Nam Shub – A Text Creation and Performance Environment

by Jörg Piringer

Abstract

Nam Shub is a tool and software art project for the creation, modification and performance of text oriented electronic art ranging from experimental literature to visual sound poetry performances or interactive art installations.

The discussed project is the second version or rather a newly developed and enhanced version of *HyperString* which was presented at *e-poetry 2005*.

This tool will be made available under an open source license in the future so that everybody can not only use but alter and expand it.

Introduction

Nam Shub is a text processor, text generator and performance system. It is designed as a tool for both creators and performers of text and language oriented arts. You could see it as a combination of a modular live performance system like for example the music programming environment *Pure Data* [1] and a text processor.
Common text processor programs (like *MS Word*, *Open Office Writer* etc.) however only offer a very limited range of real text processing tools:

- Spell-checker
- Substitution
- Summarize

So I rather drew my inspiration from music and graphic programs which usually offer features like:

- Real time interactivity
- Scriptable and chainable operations
- A large quantity of different functions

I added functions to remove vowels or consonants, change the order of letters, split words into syllables, random operations on a word and letter level, complex substitution, text synthesis and tools for displaying text. Additionally all these functions can be combined and chained through a powerful scripting language and are therefore extensible.

Precursors and Influences

The discussed program and it's concepts are of course strongly influenced by the works and ideas of literary modernist avant-garde movements like Dada,ism Surrealism, Lettrism, Oulipo, Wiener Gruppe and the Beat-poet's use of the Cut-Up technique.
These movements and groups tried to extend the field of literature through the introduction of chance or in contrast through the implementation of strict rules for the generation of texts. Almost all of them were working with the mere materiality of (found) letters, words and printed characters some of them even creating early concepts for computerized poems.
Although *Nam Shub* is inspired by these early attempts it focuses on computer specific aspects of electronic poetry: dynamic and real time generation and manipulation of text.

Text Generators

In the following I'd like to describe some text generator softwares that I feel are important for the discussion of *Nam Shub*. It will not be an exhaustive history of text generators but rather a subjective list tailored to my specific interests.

Mark V Shaney [2] was created by Bruce Ellis and Rob Pike in 1984 as a text generator for a fake Usenet personality. They implemented a basic Markov chain analyser and resynthesizer to generate text out of found texts. It was a very simple program without any possibility to control the results but through the access to a large corpus of text it could fool other Usenet users into thinking that those strange postings were produced by a real person. Markov chains as a text generating tool have since been widely used in text processing computer programs and of course plays a prominent role in my project as well.

Andrew C. Bulhak follows a different approach with his *Dada Engine* [3] and the *Postmodernism Generator* [4]: the software generates random sentences by using recursive grammars. Depending on the structure and the encoded dictionary of the grammar it can produce for example nonsensical but grammatically correct philosophical essays like the *Postmodernism Generator*. Like *Mark V Shaney* Bulhak's program offers no way of influencing the outcome of the generation other than rewriting the grammar. Generative grammars are at the moment only poorly implemented in *Nam Shub* but once improved the user will have more control through the specification of real-time parameters during the generation process.

Ray Kurzweil's *Cybernetic Poet* [5] offers the user more control over the process of generating poems. It is in fact “disguised” as a poetic assistant that offers advice such on how to continue the text the user is writing in a text-editor-like interface. The program offers assistant “personalities” ranging from Blake to Yeats each processing their own literary corpus to suggest alliterating words, rhymes, possible next words or completing the rest of the line or even the rest of the poem. The actual generation process is hidden behind a recommendation interface so it is up to the user to follow the given advice or rather choose her own word or sentence.

An even more playful project is C. P. Bryan's *Cut 'n' Mix ULTRA* [6]: it seems to be inspired by a mixing desk for audio signals. Four text tracks can be mixed together while controlling the “loudness” of each track. Additionally text effects can be applied such as randomly rearranging the resulting words, replacing words with synonyms, swapping words with randomly selected words of the same category and formatting the output to resemble song lyrics. The immediacy of changes in parameters and the real-time application of text manipulating functions are very similar to those in *Nam Shub* and follow the same idea of text generation as a kind of “poetry sculpting”.

Taylor Berg's *Darwin* [7] enables the user to create poetry through a process that mimics genetic evolution. The user plays the role of natural selection by defining the fitness for survival through acting accordingly to his aesthetic preferences. Each time a user visits the project website he is presented six different automatically generated poems. After reading each poem he is asked to select the two he likes most and to enter their number into input fields. After pressing a button the algorithm generates a new set of six poems deduced from the two selected parent poems. Each new generation is recorded and can be reproduced and refined through the same selection process by each visiting user. This evolutionary mechanism can create very complex artistic results just by user driven selection of randomly generated structures. *Nam Shub* offers a similar feature for the algorithmic programming of new text-modifiying functions.

Apart from Jean-Pierre Balpe's elaborate text generators one particular project caught my interest as it used an aspect of human-computer-interaction that seems to be perfectly relevant for electronic poetry but is rarely used. *Labylogue* [8] (by Jean-Pierre Balpe, Jean-Baptiste Barrière and Maurice Benayoun) was a networked interactive installation in the form of a virtual labyrinth built out of text walls. The users in three different cities could communicate by speaking into a microphone and move with a joystick through the text corridors. Simultaneously a computer equipped with a speech recognition software listened to the users' voices and tried to understand what they were talking about to generate new texts for the walls accordingly. My project also uses speech recognition as method to input textual data during performances and also because I think the inherent problems and flaws produced by these speech recognition systems can be a rich source for poetic content.

Nam Shub's means for the creation and modification of text draw inspiration from the above projects but also go beyond them as the unification and combination of these tools can possibly lead to a higher level of complexity in the produced results.

Visual Poetry Software

A couple of remarkable works of animated and interactive visual poetry have been published in the recent years but since this paper is about instruments for the generation and manipulation of text I will concentrate on more general software than those implemented for these very specific works of poetry.

Apart from the commercial softwares than can also be used to produce animated or generated visual text like *Macromedia Flash* or it's little brother *Swish*[1], there are not many applications that were made specifically for visual poetry but quite few that were created as typographic tools.

Estudio Paco Bascuñán's and Inklude's *RoboType* [9] for example is a typographic editor that has a set of features that enables the user to create classic visual poetry for the web browser. You can position, resize and turn letters in four different fonts and then save your creation to a public online gallery. It has however no means to create algorithmic poetry or animation.

An example for an application that explicitly generates visual poetry is *Poem Generator* [10] by Amorvita. It creates colourful constellations on the screen by either randomly choosing presets or by user supplied words and characters. The text is arranged randomly but obviously limited by constraints that give the result the appearance of concrete poetry like for example Eugen Gomringer's work.

Eugenio Tisselli's *MIDIPoet* is also capable of producing visual poetry but will be discussed in the next section because of it's performance oriented aspects.

Poetry Performance Software

There's only one performance software I know of that was specifically designed for text based art: Eugenio Tisselli's *MIDIPoet* [11] is a software that allows the manipulation of digital text and image in real-time. Works composed of interactive text and image can respond to external input like MIDI[2] messages or the computer keyboard. These inputs can influence the position and appearance of text and images.

MIDIPoet can however not generate or manipulate text other than simply substituting it so it can be seen as a visual poetry performance tool rather than a tool for real-time generation. Musical performance and programming systems like *Max/MSP* [12], *Pure Data* or *ChucK* [13] seem to offer more in regard to flexibility, versatility and text manipulation but are cumbersome to learn and handle for poetry.

Nam Shub offers the features and openness of the above musical softwares but concentrates on special tools for text generation, modification and output.

Generation and Modification

The text generation and modification tools of *Nam Shub* were designed to be as flexible as possible

1 which is even more suitable for (animated) visual poetry in my opinion than *Flash*
2 MIDI (Musical Instrument Digital Interface) is an interface and a protocol for musical data and can be used receive input from devices like musical keyboards, drum pads, motion controllers etc.

without making them to cumbersome to use. To be able to control the degree of complexity *Nam Shub* can generate or modify text on three structural levels:

- **Micro structure:** operations modify or generate character data or strings of characters
- **Macro structure:** more general parameters like grammar or rules for substitution or generative systems are altered
- **Meta-structure:** operations on the program structure or the meaning of texts

Micro Structure or Character Level

On the level of characters or character strings operations work on a non-contextual basis. The majority of these operations take no assumptions or have no knowledge about the data they operate on. They simply transform data according to simple algorithms.

Examples:

sort sorts a string:
```
this is a test -> aehiisssttt
```
rip repeats characters in a string or swaps them randomly:
```
this is a test -> thshiss is a testtt
```
spamizide makes the string look like e-mail spam:
```
this is a test -> t-hís s. tést
```
permutations generates all permutations of a string:
```
abc -> ["abc","acb","bac","bca","cab","cba"]
```

These functions be chained to more complex operations:
lambda x: map(spamizide,permutations(x)) applies *spamizide* on all *permutations* of the string:
```
abc -> ["b","b","bä","bá","b","bä"]
```
lambda x: concat(splitWords(shake(concat(map(cmabrigde,splitWords(stretchRnd(x))))))) is a complex operation:
```
this is a test -> xstthes'seaah.etett aTas
```

There are however some character level functions that work between micro and macro level like for example splitting words into syllables. This algorithm assumes that the given word is in English language and applies the rules accordingly. It would also work on transcribed Russian texts but produce no convincing results.

Macro Level

To control text on a macro level *Nam Shub* utilizes a variety of generative systems. You no longer have to take care about single characters or strings but instead of parameters that tweak the production or modification of a whole or parts of texts.

Nam Shub incorporates a couple of modules for the (semi-)autonomous generation or modification of complex text and control[3] data. Some of these modules involve a high degree of randomness and are only controllable through specifying certain boundaries.

At the moment there are four available macro level modules:

- **Markov chains:** a tool for stochastic analysis and resynthesis of texts
- **Lindenmayer systems:** a formal language for text replacement
- **Cellular automata:** very simple rules create emergent complex structures
- **Grammars:** linguistic grammatical structures for the creation of natural language texts

3 for example control data for the text turtle graphics

Markov Chains

Markov Chains were invented by the Russian mathematician Andrey Andreyevich Markov (1856 - 1922) as a tool for the stochastic analysis of dependent events.

It basically measures the conditional probability distribution of future states, given the present state and a number of past states. Applied to text that means the likelihood that a certain letter follows another. How likely is it for example that an "h" is follow by an "o"?

By analysing a corpus of text this tool can model the text's stochastic properties and almost recreate the original through weighted randomness. Markov Chains have been broadly explored in computer generated literature [2] but in my opinion still has a lot to offer especially in conjunction with other methods described here.

The first lines of William Wordsworth's poem "I wandered lonely as a Cloud[4]":

> i wander'd lonely as a cloud
> that floats on high o'er vales and hills
> when all at once i saw a crowd
> a host, of golden daffodils
> beside the lake, beneath the trees
> fluttering and dancing in the breeze

analyzed with the Markov Chain algorithm can produce results like this:

> i wand daffodills
> floats ande in at o'ering a crowd
> bes at flutter'd host, brees
> whe lake, o'er'd lake, oneats ales
> flutter valls
> float floathe thathe the breezei wancing as aleside treezei saw a

Lindenmayer Systems

L-systems [14] were introduced and developed in 1968 by the Hungarian theoretical biologist and botanist Aristid Lindenmayer (1925 -1989). These systems are formal grammars originally invented for modelling the geometric structure of plants.

They can be used as simple systems for replacing and also generating texts.

L-systems are defined as a tuple of an alphabet V, a set of constant symbols S, an axiom or start symbol ω and a set of rules P that define the way variables can be replaced with combinations of constants and variables.

For example:

V = {a,e,i,o,u}
S = {b,c,d,f,g,h,j,k,l,m,n,p,q,r,s,t,v,x,y,z}
ω is set according to the input
P = {(a => aba), (e => ebe), (i => ibi), (o => obo), (u => ubu)}

ω = "this is a test"

```
this is a test => thibis ibis aba tebest => thibibibis ibibibis
abababa tebebebest
```

L-systems are used in *Nam Shub* not only for text creation but also to control text turtle graphics.

Cellular Automata

Cellular automata model n-dimensional spatially discrete dynamic systems where the state of each

4 I removed punctuation marks and converted upper case letters to lower case to enable the Markov algorithm to associate more letters

cell at time *t* is a function only of the states of a finite number of cells at time *t-1*. This function or rule is the same for all cells. So the generation of a new line of text is only dependent on the previous line and very simple rules.

Despite the simplicity of the provided rules cellular automata can produce complex patterns as two results of my experiments with cellular automata for text generation show:

```
smrmmmtrr  rtlll  v v lzuum mzsmzmm  s m
 r r trt rrtlt  lv   lzuzmm zsmz z ms ms
r   tr tr tl ttlvvv lzu mz zsmzz  mssmss
rr trrtrrtllt lvl  lzuumzz smzm zmsmm mr
  trtt ttltttlv lllzuzmzm smz mzms s  rm
mtr r   l l lvvl  zu m  msmzzmz ss  srmm
trr  r l    vllllzuum mms zmm zs  ssrmr
rt rr l l  vlv  zuzmm  sszmz zssss rm rt
 tr  l   lvl vvzu mz ms zm z sz   rmmrt
trrrl l lv lv zuumzzmsszmmz szzz rmrrttt
rt ll   vvlvvzuzmzmmsmzmzzzszs  rm mtr
ttl  l v l lzu m  zs z  m s  ssrmm trrrr
rllll v l  zuum mzssz zm s ss rmr trt  t
lr   v l lzuzmm zszzz mms    rm rtr tttr
 rr v l  zu mz zs s  m sss  rmmrt rt  rl
r  v l lzuumzz ss  sm s   srmrrttrtttrll
rrv l  zuzmzm s  ssmms s srm mtrr r rlrr
 vvl lzu m  ms ss msss   rmm trt    l l
v ll zuum mmss   msm  s rmr tr tt  l   l
vl  zuzmm  sm s ms mms rm rtrrt  tl l lv
lllzu mz msmms mssm ssrmmrt tttttll   vl
  zuumzzms sssmsmmms rmrrtt     lt l vlv
vzuzmzmmss   m  s ssrm mtr t   lttl vl v
zu m  zsm s m ms   rmm trrt t ltlllvllv
uum mzsmms m  sss rmr trttt  lt t v vvvz
zmm zsmsssm ms   rm rtr r  tltt  v    zu
mz zsm m mm sss rmmrt r  rtl l tv v  zu
zz smm     s   rmrrttr rrtll  tvv  vzuum
m sms m   s s rm mtrrr  tlt ltvt vvzuzmz
msm sm m s   rmm trt  rtl tltv tv zu m
s msmm  s s rmr tr ttrtlltl vvtvvzuum mm
sms s ms   rm rtrrt r ltt lv t tzuzmm  s
m s  msss rmmrt tttr ltl lvvt  zu mz ms
 s smsm  rmrrtt   rrlt l vltttzuumzzmssm
s  m  mmrm mtr t r lttl vltl zuzmzmmsmmm
ssm mm r m trrt r ltlllvlt lzu m  zs s s
 mm   r m trtttr lt t v ttlzuum mzss
m  m r m tr r rrltt  v t lzuzmm zsz s
 mm r m trr    ltl tv t lzu mz zs zs s m
   r m trt r  lt ltvvt lzuumzz sszss  m
```

```
aaaaaaaaaaaaaaaaa   aa aaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa  a   a aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa   a a aa aaaaaaaaaaaaaaa
aaaaaaaaaaaaaa  a aaaa  a aaaaaaaaaaaaaa
aaaaaaaaaaaaa   aa aa   aa aaaaaaaaaaaaa
aaaaaaaaaaaa  a   a   a   a aaaaaaaaaaaa
aaaaaaaaaaa   a a a a a a aa aaaaaaaaaaa
aaaaaaaaaa  a aaaaaaaaaaaa  a aaaaaaaaaa
aaaaaaaaa   aa aaaaaaaaaa   aa aaaaaaaaa
aaaaaaaa  a   a aaaaaaaa  a   a aaaaaaaa
aaaaaaa   a a aa aaaaaa   a a aa aaaaaaa
aaaaaa  a aaaa  a aaaa  a aaaa  a aaaaaa
aaaaa   aa aa   aa aa   aa aa   aa aaaaa
aaaa  a   a   a   a   a   a   a   a aaaa
aaa   a a a a a a a a a a a a a a aa aaa
aa  a aaaaaaaaaaaaaaaaaaaaaaaaaaaa  a aa
a   aa aaaaaaaaaaaaaaaaaaaaaaaaaa   aa a
  a   a aaaaaaaaaaaaaaaaaaaaaaaa  a   a
a a a aa aaaaaaaaaaaaaaaaaaaaaa   a a a
aaaaaa  a aaaaaaaaaaaaaaaaaaaa  a aaaaaa
aaaaa   aa aaaaaaaaaaaaaaaaaa   aa aaaaa
aaaa  a   a aaaaaaaaaaaaaaaa  a   a aaaa
aaa   a a aa aaaaaaaaaaaaaa   a a aa aaa
aa  a aaaa  a aaaaaaaaaaaa  a aaaa  a aa
a   aa aa   aa aaaaaaaaaa   aa aa   aa a
  a   a   a   a aaaaaaaa  a   a   a   a
a a a a a a a aa aaaaaa   a a a a a a a
aaaaaaaaaaaaaa  a aaaa  a aaaaaaaaaaaaaa
aaaaaaaaaaaaa   aa aa   aa aaaaaaaaaaaaa
aaaaaaaaaaaa  a   a   a   a aaaaaaaaaaaa
aaaaaaaaaaa   a a a a a a aa aaaaaaaaaaa
aaaaaaaaaa  a aaaaaaaaaaaa  a aaaaaaaaaa
aaaaaaaaa   aa aaaaaaaaaa   aa aaaaaaaaa
aaaaaaaa  a   a aaaaaaaa  a   a aaaaaaaa
aaaaaaa   a a aa aaaaaa   a a aa aaaaaaa
aaaaaa  a aaaa  a aaaa  a aaaa  a aaaaaa
aaaaa   aa aa   aa aa   aa aa   aa aaaaa
aaaa  a   a   a   a   a   a   a   a aaaa
aaa   a a a a a a a a a a a a a a aa aaa
aa  a aaaaaaaaaaaaaaaaaaaaaaaaaaaa  a aa
```

Nam Shub at the moment features only a very simple one dimensional cellular automaton where each cell represents a character and a new generation in time is a new line.

For an in-depth discussion of cellular automata see [15]

Grammars

The support for grammars is at the moment limited to relatively simple context free grammars. Context free grammars consist of productions or rules of the form *V* => *w*, where *V* are non-terminal symbols and w is a string consisting of non-terminal symbols and terminals. Non-terminal symbols can be replaced by the string *w*. The term *context-free* expresses the fact that the non-terminal *V* can always be replaced by *w*, regardless of the context in which it occurs.

The rule `S => NP VP` for example means that the non-terminal symbol S can be replaced by the non-terminal NP followed by VP or in plain words that a sentence contains a noun phrase and a verb phrase. The rule for the noun phrase `NP => Art Adj N` means that it can be replaced by an article following an adjective and a noun. So by consequently replacing non-terminal symbols through productions grammatically correct sentences can be constructed by the program.

The following grammar for example

```
S => NP VP
S => S Conj S
NP => [Art N, Art Adj N]
VP => V NP
Conj => ['and','or']
Art => ['the', 'a', 'some']
Adj => ['blue', 'big', 'small', 'beautiful', 'ugly']
N => ['man', 'ball', 'woman', 'table', 'sausage', 'test', 'dog', 'house', 'toy']
V => ['hit', 'took', 'saw', 'liked', 'stole', 'found']
```

can generate nonsensical sentences like:

a beautiful test liked a woman.

a sausage stole a house or a table took a small toy and some sausage found the ugly dog and some big table took some toy.

I plan to incorporate more sophisticated linguistic toolkits and libraries like *NLTK* [16], *MontyLingua* [17] and *WordNet* [18] which feature a vast amount of linguistic data and algorithms to support the the grammatically correct generation of natural language texts.

Meta Level

Meta level modules control the macro and micro level through the modification of their parameters or the program structure itself. They either reprogram (or generate new) functions that change the behaviour of the underlying levels or they change the way a grammar for example generates natural language sentences according to a meaning or topic.

Program structure

Nam Shub was designed as an open system and is therefore programmable and extensible through the programming language Python.

> *„Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days.“* [19]

This language is broadly used in various technical and scientific fields especially in linguistics and symbol manipulation. A large community of developers and researchers is publishing open source libraries like for example *NLTK* or *MontyLingua* that can be easily incorporated in future versions of *Nam Shub*.

To make these potentialities available for artists who are more than often not trained programmers I implemented tools that enable the user to program without actually writing code.

- **Record:** Records a function like a macro. Live operations are recorded in a function and can be called again by the user or a script.
- **Mutator:** With this tool you can write scripts without typing a single command. Like in genetic algorithms you can try out different randomly generated variations of scripts and choose the ones you like.
- **Graphical programming:** is a planned feature that enables the user to program by manipulating graphical objects instead of typing code

Meaning

Another way of controlling the macro level would be to apply meaningful parameters and to generate this “meaning”[5] programmatically.

At the moment this module is not yet implemented but it will use data from common-sense knowledge bases like *ConceptNet* that can be used for affect-sensing, analogy-making, contextual expansion, causal projection and other context-oriented inferences described in [20].

Audiovisual Output

Additionally to simply printing or displaying text *Nam Shub* features some more performance adequate methods of output:

- **Speech:** *Nam Shub* incorporates a text-to-speech module that can speak text with English phonemes.
- **Text graphics:** the program can display typographic text in real-time

5 I use the term “meaning” here as place holder for a range of ideas from contextual or topical data to the human concept of meaning.

Speech output

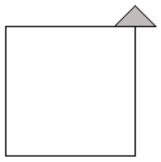
The open source text-to-speech software *Flite* [21] was modified so that it can also vocalize combinations of phonemes that do normally not occur in English language. With these modifications *Nam Shub* is now able to speak letter combinations like “fmsbwtözäu[6]” and therefore is to my knowledge the text-to-speech software that is most suitable for sound poetry. Commercial speech software tends to spell out words that could sound strange.

My software also includes a sequencer[7] that can arrange the spoken words and noises in a rhythmic grid and play them back in real-time.

Text Graphics

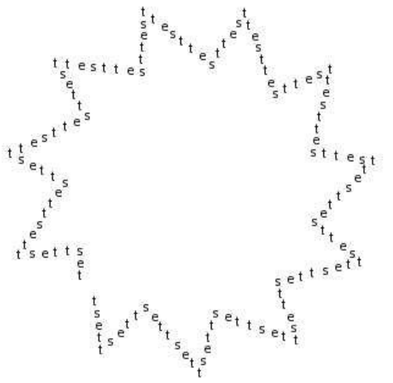
Nam Shub can output graphics via a modified turtle graphics paradigm inspired by the computer language *LOGO* [22] but applied to text output.
In *LOGO* lines can be drawn via simple commands that move a pen (turtle) on the screen.
The following *LOGO*-commands draw a square of the size 100:

```
LEFT 90
FORWARD 100
LEFT 90
FORWARD 100
LEFT 90
FORWARD 100
```



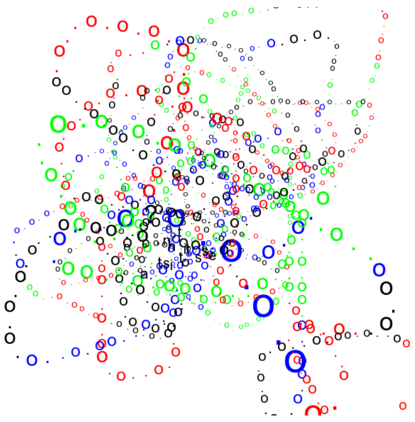
The above draws a square by turning the turtle left by 90 degrees and then moving forward and repeating this four times.

Text turtle graphics example
The turtle graphics of *Nam Shub* doesn't draw lines but sequences of text:



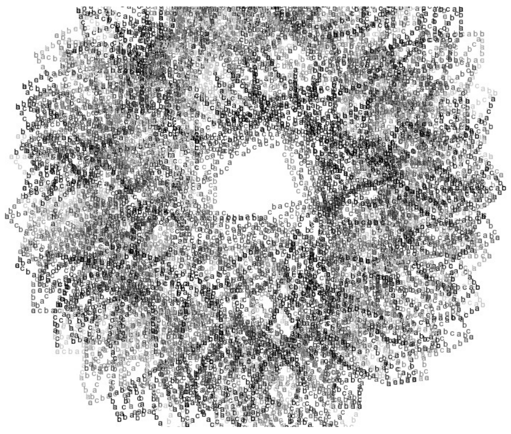
Random text turtle functions
Text can be also drawn in a more random way by applying uncertainty:

6 the first line of Raoul Hausmann's famous letter poem which was adopted by Kurt Schwitters for his *Ursonate*
7 a (musical) sequencer is a software (or device) that records or plays back sequences of timed control information



The control commands for the text turtle graphics are itself coded in a textual format and thus can be modified or generated with the text functions of *Nam Shub*.

The following image shows text turtle graphic which parameters have been generated by an L-system:



Performance

One main goal of *Nam Shub* is to enable a performer to create and modify text and language related audiovisual electronic art in real time. To achieve the necessary flexibility and responsiveness usability aspects have to play a prominent role in the design of the program.

I am currently in the process of evaluating ways for the input and output of data and commands suitable for live performance so some of the following is at the moment still in an experimental development phase.

Input

The user can control the parameters and functions of the executed scripts with a broad range of physical controllers and interfaces to control the program like a musical instruments:

- **Standard controllers** like mouse and keyboard. similar to a standard point-and-click interface known from conventional software
- **MIDI and OSC** are both protocols developed for the transfer of musical data. They are very flexible and can be used with a lot of commercially available controllers like electronic drum sets or musical keyboards.
- **Gestural controllers** enable the user to communicate with *Nam Shub* through movement of the body.

- **Web cams** can be used as an input device for gestural control or in an installation context.
- **Speech input** can be used for issuing commands and entering text with a microphone
- **Custom physical interfaces** like for an example the open-source physical computing platform *Arduino* [23] can be tailored for the users needs and communicate with the program.

Live Coding

To be also able to control the meta level in real-time the program supports live coding through a module called *Playground*. This part of *Nam Shub* enhances the usability of applying quick changes to the program code as it works like a collection of post-it-notes that contain program snippets which can be changed, stored, recalled and executed easily and in real-time.

Thus the user can incorporate the process of programming and refining the logical structure into the live performance to create a permanent work-in-progress.

Possible Applications

Because *Nam Shub* includes the programming language *Python* it could generate all texts or visual poems that could possibly be generated by a computer given enough processing power (or time) and memory. Although this can be easily proven with the given Turing-completeness[8] of modern programming languages I'd rather describe *typical* case scenarios to illustrate the various possible applications of the project.

Poetry Generation

In the following I'd like to demonstrate a simple example of text and visual poetry generation. The system is of course capable of creating much more complex texts and also visual patterns, but I chose a rather simple example to demonstrate the abilities more clearly.

The first step is to generate a basic sentence from the example grammar I described earlier in this paper:

```
some man took some table and some beautiful toy liked the ball
```

Then a randomization function is applied on every word:

```
sommsemeee man toook soooome tabbble and esmssommmmeme beauatbifeuul yytoy lllkkiikled tthe baall
```

After that each word is being looked up in *Nam Shub's* built-in spell checker dictionary and a randomly selected word out of a list of possible corrections is substituted:

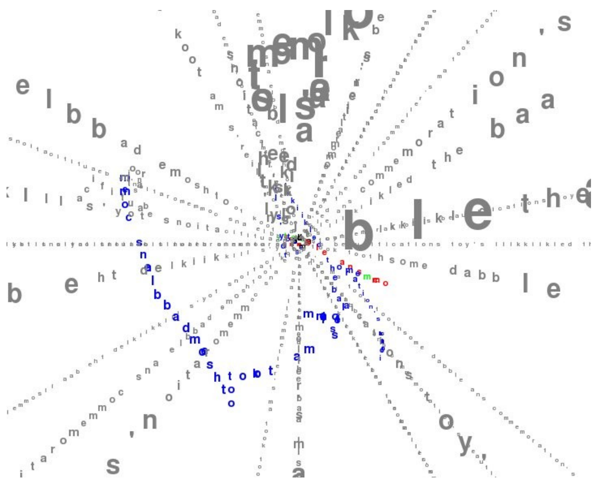
```
sommelier's ma took toothsome dabble ans commemoration's beautifications toy's lllkkiikled the baa ll
```

The final step is then an automatic conversion of the generated “sentence” into commands for the turtle graphics module i described earlier. This is done with the help of Lindenmayer substitution systems:

```
#abcommfbff+b+b<---'abc mf uvoo++ff> uvoouv-b<abcomf ++rrf+r+rbf fklmabc -lommfmo+b+b<---fuvfoklm'abc +rffffffuvf.f-
lfuvfoklmabc uvoklmnopq'abc bbb++ff>++ff>ff++ff>bf++rr uv-b<f +rff bb#f+r-lomm.+r..+f++++++++r+f++++++++r<---'f+r-l m.
ffffdefoo+f++++++++f+++++++..> ffffdefooffffdef-+r<f+r-lom. +f++++++++f++++++++b+b<---+b+b<---.+f++++++++b+b<---
+f++++++++b+b<---+r. .++ff>bmf+r-l -bomm.mo+f++++++++r+f++++++++r<---.ffffdef.o++ff>bm'f+r-l +f++++++++b+b<---......
ffffdef...-b.ffffdef.o++ff>bmf+r-l ffffdefo++ff>bmklmoggqrq'f+r-l
[...]
```

These commands result in this image:

8 for a discussion of Turing-completeness and the Church–Turing thesis see [24]



The elapsed time from the first step to the finished image would most likely be under a minute provided that the user knows what to do. But the overall goal in this case is not efficiency but rather a new kind of work flow like in electronic music or graphic design where aesthetic decisions are often taken on the basis of a trial-and-error approach. So to change the appearance or the actual text it would only be necessary to make a few adjustments and to reapply the operations. In my opinion this could free the poet from pen and paper[9] and supply her with a digital tool like those already available for music and visual arts.

Live Performance

The real-time features of *Nam Shub* enable the user not only to create text based art more elegantly but also to use the program in a live performance context. Unlike other tools that could be used for electronic poetry shows my program is no sophisticated text playing or reading device but could be used as an instrument for literary improvisation.

A typical performance piece could be staged following a script like this:

Silence. On the projected screen visible both for the performer and the audience appear a couple of words or sentences. The performer starts to speak improvised sentences associated with those on screen into a microphone. These words are analysed by *Nam Shub's* speech recognition module and linguistic functions. The results are then displayed and grouped by context data extracted from the common-sense knowledge base *ConceptNet* to mirror the human association process and to guide the performance into possible new directions. So the show leads through an ever evolving associative textual space directed both by the human performer as well as the machine and thus creates real-time man-machine artwork.

Again my project here is aimed at freeing the electronic literature from traditional strictness and immobility or as Jean-Pierre Balpe puts it:

> *"Generative literature tries to be on the side of the effusive superficiality of show. It wants to reconcile the literary activity with that of play and game: to separate literature from the sphere of reverential and deadly seriousness in which the whole classical tradition locks it. Not merely about a particular text, it questions itself infinitely about the aesthetic working of the human spirit."* [25]

Poetry Installation

Nam Shub could also be used in an arts installation in a gallery running on a computer with an

9 in my view contemporary text editing software is only an enhancement of the typewriter which itself is only a mechanized pen

attached video projector and displaying generated text, outputting speech and reacting to input from the visitors.
This could could be a very complex setting incorporating sophisticated algorithmic procedures but could also be as simple as the following[10]: a simple sentence is spoken with the program's text to speech module and simultaneously recorded and processed by *Nam Shub's* speech recognition. The recognized words are then again spoken and likewise recorded. The resulting textual feedback loop becomes more and more flawed due to the inherent limitations of both the speech output and recognition. As the produced text aberrates increasingly from the initial preset sentence the computer system reveals its preprogrammed bias and resonances. The following is an excerpt of a first test run:

```
[...]
presume why my name is Levy and I have is a line wages and I said I'm 99 that is a lot of these and I
and my my name is Levy and I have is in line with Dan Eisen I
have is a lot of these and I have made money and I have is in line
My native land I had in mind the end
enemies and I am money and I have is
My mainland I am mime
enemies and I am money and I have is my mainland I
Enemies and I I and I had in mind mainland I I
enemies and I am money and I have is my name in Miami's N. Y. I. N. I. M. I. N. I. I.
enemies and I am and I have is my name Miami's N. Y. I. N. I. I. A's N. I. I. M. I. name Miami's N. Y. I. N. I. I.
enemies and I am in Miami's mime Miami's A. I. M. A. I. N. nineteen Miami's N. Y. I. N. I. I.
enemies and I am Miami's
Miami is a I N. A. I. N. nineteen Miami's NY I I
enemies and I am Miami's Miami is a I N. A. I. N. nineteen Miami's A. I. N. F. I. Miami's Miami is a I N. A. I. N. nineteen Miami's NY I
enemies and I Miami's Miami is a I N. A. I. S. Miami's A. I. time I am Miami is if I may I nineteen nineties and I
enemies and I Miami's Miami is that I may I ask my is a lifetime I my is assigned AI nineteen nineties and I
am a genuine Miami's Miami is that I may I ask my is a lifetime line reasons I I'm nineteen nineties and I
can examine Miami's Miami is that I may I ask my is alive timeline raises I'm nineteen nineties and I
can say in Miami's Miami is that I may I S. I. is alive timeline raises I'm nineteen nineties and I
should say bias Miami is that I may I S. I. is alive timeline raises I'm nineteen nineties and I
should say my is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nineteen nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nine nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nine nineties and I
should say money is Miami is that I may I S. I. is alive timeline raises nine nineties and I
have to say is that any headway I as I have finally reunited and I
[...]
```

Technical Information

The program is being developed with the programming language *Python* and is using the multi-platform graphical toolkit *wxPython* [26] [27] and the real-time graphics library *pyGame* [28].

I plan to release a first public downloadable alpha-version for the *Windows XP* in 2007 under an open-source licence. Versions for *Mac* and *Linux* should be available later in 2007 or at the beginning of 2008.

The Name

According to Neal Stephenson's novel *Snow Crash*, the ancient Sumerian *Nam Shub* of Enki was a neurolinguistic hack aimed against the standardization and unification of society and human life through verbal rules and laws. Therefore the program *Nam Shub* can be seen as a computer linguistic hack targeted against a global unified culture and empire.

References

10 I conceived this installation under the title “selbstgespräch” (soliloquy)

1: The Pure Data Portal, http://puredata.info/
2: Mark V. Shaney wikipedia article, http://en.wikipedia.org/wiki/Mark_V._Shaney
3: Andrew C. Bulhak, Dada Engine, http://dev.null.org/dadaengine/
4: Andrew C. Bulhak, Postmodernism Generator, http://www.elsewhere.org/pomo
5: Ray Kurzweil, Cybernetic Poet, http://www.kurzweilcyberart.com
6: Cut 'n' Mix ULTRA, http://esoteric-sensationalism.com/
7: Taylor Berg, Darwin, http://www.loudthings.org/
8: Maurice Benayoun, Labylogue, http://www.benayoun.com/Labylogue.html
9: Estudio Paco Bascuñán and Inklude, RoboType, http://www.robotype.net/
10: Amorvita, Poem Generator, http://vita.uwnet.nl/poem/poem.html
11: Eugenio Tiselli, MIDIPoet, http://www.motorhueso.net/midipeng/
12: Cycling '74, Max/MSP, http://www.cycling74.com/products/maxmsp
13: ChucK : Strongly-timed,Concurrent, and On-the-flyAudio Programming Language, , http://chuck.cs.princeton.edu/
14: L-systems wikipedia article, http://en.wikipedia.org/wiki/L-system
15: Stephen Wolfram, A New Kind of Science, 2002
16: Steven Bird, Edward Loper et al., Natural Language Toolkit, http://nltk.sourceforge.net/
17: Hugo Liu, MontyLingua, http://web.media.mit.edu/~hugo/montylingua/
18: Christiane Fellbaum, Wordnet: An Electronic Lexical Database, 1998
19: Python web site, 2003, http://www.python.org
20: Hugo Liu, Push Singh,Iian Eslick, The ConceptNet Project, , http://web.media.mit.edu/~hugo/conceptnet/
21: Alan W. Black, Kevin A. Lenzo, Flite: a small, fast run time synthesis engine, http://www.speech.cs.cmu.edu/flite/
22: Logo Foundation, A Logo Primer or What's with the Turtles?, 2000, http://el.media.mit.edu/logo-foundation/logo/turtl
23: Arduino, http://www.arduino.cc/
24: Martin Davis (Editor), The Undecidable: Basic Papers on Undecidable Propositions, ..., 2004
25: Jean-Pierre Balpe, Principles and Processes of Generative Literature, http://www.dichtung-digital.com/2005/1/Balpe/
26: , wxPython web sit, , http://www.wxpython.org
27: Noel Rappin, Robin Dunn, wxPython in Action, 2006
28: pygame - Python game development, , http://www.pygame.org/