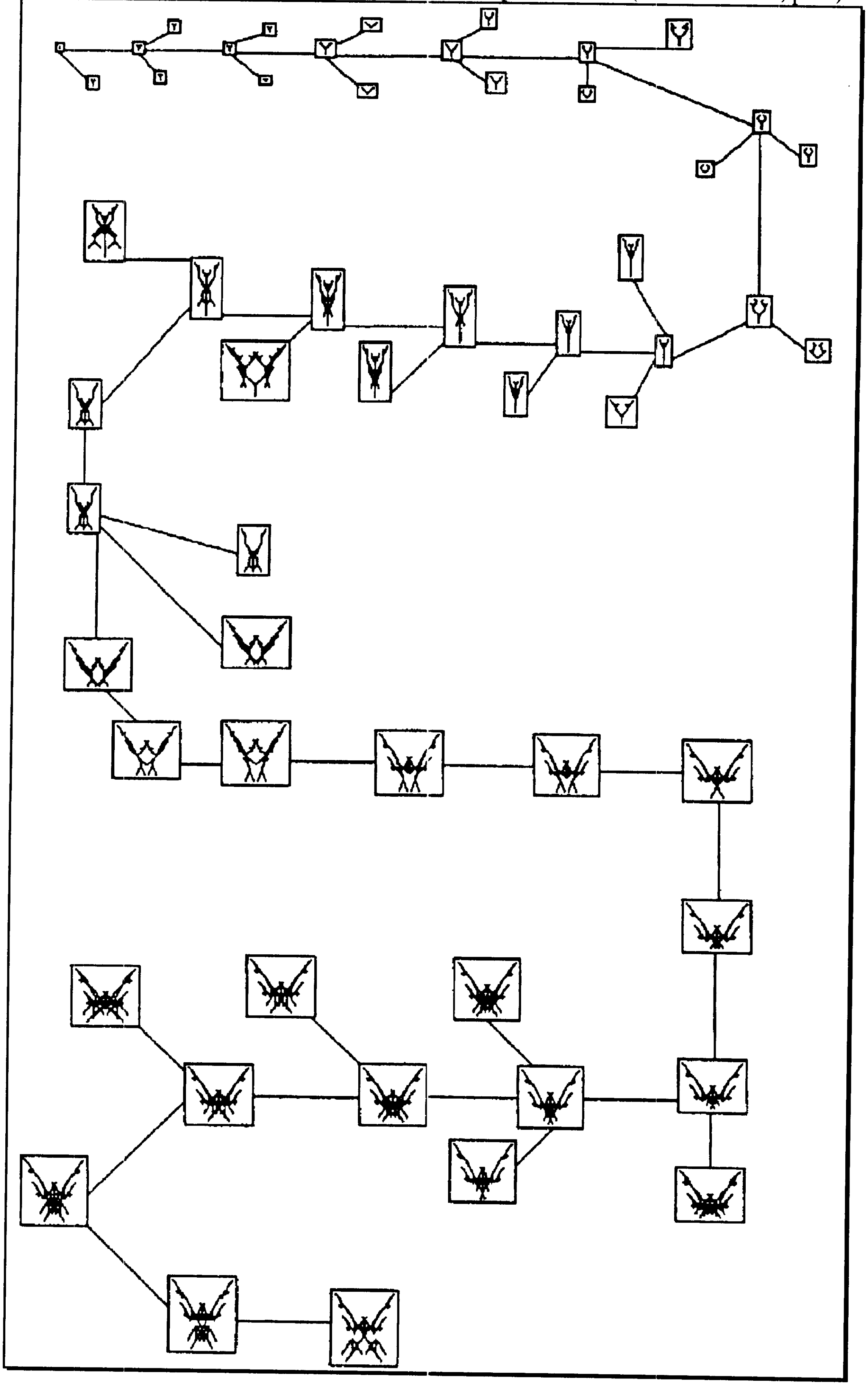
The depth of recursion of the branching rule, as a numerical value, is considered by Dawkins to be a gene that influences the growth of the phenotype, one of nine genes in the original *Blind Watchmaker* program. The remaining eight genes influence other parameters of the phenotype such as the angle of branching and the length of a branch. Figure 8.4 shows the influence of an alteration to 'gene 5' on the tree shown in Figure 8.3 (Dawkins 1986, p.54). For each gene, there is also a plus (+) and a minus (–) value that controls the direction of phenotype lines, either up or down. Such genetic alterations are considered by Dawkins to be a result of mutations in the genotype.

Figure 8.4 *Blind Watchmaker,* Archetypal phenotype with genetic mutation (gene 5+)



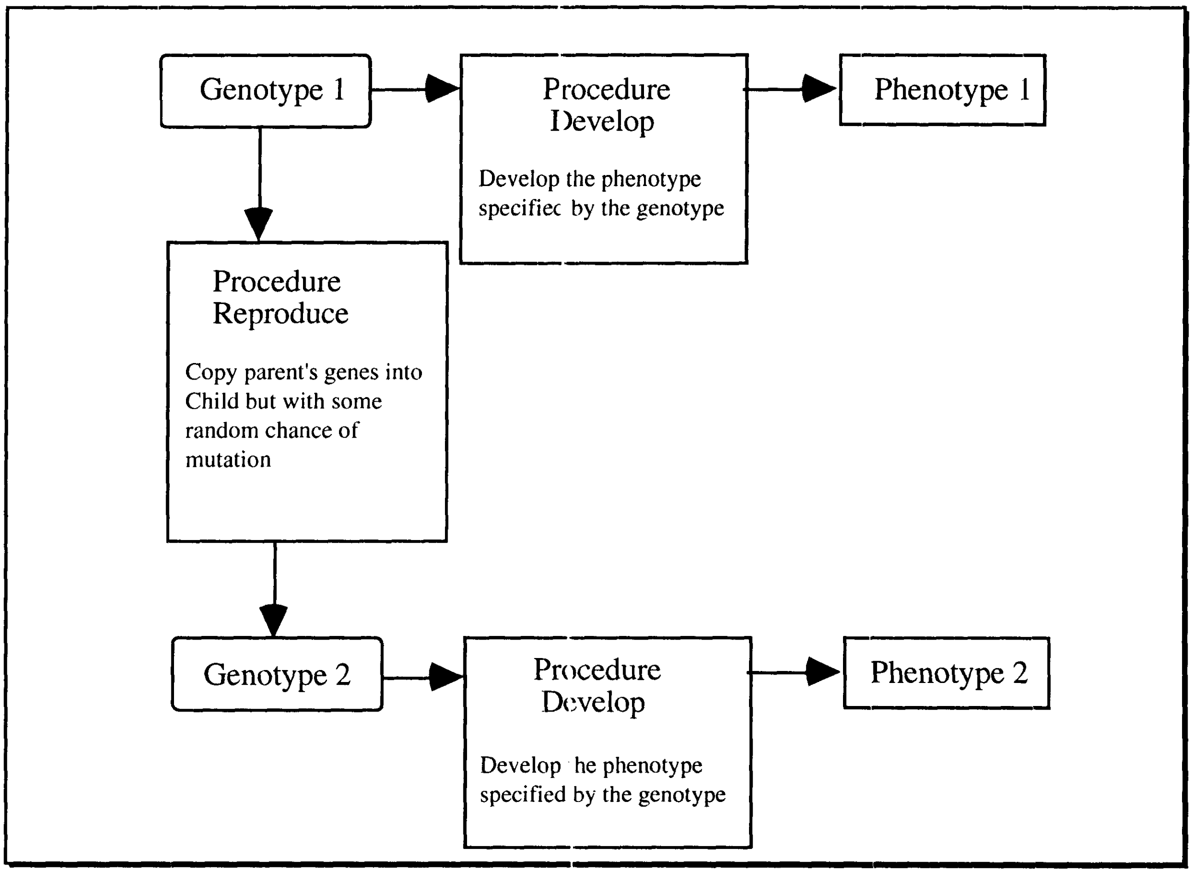
In the *Blind Watchmaker* program, the archetypal phenotype is used as the parent for a series of phenotypes (Dawkins calls the phenotypes *biomorphs*) that are drawn on the computer screen with the parent biomorph in the centre. Each biomorph drawn differs in respect to the parent biomorph by a mutation in one of its genes, for example +7 or –9. All 'child' biomorphs, like that in Figure 8.4, are a single mutational step away from the parent. The user then selects one of the child biomorphs on the screen to use as the parent of the next generation. Biomorphs in the ensuing generation are one step away from the parent and two steps away from the initial biomorph. The selecting agent in the evolution of biomorphs is the eye of the user of the program. Biomorphs are chosen to be the parent of a generation according to the requirements of the user. for example an insect-like shape can be achieved by consistently choosing biomorphs in each generation that have some resemblance to insects. Evolutionary selection in the program is thus an artificial selection as opposed to the natural selection of Darwinian evolution, and the selection process used by genetic algorithms in which artificial organisms survive according to their fitness in solving a posed problem. Figure 8.5, from Dawkins' book *The Blind Watchmaker*, shows the evolution of an insect-like biomorph through 29 generations, starting with a single pixel. The biomorphs along the central line are those chosen as the parents of each generation, while the biomorphs attached to the parent biomorphs are examples from each generation that were not chosen as parents.

Figure 8.5 *Blind Watchmaker,* Insect-like biomorph evolution (Dawkins 1986, p.58)



Like genetic algorithms, Dawkins' program follows the model of John von Neumann in which the genotype is both duplicated and used in the development of a phenotype. There 'must be replicators — entities capable, like DNA molecules, of self-replication' and 'genes must influence the development of a phenotype' (Dawkins 1989, p.202). Figure 8.6 is based on an example given by Dawkins (Dawkins 1989, p.202), and diagrammatically represents both the von Neumann model and data modules used in the Dawkins program. Arrows in the diagram represent the path of data within the program.

Figure 8.6 *Blind Watchmaker,* Genoptype/phenotype modules (Dawkins 1989, p.202)



Genotype 1
Procedure Develop
Develop the phenotype specifiec by the genotype
Phenotype 1
Procedure Reproduce
Copy parent's genes into Child but with some random chance of mutation
Genotype 2
Procedure Develop
Develop he phenotype specified by the genotype
Phenotype 2

In the *Blind Watchmaker* program, with its nine genes, and the possibility of both upward and downward mutation, there are 18 possible single-step mutations that can be derived from a parent biomorph in any one generation. These mutations exist in what is referred to as 'genetic space'. In a second generation this number expands to 324 possible biomorphs in the genetic space. These are the 18 children of each of the 18 biomorphs in the first generation (18 X 18). The *Blind Watchmaker* program allows the user to search through the genetic space of biomorphs to find shapes the user desires. Dawkins has evolved biomorphs resembling, for example, scorpions, bees, spiders, bats, frogs and birds. 'The forms could also resemble artefacts: Spitfire airplanes, lunar landers, letters of the alphabet' (Levy 1992, p.174).

In making an analogy between numerical values of drawn lines and genotypes, along with an analogy between biomorphs and phenotypes, Dawkins, in his *Blind Watchmaker*, has provided a model which can be used for a music program in which similar analogies may be made between musical intervals and genotypes, and sequences of notes, or phrases, and phenotypes. In relation to the compositional style, the evolutionary capacity of the *Blind Watchmaker* program, when transferred to a music program based on the same evolutionary principles, can be viewed as a viable medium for expressing the inherent growth of the note and numeric series matrices.

Unlike the genetic algorithm, in which a random population begins the evolutionary process, *Blind Watchmaker* begins evolution with an archetypal embryology. In a real-time music program based on *Blind Watchmaker*, a musical archetypal embryology may represent a set of rules for generating notes of the compositional style note matrices, rules for weightings on one or more intervals, or rules for generating rhythms based on the numeric matrices. As the composer has the ability to define the rules for the development of musical phenotypes, such an embryology, or rule set, will result in musical output that is immediately representative of the composer's desires. The child phenotypes that evolve from the parent phenotype will then present the composer with possibilities from the genetic space of musical phenotypes related directly to the composer's input. As shown in the excerpt analyses in Chapter Seven, notes within a phrase are chosen intuitively from the note matrices. With an ability to search quickly through the genetic space of musical phenotypes based on an embryology, the composer need not rely as heavily on intuition and may choose from any number of phrases produced by the music program. This embryological approach to the generation of musical materials is advantageous over an approach with genetic algorithms in that desired musical results are immediate with an embryological approach, and only gradual with a genetic algorithm approach. In addition, the genetic algorithm usually utilises a pre-defined analytical process or function for determining the fitness of a phenotype for reproduction. In the Dawkins model, fitnesses of individual phenotypes are determined by the user, thus allowing selection based directly on human aesthetic criteria.

Dawkins' embryological approach in evolving two dimensional computer images has been adopted by artists involved with computer graphics to generate three dimensional images and animations, the Dawkins approach being advantageous over genetic algorithms as 'it is difficult to automatically measure the aesthetic visual success of simulated objects or images' (Sims 1991, p.320). Sims, an American artist, has based work on the Dawkins model and shown how 'evolutionary techniques of variation and selection can be used to create complex simulated structures, textures and motions for use in computer graphics and animation' (Sims 1991, p.319). English artist William Latham has also based work on the Dawkins model and developed from it his *Form Synth*, 'an interactive 3-D modelling system for sculptors. The rule-based program is designed on the basis of the requirements of artists and relates to the evolutionary development of art and the 3-D modelling process used by sculptors' (Latham 1991, p.82).

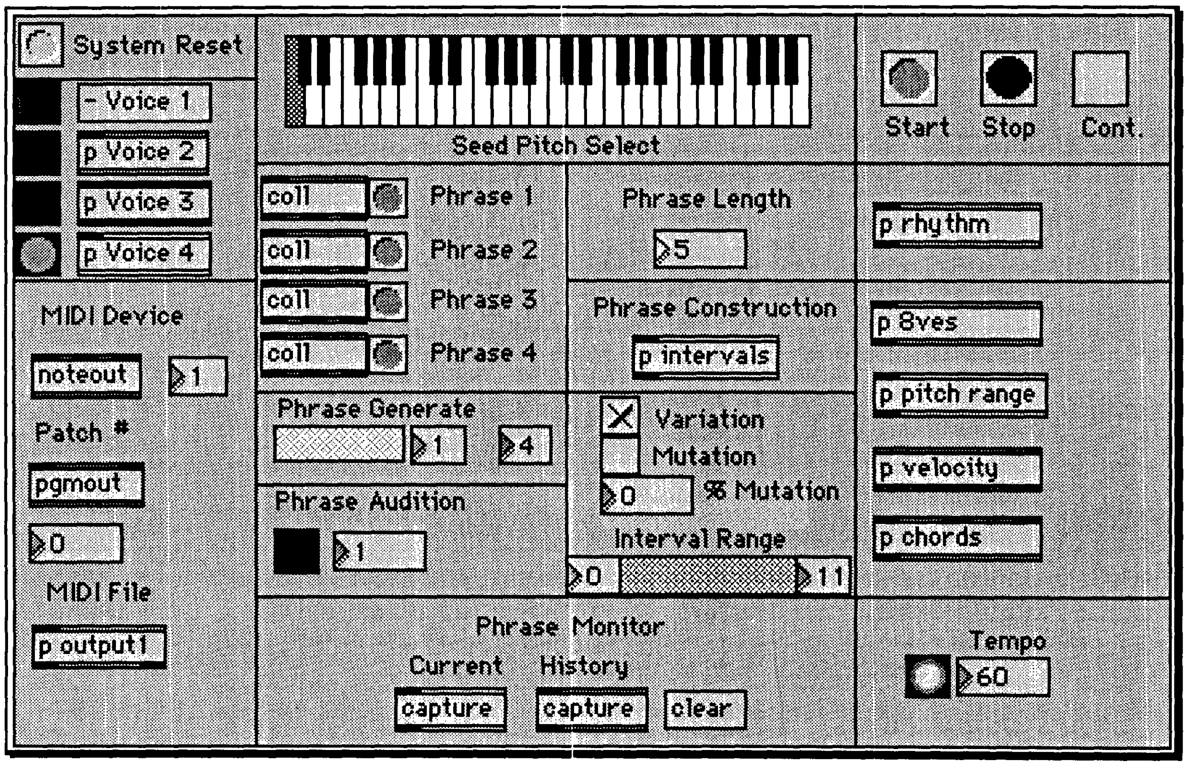
8.3. *Phrase Garden*

8.3.1. Overview

Phrase Garden primarily uses *Blind Watchmaker* as a model for the generation of pitch materials. Algorithms for rhythmic materials are based on those in *Symbolic Composer* in which rhythmic cells are user defined, the statements of rhythmic cells in *Phrase Garden* controlled by occurrences of intervals in the generated pitch material. The *Phrase Garden* Velocity algorithm is based on the Velocity-range algorithms in *M* and *Jam Factory*, providing both deterministic and stochastic settings for dynamic levels. An algorithm for vertical aggregates supplies harmonisations of linear pitch material generated by the program. In addition to these algorithms, there are algorithms for control over octave placements of pitch, pitch range, and tempo. Output of the program is heard in real-time, and in the tradition of interactive programs such as *M* and *Jam Factory*, settings in the program may be altered to affect, in real-time, generated materials. *Phrase Garden* output can also be saved to MIDI files which may be played while the program is generating new materials. The program offers four voices, or players, that are independently configurable.

Figure 8.7 shows the main screen of the *Phrase Garden* program. The piano keyboard at the top centre provides a method for pitch input to the program, and pitch input may also be provided with an external MIDI device. Rectangles (objects) with a p prefix represent *MAX* sub-patches in the program. These objects are double-clicked with the computer mouse to open a window, within which control is provided over parameters associated with the object name. Objects without the p prefix are also double-clicked either to show data collections (coll and capture) or to configure MIDI parameters (noteout and pgmout). Objects with numbers and a right pointing triangle (number boxes) allow the setting of numbers, and square objects with, or without, inner circles are switches that turn a function on or off. Shaded rectangles are bar graphs in which the computer mouse can be clicked and dragged to set numbers.

Figure 8.7 *Phrase Garden,* Main screen



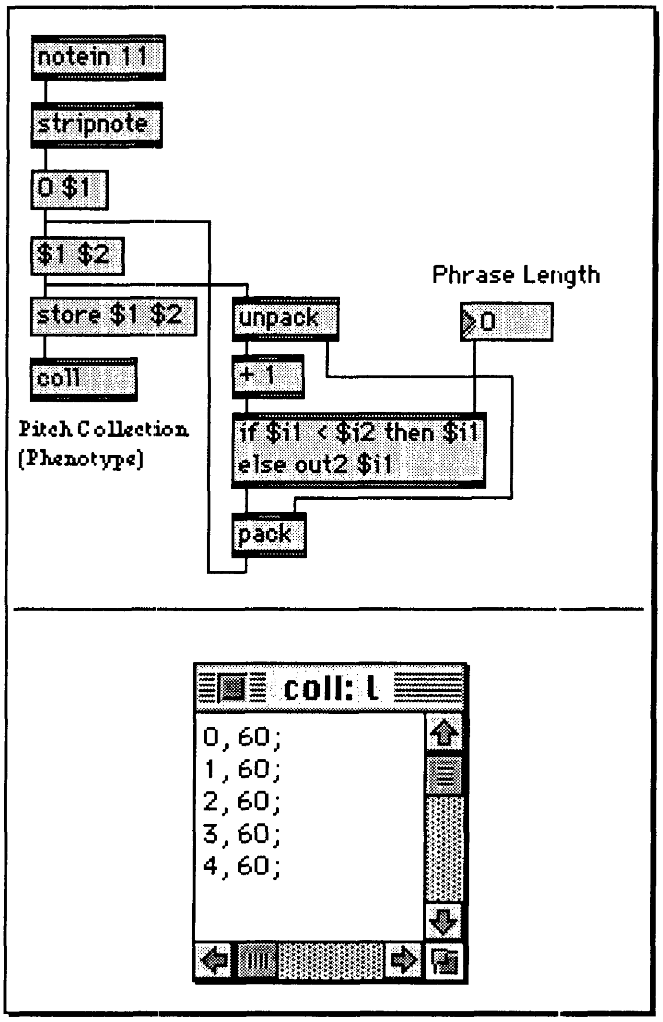
8.3.2. Pitch generation algorithms

In the initial development of Phrase Garden, a decision was made that pitch generation would be controlled by the intervallic content of the note matrices used in the compositional style, as opposed to the individual notes of the note matrices. In instrumental works composed prior to 1994, note matrices used within the compositional style included notes of an original note series, along with notes from an inversion of the original series. The use of both original and inverted note series resulted in differing pitches within a work, although interval content in a work remained constant. This shift in focus, from notes to intervals, allowed the development of *Phrase Garden* to be based on the *Blind Watchmaker* model. In *Phrase Garden*, as in *Blind Watchmaker*, phenotypes are developed from a genotype. *Phrase Garden* phenotypes are musical phrases, and its genotypes are intervals.

8.3.2.1. The Replication algorithm

At the heart of *Phrase Garden's* pitch generation is an algorithm for the self-replication of a single pitch, referred to as the Replication algorithm. Like the Dawkins example of Figure 8.5, which begins from a single pixel, *Phrase Garden* uses a single pitch as a 'seed' for the evolution of phenotypes or phrases. This pitch is supplied by the user with either the on-screen keyboard, or with an external MIDI device. The Replication algorithm takes the seed pitch, for example a middle C, and replicates the pitch a number of times according to a desired phrase length setting which is set with the Phrase Length number box on the main screen. Figure 8.8 shows the *MAX* patch used for pitch replication. With the exception of the Phrase Length number box and the coll object, all objects in the patch are hidden from view on the main screen of the program.

Figure 8.8 *Phrase Garden,* Replication algorithm and coll contents



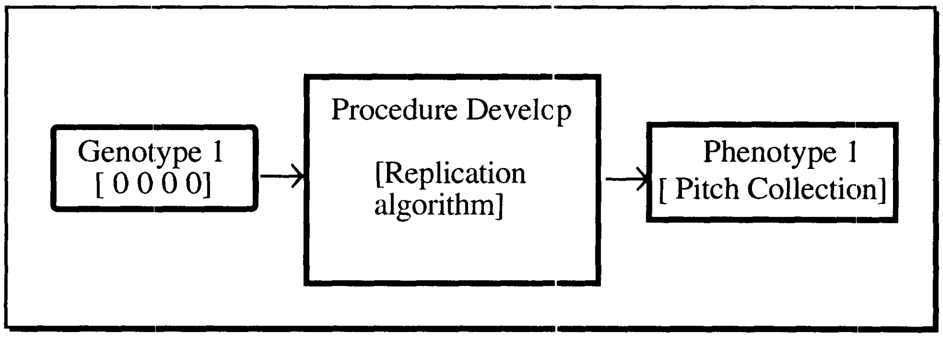
```
0,60;
1,60;
2,60;
3,60;
4,60;
```

The notein object shown in Figure 8.8 receives a MIDI note-on message when a pitch is played by the user — in this example, a middle C (MIDI note number 60) — and a MIDI note-off message when pitch is released. The stripnote object filters out MIDI note-off messages, and passes on the MIDI value of the pitch received. The MIDI pitch value is assigned an index number of 0 in a 'message box' immediately beneath the stripnote object, and the pitch value itself is defined as $1 (argument 1). In the next message box the index number is defined as $1 (argument 1) and the pitch value is $2 (argument 2). The number pair is sent to the 'store' message box which instructs the following coll object to store the index and pitch numbers of 0 (the index) and 60 (the pitch). The number pair is also sent to the unpack object which sends the index number out its left outlet and the pitch number out its right outlet. The index number is incremented by one with the + object (from 0 to 1) and combined again with the pitch value in the pack object. The new index number (1) and the pitch value (60) are sent to the $1 $2 message box, stored in the coll object, and sent again to the unpack object for another increment in the index number. This process is repeated in a cycle or loop until an end condition, set with the if object, is met. The if object receives the index number (labelled $i1) in its left inlet, and a comparison is made with a number received in its right inlet which is labelled $i2, and is specified by the user as the Phrase Length number. When the index number is less than (<) the Phrase Length number, the index number is passed onto the pack object and the cycle or loop continues. When the index number is equal to the number specified in the right inlet (i.e. after five cycles), the index number is sent out the right outlet of the if object, which is not connected to any further objects and hence the loop or cycle is broken.

With a Phrase Length setting of 5 and a pitch input of middle C (60), five pitch values are generated with the Replication algorithm and stored in the coll object with indices of 0 through to 4. The coll object appears on the *Phrase Garden* main screen, labelled Phrase 1, and the contents of the coll object are viewed by double-clicking with the mouse on the object. A window appears on-screen with the contents of the coll object, as shown in the lower portion of Figure 8.8. Further *MAX* objects, not shown in Figure 8.8, are used to trigger the pitch values stored in the coll object by stepping through the index numbers one by one and sending the pitch value associated with each index number out a coll outlet. From the coll outlet, pitch values are sent to *MAX* MIDI objects and then to external MIDI devices for playback. The five pitch values in the coll object are sent out consecutively to form a phrase, and in relation to the Dawkins model, this phrase is considered as a phenotype.

Whilst there is no genotype specified for the phenotype that is the pitch collection of five middle C pitches, there is the existence of an underlying linear interval structure of 0 0 0 0 within the pitch collection (i.e. from one pitch to the next there is an interval of 0 semitones). In *Phrase Garden*, this underlying interval structure is viewed as the genotype of phenotypes consisting of five repeated pitches. Considering the existence of such a genotype, the Replication algorithm alone satisfies a significant portion of criteria of the von Neumann model. Drawing on material in Figure 8.6, there is, as shown in Figure 8.9, a genotype (0 0 0 0) which is passed through the Replication algorithm (Procedure Develop) and a resulting phenotype of five repeated pitches.

Figure 8.9 *Phrase Garden,* Genotype to phenotype procedure



8.3.2.2. The Interval weighting algorithm (p intervals)

In order to provide phenotypes with different genotypes from that shown above, *Phrase Garden* incorporates a *MAX* sub-patch into the 'Procedure Develop' section of Figure 8.9. In the Dawkins model, a recursive branching rule is incorporated in the 'Procedure Develop' section to develop an archetypal phenotype. In *Phrase Garden*, user specified interval weightings are incorporated in the 'Procedure Develop' section to develop a phrase.

The use of interval weightings in the development of *Phrase Garden* phrases corresponds to the interval weightings used in the compositional style. As any interval, or any number of different intervals, may be weighted more heavily in phrases intuitively composed within the compositional style, the *MAX* sub-patch used for interval weighting in *Phrase Garden* similarly allows a weighting on one or more intervals. The Interval weighting algorithm, labelled p intervals, is a *MAX* sub-patch that is placed between the unpack and pack objects shown in Figure 8.8. The note values sent out of the right outlet of the unpack object are passed to the inlet of p intervals. The p intervals object appears on the *Phrase Garden* main screen, and is double-clicked to access settings for interval weightings.

Figure 8.10 shows the p intervals settings window in which interval weightings are set within a percentage range. In Figure 8.10, semitones (interval 1) are weighted most heavily (0-50%), whilst intervals of 2 semitones (interval 2) and 3 semitones (interval 3) are equally weighted with each having a 25% chance of occurrence within a phrase. Number boxes below the percentage range number boxes indicate interval direction. The number 1 (the direction setting for interval 1 in the example) results in an ascending interval, the number 2 (the direction setting for interval 2) results in a descending interval. The number 3 (the setting for interval 3) results in a probabilistic occurrence of either ascending or descending intervals, a control placed at the bottom of the window allowing the setting of a percentage probability of ascending intervals. In Figure 8.10, this control is set to provide a 50% chance of occurrences of ascending intervals.

Figure 8.10 *Phrase Garden,* p intervals settings window

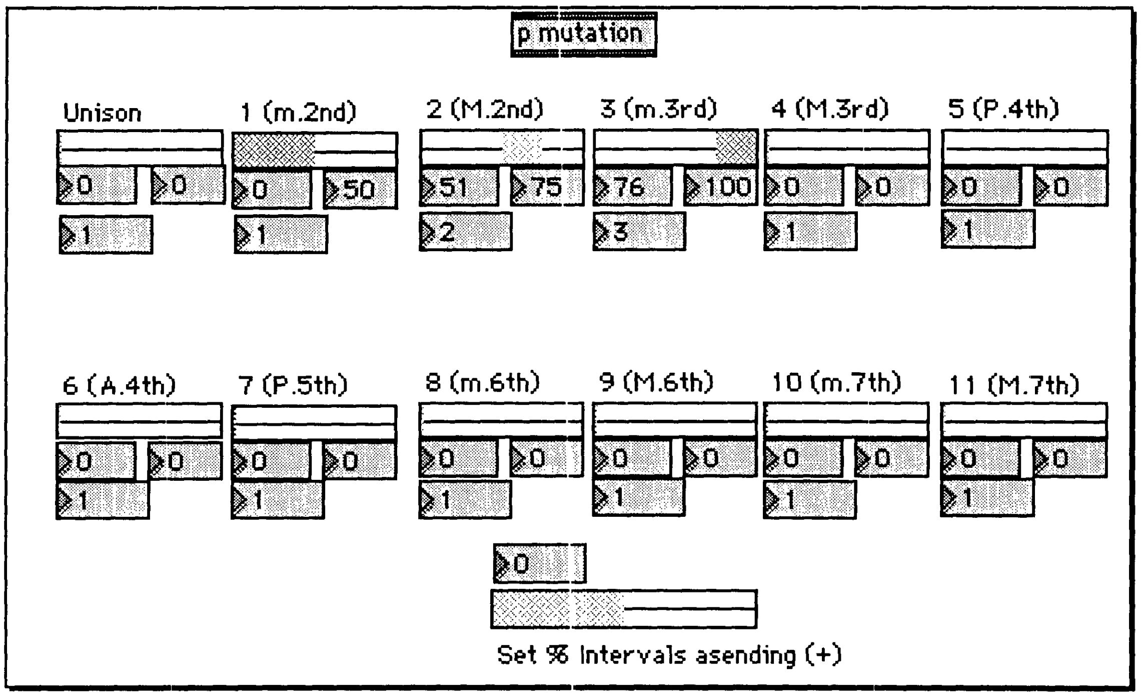
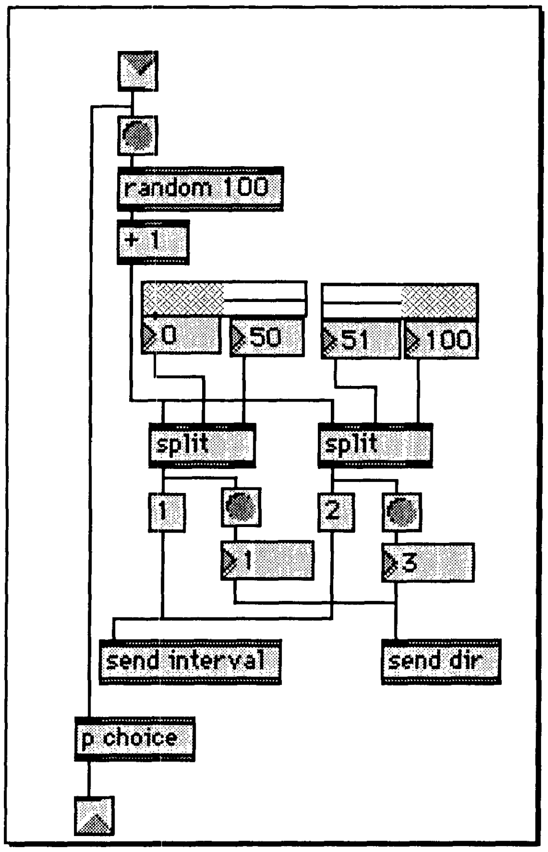


Figure 8.11a shows hidden *MAX* objects in the p intervals sub-patch, and is reduced to show hidden objects from just two of the 12 interval settings controls within the sub-patch. Pitch values from the pack object shown in Figure 8.8 enter the p intervals sub-patch via the inlet object in the top left corner. From this inlet object, pitch values are sent simultaneously to two different places: to another sub-patch labelled p choice, and to the object immediately below the inlet object which is a *MAX* button object. Any data received by the button object is converted to a positive action message (a bang in *MAX* terminology)

that is sent out the button outlet and recognised by other *MAX* objects. Pitch values received by the button object are converted to bangs which are used to trigger pseudo-random numbers in the succeeding random object. The random object outputs random numbers in a range between zero and one less than a specified number. The specified number in this example is 100 and random will output numbers between 0 and 99. The following + object increments each random number by one, so that the range is from 1 to 100, and corresponds to the percentage range used in the interval settings controls.

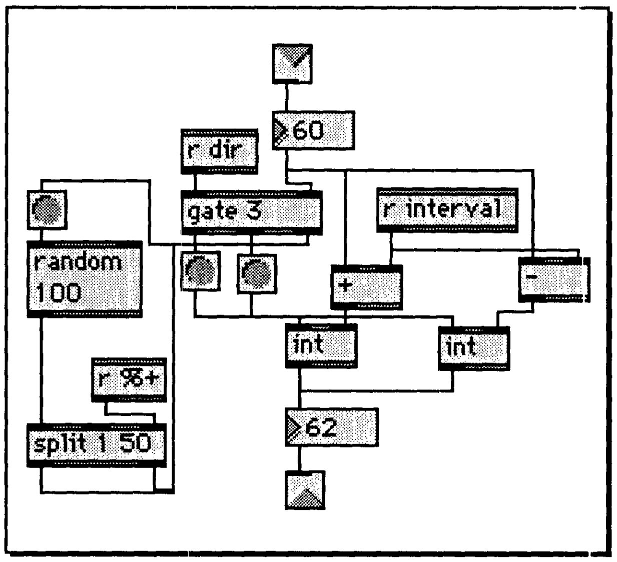
The *MAX* split object receives numbers in its middle and right inlets and stores these numbers respectively as the minimum and maximum in a range of numbers. Numbers received in the left inlet are passed out the left outlet if they fall within the specified range, otherwise they are passed out the right outlet. In Figure 8.11a a number produced by the random object is sent to all split objects simultaneously, and the split object that has the percentage range containing the random number sends the number out its left outlet. The number from a split object left outlet triggers a message box containing another number that represents the interval with which the split object range is associated. The message box interval number is then sent to a *MAX* send object (send interval). The random number is also sent, via a button object, to trigger a number box containing the interval direction number, 1, 2 or 3, this number then sent on to the send dir object. *MAX* send objects correspond to receive objects that have the same name as a suffix. The receive objects may be placed in the same patch, or in sub-patches, and receive data from send objects.

Figure 8.11a *Phrase Garden,* p intervals settings controls and hidden objects



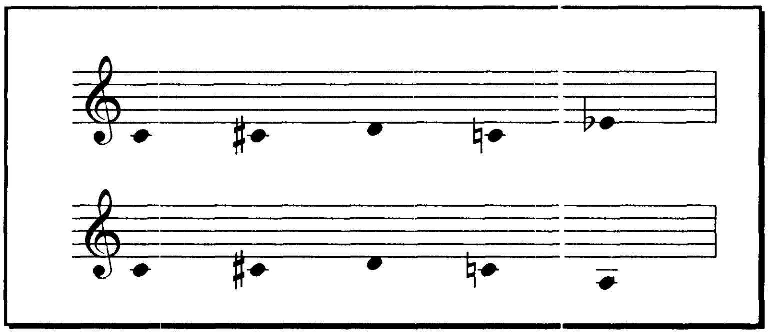
The receive interval and receive dir objects (abbreviated to r interval and r dir) are placed in the sub-patch labelled p choice, the contents of which are shown in Figure 8.11b. This sub-patch is another destination for pitch values received in the p intervals inlet, pitch values initially sent from the Replication algorithm pack object shown in Figure 8.8. The p choice sub-patch, then, receives a pitch value, a number representing an interval, and a direction for the interval. The r interval object passes the interval number to both a + and a – object. Both of these objects also receive a pitch value from the p choice inlet object, and add or subtract the interval value to the pitch. As an example with the settings in Figure 8.10, a middle C (60) may trigger a pseudo-random number of 99. This number falls into the range of numbers associated with the interval value of 2. The + object will add the interval number to the pitch value, resulting in 62, whilst the – object will subtract 2 to give 58 (i.e. a D above middle C, and a Bb below middle C respectively). Once the additions and subtractions have taken place, the results are passed to int (integer) objects which store the new pitch values. The interval direction value associated with the interval value of 2 is 3 (either up or down), which, in this example, corresponds to a probabilistic 50% chance of the interval ascending, or a 50% chance of the interval descending. The gate object in the choice sub-patch has three outlets that, from left to right, correspond to the interval direction numbers 1, 2 and 3. When the number 3 is received, as it is in this example, the right-most outlet is opened. Pitch values from the inlet object are then passed through the right-most outlet to trigger, with a button object, another pseudo-random number in a percentage range. This number is passed to another split object that, if the number is in the range of 1-50, sends it out its left outlet. The number is then converted to a bang which triggers the int object (associated with the + object that has stored the number 62) to output its stored number. If the pseudo-random number is out of the range of the split object (between 51 and 100), the number is passed out the split object's right outlet and triggers the output of the – object associated int storing the number 58. The adjusted pitch value is then sent out the outlet object in the p choice sub-patch, then out an outlet object in the p intervals sub-patch. In the Replication algorithm, the outlet of the p intervals sub-patch is connected to the pack object where it is combined with an index number and stored in the coll as part of a pitch collection or phrase.

Figure 8.11b *Phrase Garden,* p intervals, p choice sub-patch



Each new pitch value that is passed from the p intervals sub-patch back to the Replication algorithm is, aside from being stored in the coll object, returned to the p intervals sub-patch, continuing the loop of the Replication algorithm until the index numbers assigned to pitch values meet the end condition set with the if object. Referring to the interval weightings shown in Figure 8.8, a possible interval structure of a five-pitch phrase generated with these settings is 1 1 2 3. Starting with a seed pitch of middle C, this interval structure, as shown in Figure 8.12, can result in two different phrases due to the probabilistic direction setting of 3 (either up or down) for the final interval of 3 semitones. The deterministic settings for the upward direction of interval 1 and the downward direction of interval 2 result in identical pitches in the initial four pitches of the two phrases.

Figure 8.12 *Phrase Garden,* Phrases with interval structure 1 1 2 3



Further possibilities for interval structures include all of the permutations of the series 1 1 2 3, for example 1 2 3 1, 3 1 1 2 etc. Other possibilities are interval series such as 1 1 1 3 and 2 2 1 3, in both of which the original weightings differ from those set in the intervals settings window. This is due to the variety of pseudo-random numbers supplied by the random object in p intervals. With the use of the random object in *Phrase Garden,* there is no specification that the intervals set in the interval settings must all appear in every phrase produced by the program. Pseudo-random numbers produced by the random object (between 1 and 100) may all fall into a single range of, for example, 50 and 75. With the interval settings of Figure 8.10, this would result in an interval series of 2 2 2 2. However, such occurrences are rare, the pseudo-random numbers provided by the random object generally producing desired weightings of intervals.

In summary, the Replication algorithm provides a pitch collection of a length specified by the user with the Phrase Length number box on the *Phrase Garden* main screen. The p intervals sub-patch provides a means of specifying rules for the development of a phrase, the rules entailing a user specified weighting of intervals. Probabilistic settings for interval direction allow phrases with the same interval structure to develop with differing pitches.

The possibility of developing differing phrases with the use of the either up or down (3) direction setting led to a provision in *Phrase Garden* of not just a single phrase being developed by an interval series, but a series of four phrases. On the main screen, these phrases are stored in the individual coll objects labelled Phrase 1 to Phrase 4. A main screen function related to the generation of phrases is the Phrase Generate function, placed immediately below the four coll objects. This function allows the user to specify the number of phrases developed with a single interval series. The default setting of 1 to 4 allows all four coll objects to store phrases developed on a single interval series. For the development of a single phrase, a setting of 1 to 1 is used.

In the development of a first phrase, pitch values from the pack object in the Replication algorithm are sent to the p intervals sub-patch as previously explained. In the development of succeeding phrases, the p intervals sub-patch is bypassed, the pitch values going to another (hidden) sub-patch labelled p child in which the interval numbers from the first phrase are stored, along with the direction of each interval. As pitch values in a succeeding phrase enter the p child sub-patch, the interval and interval direction values of the parent phrase are applied so that the succeeding phrase shares the same interval content as the initial phrase.

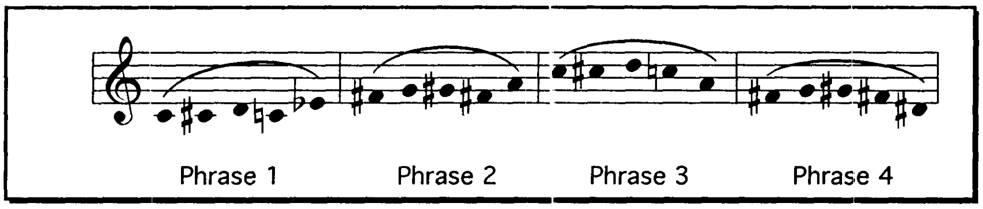
Of the four different phrases stored in the four coll objects (with a 1 to 4 setting in the Phrase Generate function), only the first phrase uses the seed pitch supplied by the user as the initial pitch in a phrase. To provide the initial pitches of the remaining phrases, another hidden sub-patch, labelled p trigger, is used. In this sub-patch is an algorithm that determines the last pitch value, interval and interval direction of a preceding phrase, and provides a new pitch based on those parameters. As an example, the last pitch in the first phrase of Figure

8.12 is an Eb, the last interval is 3 semitones, and the direction is a result of the either up or down setting for the interval 3. Using these parameters to provide the initial pitch of a next phrase, the interval of 3 semitones will provide, from the final Eb of the phrase, either a Gb above the Eb, or a middle C. Which of the two pitches that appears depends on a probabilistic decision made with a pseudo-random number supplied by a random object in the p trigger sub-patch. The random object in this sub-patch is used to provide the interval direction for the initial pitches of Phrase 2, 3 and 4 when the interval direction supplied by the p child sub-patch is either up or down.

The reason for supplying a new pitch as an initial pitch in each phrase is that *Phrase Garden* primarily relates to the compositional style, and two consecutive phrases in the compositional style rarely begin with the same starting pitch. Even rarer is any occurrence of four consecutive phrases with the same starting pitch. The Cont. button (standing for continuous playback) in the upper left corner of the main screen allows a real-time, consecutive playback of phrases stored in the coll objects. Were phrases to begin on the same starting pitch, phrases generated and played back in real-time would be always limited to a narrow range of pitches, continually returning to the same pitch at the start of each phrase. The algorithm that supplies a new pitch at the start of each phrase can result in pitch movement through a wide pitch range.

Figure 8.13 shows a possible series of phrases beginning with the first phrase shown in Figure 8.12. Each phrase shares the same interval content, 1 1 2 3, with which a limited number of pitches are possible at the end of each phrase when the interval directions are up (interval 1), down (interval 2) and either up or down (interval 3) respectively. The final notes of each phrase in this limited configuration are either Eb (D#) or A. Resulting pitches in the phrases may, however, occur in various octave placements as shown with the first and third phrases, and the second and fourth phrases, the differing octave placements being a direct result of the supply by the p trigger sub-patch of different starting pitches for each phrase.

Figure 8.13 *Phrase Garden,* Four consecutive phrases with interval structure 1 1 2 3



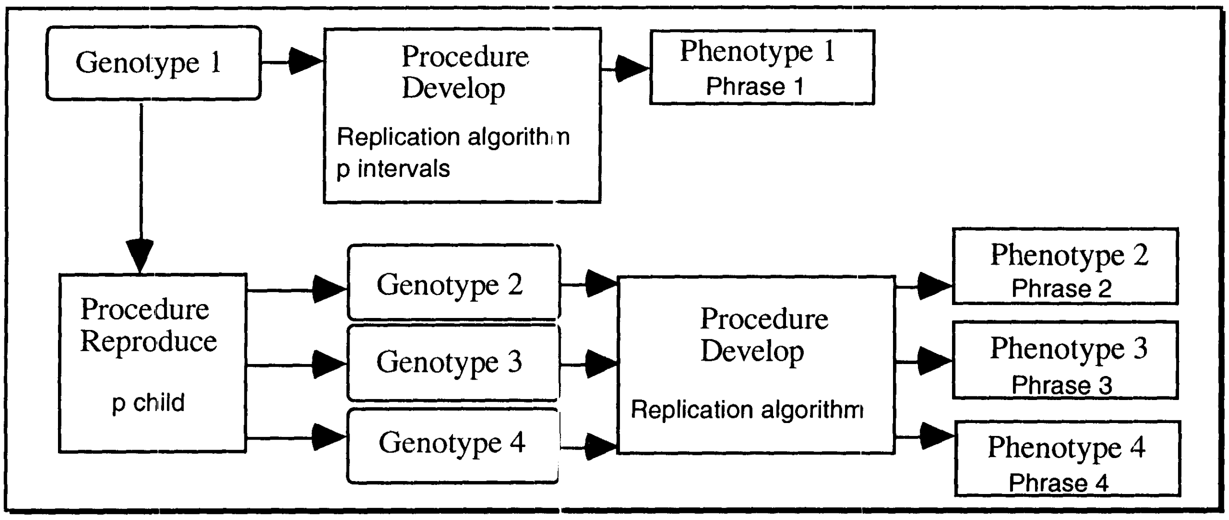
Phrase Garden provides controls for the playback of generated phrases in the upper right corner of the main screen. As previously explained, the Cont. button, when checked, allows continuous playback of consecutively numbered phrases. Playback in this mode is initialised by clicking with the mouse on the Start button. When the Cont. button is unchecked, as it is in Figure 8.7, the Start button is used to play a single phrase. After a phrase is played, a second click on the Start button initiates playback of the next consecutively numbered phrase. As such, the continuous playback resulting from checking the Cont. button results in a real-time generation and playback of phrases, and when the button is unchecked, phrases can be listened to individually at the leisure of the user.

With the Phrase Generate function, a setting of 2 to 4 may be made, following the generation of an initial phrase. With this setting, pitch values from the pack object in the Replication algorithm are continually passed to the p child sub-patch, and all phrases generated have the same interval content as the parent (Phrase 1) but are subject to mutations resulting from the use of the either up or down interval direction setting, and the change of pitch at the start of each phrase. Phrases stored in the Phrase 2 3 or 4 coll objects are overwritten with new phrases. Such a setting allows searches through the genetic space of phrases developed with a single genotype, or, with the Cont. button checked, the real-time playback of phrases that each have the same interval content.

The provision of four consecutive phrases in *Phrase Garden* is analogous to the Dawkins model in which a group of child biomorphs are drawn on-screen with the parent in the centre. The initial *Phrase Garden* phrase phenotype (Phrase 1) is considered as the parent of the ensuing phrases (Phrase 2, 3 and 4). In the Dawkins model, child biomorphs vary from the parent as a result of a mutation in the genotype. In *Phrase Garden*, child phrases vary from the parent phrase as a result of the algorithm that provides the child phrase with a different starting pitch from that of the parent. As shown in Figure 8.13, the child phrases (Phrase 2, 3 and 4) each have a differing set of pitches from those of the parent phrase. Therefore, they may be considered as mutations of the parent phrase. This process of mutation, however, is not analogous to mutation in the Dawkins model, as the algorithm for providing a new starting pitch does not affect the interval series or genotype. The mutations in the genotype that result from the use of the either up or down interval direction setting are analogous to the Dawkins model. Use of this setting results in child genotypes that are identical to the parent genotype with the exception of an up (+) value, or a down (–) value in one or more of the genes. In Figure 8.13, for example, the first phrase is considered as a parent phrase with an interval genotype of 1+ 1+ 2– 3+, whilst the third phrase is considered as a child phrase with an interval genotype of 1+ 1+ 2– 3–.

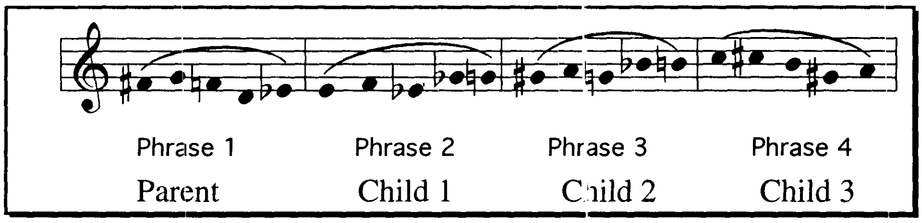
In reference to the genotype/phenotype modules of Figure 8.6, *Phrase Garden* child phrases are considered to be formed as a result of algorithms that function within the 'Procedure Reproduce' module. The parent genotype is passed onto the child phrases within the *MAX* sub-patch p child. As previously explained, this sub-patch stores the intervals and interval directions of the parent phrase and applies the intervals and directions to pitch values entering the sub-patch from the Replication algorithm pack object. Figure 8.14 is based on the genotype/phenotype procedure from Figure 8.6, and shows the corresponding *MAX* sub-patches included in the modules to which the sub-patches are relevant.

Figure 8.14 Genotype/phenotype modules with corresponding *MAX* sub-patches



Settings of the Phrase Generate function that include Phrase 1 always result in pitch values being passed through the p intervals sub-patch. If the Phrase Generate function has settings of 1 to 4, a parent phrase, as previously explained, is formed as Phrase 1, and three child phrases are generated, Phrase 2, 3 and 4. Further phrase generation initially uses the p trigger sub-patch to provide a new starting pitch for Phrase 1 based on the final pitch, interval and interval direction of Phrase 4. This pitch becomes a new seed pitch for a new parent phrase developed in the p intervals sub-patch. Within the p intervals sub-patch, a new genotype is produced; however, this genotype shares the same interval weightings of former phrases. Where the formerly discussed interval series, or genotype, of 1 1 2 3 was one possible example resulting from the settings in Figure 8.10, a new possible genotype of 1 2 3 1 may, for example, be generated with the same settings. Following on from the phrases shown in Figure 8.13, a new set of parent and child phrases, based on the interval series 1 2 3 1 may be produced, as shown in Figure 8.15.

Figure 8.15 *Phrase Garden,* Four consecutive phrases with interval structure 1 2 3 1



The provision of a new genotype with each occurrence of Phrase 1 represents a departure from the Dawkins model in that the first two generations, consisting of a parent and child phrases, have a genotype which is not passed down to a third generation. Rather, a new genotype is used in a third generation phrase, this genotype, as shown by the four phrases in Figure 8.15, being passed down to a fourth generation of child phrases. In effect, this process of providing a new genotype with each occurrence of Phrase 1, results in variations to the intervallic content of phrases from one set of parent and child phrases to the next. This process was implemented in *Phrase Garden* in accordance with the compositional style, the phrases therein having a weighting on one or more intervals, yet from one phrase to the next, interval content within the note matrices appears in various orders.

8.3.2.3. The Mutation Mode (p mutation)

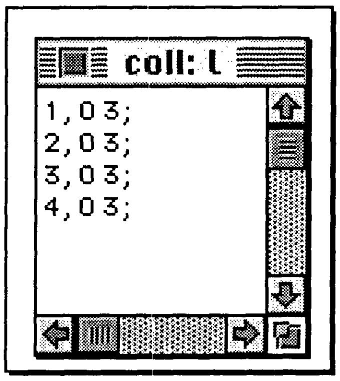
The generation of a re-ordered interval series with each occurrence of Phrase 1 takes place in a *Phrase Garden* mode of operation referred to as Variation Mode. A second operating mode is also implemented in *Phrase Garden,* and is referred to as Mutation Mode. On the *Phrase Garden* main screen, controls for switching from one mode to the other are located beneath the p intervals object, the check mark against Variation in Figure 8.7 indicating that *Phrase Garden* is in Variation Mode. Where Variation Mode is implemented in accordance with the compositional style and departs from the Dawkins model, Mutation Mode is the opposite: this mode is implemented in accordance with the Dawkins model and departs from the compositional style.

Mutations in Mutation Mode occur in the values of intervals. Referring to the genotype/phenotype modules of Figure 8.6, alterations to intervals fall into the Dawkins specification for Procedure Reproduce in which a parent genotype is passed onto the child, but with some random chance of mutation. Unlike the minor mutations that result from the applications of the either up or down interval direction setting in Variation Mode, mutation in Mutation Mode is more comprehensive, with random changes in interval values occurring between sets of parent/child phrases, along with the mutations that occur with the either up or down interval direction setting between a single parent phrase and immediately succeeding child phrases. In the Dawkins model, a child biomorph differs from the parent by a value of +1 or –1 in one of its genes. In Mutation Mode, the same numerical values are applied. Any gene or interval in an initial parent/child set may differ from genes or intervals in the following parent/child set by a value of one semitone.

The overall phrase generation process in Mutation Mode is identical to that in Variation Mode: parent/child phrase sets share the same interval genotype, minor mutations occurring with the use of the either up or down interval direction setting. Child phrases that share the same interval genotype as the parent are, in both modes, generated as many times as desired though manipulation of the Phrase Generate function settings. However, where a new parent phrase in Variation Mode contains a *re-ordering* of a previous interval genotype, a new parent phrase in Mutation Mode contains the intervals of a previous genotype, but with an *alteration* to a number of these intervals by a semitone step.

In Mutation Mode, a series of intervals and interval directions for a phrase is specified with a coll object that is part of a sub-patch labelled p mutation. This sub-patch is located at the top of the p intervals settings window, as shown in Figure 8.10. Default settings for intervals and interval directions are stored within the p mutation coll object, these settings automatically generated when a Phrase Length number is set on the main screen. The default settings provide a quantity of index numbers, corresponding to the number of intervals contained in a phrase of a length set with the Phrase Length number box. With each index number, default settings are interval values of 0, and interval directions of 3, the either up or down direction setting. Figure 8.16 shows the contents of the p mutation coll object, following a Phrase Length setting of 5. The index number is first in each horizontal row, followed by the interval value and the direction setting.

Figure 8.16 *Phrase Garden,* p mutation, coll defaults with Phrase Length 5



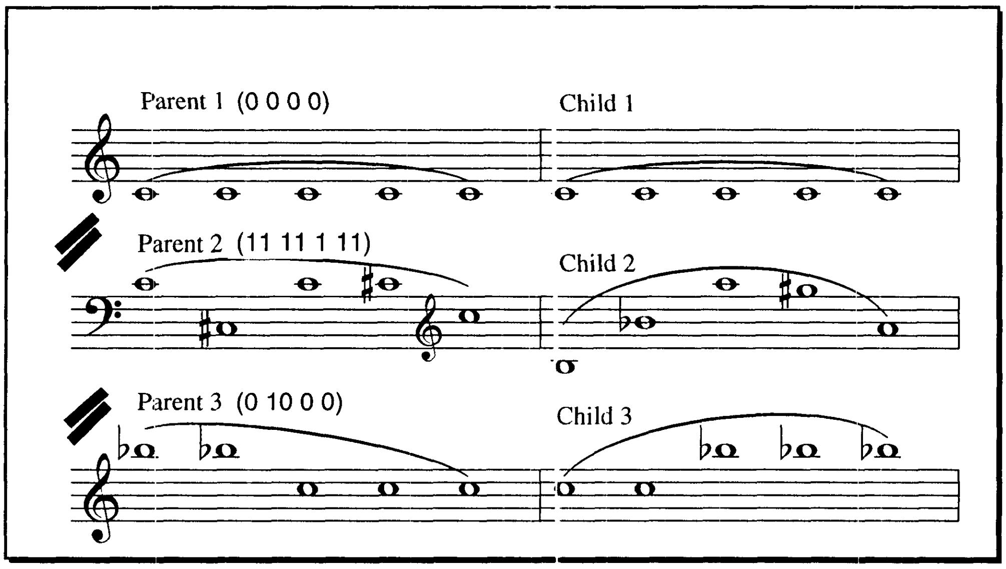
```
1,0 3;
2,0 3;
3,0 3;
4,0 3;
```

The default settings shown in Figure 8.16 may be altered to suit requirements of the user, allowing the specification of particular phrases. As an example, the formerly discussed phrase with the interval series 1 1 2 3 may be required as a starting point for phrase generation. The default interval values (0 0 0 0) in the coll object may be replaced by typing the required interval values, and the direction for each interval may be specified by replacing (or leaving) the interval direction value of 3.

Further main screen settings for the generation of phrases in Mutation Mode are the %Mutation number box and the Interval Range range bar. The %Mutation number box allows control over the number of intervals in a phrase that will mutate from one parent/child phrase set to the next. A setting of 0% (i.e. no mutation) will result in the continuous repetition of a specified (or the default) interval series for an infinite number of phrases. At the other extreme, a 100% setting will generally result in the mutation of all intervals from one parent/child set to the next. The Interval Range control allows the specification of a range of intervals that may be used in the mutation process. The default setting is the full range of 0 to 11 semitones. Semitone mutations in Mutation Mode wrap around in the Interval Range, so that, with the default, either up or down interval direction setting, a semitone step downward from the interval 0 results in an interval of 11 semitones, and an interval step upwards from 11 results in a step of 0 semitones.

Once the intervallic content of a phrase has been specified, and the Interval Range and % Mutation controls set, a seed pitch is entered with the on-screen keyboard or an external MIDI keyboard. Figure 8.17 shows interval series mutations over three consecutive parent/child phrase sets generated in Mutation Mode. A Phrase Length setting of 5 is used, and the example begins with the default interval series of 0 0 0 0. A Phrase Generate setting of 1 to 2 is used for the example, providing a parent phrase and a single child in each parent/child set. A seed pitch of middle C is entered, with settings of 100% mutation and an interval range of 0-11.

Figure 8.17 *Phrase Garden,* Example Mutation Mode output with default settings



When the Mutation control on the main screen is checked, the p mutation sub-patch receives pitch values from the Replication algorithm unpack object. These values are used to trigger numerous objects within the p mutation sub-patch, and also objects in another sub-patch within p mutation labelled p genmut. Both sub-patches are shown in Figure 8.18. The sub-patches are complex: they receive and implement values set with the %Mutation and Interval Range controls on the main screen, they store intervals and interval directions, add to, and subtract from, interval values, and send out new values. New values are sent to, and stored in, the p child sub-patch, and are sent to the p choice sub-patch within p intervals. From the p choice sub-patch the new values are sent, as they are in Variation Mode, on to the pack and coll objects in the Replication algorithm. The preceding discussion of Mutation Mode serves to detail the pitch generation processes carried out in the mode, making a detailed, object by object, discussion of the complex p mutation and p genmut sub-patches unnecessary.

Figure 8.18a *Phrase Garden,* p mutation sub-patch

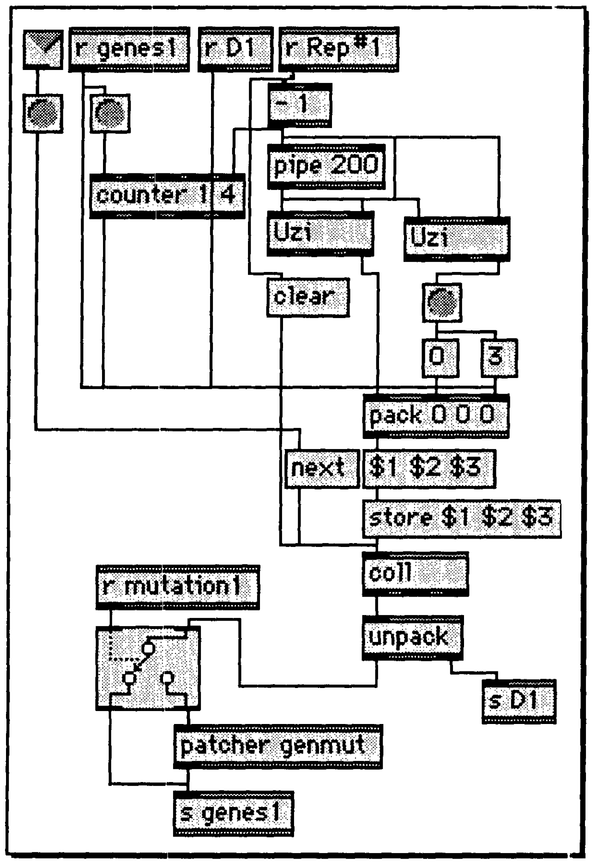


Figure 8.18b *Phrase Garden,* p genmut sub-patch

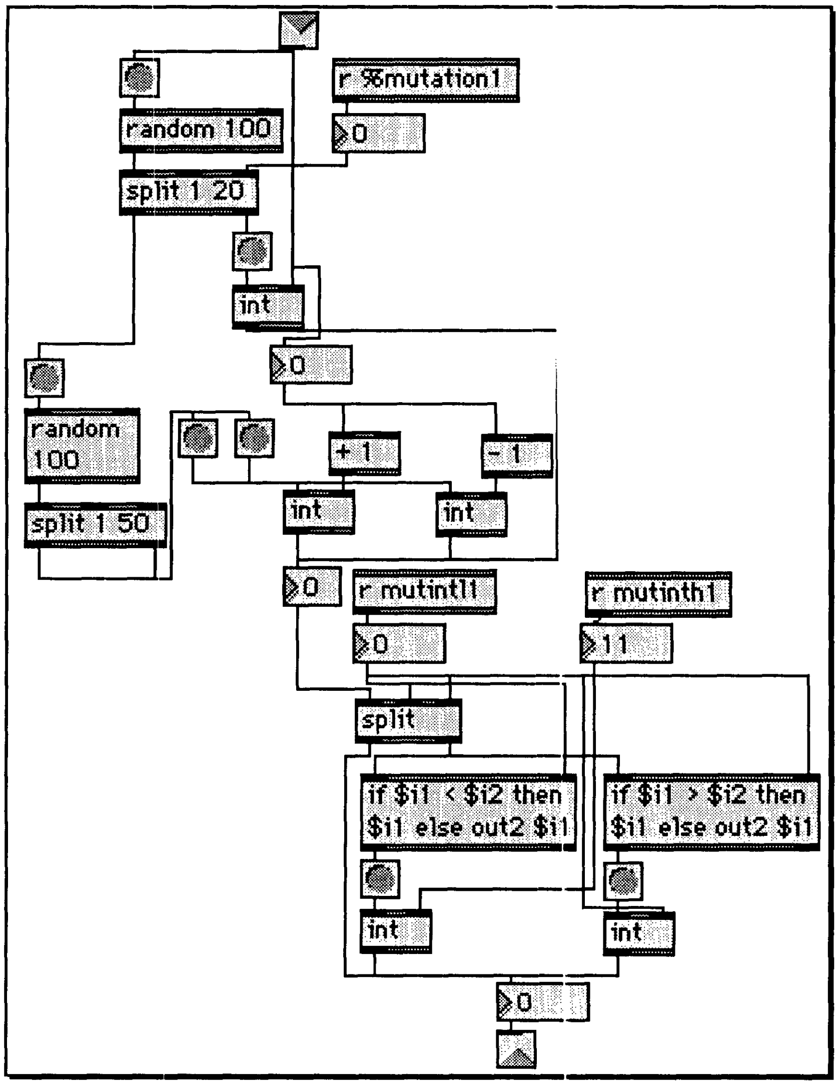
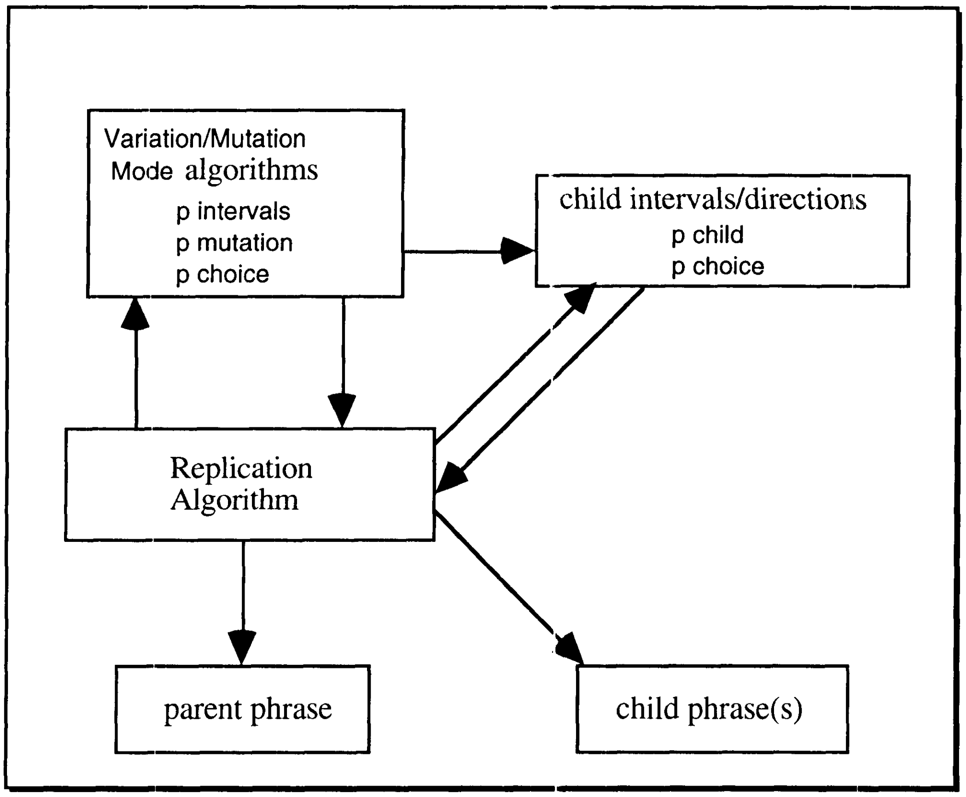


Figure 8.19 is a schematic diagram that shows the flow of data between the Replication algorithm, the two modes of pitch generation and child phrases. Vertical arrows in the Figure represent the flow of data when a parent phrase is generated. Horizontal and angled arrows represent the flow of data when a child phrase is generated. The diagram provides an overview of the manner in which *Phrase Garden* implements the algorithms previously discussed in the generation of pitch material.

Figure 8.19 *Phrase Garden,* Data flow schematic diagram



Pitch values from the Replication algorithm module in Figure 8.19 are sent to the module containing the algorithms used for pitch generation in either Variation Mode or Mutation Mode. Within the latter module, a pitch value is incremented or decremented by an amount specified by the user, either through interval weightings, in Variation Mode, or through the specification of particular intervals, in Mutation Mode. The new pitch value is passed back to the Replication algorithm, and is then stored in the parent phrase module. The pitch is also returned to the Variation/Mutation Mode module and the process is repeated. The cycle or loop in which pitches are passed from the Replication algorithm to the Variation/Mutation Mode module continues until an end condition (i.e. the number of pitches required in a phrase) is met.

Each interval value, and interval direction value, provided by the Variation/Mutation Mode algorithms is sent to the child phrase module, in which the values are stored. After the parent phrase is complete, a *MAX* gate object in the Replication algorithm module is used to direct pitch values to the child module, as opposed to sending pitch values to the

Variation/Mutation module. Pitch values entering the child module are incremented or decremented using the intervals formerly stored within the module. Pitch values are then returned to the Replication algorithm module, and stored as child phrase(s).

Once a user-specified number of child phrases have been played, the gate object in the Replication algorithm module re-directs pitch values back to the Variation/Mutation Mode module. When Variation Mode is used, a re-ordering occurs of intervals used in the former parent/child phrase set, however, the re-ordered interval set uses the same interval weightings of the former parent/child phrase set. When Mutation Mode is used, an alteration occurs to intervals used in a former parent/child phrase set, the alterations in the form of semitone steps.

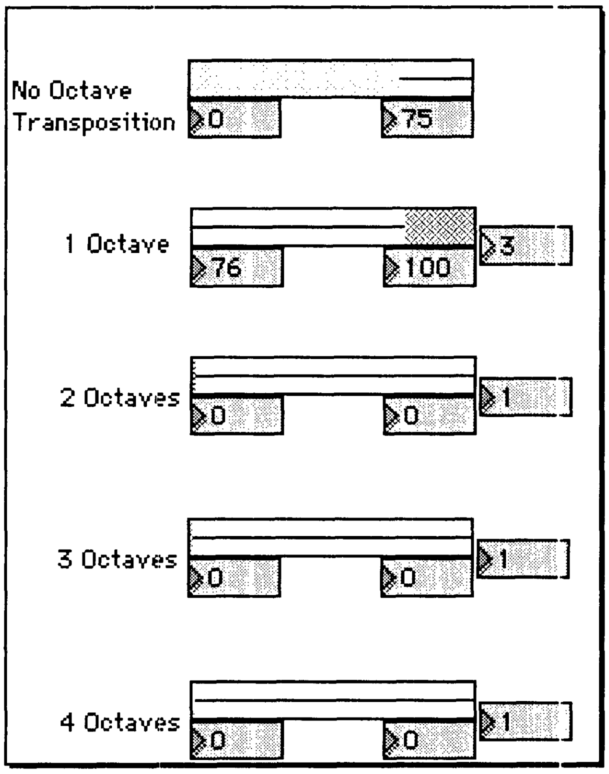
Phrase Garden also provides a function for the auditioning of generated phrases, in effect, by-passing the re-ordering of intervals in Variation Mode and the alterations to intervals in Mutation Mode. The auditioning function is controlled with a button object and a number box labelled Phrase Audition on the main screen. Once a set of parent/child phrases are generated, the Phrase Audition button may be clicked, and a phrase number selected in the number box. Clicking the main screen Start button results in the output from a Phrase 1 to 4 coll object that corresponds to the number selected in the number box. This function is used when a set of parent/child phrases contains possible solutions to a compositional problem, and the user desires either a repeat listening to a phrase, or to compare one generated phrase with one or more others.

Another *Phrase Garden* function related to pitch is the System Reset button in the top left corner of the main screen. This button is clicked whenever the user desires to clear the contents of the Phrase 1 to 4 coll objects, and begin anew with a new seed pitch. The function is also used when a change of starting pitch from one phrase to the next is not desired. As an example, a phrase that begins with middle C may be desired, and a first phrase generated by the program and stored in the Phrase 1 coll may be unsatisfactory. Clicking the System Reset button clears the unsatisfactory phrase from the Phrase 1 coll, and the program is then ready to receive another middle C seed pitch, and generate a new phrase starting with that pitch.

8.3.2.4. The Octave algorithm (p 8ves)

Another sub-patch in *Phrase Garden* that affects pitch is the sub-patch labelled p 8ves. In reference to Figure 8.19, this sub-patch is placed in the path of the downward pointing arrows that return pitch values to the Replication algorithm module. In the actual program, the sub-patch is placed between the p intervals/p child sub-patches, and the Replication algorithm pack object. The sub-patch is placed on the right-hand side of the main screen, and is double-clicked with the mouse to access settings for the control of octave leaps in generated pitch materials. Figure 8.20 shows the settings window for the p 8ves sub-patch.

Figure 8.20 *Phrase Garden,* p 8ves control settings window



No Octave Transposition 0 75

1 Octave 76 100 3

2 Octaves 0 0 1

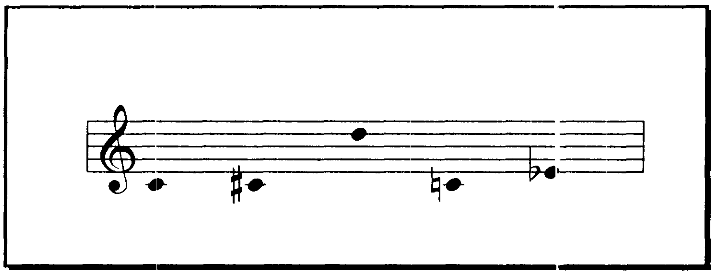
3 Octaves 0 0 1

4 Octaves 0 0 1

The settings controls in the p 8ves sub-patch allow a percentage weighting of interval leaps within a phrase, in the same way that the intervals of 0 to 11 semitones are weighted in the p intervals sub-patch. The design of the sub-patch is, in fact, virtually identical to that of the p intervals sub-patch. The direction settings of 1 (up), 2 (down) and 3 (either up or down) are also applicable with octaves, however, the either up or down setting is limited to a 50% chance of movement either way, and is not alterable.

The p 8ves sub-patch takes pitch values passed to it from either the p intervals or the p child sub-patch and applies a transposition of up to four octaves to pitch values, depending on the settings made in the sub-patch. As an example, the first five-pitch phrase of Figure 8.12 (interval series 1 1 2 3) may pass through the p 8ves sub-patch with the settings shown in Figure 8.20. With the setting of 80% No Octave Transposition, most pitch values will pass through the sub-patch without change. The 20% setting in the 1 Octave range bar will, however, result in approximately one fifth of all pitch values received being transposed by an octave. Figure 8.21 shows a possible alteration to the Figure 8.12 phrase as a result of the 20% 1 Octave setting. The third pitch (D above middle C), is transposed up an octave as a result of the p 8ves sub-patch.

Figure 8.21 *Phrase Garden,* Figure 8.12 phrase with octave transposition

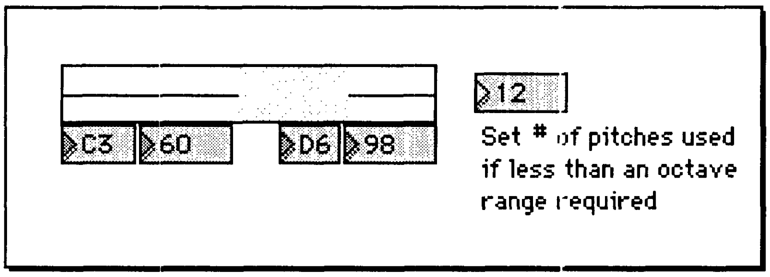


The p 8ves sub-patch was implemented in *Phrase Garden* due to limitations imposed by interval ranges of 0 to 11 semitones in phrases produced in both Variation Mode and Mutation Mode. The sub-patch was also implemented in accord with the compositional style in which interval leaps greater than an octave are commonplace. As an example, a similar phrase to the opening phrase of *Tears and Coloured Diamonds* excerpt shown in Figure 7.2 could be generated in *Phrase Garden*. The *Tears and Coloured Diamonds* phrase contains 12 pitch events, the second and last of which are vertical aggregates. Taking into account the intervals between single pitches, and between single pitches and pitches within the vertical aggregates, there are four intervals greater than an octave, equating to one third of the overall interval content. In *Phrase Garden*, a percentage setting of 67% in the No Octave Transposition range bar, and a setting of 33% in the 1 Octave range bar, will produce a phrase with similar leaps.

8.3.2.5. The Pitch Range algorithm (p pitch range)

The final sub-patch implemented in *Phrase Garden* for control of pitch material is the p pitch range sub-patch. This sub-patch is placed immediately after the p 8ves sub-patch, that is, between p 8ves and the Replication algorithm pack object. Appearing on the right-hand side of the main screen, the sub-patch is double-clicked with the mouse to access settings controls. As the label implies, this sub-patch provides control over the range of pitches present in a phrase. Figure 8.22 shows the settings controls of the sub-patch. A pitch range may be entered with either the number boxes below the range bar, or by clicking and dragging with the mouse in the range bar. Number boxes with pitch names and octave placements (C3 is middle C) are also available to set pitch range.

Figure 8.22 *Phrase Garden*, p pitch range control settings



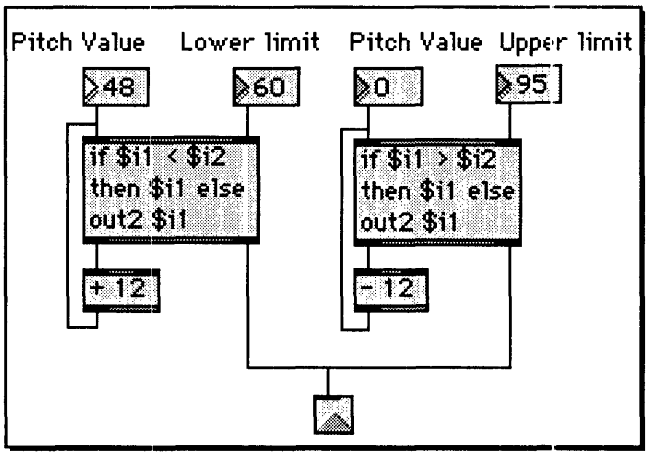
Set # of pitches used if less than an octave range required

As this sub-patch is the last in the series of sub-patches that generate and provide control over pitch, it is highly probable that numerous pitches received by the sub-patch will be out of the desired range. When a pitch is out of the desired range, it is sent to one of two hidden *MAX* if objects in the p pitch range sub-patch, depending on whether the pitch is lower or higher than the specified range. In Figure 8.23, a pitch value of 48 has been received by the p pitch range algorithm. This pitch is lower than the lowest pitch specified in the p pitch range settings of Figure 8.22 and is sent to the number box shown on the far left of Figure

8.23. The following if object compares the received pitch value to the lowest pitch value, 60 in the specified range, and when the value is lower, the if object sends the value out its left outlet. The following + object adds a value of 12 to the number, resulting in a new value of 60. The addition of the value of 12 results in the retention of the same note, but with an upward octave transposition. Following the addition, the number is returned in a loop to the inlet of the if object. Another comparison is made, in this case resulting in the value being equal to the lowest pitch permitted in the specified range. As the new pitch value is within the specified range, it is sent out the right outlet of the if object and then sent out of the p pitch range sub-patch and on to the Replication algorithm pack object.

The p pitch range loop is implemented to provide a pitch within the specified range, no matter how far a received pitch value is from the range limit. For example, if a pitch value of 12 is received, the loop through the + object would result in four increments to the value of 12, through values of 24, 36, 48, and finally the value 60, which is within the specified range. The if object on the right of Figure 8.23 functions in exactly the same way, but subtracts values of 12 from a received pitch value when the received pitch value is higher than the upper limit of the specified range.

Figure 8.23 *Phrase Garden,* p pitch range octave loop algorithm


Pitch Value
Lower limit
Pitch Value
Upper limit
48
60
0
95
if $i1 < $i2 then $i1 else out2 $i1
if $i1 > $i2 then $i1 else out2 $i1
+ 12
- 12

In general, the p pitch range sub-patch is provided to limit pitch ranges to those available on acoustic instruments when *Phrase Garden* is being used to generate phrases specifically for performance with such instruments. In Figure 8.22, for example, the range corresponds to that of the flute. The number box on the right of the p pitch range settings window, shown in Figure 8.22, is used to define a range of pitches that is less than one octave. When a number is specified in this number box, increments in pitch values in the + and – objects in Figure 8.23 are limited to the number specified in the number box. This function is used when, for example, there is a limited range of MIDI numbers assigned to different timbres, such as in a set of 11 or fewer samples in a MIDI sampling module, or a set of 11 or fewer percussion sounds in a MIDI sound module.

8.3.2.6. Summary

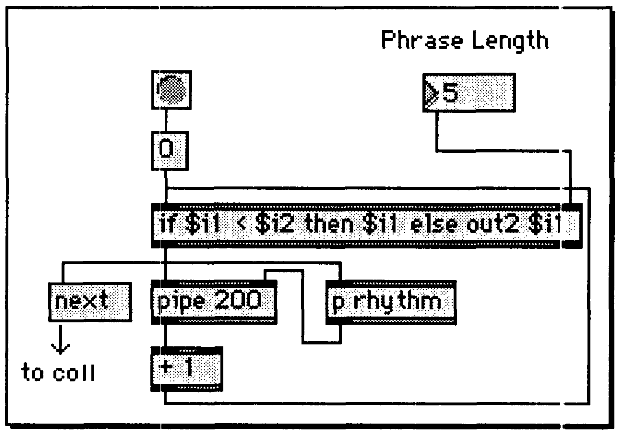
During the development of *Phrase Garden's* pitch algorithms, the Dawkins model proved to be highly valuable with regard to the initial development of phrase phenotypes similar to those used in the compositional style. The interval weightings, in Variation Mode, as genotypes, proved to be analogous to the rules Dawkins used to develop his archetypal phenotype, and, through the use of the either up or down interval direction setting, were successful in developing a range of child phenotypes within *Phrase Garden*. The phrases that result from the weighting of a set of chosen intervals are deemed to be sufficiently close in emulating phrases developed with the compositional style, and when pitches generated in Variation Mode are subject to octave transpositions, resulting phrases are very similar to those developed with the compositional style.

Interval alterations in Mutation Mode, as previously stated, represent a departure from the compositional style. This departure was deemed necessary in order to keep *Phrase Garden* completely in accord with the Dawkins model. What this departure did present, however, was a method of extending the existing compositional style. As opposed to the previous use of a single set of intervals provided in the note matrices, the alterations of intervals in *Phrase Garden's* Mutation Mode represent a method of further inherent growth of pitch materials within a composition. Such growth was explored in the development of the work entitled *When Cinderella's Monkey Comes Here, I'll Feed Him*, and is detailed in Chapter Nine of the study.

8.3.3. Rhythm generation algorithms

Figure 8.24 shows the main *Phrase Garden* Rhythm algorithm. On the main screen, the Start button object, the sub-patch p rhythm, and the Phrase Length number box appear, the remaining objects are hidden. When the Start button is clicked with the mouse, a bang (a positive action message) is sent to the message box containing the number 0. The 0 is sent on to the left inlet of the if object which has previously received, in its right inlet, a Phrase Length setting. In the if object, the value 0 is compared to the Phrase Length number. The if object determines that the number 0 is less than the Phrase Length number (the Phrase Length number must, by default, be a value more than 2) and sends the 0 out its right outlet. The number is sent to the message box containing the word next, this message recognised by a coll object in the Replication algorithm containing a generated pitch collection. The word next instructs the coll object to output the pitch value associated with the first index number, and after output, a pointer within the coll object is moved to the second index number, so that the pitch value attached to the second index number is ready to be sent out of the coll object.

Figure 8.24 *Phrase Garden,* Rhythm algorithm

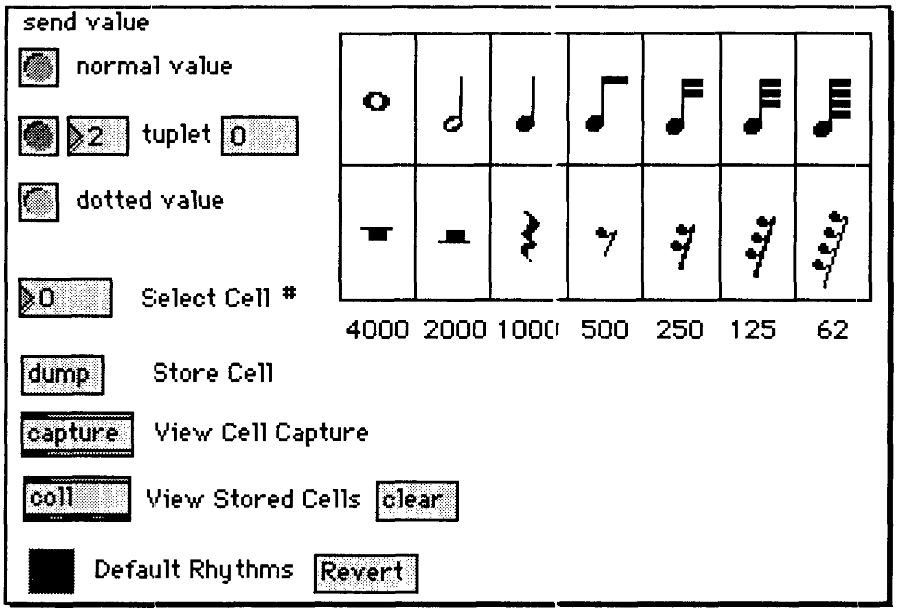


In the Rhythm algorithm, the 0 value is also passed through the pipe object, after which it is incremented in the succeeding + object by a value of 1. This new value is then sent back to the if object, forming a loop, or cycle. This cycle is repeated, with increments of 1 occurring to each number passing through the cycle, and the cycle continues until the end condition specified with the if object is met (i.e. when an incremented number is equal to the Phrase Length number).

The initial 0 value, and subsequent incremented values, in the Rhythm algorithm are also sent to the sub-patch labelled p rhythm. Within this sub-patch the values are converted to bang messages which trigger user specified values corresponding to a number of milliseconds. In the p rhythm sub-patch, a set of numeric values correspond to various millisecond lengths, and are also represented in the sub-patch with common notation symbols, the 1/4 note corresponding to 1000 milliseconds. Within the p rhythm sub-patch, the millisecond values are passed through another sub-patch labelled p tempo, in which the user specified millisecond values are converted to correspond to rhythmic values within a user specified tempo. The 1/4 note representation of 1000 milliseconds, for example, corresponds to one 1/4 note per second (MM 1/4 = 60), and a tempo of MM 1/4 = 120 would result in a millisecond value of 500 milliseconds. The millisecond values are sent out of the two sub-patches p tempo and p rhythm to the right inlet of the Rhythm algorithm pipe object and any numbers received in the left inlet of the pipe object are delayed by the number of milliseconds received in the right inlet. This delay in numbers provides rhythmic values for pitch values sent out of the Replication algorithm coll object. As an example, if a tempo of MM 1/4 = 60 is set, and a rhythmic value of 1/16th notes at that tempo are desired, the pitch values sent out of the coll object will be sent out 250 milliseconds apart.

Rhythmic values in *Phrase Garden* are set within the p rhythm sub-patch located beneath the playback controls on the right-hand side of the main screen. The sub-patch is double-clicked with the mouse to open the settings controls shown in Figure 8.25.

Figure 8.25 *Phrase Garden,* Rhythm control settings



Within *Phrase Garden*, rhythmic cells are mapped to interval values of 0 to 11, in a similar manner to that in which rhythmic cells are mapped to user defined symbols or words in *Symbolic Composer*. Each interval value is assigned one or more numeric values that correspond to millisecond lengths. As shown in Figure 8.25, a 1/4 note value corresponds to a numeric value of 1000, an 1/8th note value corresponds to a numeric value of 500, and so forth. Rhythmic values are stored in the coll object shown in Figure 8.25, and this object is double-clicked with the mouse to show stored values. Figure 8.26 shows the *Phrase Garden* default set of interval to rhythm mappings. Interval values are the first number in each horizontal row, and the one or more numbers that follow represent the rhythmic values (in milliseconds) mapped to the interval value.

Figure 8.26 *Phrase Garden,* Default interval to rhythmic cell mappings

```
0, 500;
1, 1000;
2, 500 500;
3, 333 333 333;
4, 250 250 250 250;
5, 200 200 200 200 200;
6, 166 166 166 166 166 166;
7, 500 500 200 200 200 200 200;
8, 250 250 250 250 250 250 250 250;
9, 166 166 166 166 166 166 333 333 333;
10, 200 200 200 200 200 200 200 200 200 200;
11, 166 166 166 166 166 166 200 200 200 200 200;
```

The rhythmic values shown in Figure 8.26 include values not shown in Figure 8.25. These values, 166, 200, and 333 represent tuplet values. As a 1/4 note value is represented by the value of 1000, a value of 333 will represent an eighth note triplet value (i.e., one-third of a 1/4 note value). The value 200 represents a quintuplet value, and the value of 166 represents a 1/16th note triplet value.

Clicking the Default Rhythms button, the button at the bottom of Figure 8.25, clears the default rhythms in the coll object, enabling the user to input desired rhythmic mappings. In the following example, the interval weightings shown in Figure 8.8 are used, providing the intervals of 1, 2 and 3 in a phrase. The interval 1 is mapped to a 1/4 note value, the interval 2, to two 1/8th note values, and the interval 3, to a 1/4 note triplet.

The first step in mapping rhythmic cells to intervals is to define, in the rhythm control settings window, each interval used in the generation of pitch material. Each interval is defined with the Select Cell # number box, and the defined intervals are stored temporarily in the following capture object. Following the example, the interval 1 is defined first. The next step is to select a rhythmic value to be mapped to the interval value. Rhythmic values are selected by clicking with the mouse on the desired common notation representations in the settings window. In the example, the desired value is a 1/4 note (1000). Once this value is selected, the button (normal value) in the top left corner of the window is clicked to send the rhythmic value to the capture object where the value is combined with the associated interval value. The next step is to send the contents of the capture object to the coll object, by clicking the dump message box. Once the contents of the capture object are sent to the coll object, the capture object is automatically cleared, ready to receive the next interval value. In the example, the next interval value is 2, defined with the Select Cell # number box. The required rhythmic cell contains two 1/8th notes. The common notation 1/8th note representation is clicked to select the 1/8th note value (500). The normal value button is then clicked twice, once for each of the two required 1/8th note values in the rhythmic cell. The interval value and two 1/8th note values are stored in the coll object by clicking the dump message box.

For the 1/4 note triplet values mapped to the interval 3, the common notation 1/4 note representation is selected. The button and number boxes below the normal value button are then used to define the triplet value. The left number box is used to define the number of tuplet values the 1/4 note value is divided by, in this case the number is 3. When this number is selected, the number box on the right displays the numerical value of 333, that is, one-third of the 1/4 note value of 1000. The button object on the left is then clicked thrice, once for each tuplet rhythmic value in the desired cell. Again, the dump message box is clicked to send the associated interval value and the defined rhythmic cell to the coll object. Figure 8.27 shows the contents of the coll for the example.

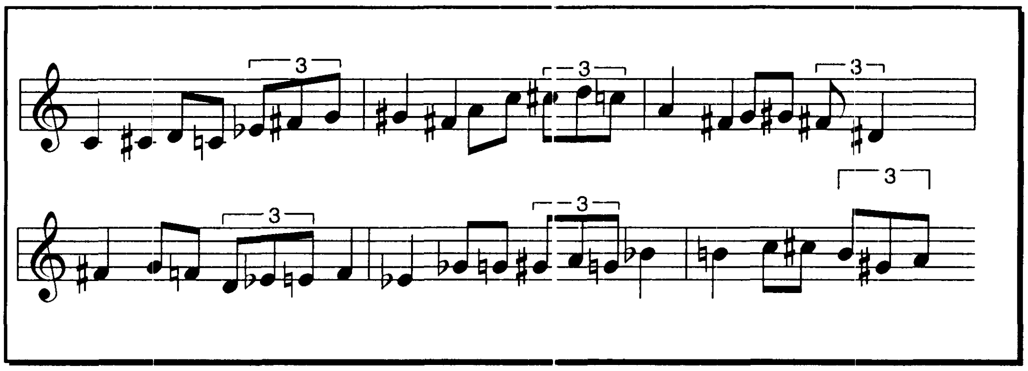
Figure 8.27 *Phrase Garden,* Example interval to rhythmic cell mapping

```
1, 1000;
2, 500 500;
3, 333 333 333;
```

Rest values are represented in *Phrase Garden* as negative numerical values. For example, a 1/4 note rest is represented with the numeral –1000, an 1/8th note rest with the value –500 etc. Rests are selected in the Rhythm algorithm settings window with the common notation rest symbols, and stored in the coll object with the same procedure followed for storing note values.

Once interval weightings in Variation Mode, or interval series in Mutation Mode, have been defined, and rhythmic mappings from intervals to rhythmic cells have also been defined, playback may be initiated with the Start button on the *Phrase Garden* main screen. Figure 8.28a uses the phrases shown in Figures 8.13 and 8.15 to illustrate the effects of the rhythmic mapping detailed above. In the first staff, the interval series of 1 1 2 3 is used. This provides a rhythmic series (1/4, 1/4, 1/8, 1/8, 1/8th note triplet) which is repeated to the end of the staff. In the second staff, the interval series is 1 2 3 1, providing a rhythmic series (1/4, 1/8 1/8, 1/8th note triplet, 1/4) which is also repeated.

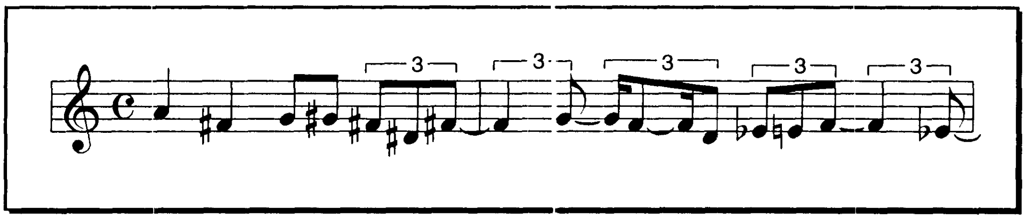
Figure 8.28a *Phrase Garden,* Interval series 1 1 2 3 and 1 2 3 1 with rhythmic mapping



Notable in Figure 8.28a is the 1/4 note value within the final triplet of the first staff. This value is the result of an edit made with *Finale* notation software, the edit made to provide a logical sequence of common notation rhythmic values that does not actually occur in the *Phrase Garden* output. As shown in Figure 8.13, the four phrases generated with the 1 1 2 3 interval series each contain five pitches, this number resulting from the Phrase Length setting of 5. This number is not in accord with the seven rhythmic values contained in the repeated rhythmic sequence. In *Phrase Garden*, the five-pitch phrases proceed using the defined rhythmic values until all 20 pitches (i.e. four sets of five pitches) are used. The rhythmic sequences are repeated only until all 20 pitches have been stated, and then immediately cease. This results in the possibility of rhythmic sequences ceasing before completion. In Figure 8.28a, the rhythmic sequence used on the first staff is stated in complete form twice, corresponding to 14 of the 20 pitches generated with the 1 1 2 3 interval series. The final six pitches correspond to the first six values of the rhythmic sequence, after which the rhythmic and pitch materials generated by the 1 1 2 3 interval series cease. Thus the seventh rhythmic value (a single triplet 1/8th note) is not used, and results in an incomplete triplet.

In the actual *Phrase Garden* output, the final 1/8th note triplet value of the first staff is replaced with the first rhythmic value generated in the next set of phrases. Figure 8.28b shows the actual *Phrase Garden* output in the final measure of the first staff, and the first measure of the second staff in common notation. The placement of the first rhythmic value (a 1/4 note value) within the final triplet of the first rhythmic sequence results in unusual, and generally undesirable, common notation groupings.

Figure 8.28b *Phrase Garden,* Figure 8.28a (mm.3-4) actual output



The method of generating independent pitch and rhythmic materials in *Phrase Garden* is analogous the 14th Century composition formalism of isorhythm. Generally in 14th Century motets, the *color* (pitch pattern) and *talea* (rhythmic pattern) are devised by the composer to be of equal length, or else there is some subdivision that enables the length of two or more *talea* to coincide with the length of a *color*. Such devising of rhythmic and pitch patterns to coincide with one another is possible in *Phrase Garden*, representing one solution to the problem of non-coinciding pitch and rhythmic patterns. Using the former interval series of 1 1 2 3, for example, the Phrase Length setting may be adjusted to provide seven pitches, to coincide with the seven rhythmic values provided by the rhythmic mappings. Alternatively, the rhythmic mappings may be altered to provide five rhythmic values as opposed to seven. A possible mapping is: interval 1 mapped to a 1/4 note value, interval 2 mapped to an 1/8th note value, and interval 3 mapped to two 1/16th note values. This mapping provides a rhythmic series of 1/4 1/4 1/8 1/16 1/16 with the interval series 1 1 2 3, the five rhythmic values coinciding with the five pitches generated by the interval series.

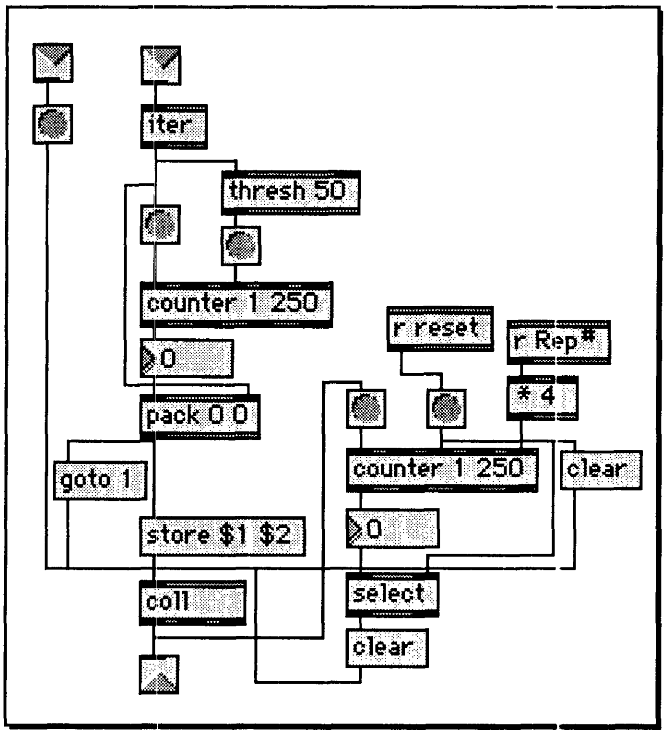
Some isorhythmic motets of the 14th Century do present unequal lengths of *color* and *talea*, for example the motet *S'Amours tous amans joir,* by Guillame de Machaut (Schrade 1956, pp.127-9). In this motet, a *talea* of five measures is repeated thrice. At the end of the third statement, the first measure of the pattern is repeated, extending the third repetition to a six measure pattern. This extension of the *talea* is analogous to the editing of *Phrase Garden* output within notation or MIDI sequencing software, as shown in Figure

8.28a, and represents a second solution to the problem of non-coinciding pitch and rhythmic patterns.

The editing of the rhythmic output from *Phrase Garden* within notation or MIDI sequencing software is a necessity when the output of the program is to be used by live performers using common music notation. Pure MIDI performances of *Phrase Garden* output do not, however, require any editing unless the MIDI output is used in a real-time collaboration with a live performer. A real-time MIDI performance will proceed regardless of whether or not the end of a rhythmic pattern coincides with the end of a pitch pattern. Whilst the current version of *Phrase Garden* does not have any algorithm implemented for solving the problem of rhythmic material coinciding with pitch material, such an algorithm could be implemented. The difficulty with the design of such an algorithm lies in determining a length for the final rhythmic value in a rhythmic pattern. One possibility that could be implemented is with a process of tracking the number of rhythmic values within a time signature. For this process, a measure of Common time would, for example, equate to a value of 4000 milliseconds with a tempo setting of MM 1/4 = 60. Where the number of values in the final measure of a rhythmic sequence is not equal to the number of millisecond values of a measure, the final rhythmic value could be extended to fill the measure. This solution is problematic however, due to the possibility that rhythmic sequences may end early in a measure, on the first beat for example. Filling the remainder of a 4/4 measure with a rhythmic value of three beats' duration may result in an undesired suspension of rhythmic movement. If the *Phrase Garden* output is to be edited in notation or MIDI sequencing software, the presence of the undesired rhythmic values is not problematic in that they may be altered to a desired length. In a real-time MIDI performance, the presence of the long rhythmic values is problematic in that they may result in an unchangeable presence of undesired suspensions of rhythmic movement. In both cases, the implementation of the algorithm is not beneficial, and for this reason, no such algorithm was implemented in the program.

The p rhythm sub-patch is sufficiently discussed in the preceding paragraphs as not to warrant a detailed explanation of the numerous *MAX* objects used to implement the interval to rhythmic cell mappings. The ordering and implementation of rhythmic cells within phrases according to the interval series is discussed below. This ordering is carried out in p rhythm, and also in another sub-patch within p rhythm labelled p cellorder, shown in Figure 8.29.

Figure 8.29 *Phrase Garden,* p cellorder sub-patch



In the p cellorder sub-patch are two inlet objects, shown at the top of Figure 8.29. The left inlet object receives the 0 value, and subsequent incremented values, from the Rhythm algorithm, following the clicking of the Start button on the main screen. The right inlet object receives rhythmic values stored in the coll object in p rhythm. When an interval series is generated with a seed pitch from the *Phrase Garden* keyboard, or a MIDI keyboard, the intervals are received by the p rhythm coll object which immediately sends out the stored rhythmic values corresponding to the interval values. As an example with the mappings shown in Figure 8.27, when the interval 1 is received by the coll object, the corresponding rhythmic cell containing a value of 1000 (a 1/4 note value) is sent out of the p rhythm coll object outlet to the p cellorder right inlet. Similarly, when the interval 2 is received, the corresponding rhythmic cell containing the two values of 500 (two 1/8th note values) are sent from the p rhythm outlet to the p cellorder inlet. With each rhythmic cell received, an index number is created in p cellorder for the purposes of storing the rhythmic values in a coll object in the p cellorder sub-patch. The index numbers correspond to singular rhythmic values received. For example, if the interval 2 is received, there will be two index numbers, one for each of the two 1/8th note rhythmic values in the cell. Figure 8.30 shows the contents of the p cellorder coll object after receiving the interval series 1 1 2 3 with the rhythmic mappings detailed previously.

Figure 8.30 *Phrase Garden,* p cellorder sub-patch contents with 1 1 2 3 interval mappings

```
1, 1000;
2, 1000;
3, 500;
4, 500;
5, 333;
6, 333;
7, 333;
```

When the Start button is clicked on the main screen, the 0 value is received in the p cellorder sub-patch left inlet, and triggers an output of the rhythmic value associated with the first index number in the p cellorder sub-patch. The number sent out is then subject to conversion in the p tempo sub-patch, to equate the number with a corresponding, common notation-based, millisecond delay at a desired tempo. From the p tempo sub-patch, the number is, as previously explained, used as a period of delay in the Rhythm algorithm pipe object, before the next number in the p cellorder coll object is stored as a new delay value. The successive incremented numbers from the Rhythm algorithm then trigger the output of consecutive rhythmic values in the p cellorder coll object, and the process is repeated.

The manner in which intervals are mapped to rhythmic cells in *Phrase Garden* corresponds to the initial manner in which intervals are mapped to rhythmic cells in the compositional style, for example, the interval 1 may, in both cases, be mapped to a single 1/4 note value. As a broader example, the initial numeric series for the *Tears and Coloured*

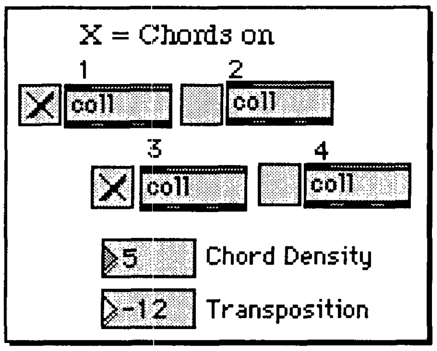
Diamonds excerpt shown in Figure 7.5 (1 2 4 5 4) may be mapped to rhythmic cells in *Phrase Garden* of 1 = 1/4, 2 = 1/8, 1/8, 4 = 1/16, 1/16, 1/16, 1/16, 5 = 1/16th note quintuplet. In Variation Mode, if interval weightings of 20% for intervals 1, 2 and 5 are defined, along with a 40% weighting for interval 4, the rhythmic output of the program will be limited to the defined cells, in the same manner as a pitch series from a note matrix in the compositional style corresponds to rhythmic cells mapped to a numeric series from a numeric series matrix.

Where *Phrase Garden* departs from the compositional style with regard to rhythm is in the development of further numeric series following the initial numeric series in a matrix, 3 6 9 9 5 for example, from the *Tears and Coloured Diamonds* excerpt. *Phrase Garden* will, at any one time, use only the rhythm mappings stored in the p rhythm coll object. However, the default rhythmic mappings may be altered to provide a second numeric series related to a composition in progress. By switching from a user defined set of mappings to an altered set of default rhythmic values with the Default Rhythms button, two sets of rhythmic series may be used consecutively in real-time. Generally however, *Phrase Garden* is designed to present numerous possibilities for phrases based on a particular set of intervals, allowing the composer to search the genetic space of phrases that are viable for a particular period within a composition. In the compositional style, there is only ever one numeric series used in any one phrase, and as this can be stored in the p rhythm coll object, further interval to rhythmic mappings are unnecessary until a new phrase is required. When a new phrase is required, a new set of interval to rhythm mappings may be defined.

8.3.4. Vertical aggregate algorithm

Vertical aggregates in *Phrase Garden* are controlled with the main screen sub-patch labelled p chords. The sub-patch is double-clicked to show the vertical aggregates settings window, shown in Figure 8.31.

Figure 8.31 *Phrase Garden,* p chords settings window



5 Chord Density
-12 Transposition

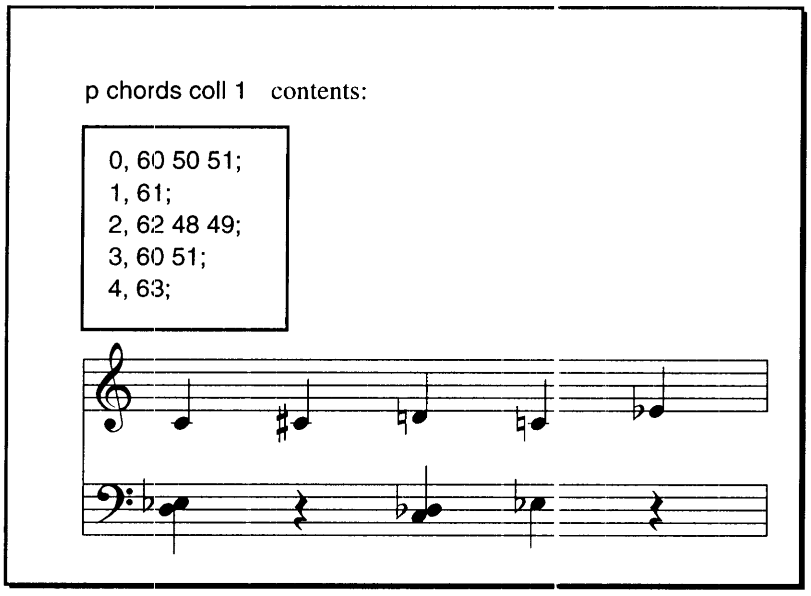
The four coll objects in p chords receive and store identical data to the four coll objects on the main screen labelled Phrase 1 to Phrase 4. Hidden within the p chords sub-patch, however, are numerous *MAX* objects that determine the pitch data being received in the coll objects, and add to this data further pitch data based solely on the data received. The amount of added pitch data is controlled with the Chord Density setting. If a setting of 5 is used, five pitch values will be randomly added to the pitch collection in each coll object. Clicking the box next to each coll object places a check mark in the box and indicates that the coll object is enabled to output any data it contains. When there is no check mark, data stored in the main screen Phrase 1 to Phrase 4 coll objects is used for output. Figure 8.32 shows the contents of the main screen Phrase 1 coll object, and the contents of the p chords coll object labelled 1. The content is based on the former interval weightings and directions, that provided the interval series 1 1 2 3.

Figure 8.32 *Phrase Garden,* Phrase 1 and p chords coll 1 contents

```
Phrase 1 coll contents:

0, 60;
1, 61;
2, 62;
3, 60;
4, 63;

p chords coll 1 contents:

0, 60 62 63;
1, 61;
2, 62 60 61;
3, 60 63;
4, 63;
```

A further control in the p chords settings window is the Transposition control. With the setting of –12 shown in Figure 8.31, each pitch value that is added to the coll is transposed down 12 semitones. A non-negative setting of 12 would transpose each added pitch value up 12 semitones. The Transposition setting may contain any number, for example a setting of –1, or a setting of 19 are both possibilities. The added pitch values shown in Figure 8.32 are shown in transposed form (–12) in Figure 8.33, along with the common notation of the phrase with the added pitches.

Figure 8.33 *Phrase Garden,* p chords coll contents with transposed pitch values



p chords coll 1 contents:

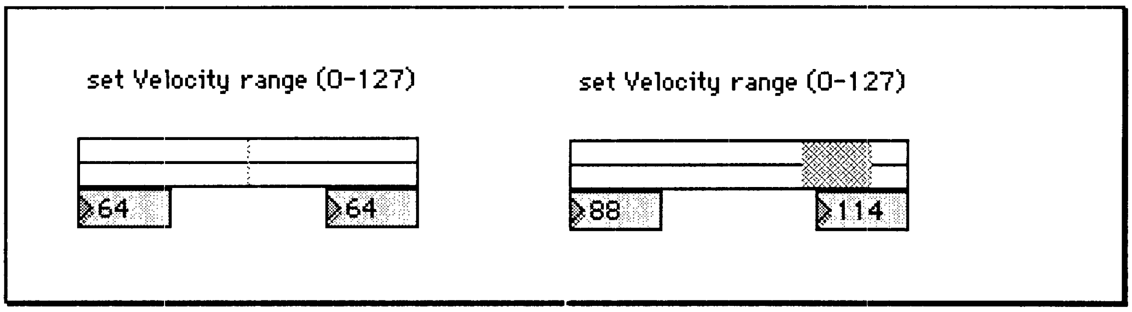
```
0, 60 50 51;
1, 61;
2, 62 48 49;
3, 60 51;
4, 63;
```

As previously described, vertical aggregates in the compositional style are formed intuitively from notes in a matrix note set that is currently in use. Considering the general lack of emphasis placed on vertical aggregates in the compositional style, the algorithm for vertical aggregates in *Phrase Garden* is deemed sufficient for the generation of vertical aggregates, with pitch values from a current pitch collection added randomly to the collection to form vertical aggregates. Where the results of the random placement of added pitches is unsatisfactory, these pitches may be edited in notation or MIDI sequencing software, or further phrases may be generated using the same interval structures until desired results are achieved.

8.3.5. The Velocity algorithm (p velocity)

The Velocity algorithm in *Phrase Garden* is similar to the Velocity-range algorithms in *M* and *Jam Factory*. Both deterministic and stochastic settings are available, and the algorithm may be controlled in real-time. The Velocity algorithm in *Phrase Garden* is provided purely for the enhancement of the real-time performance and auditioning of generated phrases. The Velocity algorithm is contained within the sub-patch p velocity, located on the lower right of the main screen, and is double clicked to access velocity settings. Figure 8.34 shows two examples of settings in the Velocity algorithm settings window. The left setting is deterministic, with the minimum and maximum in the velocity range set to 64 in the MIDI scale of 0-127, this number representing a continual output of *Phrase Garden* pitches at a dynamic level of *mp*. The right setting is stochastic, the dynamic range of the output with this setting randomly varied in a dynamic range of *f* to *fff*.

Figure 8.34 *Phrase Garden,* p velocity settings



set Velocity range (0-127)
64 64

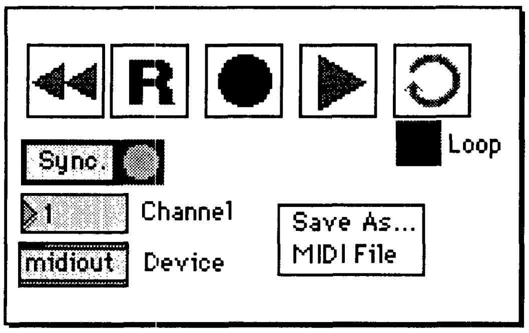
set Velocity range (0-127)
88 114

8.3.6. Output

Data from the coll objects on the *Phrase Garden* main screen, and/or from the coll objects in the sub-patch p chords, is sent to a sub-patch labelled p output shown on the left-hand side of the main screen. This sub-patch contains a hidden *MAX* object labelled makenote which supplies a MIDI note-off message for pitch values received from coll objects, and a length of delay between the MIDI note-on message and the supplied MIDI note-off message is received by makenote from the Rhythm algorithm pipe object. The makenote object also receives velocity values from the p velocity sub-patch and combines the velocity values with received pitch values. The combined data is sent from makenote to the noteout object shown on the left of the main screen, the noteout object then sending the data through a Macintosh serial port to an external MIDI device.

Within the p output object, controls for recording data from the makenote object are accessed when the object is double-clicked. The controls, shown in Figure 8.35, consist of, from left to right, a button that deletes any blank space that may be recorded before the start of recorded MIDI data, the record button, the stop button, the play button and a loop button.

Figure 8.35 *Phrase Garden,* p output controls window



A recording using the p output recording controls is initiated with the record button, before the Start button on the main screen is clicked to provide MIDI data for a sequence. MIDI data is supplied to the makenote object and is heard in real-time, and following the clicking of the p output record button, MIDI data is also received by a hidden *MAX* seq object that stores the real-time performance as a MIDI sequence. Recording may be stopped by clicking the p output stop button. Stored sequences can be played back by pressing the play button, and if the loop button is clicked, the stored sequence is played back repeatedly.

The Sync button allows the stop and play buttons in the p output sub-patch to be synchronised with the Stop and Start buttons on the main screen. Synchronisation is linked to computer keyboard keys, the space bar initiating playback of a stored sequence in p output along with further generated MIDI data, and the return key stopping playback of a stored sequence and further generated MIDI data. The midiout object and the Channel number box allow the selection of an external MIDI device and MIDI channel for playback of stored sequences. By changing the MIDI channel and/or MIDI device, a stored sequence may be played in combination with newly generated MIDI data, with each using a different MIDI timbre. Clicking the Save As... MIDI File box brings up a standard Macintosh *Save As* dialogue box, enabling a stored MIDI sequence to be saved as a Standard (Type 0) MIDI File.

Phrase Garden has four independent voices that each has its own version of the main screen, which includes all objects previously discussed. The windows for each voice are shown by selection of a voice number from a 'Voices' menu on the Macintosh menu bar. Each voice functions independently, and the output of each voice may be assigned to different MIDI devices, instruments and channels. Each voice has a p output sub-patch, and therefore, there may be four sequences recorded, one in each voice, and each voice may then generate new MIDI data whilst the formerly recorded sequences are playing. In total then, there is the possibility of having eight simultaneous performances of MIDI data in real-time.

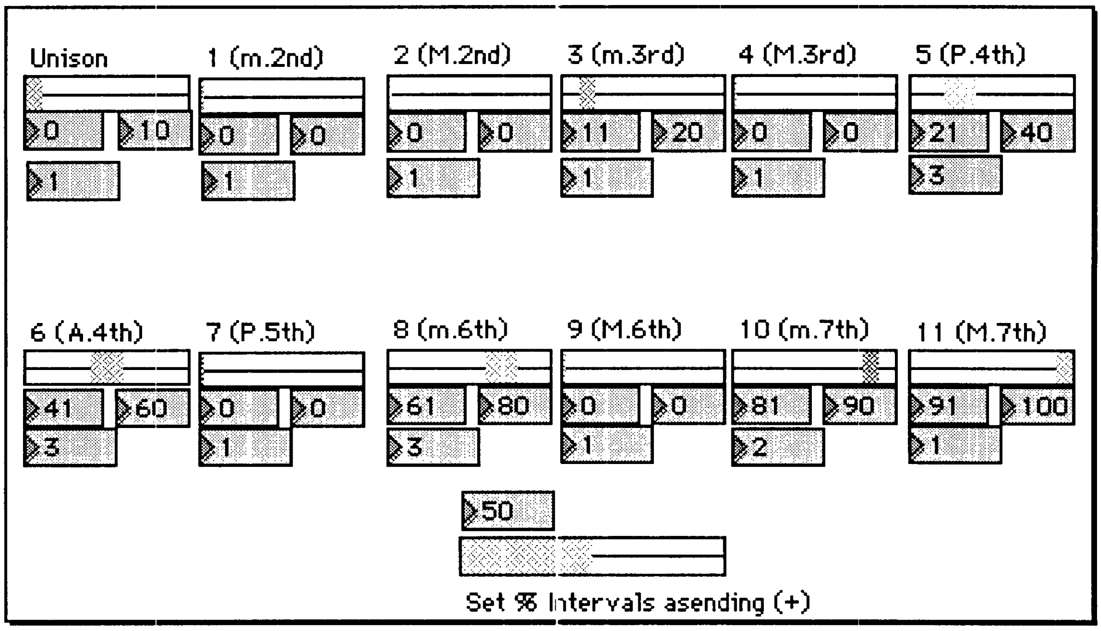
8.4. Summary

The inherent growth noted in the compositional style note and numeric matrices led to an examination of various computer-based systems that simulate biological growth, in particular, systems from the field of A-Life. Of the systems examined, Richard Dawkins' program *Blind Watchmaker* proved viable as a model for the development of *Phrase Garden*.

As a summary of the *Phrase Garden* algorithms detailed in this chapter, a further example is presented in which the opening phrase of *Tears and Coloured Diamonds*, as shown in Figure 7.2, is used as a basis for a development of phrases in the *Phrase Garden* environment. Certainly the phrase as it appears in the work could be closely duplicated in *Phrase Garden* by using Mutation Mode and precisely defining, interval by interval, the interval content of the original phrase, along with the direction each interval takes. Some variation to the phrase could appear however, with the random-based, percentage weighting of octave occurrences within the phrase Whilst close approximations of pre-conceived musical material are a possibility in *Phrase Garden*, to endeavour to duplicate phrases is not a primary focus of the program. Rather, the program is designed to present musical materials based on underlying interval content.

More suited to the example is the use of Variation Mode, with settings of interval weightings identical to those in the original example. For the generation of a new phrase, the initial grace note of the original is ignored, providing an uncomplicated ten interval series that maps directly to percentages, and in the two, two-pitch, vertical aggregates of the original, the upper pitch is momentarily ignored. In the original, the intervals are, in order of appearance, 0 (actually an octave), 8, 5, 6, 5, 10, 8, 11, 6, 3. Intervals 0, 3, 10 and 11 all occur once and have a percentage occurrence of 10%, the remaining three intervals all occur twice and have a percentage occurrence of 20%. With a ten interval series, a Phrase Length setting of 11 is set on the *Phrase Garden* main screen. The percentages are then used as the weightings in the p intervals settings window, as shown in Figure 8.36. The directions of the intervals that occur once are set to correspond to the directions those intervals take in the original, and for the remaining intervals, the either up or down direction setting (3) is used.

Figure 8.36 *Phrase Garden,* Example interval weightings



Unison	1 (m.2nd)	2 (M.2nd)	3 (m.3rd)	4 (M.3rd)	5 (P.4th)
0 10	0 0	0 0	11 20	0 0	21 40
1	1	1	1	1	3
6 (A.4th)	7 (P.5th)	8 (m.6th)	9 (M.6th)	10 (m.7th)	11 (M.7th)
41 60	0 0	61 80	0 0	81 90	91 100
3	1	3	1	2	1

50

Set % Intervals asending (+)

In the original, there are three leaps of an octave or more (i.e. a 33% probability of occurrence). To correspond to these leaps, a setting is made in the *Phrase Garden* p 8ves sub-patch settings window of 67% No Octave Transposition, and 33% 1 Octave.

The rhythmic content for the example is a single cell that corresponds to the entire rhythmic content of the original. This cell is mapped to each interval that occurs so that no matter which interval occurs first in the generated phrases, it will trigger a rhythmic cell that will last for the duration of the phrase. Figure 8.37 shows the contents of the p rhythm coll object.

Figure 8.37 *Phrase Garden,* p rhythm coll contents

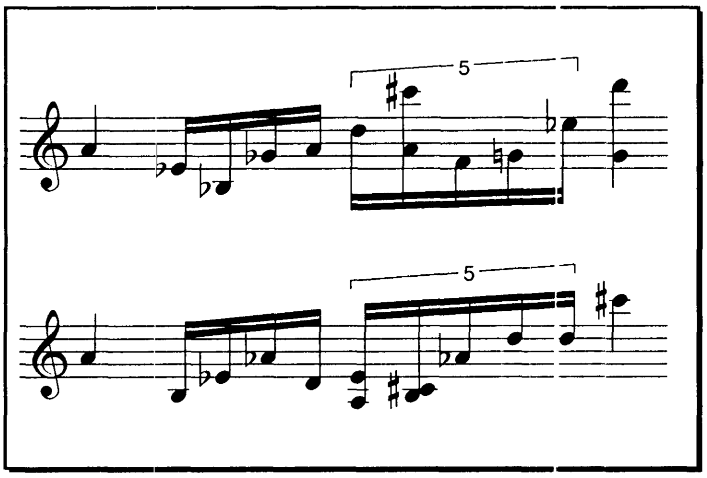
```
0, 1000 125 125 125 125 100 100 100 100 100 1000;
1, 1000 125 125 125 125 100 100 100 100 100 1000;
3, 1000 125 125 125 125 100 100 100 100 100 1000;
5, 1000 125 125 125 125 100 100 100 100 100 1000;
6, 1000 125 125 125 125 100 100 100 100 100 1000;
8, 1000 125 125 125 125 100 100 100 100 100 1000;
10, 1000 125 125 125 125 100 100 100 100 100 1000;
11, 1000 125 125 125 125 100 100 100 100 100 1000;
```

In the p pitch range sub-patch, the range of pitch for the example is set to correspond to the range of pitch in the original (i.e. from A2 up to E5), where middle C is C3. In the original, there are the two occurrences of vertical aggregates, the upper pitches of which are within an octave of the lower pitch. In the p chords settings, there is a Chord Density setting of 2, which provides pitch values for the vertical aggregates that were previously ignored in the interval settings. The placement of the upper pitches within the octave, and also the fact that both upper pitches occur elsewhere in the original phrase, indicates that a Transposition setting of 0 (no transposition) is required in the p chords settings.

Figure 8.38 shows two different phrases generated with the above settings. In the first phrase an interval series of 6 5 8 3 5 11 8 10 10 11 is generated. A downward octave leap with the second interval of 8 occurs, resulting in a leap of 20 semitones, and an upward octave leap occurs with the first interval of 10. As the interval of 10 moves downward, the resulting G pitch, with the upward octave transposition, is only two semitones higher than the former F pitch. In both vertical aggregates in the phrase, the added pitch appears below the pitch generated in the single line material.

In the second phrase, an interval series of 10 8 5 6 5 8 5 6 0 11 is generated. With both intervals of 8, the interval is descending, and an upward octave transposition occurs. The final interval of 5 is descending and also occurs with an upward octave transposition. The position of the added pitch in the vertical aggregates is, in the first, the E above the generated single line pitch of A, and in the second, is the B below the C#.

Figure 8.38 *Phrase Garden,* Phrases based on *Tears and Coloured Diamonds* opening



Whilst the actual pitches presented in the two phrases of Figure 8.38 differ from those in the original *Tears and Coloured Diamonds* phrase, all three phrases share similarities in their underlying interval content. Such similarities make phrases such as the two in Figure 8.38 viable alternatives to phrases such as those presented in a composition like *Tears and Coloured Diamonds. Phrase Garden's* strength lies in this presentation of viable alternatives for phrases at any point in a composition, the program, in general, allowing the composer to search the genetic space of phrases based on desired interval content.

With regard to the compositional style, the numerous algorithms within the program combine, as shown in the former example, to present phrases that closely resemble phrases developed with the compositional style: the use of Variation Mode allows the specification of a desired interval content that may be weighted to present phrases that are similar to phrases presented with the compositional style; *Phrase Garden* rhythmic cells are linked to interval content in a similar manner to the linking of rhythmic cells and intervals in the compositional style; disjunct interval movement in *Phrase Garden* is achieved with weightings on octave transpositions of pitch which simulate the disjunct interval movement of the compositional style; and vertical aggregates in *Phrase Garden* are derived from pitch materials already existing in a phrase, in a similar way to the manner in which vertical aggregates in the compositional style are derived from notes of a currently used note series.