Aho, Alan, Lavoie, Don, & Paprocki, James. 1975. "MUSIM - Simulation of Music Composition Using GASP," Proceedings of the 1974 Winter Simulation Conference, Vol. 2, WSC/SIGSIM of the Association for Computing Machinery, Inc., 1975.

MUSIM: SIMULATION OF MUSIC COMPOSITION USING GASP

Alan C. Aho,[1] Donald C. Lavoie,[2]

and James N. Paprocki[3]

[1]Travelers Insurance Company

[2]Consolidated Edison of New York

[3]Eastman Kodak

ABSTRACT

Most of the efforts at computer composition of music which we read about were either semi-random generation limited by parameters obtained through statistical analysis of previous works or based on the relatively straightforward (mathematically) twelve-tone system. It was our intent to create a fusion of these two methods. MUSIM combines semi-random generation with the harmonic rules of 19th century classical music in the hope of obtaining pleasing sounding music with enough surprises to keep it from becoming boring. Of course, this is also partially a function of the complexity of the rules you impose on the system. Our program is such that it could easily be adapted to twelve-tone composition rules.

I. GENERAL INFORMATION

The tonal range of notes is represented by the integers from 0 to 96. The logical array SKALE chooses which of the 12 notes in each octave comprise the standard scale, thus by changing this array any mode, including 12 tone with all Trues, can be used. The two dimensional logical array CHORD is a set of preferred chordal patterns the frequencies of occurrence of which are determined from other variables. The notes as they are generated are matched against these patterns in the computation of their weights.

Melody notes are scheduled as GASP events. Characteristics of the note, stored in the GASP array ATRIB are set at the time of scheduling. The tone of the note, the time of its occurrence, its duration and the chordal pattern which prevails at this time are temporarily stored. At the end of each measure the accompanying voices are added and the completed measure is written out to what will be the input file for the plot routine. At present the accompanying voices are weighted in accordance with basic rules of four part harmony (see Piston, Harmony). By adding conditions and varying weights up to 9 accompanying voices could be chosen on any criteria desired.

Weighted Random Conditions Method

To choose melody notes, bass notes, and other accompanying voices a weighted random condition method is used. Various conditions, as for instance the proximity to the last note, adherence to particular rules of Harmony, etc., are given weights. For each possible note (or, in the case of accompanying voices, for each possible permutation of notes) a total weight is computed. From the best few total weights the decision is then made randomly (favoring the best). This method is quite flexible, allowing for the adding of conditions and the varying of their weights.

Weighted Random Variables Method

Many specific logic decisions are made by simply setting a variable to a constant probability and comparing a random number against this variable. Such variables are, of course, subject to change by the user at any time during the simulation.

II. GASP

GASP is a package of FORTRAN subroutines designed specifically for discrete simulation of physical systems, with special emphasis on statistics-gathering, which is generally used to study characteristics of the system under consideration. We used GASP for our project because its filing system and handling of time were particularly well suited for this application.

The major filing array is NSET, a two-dimensional array. Each column of NSET can contain one event, each of which has certain attributes. In our case a typical event was a note.

NSET can be divided into up to eight files. File one is assumed to be the event file. By this it is meant that pending future events are always stored in file one. The other files can be used either for storage, as in our case, or as secondary event files.

A typical GASP program consists of a user written main program, and a subroutine called EVNTS. Also subroutines are written by the user to take certain actions on specified events. The main program's primary purpose is to call subroutine GASP, which initializes NSET & sets constants to values specified by the user, such as the number of statistics to be gathered and other data.

Once this is done, NSET is searched for the first chronological event. This event is removed, TNOW, the current time, is set to the value of the first attribute of the event just removed and this event is deposited in the array ATRIB.
ATRIB(2) is called the event code. Once an event has been removed by GASP, control is passed to

EVNTS. EVNTS is basically a computed GO TO which determines the flow of the program on the basis of the value of ATRIB(2).

After taking appropriate action, the program returns to EVNTS which returns to GASP, and the sequence starts all over.

If the user wishes to file events into NSET or remove them, he simply calls FILEM or RMOVE. There are two types of statistics which can be gathered. One is normal frequency data, like the number of events processed. This is done by calling COLCT, with the facility for handling up to 30 different statistics of this type. Also, time integrated statistics can be taken by calling TMST. This would be done to find the time weighted average of a parameter; for example, to see how much of the time a certain queue is empty, or to find the time weighted average of the number people in a certain queue.

Since the user must schedule events in the future for the simulation to continue, various distributions are provided, such as ehrlang, poisson, normal, and uniform, in order to approximate the characteristics of a given system.

III. MUSIM'S GASP ATTRIBUTES AND FILES

In GASP the most crucial information about any scheduled events must be stored in the major storage array NSET. Each column of this array represents all the attributes of one stored element. When these elements are removed from NSET they are sent to the array ATRIB.

Thus ATRIB(1) is the first row of NSET corresponding to whatever column has just been removed. ATRIB(1) is the value upon which the order of storage in NSET is determined.

ATRIB(2) for the Event File is the event code which directs the logic to one of a number of subroutines. In MUSIM the options are Inishl, Change, Endsm, Melody, Submel, Startr and Stopr (see Flow Chart). GASP removes a scheduled event from the Event File and ATRIB(2) instructs the program as to which type of event this particular one is.

The Event File for MUSIM uses the following ATRIBs for the MELODY and SUBMEL events (the others use only ATRIBs 1, 2 and/or 7):

ATRIB(1) Time of event, that is, the time of occurrence of the note

ATRIB(2) Event code 1 for MELODY, 7 for SUBMEL

ATRIB(3) Voice number. 1 for all melody notes. Accompanying voices will be numbered down from the melody to the bass.

ATRIB(4) Tone of the note. A number from 0 to 96. 48 is middle C.

ATRIB(5) Duration of the note, quarter .250, eighth .125, half .50 etc.

ATRIB(6) Predominate chord to be played with this melody note. Used in choosing harmonies later. Integers 1-7 represent standard triads, more complex chords can be added.

In MUSIM File 2 is used to accumulate the melody notes for a measure. File 4 is used to integrate these melody notes with their harmonies as a preliminary for outputting all this information. File 3 is for all theme, melody pattern and rhythm pattern storage and later retrieval.

The bulk of the programming is through either of these two routes, MELODY or SUBMEL. From either route at the end of each measure all accompanying voices are filled in by subroutine HRMONY. Subroutine TIMING handles all aspects of tempo and rhythm and maintains appropriate values in ATRIBs 1 and 5, time of occurrence and duration. Both subroutines MELODY and SUBMEL reschedule events of their respective types. MELODY generates melody notes using the weighted random conditions method and SUBMEL handles repeated rhythm patterns, melodic patterns or both which it finds in File 3.

Subroutine STARTR and STOPR respectively start and stop the recording of these melodic or rhythmic patterns in File 3. ENDSM, meaning end of simulation, wraps up final statistics. INISHL, obviously, initializes the MUSIM variables. INISHL also calls subroutine CHANGE. This subroutine interacts with the teletype to allow for any changes of any of the significant parameters of MUSIM. This routine is a regular event and can be scheduled at any time. Thus for example the conditions for the production of melody notes can be changed if the melody up to this point does not appeal to the composer at the terminal. In this way a user of MUSIM can run a long simulation changing parameters throughout until he finds a combination of conditions which produces interesting music. Theoretically.

The timing routine generates the rhythm for one measure which is used to construct the melody pattern. The rhythm sequence is generated at the beginning of each measure returning a time duration for each melody note each time the routine is entered. The rhythm of a measure is chosen by randomly picking predetermined rhythm patterns of one to four beats of varying durations which will occur on the beat. In this manner another measure is filled and the time duration is returned.

IV. PLOT ROUTINE

One common fault of computer generated music is that it is described in terms of numbers and letters. This is the way the output from this simulation is represented, as numbers indicating tone, duration, and the time that tone occurs. Using this output as input to a plot routine we can write the music in standard notation so our concert pianist can read and play it.

The plot routine reads the simulated music one

measure at a time and determines the placement of the notes along the horizontal line. Thus a measure will not be split in two. It then determines the vertical placement of each note, connecting the stems of a chord where appropriate. A difference of duration between two notes at the same time is recognized in which case the stems will not be connected. Quick beats (i.e. less than a quarter note) will be recorded and grouped to the equivalent count of a quarter note. Thus four sixteenth notes that comprise of one count will have their tails connected.

Each note is raised or lowered on the staff depending on the tone and initially each note is assumed to be in the key of C. An array containing the step increments determines whether the note is an accidental or not. And if some other key than C is indicated this array is just transposed along the scale. Rests are the absence of notes in our case. Therefore when reading the notes for the measure the vacancies are recorded and the appropriate sign for a rest is written. If any note has a duration continuing past the end of a measure it must be continued into the next measure. Thus the note is broken into two parts and tied to each other between measures. A tie might also occur where there are two notes in a chord of different duration. The longer note will be split, the first part being equivalent to the other note in the chord and the other will be tied to the first.

In this way we can see what our music looks like and have it played for us, getting an idea what we might change to improve the composition.

V. MELODY AND RHYTHM REPETITION

In order to create a greater sense of continuity and coherence in the output music, it was decided to provide facilities for the repetition of melody and rhythm patterns during the process of composition. This is accomplished by scheduling an event, which when encountered in the course of a run, initiates the storage of either a short melody pattern, a short rhythm pattern, or a longer 'theme'. The time of the end of this sequence is determined before recording begins and an end-of-recording event is scheduled for that time.

Immediately after the end of a recording sequence, control is passed to a subroutine which is responsible for preparations necessary for the repetition of the previously stored sequence. In the case of a short melody sequence, a rhythm has to be found to use simultaneously. The new rhythm can be either a pattern previously stored or an entirely new one made by calling TIMING as in the primary composition process. The number of desired repetitions of the sequence is then determined. Pointers are set up to find the appropriate patterns in storage. Within the repetition subroutine are various switches which are set at the beginning of each repetition. These include switches which determine whether or not the melody or thythm pattern will be reversed, and whether or not an exact repetition of the sequence will occur. Normally the melody pattern is displaced by a different amount on each repetition, so a step size is determined.

In a normal melody repetition, the general sequence of events is as follows:

1) File previous note in music output file (which is cleared at end of each measure).
2) Switches for reversal set.
3) If no rhythm sequence, TIMING called on every note, otherwise only at beginning of measure-HRMONY called by TIMING at end of measure.
4) CADNTZ called if first note in melody sequence encountered during final repetition.
5) TIMING called if no rhythm pattern selected.
6) Melody note determined 2 ways:
   a. MELODY called on first note of each repetition and tonal displacement for pattern determined at this time. For the rest of the melody repetition the displacement added to the value of note in pattern to determine new tone value.
   b. MELODY called on every note.
7) KORD called to determine chord to be used, based on newly determined note and value of last chord using chord progression rules.
8) Assignment of attribute values and filing of note.

In a rhythm sequence, one has the option of specifying a simple melody sequence to run concurrently or generating a different melody as in primary music generation. In the first case, the procedure is almost identical to a melody sequence with a rhythm pattern, except that the thythm pattern is used as the unit of repetition rather than the melody pattern. If no melody pattern is specified, KORD is called and the new melody note is determined afterwards by calling MELODY. Calling KORD first insures a better chord progression. This cannot be done if a melody pattern is used, because the melody note is predetermined and the chord is partially dependent on the value of the new note.

VI. THEME REPETITION

Although facilities for the repetition of themes have not as yet been implemented, the general structure of such a section has been established. Such a stored theme is different from the other types in that all harmony is included with the theme, whereas a melody repetition contains only the melody notes. Many of the same subroutines will be used, in addition to several new ones. The initial stages, including setting up pointers and determining the number of repetitions, will be the same. Once control passes to the main theme repetition subroutine the procedure is much different.

There are many different techniques used in varying a theme. Among them are:

Melody Variation
1) Preserving original melody.
2) Mirror writing (reflecting melody through tonic).
3) Preserving melody with addition of a pattern of ornamentation.
4) Drop melody and use second voice.

5) Create entirely new melody.

Harmony Variation
1) Keep harmony.
2) Preserve chords, change actual notes in chords.
3) Completely alter harmony.
4) Keep chords at beginning of measure.

Rhythm Variation
1) Keep timing same.
2) Change timing while partially preserving old rhythm (halving or doubling time).
3) Alter timing completely.

There will be an array called SWITCH(3) with one element each for harmony, rhythm and melody. At the beginning of each variation, each element of SWITCH will be set according to the number of the corresponding variation technique to be used, being careful not to select incompatible techniques, such as mirror-writing and preserving harmony.

After the techniques have been selected, the program will branch to the appropriate subroutine to perform the indicated function. The main theme routine could be divided into sections for melody, rhythm and harmony. For each permutation of techniques an order of execution for these three sections would be established. For example, if the melody was to be mirrored, and the harmony changed, the melody section would be executed first so that the new harmony would contain the right melody. If it was decided to keep the harmony and change the melody, the melody would have to be made available first so that the harmony could be based on it.

This type of variation would be another step up from semi-random generation. Not only would melody and rhythm patterns be incorporated into the fabric of the work, but the harmonic and melodic structure could be subtly changed while maintaining the continuity of the music, thus lending it more diversity while retaining coherence and form. This structure has the obvious advantage of being easily expandable. Any additional variation techniques can be added as separate modules.

VII. CONCLUSION

There are, of course, practically an infinity of directions to take a program of such broad scope. Limitations of time prevented us from pursuing more than a few approaches.

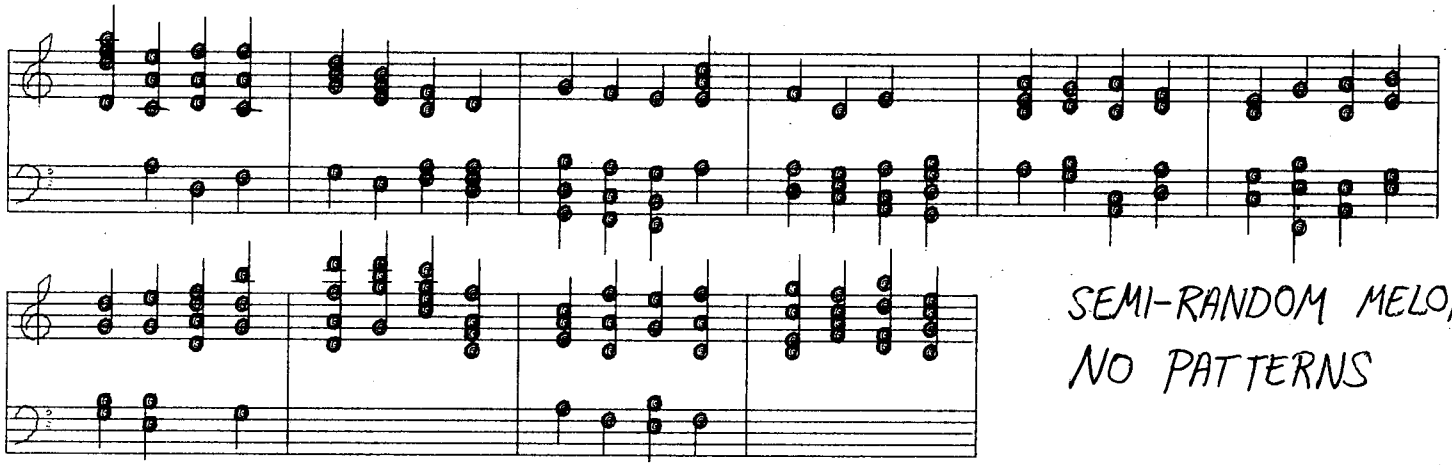
Vivid imagination can usually envision great possibilities for computer music generation. We set out to do far more than we ever got around to. And, we insist, if we only had a little more time.....

The structure of our program is conducive to some interesting possibilities. Repeating a melodic pattern at different voice levels could be the beginnings of a fugue. Harmonizing around a repeated bass line could produce a passacaglia.

One of the directions which ought to be investigated before such lofty goals is the implementation of a dissonance-to-consonance flow routine. Music without dissonance tends to be uninteresting, but music with too much or improperly structured dissonance tends to be harsh sounding. The balance here is at the same time crucial to harmonically structured music and very difficult to break down into objective logical programming.

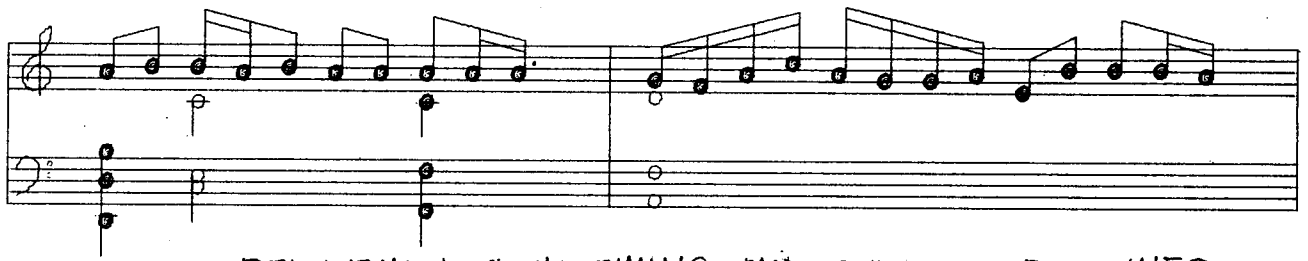
MUSIM is just a beginning. The foundation is there. The general structure for a complex discrete simulation of the process of creative music composition has been built. It is, we feel, a good structure to serve as the basis for the deeper exploration of computer music.

SONG2

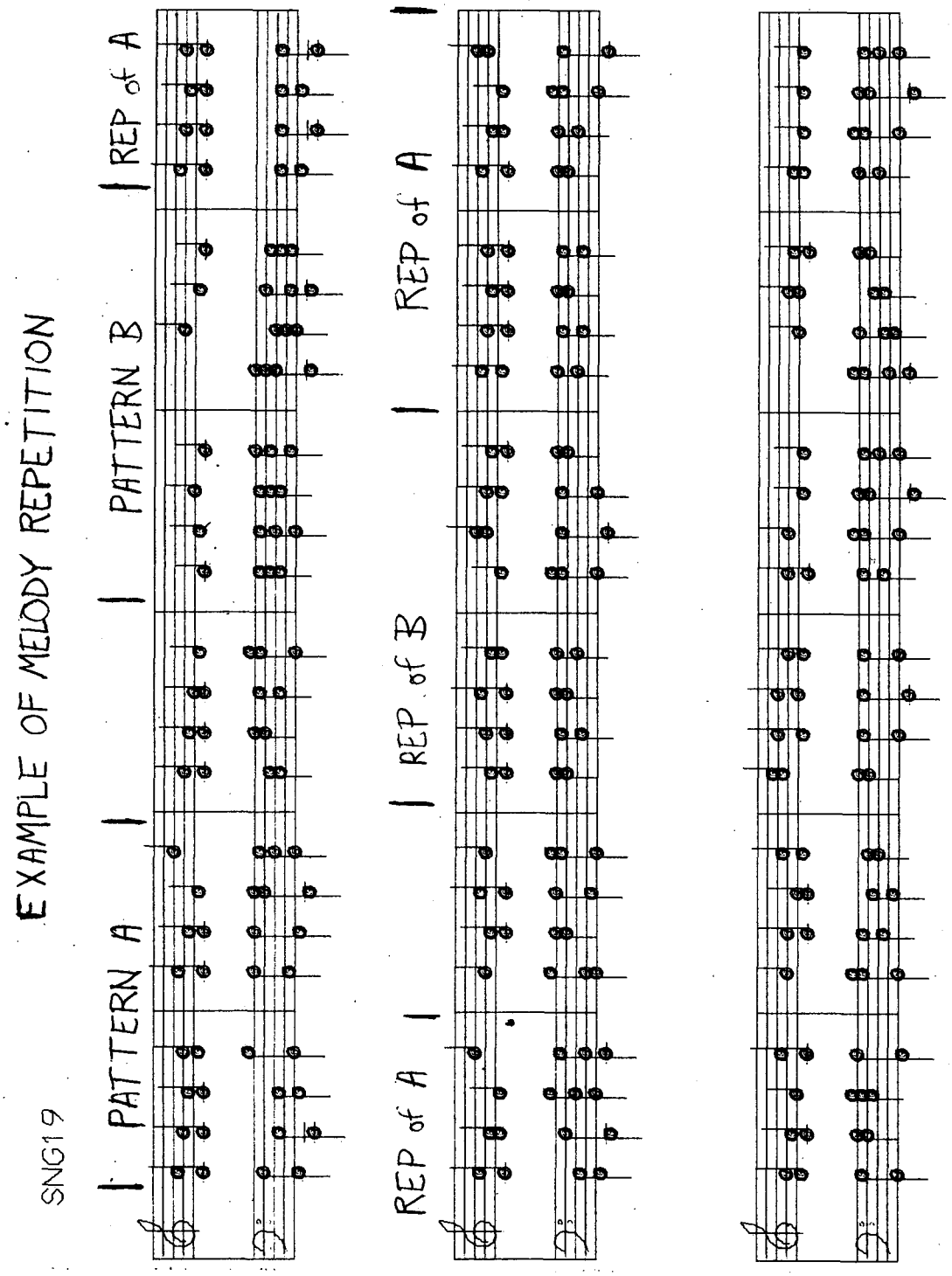


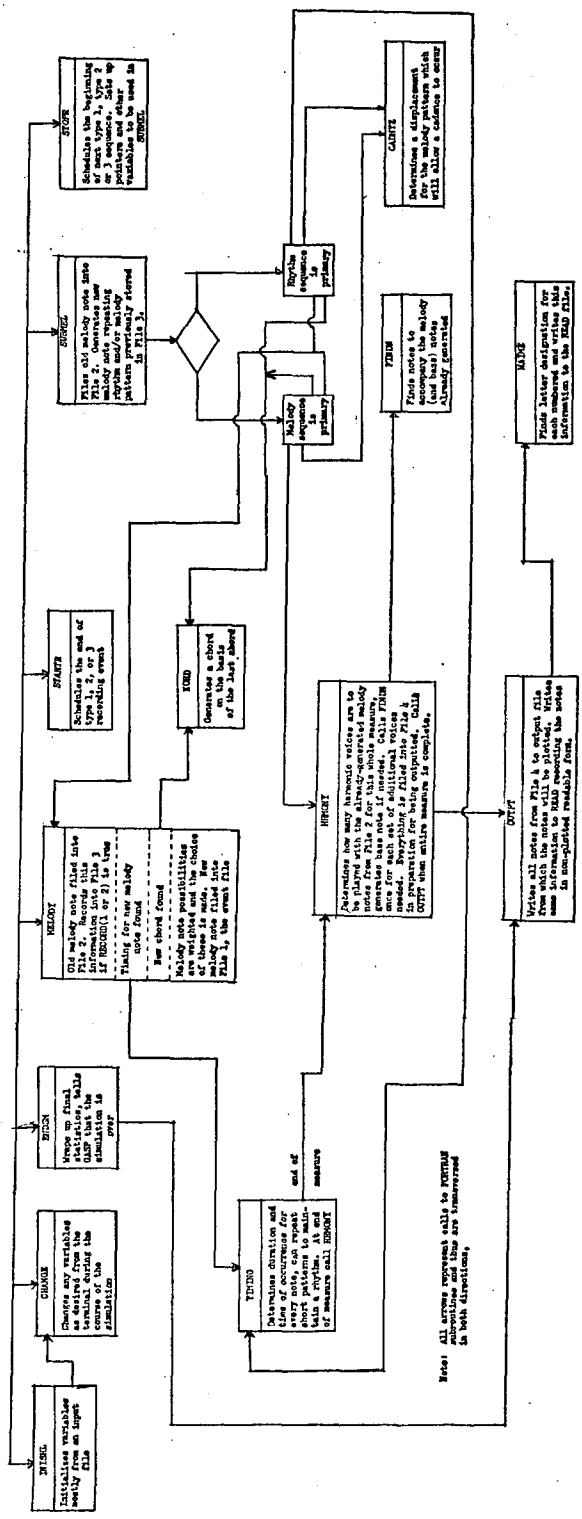
SEMI-RANDOM MELODY WITH NO PATTERNS

SNG52



REFINEMENTS IN TIMING AND OUTPUT ROUTINES


EXAMPLE OF MELODY REPETITION
SNG19
PATTERN A
PATTERN B
REP of A
REP of A
REP of B
REP of A



FLOW DIAGRAM OF MUSIM

DONALD C. LAVOIE, ALAN C. AHO and JAMES N. PAPROCKI all graduated in 1973 from Worcester Polytechnic Institute in Worcester, Mass. with Bachelor of Science degrees in Computer Science.

Don has done work at school which includes a GASP simulation to study an algorithm for the efficient dynamic allocation of storage in stacking, queues, and deques. He also has taken piano lessons for several years which was a great asset in this project. Currently Don is working at Consolidated Edison of New York as a programmer with their new Nuclear Power Plant Simulator and is also studying for an advanced degree in Economics at N.Y.U.

Alan brings a similar interest in simulation programming to this project. In the past he has written an assembly language simulation of a STOPGAP-like line editor, and a plotter program which simulates the rotation of an object around any axis in three-dimensional space. His interest in classical music also drew him into this project. Currently Alan is employed as a programmer for Travelers Insurance Company in their Commercial Lines Systems Division in Hartford, Conn.

Jim has been very interested in the applications of the computer in the world of art. He has written, among other things, a program to execute line printer pictures, where the user has complete control over the shading that is used. He has experimented with creating line drawings and Moire patterns on the plotter, and does oil painting for a hobby. Jim is currently employed as a programmer for Eastman Kodak in Rochester, New York.