XS, S, M, L

Creative Text Generators of Different Scales

Nick Montfort

January 2012 TROPE–12–02

Abstract

Creative text generation projects of different sizes (in terms of lines of code and length of development time) are described. “Extra-small,” “small,” “medium,” and “large” projects are discussed as participating in the practice of creative computing differently. Different ways in which these projects have circulated and are being used in the community of practice are identified. While large-scale projects have clearly been important in advancing creative text generation, the argument presented here is that the other types of projects are also valuable and that they are undervalued (particularly in computer science and strongly related fields) by current structures of higher education and academic communication – structures which could be changed.

A technical report from

The Trope Tank
Massachusetts Institute of Technology
77 Massachusetts Ave, 14N-233
Cambridge, MA 02139 USA
http://trope-tank.mit.edu

Introduction

In *S, M, L, XL*, Rem Koolhaus discusses his realized and unrealized architectural projects, organizing them by scale. At 1376 pages, this book, designed by Bruce Mau, is itself certainly extra-large. *S, M, L, XL* is widely used for interior decorating purposes; its formidable thickness can be shown off on a shelf to good effect. But despite its capaciousness and wide-ranging discussion, Koolhaus's more focused 320-page *Delirious New York,* published the year before, has actually been cited much more often. As a communication about architecture rather than an audacious publishing project, the more "M-sized" book seems to have had greater success.

The creative computing projects that tend to be discussed at conferences and focused upon by researchers are of the "large" variety: Dissertation-sized systems with elaborate architectures and subsystems that can be extended by researchers to investigate new questions. The significance of projects on this scale is rather obvious; consider, for instance, Terry Winograd's SHRDLU, Eduard Hovy's PAULINE, Scott Turner's MINSTREL, Michael Mateas and Andrew Stern's *Façade,* and D. Fox Harrell's GRIOT.

Although this report will not look impressive on a shelf, in it, I follow Rem Koolhaus's general scheme of organizing projects by scale. I also follow Koolhaus in discussing my own projects, the ones I have the most insight into when it comes to motivation and the development process.

There are certainly "extra-large" projects as well, such as (to leave the creative text domain) Douglas Lenat's Cyc and Rodney Books's Cog. But I have no direct experience with any of these myself and have certainly not initiated any of them. Instead, I prefer to consider the "extra-small" category of creative programs that are less than even a single page long.

XS

ppg256

I have been writing a series of very short Perl programs to generate poems since late 2007. Each program in the ppg256 (Perl poetry generator in 256 characters) series [3, 4, 5] consists of exactly 256 characters of code. Each one can be run on the command line on any system that has Perl installed. The programs use no data sources except for strings in the code itself; no other programs or special libraries are invoked.

The use of non-traditional constraints in the writing process has been championed by the French mathematical and literary group, the Oulipo [6, 7]. Raymond Queneau, founder of this group, described members of the Oulipo as "rats who must escape from the labyrinth they construct," while Italo Calvino said "an Oulipian writer ... runs faster when there are hurdles on the track." Beyond the Oulipo, certain writers have recently developed significant new methods of constrained writing. Those who have done so in English include Walter Abish, Christian Bök, William Gillespie, Doug Nufer, Jackson Mac Low, and George Starbuck. In ppg256, the concept of writing under constraint is joined to the practice of programming.

Writing ppg256 in Perl was one significant constraint on programming determined at the beginning of the series. Perl is not known for supporting clarity or consistency, and has been called a “write-only” language. On the other hand, it is widely used, flexible, and well-adapted to text processing. And, programs can be written very compactly in Perl. This is reflected in an online, competitive recreation, Perl golf, in which programmers vie to write the smallest program that accomplishes a specific task.

The other main constraint, the 256 character limit, was particularly inspired by the demoscene [8], where coders work to develop amazing process-intensive generators of music videos for specific platforms. On the major demoscene site pouet.net, it’s possible to download numerous length-constrained demos, including 256 bytes; all of these limits are powers of two. The 160-character limit of SMS, 140-character limit of Twitter, and the 80-character line provide other similar constraints. The Twitter-based limit has been used to produce an album of Supercollider programs, sc140, and was used in a modified form for the interactive fiction competition TWIFcomp.

In developing ppg256-1 I first attempted to create a program whose output would be recognized by a receptive reader as a series of poems. I did not attempt to create poems like those that people in general, or any one poet specifically, have produced. To accomplish this, I found it essential to use some of the very limited amount of code producing titles, shaping the language into lines, and printing blank lines between the single-stanza poems. Generating an interesting stream of words was not enough to signal that the text was meant to be a poem or series of poems. Sketching different versions of the generator revealed to me, experimentally, the importance of these formal poetic markers. I have maintained most of them throughout most of the generators in the series.

In ppg256-1, I also sought to produce a very large vocabulary, at least a majority of which consists of English dictionary words and almost all of which at least sounds like English. I developed an unusual method of generating words because the most obvious, well-known method of generating words – using the conditional probability of letters given n-grams – will not fit in a 256-character Perl program, even for n=2. (To specify conditional probabilities for each pair of letters requires storing 26×26 = 676 probabilities.) I determined that common initial bigrams and common final bigrams of four-letter words could be joined uniformly at random to produce 450 distinct four-letter words, 273 of which (more than 60%) were dictionary words. By repeating the most common initial bigrams to weight the distribution and by adding a small bit of code to fix up a set of words, the probability of a dictionary word was improved.

The following string contains the 256 characters of ppg256-1 (between single quotes) and can be run on the command line as it appears:

```
perl -le 'sub b{@_=unpack"(A2)*",pop;$_[rand@_]}sub w{" ".b(
"cococacamamadebapabohamolaburatamihopodito").b("estsnslldsc
kregspsstedbsnelengkemsattewsntarshnknd")}{$_="\n\nthe".w."\
n";$_=w." ".b("attoonnoof").w if$l;s/[au][ae]/a/;print$l=0if
$l++>rand 9;sleep 1;redo}'
```

As with the other programs in the series, this one generates an unbounded series of poems. Although the texture of this continuously-produced language and not any particular excerpt is the point of this project, here is one example of a generated poem:

the cots

diws on dite
cabs no cons
coar to deed
mick at pock
mank at cats

The other programs in the series were developed with different specific goals, although they are all meant to generate poems. The requirement for a large vocabulary was relaxed in ppg256-2, which instead exhibits greater variety in the syntax of lines, the shape of stanzas, and the length of words:

```
perl -le 'sub p{split/,/,pop;$_[rand@_]}{$_=p("sw,-aw,&w,saw
".", "x$l);s//p("aw,w")/e;s// /g;$_="\n\nthe s\n"if$l;s/s/ws
/;s/a/p("a,the,to,of")/e;s/w/p("b,ch,f,gr,k,p,sh,s,sk,sp,tw"
)."i".p("ll,n,t")/eg;s/(b|p|f)i/$1.p("a,i")/e;print;$l=0if$l
++>6+rand 9;sleep 1;redo}'
```

the balls

the span
to grit
grit kins skit
shill shits of spill
to pill
a twit chits a bill
of sit grills shill
spall skins skin
of chill

In ppg256-3, which generates kennings in the form of compound words, the output provides very brief stanzas that are also stories – "stanzories" – that describe a potentially provocative situation and have some narrativity:

```
perl -le 'sub p{(unpack"(A3)*",pop)[rand 18]}sub w{p("apebot
boyelfgodmannunorcgunhateel"x2)}sub n{p("theone"x8)._.p("big
dimdunfathiplitredwanwax")._.w.w."\n"}{print"\n".n."and\n".n
.p("cutgothitjammetputransettop"x2)._.p("herhimin it offon o
utup us "x2);sleep 4;redo}'
```

one_dun_manelf
and
the_dim_eelbot
cut_off

one_wan_mangod
and
the__gunman
met_us

ppg256-4 was developed to drive an LED sign in an art gallery and contains all of the control code necessary to drive the sign. This generator produces a set of utterances – vaguely aggressive commands that include coined kennings. The utterances are meant to suggest a speaker of a certain gender (male) and to highlight how racial standing influences everyday speech with terms such as “dogg” and “vato.”. The possible lexical combinations for a syllable include “fag,” an offensive term that is in keeping with the tone of the generator and can remind a reader that offhand expressions include reference to sexuality as well as gender and race. These texts are further constrained to fit on the LED display:

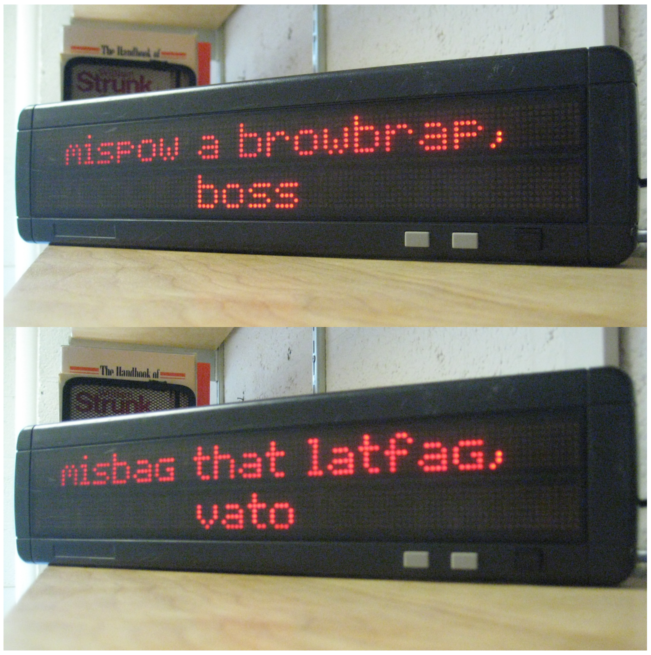

Figure 1. Example output from ppg256-4.

```
perl -e 'sub c{$_=pop;$_[rand split]}sub w{c("b br d f fl l
m p s tr w").c"ad ag ap at ay ip on ot ow"}{$=print"\0\0\0\0
\0\1Z00\2AA\33 b".c("be de mis re pre ").w." ".c("a on the t
hat")." ".w.w.", ".c("boss bro buddy dogg dude guy man pal
vato")."\4";sleep 4;redo}' > /dev/alpha
```

ppg256-5 was premiered at a poetry festival at the Banff Centre. It was written to generate a continual riff on Tristan Tzara’s Feburary 1921 “Dada Manifesto,” one which degrades over time with the introduction of misspellings and refers to contemporary poetry movements:

```
perl -le '@a=split/,/,"conceptual,digit,flarf,maximal,modern
,pixel,quiet,real";sub f{pop if rand>.5}sub w{$a[rand@a]}{pr
int f("post").w."ism ".w."s ".f("the ").w."\n".(" "x45)."WHA
T DOES ppg DO?";$a[rand@a]=~s/[aeio]/substr("aeio",rand4,1)/
e if$l++>5;sleep 5;redo}'
```

flarfism pixels the real

WHAT DOES ppg DO?

conceptualism flarfs quiet

WHAT DOES ppg DO?

postconceptualism conceptuals digit

WHAT DOES ppg DO?

ppg256-6, the latest program in the series, was developed as an example of a deterministic poetry generator. In this case, the generator always produces the same text when run at the same time. It has a very limited vocabulary but does not repeat a full line until 12 hours have elapsed:

```
perl -le '@d=split/ /,"eros won to tree for fire sex sever a
te nice tin elfin wealth";@t=split//,"_bhlmnpstw";{$_=localt
ime;/(..):(.)(.):(.)(.)/;print"\n$t[$3]".($4%2)."ck $t[$4]".
($3%2)."ck\n"if!$5;print"\\"x$5." $d[$1%12] $d[$2] $d[$3] $d
[$4] $d[$5]";sleep 1;redo}'
```

```
l0ck _1ck

 eros for tree eros eros
\ eros for tree eros won
\\ eros for tree eros to
\\\ eros for tree eros tree
\\\\ eros for tree eros for
\\\\\ eros for tree eros fire
\\\\\\ eros for tree eros sex
\\\\\\\ eros for tree eros sever
\\\\\\\\ eros for tree eros ate
\\\\\\\\\ eros for tree eros nice
```

These programs, in addition to being tiny, each resulted from at most a few days of work. They demonstrate not only that very tiny poetry generators are possible, but that a range of such generators can be developed to explore language in different ways. While these generators do not have the architectures and subsystems that facilitate the understanding of large systems, and while very compressed Perl does not make for the clearest monolithic program, these generators can be very easily transmitted. For instance, one can be sent to someone by copying and pasting it off of a Web page and into an instant message or onto an IRC channel. The code is of course not well-commented, but these programs have been documented in various blog and academic conference contexts, an approach that works well for programs that are this short. They are free and open for people to inspect and understand. A few people have sent or posted modified versions of some of the ppg256 programs; for instance, Travis Kirton sent a modified version to ppg256-5 which generates answers to that program's repeated question, "WHAT DOES ppg DO?," such as "detxtulizes typfcation / IS WHAT ppg DOES! / mutizes illmnations / IS WHAT ppg DOES! "

S

The Two

I posted three Python programs, each 1k (1024 characters) long, on November 30, 2008.[1] Each of these are very simple, but I believe interesting, story generators. The first one was modified into "Excerpts from the Chronicles of Pookie & JR" by J. R. Carpenter and used to generate texts for her book *Generation[s],* while the third one is a collaboration with an author that builds on the first. Here I focus on the second program, which I later ported to JavaScript so that it can be run simply by loading a Web page. [2] The Web version of this program is called "The Two." Like ppg-256-3, the program generates stanzories. "The Two" produces this sort of output:

The student knocks on the teacher's door.
She seeks help from her.
They pray together.

The indigent turns to the librarian.
She seeks help from him.
One ends up with a broken will.

The student knocks on the teacher's door.
She begs him.
Rude gesture meets rude gesture.

The first line is simply one selected from several possibilities; the second line has "He" or "She" followed by a verb and associated words and then "him" or "her"; the final line is also one selected from several. All choices are made uniformly at random.

The point of this generator is not a compelling new sort of computation or the production of novel situations or plots. Rather, the system is meant to challenge the reader to resolve referring expressions. In the second story above, the reader must decide whether "she" is the librarian or the indigent, and, jointly, which one is "him." The subject and object of the first line may contribute to the reader's interpretation; there is a tendency to maintain the subject of the first sentence as the subject of the second. However, our ideas of power relations (why would a librarian seek help from an indigent?) are also a factor. And, additionally, our gender stereotypes of different roles are significant: When no other information is presented, most people in United States will think of a woman when imagining a librarian, a man when thinking of an indigent.

As challenging as this can be (depending upon the particular generated story) for the reader, it can be even more challenging for a translator. Serge Bouchardon undertook a translation of this generator into French, where the gender of a character is usually indicated as soon as that character is introduced. By using the names of professions that begin with vowels and eliding what would otherwise be written beginning with the gendered definite articles "La" and "Le," Bouchardon kept the gender of characters ambiguous in the first line so that the reader is compelled to resolve pronouns according to syntax, power relations, and gender stereotypes:

L'économiste interpelle l'entrepreneur.
Il la congratule.
Six ans après, ni l'un ni l'autre ne se souvient de l'incident.

Since then, "Two Two" has also been translated into Spanish (by Carlos León) and Russian (by Natalia Fedorova).

"The Two" was written in a few hours. However limited of a system it is, it has posed some interesting questions to readers of its output. The process of translating it to another language also revealed some interesting qualities of English and French (beyond the obvious differences in grammatical gender) related to ambiguity and the gender of characters. There are additional challenges to maintaining such ambiguities in French, and even more severe ones in Spanish, but they are evidently surmountable.

Taroko Gorge

This is a poetry generator written in Taiwan's Taroko Gorge National Park and on the plane afterwards on one day, January 8, 2009. It was written initially in Python. The code with comments is one page long (that is, it does not exceed 66 80-column lines and can be printed on a single dot matrix printer page). I later ported this generator to JavaScript so that it could easily be appreciated on the Web. [9] Here is a sample of its output:

Stone ranges the rocks.
Heights sweep the vein.

  progress through the encompassing arched clear —

Shape sweeps the flow.
Heights dwell.
The crags hold.
Mist roams the shapes.

  enter the straight objective —

There are three types of lines generated. The first two, and the first and last line in the third stanza, are "path" lines meant to suggest the experience of walking along an outdoor path and seeing natural features either above or below. One or more "site" lines may be generated between these, as happens in the third stanza. These lines are less active and are meant to suggest the experience of stopping at a viewing area. Finally, the indented one-line stanzas are "cave" lines meant to suggest the experience of moving through the tunnels that, like the paths, were carved through the rock by Chiang Kai-shek's Nationalist army. In many of these tunnels, which are not lit with artificial light, it is not possible to see the exit from the entrance on the other side. This is why the ending of the sentence in a "cave" line is also unseen.

In part, "Taroko Gorge" was developed to show that just as one can visit a pleasing natural space and write a nature poem about it, one can write a generator of nature poetry and even do so at the natural site. Given this goal, it may be less of a surprise that the output of this generator is more conventional that that of the ppg256 programs and of the next system that I discuss. I was not seeking to radically explore language or to connect to the concerns and techniques of the avant-garde, but rather to extend a traditional mode of poetic composition into creative text generation. The stanzas generated here are meant to be expressive and related to experience.

On March 27, 2009, Scott Rettberg announced his "remix" of this poetry generator, which he called "Tokyo Garage." [10] Rettberg changed the strings and added more of them without modifying the code. He did not use the original one page limit, producing a 6.3k file in comparison to the original 2.7k one. As Rettberg wrote in describing his project,

> Where Montfort's poem is an elegant nature poetry generator, I attempted to develop a poetry generator about ... a city I have never been to, Tokyo. "Tokyo

> Garage” is about the idea of Tokyo, barraging us with images and archetypes absurd and sublime, in seemingly infinite variety. Reading the poem should feel much being dropped by a spaceship onto a busy corner of a strange city in the neon glow of a bustling, baffling night as the denizens stream past you now. [11]

Rettberg’s version of the generator produces text of this sort:

> *Chat client rescues the casinos.*
> *Transvestites walk.*
> *Zombies rest.*
> *Private security agents bribe the dancers.*
>
> *paint the disorganized strangely quiet disoriented peripheral--*

By May 2010 J. R. Carpenter also had remixed “Taroko Gorge” as “Gorge,” a piece about overeating and the sickeningly sweet textures of gastronomic excess. [12] Or, as Carpenter writes in describing the piece: “This never-ending tract spews verse approximations, poetic paroxysms on food, consumption, decadence and desire.” [13] The following is example output:

> *Thirst agitates the vinaigrette.*
> *Veins dissolve.*
> *Blood vessels perfume.*
> *Incisor strengthens the finger tip.*
>
> *translate the faint cardamom cinnamon liquorice -*

It is output from “Gorge” and from “The Chronicles of Pookie & JR” (as mentioned, a remix of “The Two”) that forms the basis of J. R. Carpenter’s book *Generation[s].*

Carpenter and Rettberg are both wide-ranging authors of electronic literature who, although I have not known them to typically identify as programmers, are technically proficient, expert at working with markup languages and style sheets, and have modified programs before. “Taroko Gorge” can be seen as a program, but it makes as much sense to see it as defining a boundless poetic form, like an infinite sonnet or pantoum: a path line followed by zero or more site lines followed by a path line followed by a line break and a cave line and a line break. Rettberg and Carpenter were clearly interested in playing with this form, in maintaining most of it while switching its original purpose from natural observation to urban experience or endless ingestion. They used the formal, poetic core of the program in their appropriations of it.

At the time of this writing, more than ten remixes of Taroko Gorge have been published online. In addition to the two mentioned previously, there are “Alone Engaged” by Maria Engberg, “Along the Briny Beach” by J. R. Carpenter, “Argot Ogre, OK!” by Andrew Plotkin, “Fred & George” by Flourish Klink, “Inside the House” by Adam Sylvain, “Takei, George” by Mark Sample, “Toy Garbage” by Talan Memmott, “Whisper Wire” by J. R. Carpenter, and “Yoko Engorged” by Eric Snodgrass. These are quite various in tone and suggested subject matter. “Argot Ogre, OK!” is a meta-remix which automatically remixes all of the versions presented up to that point individually and in pairs. It is likely that the small size of the original generator contributed to its widespread use for remixing and appropriation.

M

Sea and Spar Between

I collaborated with poet Stephanie Strickland, who has also done extensive work in digital media, to create a poetry generator that uses language from Emily Dickinson's poems and Herman Melville's *Moby Dick.* The processing of source text, creation of syntactical structures, and development of the interface (since this system is interactive) was significantly more involved than in my small projects and required in-person collaborative work, discussion on the phone, and many exchanges of drafts and emails. I sketched the system in Python and did the final implementation in JavaScript and HTML with canvas. The resulting JavaScript program, extensively commented, is 502 lines long. It was almost exactly a year from our first discussions on December 15, 2009 to the publication of this piece, *Sea and Spar Between,* in the very appropriately-named journal *Dear Navigator.* [14]

Like ppg256-5, this generator is deterministic; unlike that small program, it is interactive, allowing a user to navigate in unconventional ways. Instead of randomly drawing a poem from a distribution, the Sea and Spar Between defines a very large, fixed, two-dimensional space of stanzas and (in the only use of randomness) places the user initially at a random point. The generator can be invoked with coordinates as arguments, in which case it presents the specified stanza and is completely deterministic.

Since we developed a short description of the system together which is included with the project, I will quote it rather than giving my own rephrasing:

> *Sea and Spar Between* is a poetry generator which defines a space of language populated by a number of stanzas comparable to the number of fish in the sea, around 225 trillion. Each stanza is indicated by two coordinates, as with latitude and longitude. They range from 0 : 0 to 14992383 : 14992383. To operate the system, you may:
>
> - move your mouse;
> - press the spacebar to mark the stanza that is in the center of the screen of that moment, bringing its coordinates into the navigation box at the bottom in order to note them and return to this view;
> - click your mouse at the right edge of the screen to move right to a new region of texts (to increase the first coordinate); click your mouse at the bottom, left, or top to move similarly in those directions;
> - tap the arrow keys to move the visible lattice of stanzas up, right, down, or left by a single stanza;
> - scroll the wheel on your mouse or tap the A and Z keys on the keyboard to zoom in and out;
> - type a pair of coordinates into the navigation box at the bottom and press *enter* to move anywhere in the sea of text.

Here are some of the stanzas defined by the system, a few of the fish in the sea, along with their coordinates:

swerve me?
for pangless is the earth

one care one show one star one pain
paradise! dying!

(3034896, 6450568)

cut to fit the bardblot course
nailed to the coffin

wheel on
hueless sing and sleep

(3058586, 4994118)

loose-fish
another! alone!

loose-fish
nailed to the groove

(10486167, 5840830)

In the browser, these appear (when one types in the coordinates or navigates to them) as the central stanzas in a lattice of stanzas, so there is more to read on the screen at any point – unless the user has zoomed in so far that only one stanza can be be seen.

Strickland and I worked on all aspects of the project together, jointly determining the strings that are assembled into lines, the way those lines are assembled, the shape of the stanzas, and the overall interface and visual appearance of the piece. Although I wrote the Python and later JavaScript code, we worked together closely on all of these aspects and even on properly commenting the code as the new issue of *Dear Navigator* was preparing for launch.

Sea and Spar Between was widely announced on major poetry blogs and has received more attention in the poetry community than any of these other projects. It has not yet been taken up by others and transformed, however. This generator has only been available for a few months, and, as of this writing, we have only presented it together on one occasion. It is explicitly licensed as free software, with the hope that poets and programmers will use some or all of it in their own projects.

While medium-length projects may be more difficult for others to casually appropriate and remix, they may begin to provide a framework for future reuse, allowing modules, rather than the entire piece, to be taken and reworked. They also allow for the extended investigation of a poetic concept.

L

Curveship

This is an interactive fiction system that can automatically change the way that it narrates. It was developed beginning in early 2006, initially under the name "nn," and used in my dissertation research, which was completed in June 2007. The system is still in development; it was first released as free software in February 2011 at <http://curveship.com>.

The system, as with any full-featured interactive fiction system, includes a world model, including representations of characters, objects, and locations. To this Curveship adds first-order representations of actions and a model of the narrative discourse, so that the narration and description of the simulated world can change. A "string-with-slots" formalism allows individual sentences representing actions or describing items to be modeled in a way that is reasonably easy to author while providing the ability to flexibly realize many different output strings and accomplish several significant narrative variations. Curveship can tell events out of order, using flashback and other techniques, and can tell the story from the standpoint of particular characters, based on their perceptions and understandings.

Although Curveship is an interactive fiction system, it is easiest to explain its facilities for narrative variation if a fixed story (the same sequence of actions in the same storyworld) is considered. While this leaves aside a major aspect of the system, it allows for clear discussion of Curveship's creative text generation abilities. The system is provided with an example file that allows it to tell a (non-interactive) story about a bank robbery in many ways. This is the default narrative, told in chronological order, narrated as if during the events themselves, and without a diegetic narrator or narratee:

The twitchy man puts on a Dora the Explorer mask.
The twitchy man leaves the street.
The twitchy man examines the lobby.
From here, the twitchy man is able to see: inside the vestibule, some deposit slips, the bank teller, and a black bag.
The bank teller waves to the twitchy man.
The twitchy man brandishes a gun-shaped object at the bank teller.
The bank teller laughs.
Something sees the twitchy man.
The bank teller puts something in the black bag.
The twitchy man turns to the burly guard.
The burly guard shoots the twitchy man in the chest.
The burly guard shoots the twitchy man in the chest.
The twitchy man falls.
The twitchy man dies.
The bank teller cries.

The same underlying events can be narrated with a different *spin.* For example, the story can be told in reverse-chronological order, with an appropriate shift of tense after the first sentence:

The bank teller cries.
Just beforehand, the twitchy man died.
Previously, the twitchy man fell.
Earlier, the burly guard shot the twitchy man in the chest.
Previously, the burly guard shot the twitchy man in the chest.
Previous to that, the twitchy man turned to the burly guard.
That was after the bank teller put something in a black bag.
Just beforehand, something saw the twitchy man.
Previously, the bank teller laughed.
Previous to that, the twitchy man brandished a gun-shaped object at the bank teller.
A moment before the bank teller waved to the twitchy man.
Before that, the twitchy man examined the lobby.
From there, the twitchy man was able to see: inside the vestibule, some deposit slips, the bank teller, and the black bag.
Earlier, the twitchy man left the street.
Previously, the twitchy man put on a Dora the Explorer mask.

A narrative that is focalized by the guard (so that it includes only those events known to the guard) and also told by the guard (so the guard becomes the "I") to the bank teller (the "you") can also be produced:

I was snoozing.
The twitchy man brandished a gun-shaped object at you.
I woke.
I saw the twitchy man.
I saw that the twitchy man possessed the gun-shaped object and was wearing a Dora the Explorer mask.
I left the guard post.
I examined the lobby.
I saw the twitchy man. The gun-shaped object was held by the twitchy man. The Dora the Explorer mask was on the twitchy man.
From there, I was able to see: inside the vestibule, some deposit slips, you, and a black bag.
You put something in the black bag.
The twitchy man turned to me.
I shot the twitchy man in the chest.
I shot the twitchy man in the chest.
The twitchy man fell.
The twitchy man died.
You cried.

This is the beginning of a narrative focalized by the guard and told by the guard to the the bank teller, hesitantly:

I, uh, was snoozing.
The twitchy man brandished a gun-shaped object at you.
I woke.

I saw the twitchy man.
I, uh, saw that the twitchy man possessed the gun-shaped object and was wearing a Dora the Explorer mask.
I left the guard post.
I examined the lobby.
I saw the twitchy man. The gun-shaped object was held, uh, by the twitchy man. The Dora the Explorer mask was on the twitchy man.

The next excerpt is told in a prophetic way (the narrator speaks from a time before the events have happened) by a narrator who is hesitant, somewhat pleasantly surprised, and speaks in the stereotypical style of a "valley girl":

The twitchy man will put on a Dora the Explorer mask!
The twitchy man will leave the street! Wow!
The twitchy man, um, like, will examine the lobby!
From, like, uh, there, like, the twitchy man will be able, like, to see: inside, like, the vestibule, some deposit slips, um, er, the bank teller, like, and, like, a black bag, man! Wow!
The bank teller, like, uh, will wave to the twitchy man!
The twitchy man, like, will brandish, like, a gun-shaped object at the bank teller!

Curveship can do all of this and more not just with fixed representations of stories but also based on user interaction in the context of a simulated world, as is typical in interactive fiction. The system models the world as a tree of items, with edges representing different child relationships so that actors can be contained with rooms, objects can be on tables, etc. There is also a rich model of every action that has transpired in the storyworld. The system checks to see whether attempted actions will succeed in the world as it has been defined. Its persistent representation of actions allows for flashbacks and other anachronies. The simulation of the world is abstracted from the generation of language.

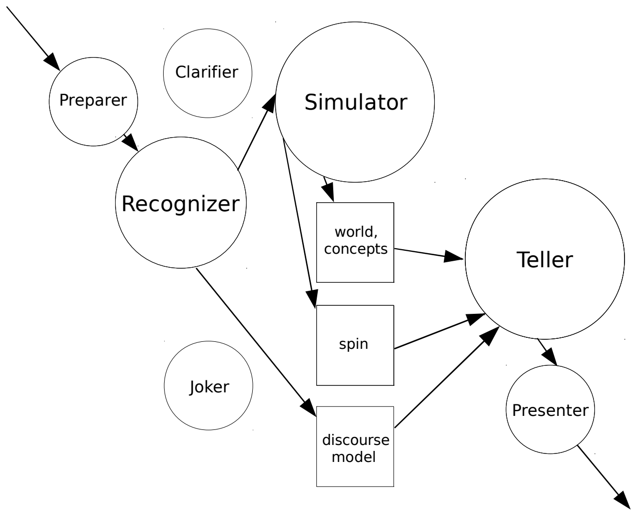

Figure 2. The architecture of Curveship. The important functions of an IF system are modularized, with the Simulator dealing exclusively with changes to the world model and to concepts and the Teller dealing exclusively with producing the narrative text.

In addition to maintaining a world model, Curveship also maintains a *concept* for each actor, representing that actor's theory of the world based on knowledge (specified initially) and perceptions (as the simulation proceeds). A concept is be used to focalize a particular actor.

From a research standpoint, one of the major goals of the project was to computationally connect high-level narrative concepts to low-level grammatical specifics. The programmer creating an interactive fiction in Curveship does not ever have to specify person or tense. If that were necessary, it would, in general, be necessary for each sentence. Instead, the assignment of narrator and narratee, the positioning of the narrator in time, the ordering of events, and other narrative factors are used by Curveship to determine the grammatical particulars.

One of the major contributions of Curveship, in terms of narratology and text generation, is a new representation of the order of events in the telling, allowing anachronies such as flashback, retrograde narration, and sylleptic narration (narration by category) to be generated. Instead of using Gérard Genette's simple sequence of re-ordered actions (e.g., "341256") Curveship uses an ordered tree representation which captures *why* actions are being reordered, so that an main sequence with flashback can be narrated differently than a syllepsis. To generate the appropriate tenses from a high-level narrative representation, the programmer indicates whether how the time of reference and time of speech should be defined for an ordered tree overall – not for each action being represented. The event time is determined by the simulation. Using Hans Reichenbach's theory of tense, these times specify how verbs in the output sentence should be realized. [15] It is unlikely that this result could have been obtained in a small- or medium-sized system. Much of Curveship's heavy machinery (its three-stage pipelined natural language generator and its simulation of events with timestamps, for instance) is necessary for this result.

Curveship has the ability to add non-narrative stylistic variation through the use of output filters. The examples of these that were discussed previously are the hesitant, surprise, and valley girl filters. These almost always impressive when shown in demonstrations, particularly when it is shown that they can compose arbitrarily. They are not, however, part of the core research project, since they are not variations in narrative specifically and can be applied to any text. There is also little in Curveship that enables them. The system does have a lightweight model of sentence structure and prevents interjections from occurring within verb phrases, for instance. Also, it knows where sentence boundaries are. But it would probably not be much harder to apply these filters to arbitrary plain text. Nevertheless, these output filters – each one a decidedly small-scale project in itself – are effective in the context of Curveship, which also offers more difficult-to-achieve types of narrative variation. They demonstrate that "S" subsystems (which might have been developed independently) can add value and interest to "L" systems.

As of this writing, a small number of "spins" and one complete interactive fiction in Curveship have been written by people other than the developer of the system, and the system has been used in teaching narrative theory and interactive narrative. Several publications have resulted, including one reporting preliminary results from integrating Curveship with a plot generator, MEXICA. [16]

Inquiry into Creativity and Language

With the exception of Curveship, these projects were not undertaken with scientific inquiry as a major purpose. They are not meant as trivial curiosities, either. Rather, they are critical investigations of creativity and language developed through the practices of programming and the literary arts. The small and medium projects described here are offered as a complement (not an alternative) to large-scale, rigorous systems that are driven by ontologies or large stores of data or have elaborate architectures motivated by cognitive models or concepts from other fields.

For example, a very simple system such as “The Two,” although not developed as a scientific project, can nevertheless inform a more elaborate story generator that is designed to tell stories with male and female characters, to generate pronouns and other referring expressions, and to use gender ambiguity in its production of narrative discourse. “The Two” and “Les Deux” show that there are general ways of producing such ambiguous reference in two languages, one in which the scope for such reference is significantly more limited. If one believes that the production of different sorts of ambiguities is important to the creation of poetry and other sorts of literary art, as has often been asserted (e.g., [17]), it could certainly be useful to incorporate this non-scientific result.

These projects all arise from the same interest in exploring language and computation. Instead of relegating them exclusively to distinct contents (this one on a blog, this one at a literary festival, this one in a predominantly computer science conference), I suggest that it is useful to see the relationship of the projects to one another and to examine how insights can be achieved from working on different scales.

Remixing as a Metric for a System’s Value

Citation analysis is a basic technique in the field of bibliometrics, used to determine the impact of an academic publication. When sophisticated bibliometric techniques are used, citations are usually weighted to approximate their value. In addition to looking at the different value of different sorts of publications, ideally, the citation of a book as an offhand example should be weighted less strongly than a citation that indicates core importance and intellectual influence.

One of the forms of citation that expresses the greatest influence is remixing or other direct use of an pre-existing system (as a subsystem of or direct inspiration for a new system, for instance). That a student is reimplementing and modifying MISTREL as a Ph.D. dissertation project strongly suggests that MINSTREL is a system of lasting value, as much as any citation suggests this. [18] The main value of WordNet and ConceptNet are seen in their use in other systems. But mainstream citation measures do not perfectly track the remixing, reimplementation, and reuse of systems, particularly small-scale systems that are important in artistic as well as scientific contexts. For instance, as of the end of April 2011, Google scholar shows only 9 citations to Olia Lialina’s significant early net.art piece “My Boyfriend Came Back from the War,” from 1996. Liliana’s site links to her work and documents 22 remixes, reworkings, and pieces inspired by her piece. [19]

The value of having pieces translated, remixed, recombined, and further developed is clearly great, but the extent of this activity can be greatly understated by standard citation metrics. One way to value small systems better is to consciously and explicitly value remixing in addition to standard citation.

Gaining Space for the Small (and Extra-Small)

Small-scale systems can have many types of significance. They can be picked up and modified by others, contributing directly to new types of aesthetic cultural production. They can be instructive and provocative, challenging the ideas that have been developed using exclusively large-scale systems. They can be used in teaching to prompt discussion or as the basis for student work. And, of course, they can be used to sketch and explore so that one's efforts are better applied when later undertaking medium- or large-scale work.

Although not discussed in this report, I have developed a set of poetry generators even smaller than ppg256. This set, "Concrete Perl," consists of four 32-character Perl programs that generate moving concrete poems in a terminal window or on a console. These are not as clearly oriented toward narrative or standard poetic questions as are the systems covered in this paper, but they show that the lower boundary for extra-small systems is probably even lower than 256 characters.

To turn to projects on the other extreme: The potential of large-scale projects such as Curveship is without doubt. The way that large-scale projects are valued is also not at risk. Nor am I proposing that smaller-scale projects replace larger ones.

Despite the worth of small-scale projects, the contexts of computer science and engineering, and interdisciplinary contexts based on these disciplines, discourage such projects and suggest that time be spent on large-scale systems. Only if a small-scale system is developed as an example during a longer discussion, or as a subsystem of a larger system, is it likely to be seen as valuable.

The dissertation is often the occasion for the development of a large-scale system, just as a master's thesis and undergraduate thesis often occasion medium-sized systems that occupy as much student time as possible. It seems that occasional smaller-scale projects could be intellectually invigorating as well as productive, but the drive toward a monolithic dissertation project discourages such work and discourages students to connect insights from any smaller projects with those in their large-scale one. One way past this barrier is to encourage the inclusion of smaller-scale projects that relate to the major project in the dissertation. Michael Mateas did this in his dissertation, which focused on the large-scale system *Façade.* He wrote about *Subjective Avatars, Office Plant #1,* and *Terminal Time.* These three were not small-scale projects, and the main discussion was relegated to appendices, but they show how independent, smaller-scale expressive AI endeavors can be usefully related to the main system. Students are expected to understand their research area well by understanding the literature; why is it inappropriate for them to occasionally spend as much time as they would spend struggling with an conference submission in the creation of a small-scale project that relates to their research question?

Open-ended, short term "competitions," along the lines of a hackathon, codefest, or demo party, could give additional space for small-scale projects to be created, shown, and appreciated. These can be arranged at conferences as well as universities. Conferences currently offer a venue for "short papers" (often position papers rather than papers about small-scale systems) and "late-breaking results," but inviting a wider variety of systems in demo sessions and hosting friendly contests that encourage the development of such systems, perhaps collaboratively, would enhance the opportunities for producing and discussing small-scale systems.

If I did not have the experience of writing small-scale systems, it is unlikely that I would have added one of Curveship's most impressive, if not very research-related, features: output filters. The small-scale systems I have created have also proved more inviting and understandable to people outside computer science, which has allowed for a broader discussion and the effective and surprising remixing of my work. Using only remixing, translation, and other direct reuse of the system as a metric, benefits do seem to accrue from the development of small-scale systems. Since the cost of developing these systems in terms of time and effort is also so low, it seems reasonable to make more space for them in undergraduate and graduate education and in our conferences.

References

1. Montfort, Nick. "Three 1K Story Generators." *Grand Text Auto.* November 30, 2008. http://grandtextauto.org/2008/11/30/three-1k-story-generators/
2. Montfort, Nick. "The Two." *nickm.com.* [JavaScript and Python, with French, Spanish, and Russian translations.] http://nickm.com/poems/the_two.html.
3. Montfort, Nick. ppg256 series. *nickm.com.* n.d. http://nickm.com/poems/ppg256.html
4. Montfort, N. "The ppg256 Series of Minimal Poetry Generators." *Proceedings of the Digital Arts and Culture Conference,* 2009. http://www.escholarship.org/uc/item/4v2465kn?display=all
5. Marino, Mark. "The ppg256 Perl Primer: The Poetry of Techneculture." *Emerging Language Practices* 1. 2010. http://epc.buffalo.edu/ezines/elp/issue-1/ppg256.php
6. *Site officiel de l'OuLiPo, Ouvroir de Littérature Potentielle.* http://oulipo.net
7. Mathews, Harry and Alistair Brotchie *Oulipo Compendium*. London: Atlas, 1998.
8. Tasajärvi, Lassi. *DEMOSCENE: The Art of Real-Time.* Evenlake Studios. 2004.
9. Montfort, Nick. "Taroko Gorge." *nickm.com.* n.d. http://nickm.com/poems/taroko_gorge.html
10. Rettberg, Scott. "Tokyo Garage." *retts.net*. n.d. http://retts.net/tokyogarage.html
11. DAC 09 Literary Arts Extravaganza Gallery. *Writer Response Theory.* n.d. [Click on "Scott Rettberg."] http://writerresponsetheory.org/dac09/gallery.htm
12. Carpenter, J. R. "Gorge." [Poetry generator.] *Luckysoap & Co.* n.d. http://luckysoap.com/generations/gorge.html
13. Carpenter, J. R. "Gorge." Blog post. *Luckysoap & Co.* May 26, 2010. http://luckysoap.com/lapsuslinguae/2010/05/gorge/
14. Montfort, Nick and Stephanie Strickland. "Sea and Spar Between." *Dear Navigator* 2. Winter 2010. http://blogs.saic.edu/dearnavigator/winter2010/nick-montfort-stephanie-strickland-sea-and-spar-between/
15. Montfort, Nick. "Ordering Events in Interactive Fiction Narratives." *Intelligent Narrative Technologies: Papers from the 2007 AAAI Fall Symposium,* pp. 87-94. Eds. B. Magerko and M. Reidl. 9 November 2007.
16. Montfort, Nick and Rafael Pérez y Pérez. "Integrating a Plot Generator and an Automatic Narrator to Create and Tell Stories." *Proceedings of the 5th International Joint Workshop on Computational Creativity,* pp. 61-70. Eds. P. Gervás, R. Pérez y Pérez and T. Veale. 17-19 September 2008.
17. Su, Soon Peng. *Lexical Ambiguity in Poetry.* Longman: New York. 1994.
18. Tearse, Brandon, Michael Mateas and Noah Wardrip-Fruin 2010. "MINSTREL Remixed: a rational reconstruction." *Proceedings of the Intelligent Narrative Technologies III Workshop* (INT3 '10). Article 12.
19. Lialina, Olia. *Last Real Net Art Museum.* n.d. http://myboyfriendcamebackfromth.ewar.ru/
20. Mateas, Michael. "Interactive Drama, Art and Artificial Intelligence." Ph.D. Diss, Computer Science Department, Carnegie Mellon University. December 2002.