TUTORIAL/STUDENT ARTICLE

Max and Tabula Vigilans: two programs for algorithmic and rule-based composition

ODILON MARCENARO

Head of Music, North Shropshire College, College Road, Oswestry, Shropshire SY11 2SA, UK
E-mail: odilon@euronet.nl

1. INTRODUCTION

In this article I intend to investigate the differences and common factors between Max and Tabula Vigilans, two programs for rule-based and algorithmic composition. Before I begin this investigation, I will introduce both programs briefly.

I was first introduced to Max in the first year of my Music Technology studies, when a program then called 'Patcher' was demonstrated at my faculty by Zack Settel of IRCAM in 1989. I was immediately impressed by its philosophy: construct a MIDI-based programming language, using the analogue studio as a metaphor: objects are visually 'patched' together.

'Patcher' has evolved considerably since then; it is now called 'Max', a tribute to Max V. Mathews (a pioneer of computer music). It is developed at IRCAM and marketed by Opcode, and widely commercially available at around £400. As I write this article, the current software version is 3.0. Max has evolved to be a standard by which many other programs are judged, and is widely used by composers all over the world, both for MIDI and DSP applications. In a DSP context, Max is mainly used on the NeXT/ISPWS (IRCAM Signal Processing Workstation) platform.

Tabula Vigilans ('Vigilant Table') (TV) was introduced at the San José ICMC conference in 1992. It started as a project by Richard Orton, a reader in Music Technology at the University of York, out of a 'perceived requirement to control musical data under conditions of actual or simulated performance' (Orton and Kirk 1992).

My first introduction to TV was at MediaMix in York, 1994, when a piece by Richard Orton rekindled my interest in algorithmic music with which I had become totally disenchanted. During MediaMix I stayed with one of Orton's students, Nick Fells, who kindly introduced me to the program. I wrote to Richard Orton asking for a copy of TV, which was subsequently released by the Composers' Desktop Project in Chippenham. It is now available from CDP at around £100. The current software version is 1.1β.

Both programs are intended for rule-based and algorithmic composition, and are script-driven (i.e. the user writes a script, which is then submitted to the program). This is where the similarity ends.

Max is designed for use on Apple Macintosh and Next ('Signal-Max') computers, and is a stand-alone program. Tabula Vigilans runs on fast Atari (TT, Falcon) and PC computers and on Silicon Graphics Unix workstations, and is a command-line program, designed to be part of the CDP suite of command-line programs (a command-line program is a program that is run by typing its name, followed by various 'arguments' into a command-line interpreter or commandshell).

I tested Max on various machines, but mostly on an Apple PowerBook 170. TV ran on an Atari TT. These machines are comparable in speed and processing power (PowerBook 170: 68030/25 MHz with FPU; Atari TT: 68030/32 MHz with FPU), the TT being the slightly faster machine.

2. A (VERY) BRIEF INTRODUCTION TO ALGORITHMIC AND RULE-BASED COMPOSITION

As both Max and TV deal with algorithmic and rule-based composition, a short introduction might be useful. This is, however, a very large subject, and for a more detailed explanation I have to refer the interested reader to Iannis Xenakis' *Formalized Music* (Xenakis 1971).

An algorithm is a prescribed set of well-defined rules or instructions for the solution of a problem. An algorithm is by definition generative ('I know what I want, now how do I *get* it?'). Algorithms can be used to describe musical processes in a mathematical way. This does not necessarily have to be complicated, as will appear from the following examples:

semitone = 1

note(pitch) = note(pitch + semitone*7)

This is a perfectly valid mathematical way of saying: transpose every note played by a perfect fifth. Admittedly not fascinating, but still a valid musical

Organised Sound **2**(1): 43–56 © 1997 Cambridge University Press. Printed in the United Kingdom.

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

statement. Now let's expand it a little:

semitone = 1

note(pitch,duration)

= note(pitch + semitone*7, pitch/10)

Now we have effectively formalised pitch and duration using pitch as a rule. The duration of each note generated will be equal to the pitch of the note played divided by ten. So (assuming MIDI input), if middle C (note number 60) is played, the resulting note will be the G above middle C, which will last for six seconds.

These examples have, of course, a very predictable output. Now let's try something slightly more daring:

0 < random < 1

note(pitch,duration)

= note(random*100, random*100)

This illustrates another very important concept: random processes. A random function typically generates numbers between 0 and 1, so in the above example every note generated will have a pitch and duration that lie anywhere between 0(!) and 100. The result will be extremely chaotic. To make this musically acceptable, some form of limiting the output will have to be devised. We could, for example, define a function that allows only numbers within certain limits to be generated. Many random functions in algorithmic programs have such limiting facilities built in. In TV, for example, the statement

random(20,25)

means that random numbers *between 20 and 25 only* will be generated.

Another powerful concept is the use of time-varying functions. We could, for example, specify two functions, one that moves from 5 to 20 and another that moves from 10 to 25 over ten seconds. We could then pass these functions to a limited random function, which in turn is passed to a note generator:

note(pitch,duration)

= note (random(function1,function2),1)

This will result in a cluster of notes that moves upward fifteen semitones over ten seconds. The cluster will be limited to five semitones.

A brief explanation is needed here to avoid confusion due to the terms 'algorithmic' and 'rule-based'. 'Algorithmic' refers to the actual compositional process, i.e. formulaic generation of music. 'Rules' are bidirectional entities. For example, given the function $A = B + C$, one could define a rule that can work out *any one* of these variables given the two other ones.

It is important to realise that the term 'algorithmic' does not necessarily imply a mathematical process. Any method by which a composer produces a piece of music can be called an algorithm. A good example is John Cage's *Music of Changes*, in which the progress of the piece is determined from 'I Ching' readings.

I should point out that rule-based programs *can* be used for algorithmic composition, but they can also serve other purposes (this is especially true in the case of Max). At the same time, algorithmic composition does not necessarily require the use of a computer program. The use of a computer program becomes necessary if music is to be generated in 'real time', or interactively (transformation of input from the performer), or is too sophisticated to be done by hand.

Please bear in mind that the examples used here are generalised and do not represent an existing programming language or program.

3. SYNTAX

As both Max and TV serve similar purposes (i.e. algorithmic and rule-based composition), one may expect considerable similarities in their respective syntaxes. This investigation, however, will demonstrate that both programs have clearly distinguishable biases.

The following lists consist of all the functions available in Max and TV, indexed by topic and followed by a brief explanation. Similar and related functions are listed on the same line.

MAX	TV	EXPLANATION
mathematical:		
ABS	ABS	convert to absolute value
ACCUM	MULT, SUM	multiply or sum arguments
CHANGE		filter out repetitions of a number
DRUNK		random walk algorithm
EXPR		evaluate expression
MAXIMUM	MAX	maximum of a number of arguments
MINIMUM	MIN	minimum of a number of arguments
PAST		report when input increases beyond a certain number
PEAK		output when number is greater than previous numbers
RANDOM	RAND, RANDOM	output a random number

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

SPLIT		look for a range of numbers
TOGEDGE		report zero/non zero transitions
TROUGH		output when a number is less than previous numbers
URN		generate random numbers without duplicates
&& ||	&& ||	logical AND/OR
	ADD_DEC	increment or decrement
	GAMMA	gamma random distribution
	GAUSS	gauss random distribution
	POP	population fractal algorithm

variables:

APPEND		append arguments at the end of a message
FLOAT		store a decimal number
INT		store an integer value
PREPEND		put one message at the beginning of another
SWAP	SWAP	swap the values of two arguments
PV	LOCAL	declare a variable to be local
VALUE	LOCAL	declare a variable to be local
	COPY	copy the value of a variable
	LIM	limit the value of a variable
	LOCK	lock the value a variable
	UNLOCK	unlock the value of a variable

midi:

BENDIN		read pitchbend
BENDOUT	PITCHBEND	output pitchbend
BORAX		report state of note-ons and note-offs
CTLIN		read controller
CTLOUT	CONTROL_OUT	output controller
FLUSH		provide note-offs for held notes
FOLLOW		compare live to recorded performance
MAKENOTE		generate note-off following each note
MIDIFLUSH		send note-offs for stuck notes
MIDIFORMAT		format data as midi
MIDIIN	MIDIIN	read midi
MIDIOUT	MIDIOUT	output midi
MIDIPARSE		interpret raw midi
MSD		midi sample dump
NOTEIN		read a note
NOTEOUT		output a note
PGMIN		read program change
PGMOUT	MIDISET	output program change
POLY		allocate notes to different voices
POLYIN		read poly pressure
POLYOUT		output poly pressure
RTIN		output system real time messages
STRIPNOTE		filter out note-offs
SUSTAIN		emulate sustain pedal
SXFORMAT		prepare system exclusive
SXIN		read system exclusive
TOUCHIN		read aftertouch
TOUCHOUT		output aftertouch
XBENDIN		read extra precision pitchbend
XBENDOUT		format extra precision pitchbend
XNOTEIN		read notes with release velocity
XNOTEOUT		format notes with release velocity
	MIDIECHO	read and output midi
	MIDICHORD	output a chord
	SCHEDULE	schedule a midi event for later performance

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

tables, lists and arrays:

FUNBUFF		store x,y pairs of numbers
FUNNEL		map a number to a list which identifies its inlet
ITER		break a list up into a series of numbers
MATCH		look for a series of numbers, output as list
PACK		combine numbers into list
SWAP		reverse the order of two numbers
TABLE	TABLE	set up a table
UNPACK		break a list up into individual numbers
	ALLOCKED	check if all the elements of a table are locked
	ANYLOCKED	check if any elemt of a table is locked
	COMPARE	compare two tables
	COPY_TABLE	copy a table
	DIMENSIONS	return the dimensions of a table
	DIMSIZE	return the size of a dimension
	EMBED, FOLD	embed or enfold two tables
	FILL-TABLE	fill a table from a file
	GENERATE	generate elemnts for a table
	INTERP_TABLE	interpolate two tables
	MULT_TABLE	multiply two tables
	OFFSET_TABLE	offset the elements of a table by a specified value
	PERM	permutate a table
	SCALE_TABLE	scale a table
	SHIFT	shift the elements of a table by a specified factor
	SORT	sort the elements of a table
	SUM_TABLE	sum two tables
	XAD	extract the difference between adjacent elements
	XAR	extract the ratio of adjacent elements

user interfacing and -interaction:

ACTIVE		check whether window is active
BPATCHER		embed a patcher inside a box
BUTTON		flash on any message, send a bang
CAPTURE		store numbers to view and edit
COLL		store and edit a variety of messages
COMMENT		put explanatory notes in window
DIAL		output numbers by emulating dial onscreen
DIALOG		open a dialog box for text entry
GSWITCH		switch between two inlets
HSLIDER		horizontal slider
INCDEC		increment/decrement in user interface object
KEY		read keys typed on keyboard
KEYUP		report released keys
KSLIDER		output numbers from an emulated piano keyboard onscreen
LED		display on/off
MENU		pop-up menu
MENUBAR		user-scriptable menubar
MESSAGE		send any message
MOUSEFILTER		pass numbers only when mouse button is up
MOUSESTATE	MOUSE	probe the mouse
MULTISLIDER		multiple slider and scrolling display
NUMBERBOX		display and output number
NUMKEY		read numbers typed on keyboard
PRESET		store and recall settings of objects
SLIDER		output numbers by moving an emulated slider
SPELL		convert input to ASCII
TOGGLE		switch between on and off
UBUTTON		transparent button
USLIDER		output numbers by moving a slider onscreen
	ARG	read a command line argument

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

digital signal processing:

DSP LOAD		load DSP code
DSP GO		start DSP application
DSP PEEK		read DSP values
DSP POKE		write DSP values

triggering:

BANG		send a trigger
BANGBANG		send many bangs
BUTTON		send a bang
CLOSEBANG		bang when closing window
DECODE		send 1 or 0 from specific outlet
GATE		pass the input out a specific outlet
GGATE		pass input out of one of two outlets
LOADBANG		bang on loading of patch
METRO	TRIGGER	set a time-constant trigger
TRIGGER		send a number to many places
UZI		send a specific number of bangs

external devices:

CD		access CD ROM drive for audio
GLOVE		access Mattell Data Glove
OMSINFO		set popup menu with Opcode Midi System devices
SERIAL		access serial port
SND		use sound manager
TIMEIN		report time code
TIMEOUT		output time code
VDP		control videodisk player

librarian:

LIB		script-driven patch librarian
LIBBANG		bang after librarian init
LIBFROM		output patch value
LIBNAMES		output patch names for a menu
LIBSWITCH		change patch being edited
LIBTO		store data within an edited patch
LIBTOFROM		combines libto and libfrom

time-based:

ENV	SEG	set up an envelope
LINE	LIN	trace a linear function
METRO		start a metronome
TEMPO		output numbers at a metronomic tempo
	EXP	trace an exponential function
	LOG	trace a logarithmic function
THISTRACK		function for time-related triggering of events
TICMD		function for time-related triggering of events
TIMELINE		function for time-related triggering of events
TIOUT		function for time-related triggering of events
THISTIMELINE		function for time-related triggering of events

timer:

CLOCKER		report elapsed time at regular intervals
TIMER	TIME	start a clock

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

file I/O:

FUNBUFF		writes x,y pairs of numbers to disk
MTR		writes midifiles to disk
SEQ		writes midifiles to disk
TABLE		writes contents to disk
TEXT	STOREFILE	store a file to disk
	STORI	store an integer to disk
	STORE	store a value to disk
	STORSTR	store a string to disk

binary:

&	bitwise AND
|	bitwise OR
<<	bitshift left
>>	bitshift right

program specific:

COUNTER		count received bangs
FORWARD		send remote messages
GRAB		intercept output of object
INLET		receive messages from outside patcher
OUTLET		send messages out of a patcher
PATCHER		make a subpatcher object
PCONTROL		open and close subpatcher
RECEIVE		receive messages, without patch cords
ROUTE		selectively pass the input out a specific outlet
SELECT		select certain inputs
SEND		send messages, without patch cords
SWITCH		receive input in a specific inlet
THISPATCHER		send messages to a subpatcher
	BREAK	interrupt a loop
	CALL	call a procedure
	CONTINUE	continue with the last procedure
	END	stop performance
	FOR	for() conditional loop
	IF	if...else conditional construct
	LOOP	loop a procedure
	RETURN	return to last procedure
	WAIT	interrupt performance for specified time
	WHILE	conditional loop

graphics & display:

FPIC		display a PICT file
FRAME		draw a rectangle
GRAPHIC		graphic window
IMOVIE		play a QuickTime movie in a patcher
MOVIE		play a QuickTime movie in a window
OVAL		draw an oval
PICS		animation in graphic window
PICT		draw picture
PLAYBAR		QuickTime control bar
PRINT	MESSAGE, MESSAG1	print to screen
	PRINT, PROBE, PROBI	print to screen
RECT		draw a rectangle
RING		draw a circle
	CLS	clear screen

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

scheduling & delay:

BAG	store a collection of numbers
BUDDY	synchronize arriving numbers
DELAY	delay a bang
PIPE	delay a message
SCHEDULE	schedule a midi event for later performance

sequencing:

MTR	multi-track sequencer
SEQ	single-track sequencer

statistical:

HISTO	histogram
MEAN	statistical averaging
SUBST	statistical substitution

Both programs have standard arithmetic and trigonometric operators implemented (+, –, sine, cosine, etc.). These have been omitted in this list.

An examination of the list shows clearly the different biases of both programs. In terms of mathematical rules, both programs are equally well equipped. Tabula Vigilans is the stronger program when it comes to operations on variables, but Max is capable of doing binary arithmetic (bitwise AND & OR and bit shifting). Both programs have basic statistical rules: a 'histogram' rule for Max and statistical averaging and weighting for TV.

Max has a very comprehensive suite of MIDI rules, whereas TV's MIDI rules are rather basic. Tabula Vigilans has just one MIDI rule that Max does not have, i.e. 'midichord', which outputs a chord, as the name suggests. Max provides basic single-track and multi-track sequencers.

Tabula Vigilans' true strength is shown by the amount of rules that are available for operations on lists and arrays. Many of these rules are extremely sophisticated (e.g. time-based interpolation of two tables).

Rules for user interfacing are very basic in TV, whereas the amount of rules in Max suggests that this program is much better suited for the production of stand-alone applications with a user-friendly front end. This is also illustrated by the amount of graphic rules that are available for Max. In TV, these rules are very basic 'print to screen' operations, more suited for diagnostic messages than for real user interfacing. Max is also capable of producing complete librarians and editors for MIDI devices, using sophisticated librarian and 'system exclusive' rules, and can use external devices (e.g. the Mattel Data Glove or a CD player), and access computer interfaces.

Max has a few rules that can access DSP cards. Tabula Vigilans does not have these functions, but has been designed to work with the MIDAS DSP system (Kirk, Orton, Hunt and Katrami 1994).

Both programs have a few time-based rules. Tabula Vigilans can trace many different time-varying functions (linear, exponential, logarithmic), but Max has a very powerful script-driven envelope generator, and in its most recent update, powerful graphic macros for time-dependent triggering of events.

Both programs can read/write files to and from disk. Max has the capability of producing standard MIDI files, which TV has not.

Both programs are capable of setting up a timer. Max has an extra 'stopwatch' rule, but the TV clock is more precise.

Tabula Vigilans has a sophisticated scheduling rule, which allows MIDI events to be stored for later performance. Max allows for a graphic representation of events to be triggered, using a sequencer-like graphic metaphor.

Tabula Vigilans has one rule for triggering. This rule results in a positive number at regular adjustable time intervals. More rules for triggering are not necessary, as triggering will normally be done by function-calls or control-flow calls. Max has many rules for triggering, of which a few are graphic rules which allow the user to press a 'button' or throw a 'switch' on-screen.

The program-specific rules show a clearly different approach in the programs. Most of the program-specific rules in Max are used to make the screen layout more simple, e.g. 'send' and 'receive' allow objects to be connected without making an actual 'physical' connection. Other rules deal with 'nesting' of patchers (one patcher may contain other patchers). This clearly demonstrates the highly graphic nature of Max.

The program-specific rules in TV show that TV is more like an 'ordinary' non-graphic programming language. These rules deal with setting up control-flow procedures (call, return, loop, end, etc.) or with conditional evaluations ('for . . .' loops and 'if. . . else' constructs).

This comparison shows that while Max has a lot of extremely powerful rules for the generation of

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

music and for MIDI processing, it has a heavy emphasis on user-friendliness and graphics, which makes it equally suitable for some non-musical applications. Tabula Vigilans, on the other hand, is a bare-bones language for music generation, and is not suitable for any other use. An outstanding feature of TV is its extremely powerful set of rules for operations on tables. The fact that TV dedicates all processor speed to music makes it very accurate, while timing in Max may become sloppy if a lot of graphics are being used.

4. GRAMMAR AND USER INTERFACE

4.1. Tabula Vigilans

The first thing one will notice when reading through some TV scores is how much the data format looks like the C programming language. This supposedly makes the program easier to access for people who are computer-literate. However, a thorough knowledge of C or any other programming language is not required, and can even be somewhat cumbersome, as TV and C differ on some crucial points (most notably, variables in TV are global unless declared local; in C this is the other way around).

Tabula Vigilans is run by submitting a 'script' to the program. The script is a text (ASCII) file, which can be produced on any word processor. This script consists of 'procedures', one of which must be named 'start( )'.

Procedures consist of a procedure name, followed by '()' or by the name of a 'cell' enclosed in parentheses. A cell is a variable which may be any number and may have any name. A procedure may call a subsequent procedure, passing on a cell (or cells) as a parameter.

The value of a cell may be locked or limited, thus forcing an algorithm to 'conform' its output. This makes TV rule-based. 'Tables' are groups of cells. They have the format of a C array.

Events within procedures can be of three types: 'assignments' to cells (e.g. a = 1), 'rules' operating on cells and tables, and program control flow statements. Rules perform one specific action, and are identified by their names. They usually have input and output cells or tables (e.g. 'a = abs(b)' means: return the absolute value of cell b in cell a).

The basic format of a TV script looks like this:

– Initialisation rules (e.g. setting up a table)
– 'start( )' procedure
– other procedures

In TV format:

```
table A_TABLE[10]

start( )
{
     (rules, one to a line)
     (control flow procedures)
}

other_procedure (a_cell)
{
     (rules, one to a line)
     (control flow procedures)
}
```

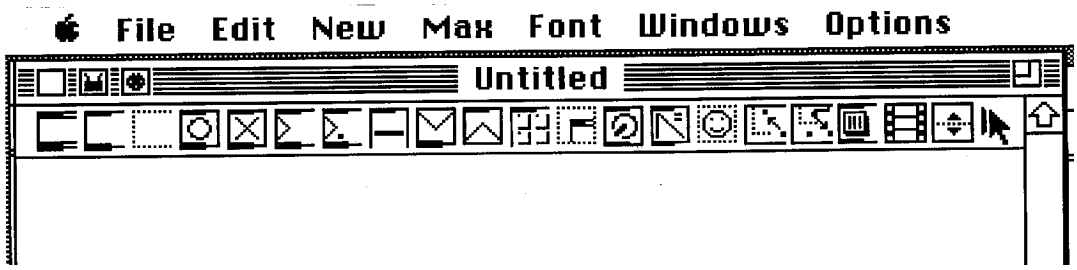
This script is subsequently executed by running TV from a command-line shell with the name of the script as argument. If the script were called 'script1', one would give the following command:

TV script1

This will initiate performance, which will continue until the user presses a key on the computer keyboard, or until an 'end' statement is encountered. After performance, TV reports its running time. If an error is encountered, the user is informed immediately after executing the script, which will not run until all errors are corrected. Tabula Vigilans provides limited diagnostic error messages, indicating the line number on which a syntax error may be found, or the name of the rule if an incorrect number of parameters has been given.

4.2. Max

When starting up Max, one is presented with two windows: the Max window, which provides diagnostic messages, and a blank 'Patcher' window, which at the top contains a row of boxes. These boxes are the program modules that can be used in Max (called 'objects'), and include, most importantly, an object for rules, number boxes, message boxes and various graphic 'switches' and 'sliders'.

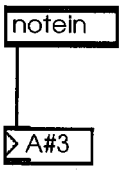


https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

These boxes can be put on the screen by dragging them onto the Patcher window. All objects have inlets and/or outlets, through which they can be connected to other objects. This is done by clicking the mouse in an outlet, and dragging to an inlet. This results in a graphic representation of a 'cable'. This process is called 'patching', and is clearly a metaphor for the patch-panels found in analogue studios.

The most important object evidently is the program module box. If this box is positioned on the window, a list pops up containing all available rules. If one clicks on one of these rules, it is copied into the program module (this feature can be disabled, so that the experienced user can type in names of rules directly).

A finished program in Max is made by patching a number of these boxes together to perform some application. For instance, the following setup monitors incoming MIDI data for pitch:



This is of course a very simple application. As there is no limit to the number of objects that may be used (and patcher windows may contain other patchers), complicated applications are easily realised.

When the application is finished, the user clicks on the 'lock' icon. The application is then immediately available for performance.

This highly graphic user interface, along with features like a powerful online help facility, makes Max extremely easy and intuitive to use. No knowledge whatsoever of programming languages is required, and the user need not type a file which then has to be submitted to a separate program. In my teaching practice, I've found that novice users can learn the basic use of Max in about an hour.

From the above, one may conclude that, as Max is the easier program to use, it will have a greater appeal, especially for users who do not wish to deal with learning a text-based programming language. These users possibly will not use Max for algorithmic composing (which is quite mathematical). Tabula Vigilans will appeal to users who already have a good grasp of algorithmic composition and computer programming. These users will find TV's command-line interface useful because it can be run from a shell which contains other programs (the Composers' Desktop Project philosophy); they will not want to bother with a graphic interface.

Clearly both philosophies have their merits. It must be remarked, however, that a user who is experienced in computer programming and algorithmic composition will be able to make very effective use of Max, whereas an inexperienced user will find it nearly impossible to use TV.

Finally, allow me to emphasise that this article provides no thorough insight on how to use both programs. For that I refer the reader to the respective manuals.

5. APPLICATION

As in any composition, an algorithmic composition is all about getting from point a to point b in a sensible way without losing sight of the total structure. In algorithmic computer composition this is done by setting up events which happen at certain predetermined or non-predetermined points in time. What happens within these events is, of course, up to the composer.

In TV this is quite easy to set up. One simply makes a function-call to a clock, and instructs the program to only perform a specific event after, for example, two minutes have passed, or only within thirty seconds and one minute after the start of the piece.

In Max once again the graphic approach is taken. When using the macro **timeline** the user is presented with a sequencer-like window, divided into tracks, onto which events to be triggered can be placed. This is a vast improvement over previous versions, in which the only option for time-dependent triggering was the use of delays, which makes a heavy demand on processor time.

Max excels at interactivity with the user. The actual application is displayed on the screen while running, and variables may be modified at any time during the performance. Variables in number boxes can be changed with the mouse, menus may be used, the computer keyboard may be used for data entry, etc. In this way it is of course possible to make use of different events at different points in time.

The main difference between Max and TV in this respect would appear to be that, when it comes to organising events in time, TV is best suited for use with a predetermined time line. When the actual changes occur in Max depends on the user, so Max works best in a non-predetermined situation. This leads to the conclusion that Max may be better suited for improvisation than TV. In my own work I have indeed found this to be true. Making music with TV is often a matter of typing the file, and then sitting back while the file is performed, and going back to the file after performance to correct mistakes. When a Max application is finished, the actual 'tweaking of the variables' I often do while running the application. The adjusting of variables while playing results in the performance, which of course may be different each time when performed: one of the most important concepts of improvisation.

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

Another aspect of Max and TV that has a clearly distinguishable influence on how the respective programs are used, is the format in which the actual scripts are constructed. As discussed earlier, TV uses a script in text format, while a Max script is graphic. Events or groups of events in TV are placed in procedures, which are called by a main procedure (in computer programming terminology, a main event loop). The order in which they are called from the main procedure determines for a large part the time line: if the main procedure is set up to call procedure A first, and then procedure B, procedure B will not start until procedure A is finished (unless procedure A makes calls to procedure B, or hands control back to the main procedure immediately). So, the order in which the procedures are entered into the file determines the time line to a large extent. Also, the fact that the procedures are typed below each other, has the psychological effect of having the composer think along a time axis. It is in fact extremely useful to draw a flow-chart diagram of the procedures to be used before one starts programming (a technique widely used in computer programming).

In Max this is totally different. Procedures are groups of graphic objects, and they are active all of the time. They may be placed anywhere on the screen, and their placement has no effect on the order of their execution.[1] The procedures are not called from a main procedure but are all main procedures in their own right, unless the **timeline** function is used for triggering. This means that there aren't necessarily any visual clues to a time line: a Max application is like one large procedure in which the variables may be adjusted at any time. This makes it possible for Max to be used for different purposes than composition. One can, for example, program a complete synthesizer patch editor, or a complex MIDI mapper.

This demonstrates that the script formats of the respective programs have a clear influence on the use of the programs. The TV script format suggests the use of a predetermined time line, whereas the Max script format doesn't suggest a time line in itself, but leaves it to the user.

It might be interesting to use Max and TV 'in tandem'. The fact that the user has little influence over a TV performance could be ameliorated by passing the output of TV to Max, which could then be used to dynamically redistribute and process the 'raw material' from TV in a user-interactive fashion. This would require two computers, which might be a bit bulky for a useful live performance setup.

[1] Actually, this is not entirely true. Max is organised in a right-to-left order, which means that if an object has two outlets, data is sent from the right outlet first. This is something to bear in mind when doing the actual programming, because a misinterpretation of the order in which data is received may cause serious problems: there is a difference between 4/2 and 2/4! When everything is set up correctly, the right-to-left order will have no noticeable influence on performance.

6. CONCLUSION

Having investigated Max and Tabula Vigilans on the levels of syntax, grammar, user interface and application, I will conclude this study with a summary of both programs, and of their respective merits and deficiencies.

Max, a product of the IRCAM and the Opcode software company, is a graphic programming language. It comprises not only a very comprehensive set of MIDI commands, but also a great number of commands which make Max suitable for non-MIDI applications. Its user interface is user-interactive, which makes Max eminently suitable both for improvisation, and for the creation of applications which can be used intuitively. One of the most useful features found in Max is the on-line help function, which greatly facilitates the use of the program. Max can interface to a number of external devices.

The main shortcoming of Max is that the maintaining of the graphic user interface takes up a large part of the available computer processor speed; to run complicated applications efficiently, a powerful computer is required. This, combined with the high price of the program, makes Max an expensive platform to use.

Another problem with the graphic demand on the computer is that timing functions in Max tend to be inaccurate. This problem accumulates as the applications get more complicated. In sum, Max is an extremely powerful tool which can be used for many forms of MIDI-based music making.

Tabula Vigilans, a program by Richard Orton of the University of York and distributed by the Composers' Desktop Project, is a procedural programming language with a command line interface. It is designed to be a part of the suite of command line programs for sound and music which is known as the Composers' Desktop Project System. Its most powerful feature by far is its capability to perform complex operations on tables of numbers. It is very well suited for the generation of algorithmic music with a predetermined time line. Improvisation can take place by transforming and reacting to MIDI input from the user, and by interaction with the computer's mouse. As the user interface is not interactive at all, no improvisation can take place on that level. Tabula Vigilans is very accurate in its timing functions. As it is designed to be part of a larger system, it is very cost-effective.

In comparison with Max, TV is currently clearly the less comprehensive program. Most importantly, it lacks the possibility of having an interactive user

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

interface. The presently limited diagnostic error messages need extending to a more comprehensive form of help for the user. Tabula Vigilans' functions for MIDI are rather rudimentary and are restricted to note and controller data.

The fact that TV's script format looks and acts like a computer programming language may scare off potential users. On the other hand, it makes the program more attractive to those users who have experience with computer programming. In all, TV is an excellent tool for algorithmic composition when used by an expert user.

It must be remarked that comparing Max and TV on comprehensiveness is slightly unfair. The development of Max has been done by a team of programmers (the name of Miller Puckette as the main developer must be mentioned here) at an institute with almost unlimited funding, and Max is marketed by one of the largest music software companies. Tabula Vigilans, on the other hand, is mostly a solo effort by Richard Orton, supported by his colleagues and students. The Composers' Desktop Project is a non-profit, highly idealistic organisation. Considering this observation, TV is a remarkable program indeed, and the Composers' Desktop Project must be commended for making yet another valuable contribution to the world of experimental music.

On a more personal note: as the programs are so different, I do not feel that I have to recommend the one over the other. I am in the fortunate position that I am able to use both programs, and I haven't had to make the decision to stick with the one or the other. Probably I use Max more often, as it can be used for such a large variety of applications. However, I find it harder to produce a presentable piece of music with Max than with TV; quite possibly, TV is better matched to my way of thinking. When improvising, however, I invariably use Max.

For anyone who is interested in taking their first steps in algorithmic composition, I would recommend Max. Tabula Vigilans would be more interesting for the user who already has some experience. A great benefit of TV is that it is part of the CDP System, which as a tool for composition has proved invaluable to me.

Finally, I would like to take this opportunity to thank Archer Endrich, the Director of the Composers' Desktop Project. Without his help and support this study could never have been realised.

7. EXAMPLES OF TABULA VIGILANS AND MAX

These are examples of simple algorithms in both Tabula Vigilans and Max format. Example 1 simply transposes MIDI input by a perfect fifth. Example 2 transposes the input by a perfect fifth, and formalises the duration of the resultant note by using the function **duration = pitch/10**. Example 3 generates a note with a random pitch and duration whenever a note is input. Example 4 generates a ten second long ascending cluster of five semitones, starting on the note received at the input.

Example 1:

```
start() ;start program
{
     if(try(channel,pitch,velocity midiin)==1)
     ;get a note, if available
     {
          midiecho channel,pitch+7,velocity
          ;output the note, 7 semitones higher
     }
     loop  ;keep doing this
}
```

Example 2:

```
start()
{
     if(try(channel,pitch,velocity midiin)==1)
     {
          polyphony=1   ;maximum of 1 note
          duration=pitch/10   ;make duration pitch-dependent
          midiout channel,pitch+7,velocity,duration,polyphony
          ;output the result
     }
     loop
}
```

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

Example 3:

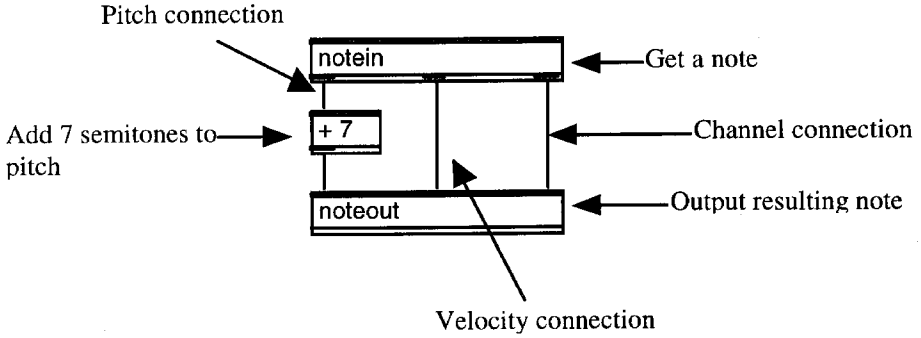
```
start()
{
     if(try(channel,pitch,velocity midiin)==1)
     {
          polyphony=16   ;maximum of 16 notes
          pitch=random(0,100)      ;generate random pitch
          duration=random(10,100)/1000
          ;generate random duration and convert to Msecs.
          midiout channel,pitch,velocity,duration,polyphony
          ;output the result
     }
     loop
}
```

Example 4:

```
start()
{
     if(try(channel,pitch,velocity midiin)==1)
     {
          minimum=5     ;reset the minimum
          call lines (channel, pitch, velocity)
          ;call "lines" function below
     }
     loop
}

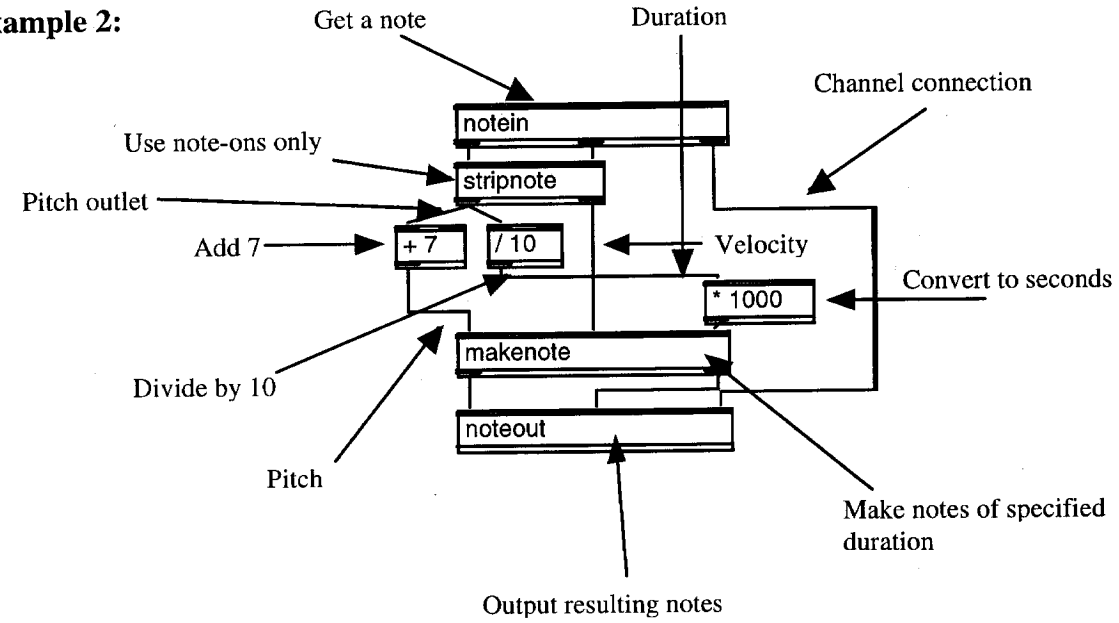
lines (channel, pitch, velocity)     ;"lines" function
{
     while(minimum<20)     ;do until line-tracing is complete
     {
     minimum lin 10,0,5,20;trace  linear function between values 5 and 20 over 10 sec
     maximum lin 10,0,10,25;trace linear function between values 10 and 25 over 10 sec
     midiout channel,pitch+random(minimum,maximum),velocity,1,16
     ;output the result, with the output from the linear functions used in a random pitch
generator
     wait 0.1
     ;wait for 0.1 sec
     }
}
```

Example 1:

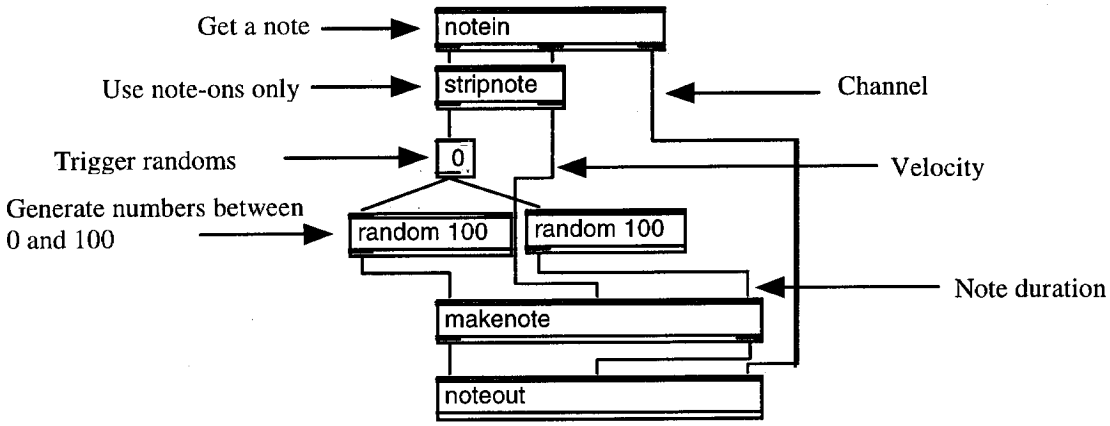


https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

Example 2:

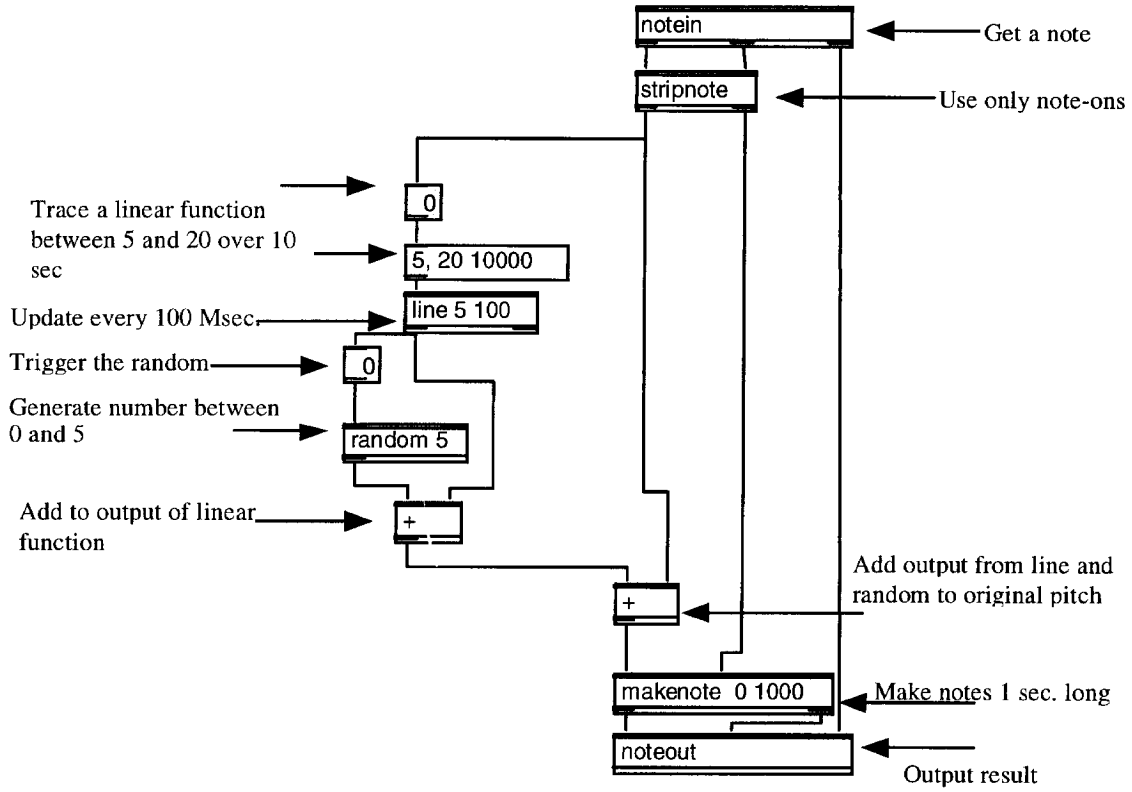


Example 3:



https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press

Example 4:



Note: This is the only way to do this in Max. It is not possible to give the generator a minimum other than zero. I have realised the bandwidth of 5 generating a number between 0 and 5 and adding this number to a line traced between 5 20 over 10 seconds.

REFERENCES

Dodge, C., and Jerse, T. A. 1985. *Computer Music: Synthesis, Composition, and Performance.* New York: Schirmer Books.

Endrich, T. J. (ed.). 1994. *CDP System Manual: Falcon030 Level: Release 2.* York: Composers' Desktop Project.

Kirk, R., Orton, R., Hunt, A., and Katrami, A. 1994. Evolutions of timbres through the use of Tabula Vigilans on the Midas system. In *Contemporary Music Review* **10**(2). York: Harwood Academic Publishers.

Moore, F. R. 1990. *Elements of Computer Music.* New Jersey: Prentice Hall.

Orton, R. 1994. *Tabula Vigilans Manual, Version 1.0.* York: Composers' Desktop Project.

Orton, R., and Kirk, R. 1992. Tabula Vigilans. In *Proc. of the ICMC.*

Pope, S. T. (ed.). *Computer Music Journal.* Cambridge, MA: MIT Press, published quarterly.

Puckette, M. *et al.* 1992. *Max 3.0 Manual.* Paris: IRCAM/Opcode.

Roads, C. (ed.). 1990. *The Music Machine.* Cambridge, MA: MIT Press.

Xenakis, I. 1971. *Formalized Music.* Bloomington: Indiana University Press.

Composers' Desktop Project: c/o 12 Goodwood Way, Cepen Park South, Chippenham, Wiltshire SN14 0SY, UK.

This article was originally submitted as part of the author's MA Contemporary Performing Arts study at University College Bretton Hall, UK.

https://doi.org/10.1017/S1355771897000095 Published online by Cambridge University Press