A Response to Nick Montfort's "Programming for Fun, Together"

I want to underline and further explore a few of the ideas in Montfort's talk that I think opens some important questions for those of us creating literary experiences made for digital media. I need to say at the outset that I am not Rita Raley, and I am not even close to being a simulacrum of her, and I apologize for that. In fact, I deeply wish that a ferocious hurricane had not thrown us all into the situation of finding a replacement for her on short notice, basically anyone on hand with a Ph.D. who can respond to Nick Montfort in intelligible sentences, perhaps while wearing a blue suit. I am unfortunately that man, though I am not packing a blue shirt.

Were Rita here, I think she might have some provocative questions for Montfort, and perhaps some of these provocations would be critical in nature. For instance, she might have attacked the very fundament of Montfort's reasoning, challenged him in a way that could have provoked him to scale new heights or hop over seemingly impossibly high fences or for that matter to use vines to swing over pools filled with crocodiles or to climb ladders to ascend new platforms in order to cast wooden barrels upon opponents and remove strange obstacles put before him. She might have required him to get a lamp to find his way out of a dark room or through a maze of twisty little passages, all alike, or to otherwise solve a complex puzzle or riddle.

For my part, I can mostly only agree, provide a couple of expanding examples and ask for further elucidation, for more light. I need to explain that I have collaborated with Nick Montfort on a number of different creative writing projects, some of which involved the mere act of writing together towards a certain end, and did not involve programming in any computational sense. Let me start there with a couple of examples.

Sometimes these projects have involved procedures and constraints: to provide an example, we coauthored a novel "Implementation" designed to be published and distributed on 240 business sized shipping labels, which would then be placed in various public locations in the physical world and photographed. After some testing, we agreed that the amount of text that could fit on a business size shipping label is roughly the same as would fit on a handwritten 3x5 index card. We agreed to some themes, some schedules, some sketches of characters. Then we began to write together, each sketching a short narrative on a card and then passing them to the other. We would then edit, scratch out, insert, revise, the others' text, and then copy it over to other card, or, in certain cases, simply draw an X through the text written on the card and explain why it was completely unsuitable, lame-brained, stilted, inconsistent, ill-phrased, unpoetic, tin-eared, deeply offensive, or irrelevant to the progress of the story at hand. Then we would pass it over again. Some of these card would go through multiple rounds of back and forth. Sometimes we would spend somewhat ridiculous amounts of time in argument over a single phrase or even a particular word. The limitations of the space of a sticker requires a certain attention to economy, and when writing with a friend, you must from time to time murder their little darlings as well as your own.

This was constrained writing, writing according to a certain procedural practice, and it was clearly collaborative, but it was not programming.

Something worth considering and talking more about, perhaps, is the relationship between constrained collaborative writing and social programming.

Another less direct example I will provide is that of Nick's "Taroko Gorge" project, a poetry generator written in javascript for the web. It is a relatively simple, comparatively elegant computer program that, when run in a web browser, produces a nature poem, about the main feature of a national park in Taiwan, a gorge with a beautiful waterfall in nature surroundings. As the poem begins down the the browser, the reader at first might think this is a linear poem, which will meditate on this natural wonder for a while, and then arrive at a sensible end. However, slowly, as certain phrases begin to reappear in new configurations, with other words and phrases we have seen before, there is a moment of reveal. At some point we understand that this is not a simply animated nature poem, but instead the product of a poetry generator, a combinatory script that is writing a new version of the poem every time it is run, and further one that will not stop writing itself and filling the screen with language until we shut the browser window and banish it from our lives.

When I first saw "Taroko Gorge" I was immediately delight with this moment of reveal, and with the indefatigable energy of this little poetry program. I also appreciated the irony of its inversion—what seemed on first glance to be a meditative nature poem was actually the product of a computer program, an algorithmic cybertext. However, I have never been much of a fan of nature poetry, and I near always feel an urge to jump in and improve Nick's text. Opening the program in my web browser through the simple act of selecting "view source" I was able to quickly gain access to the textons of Taroko Gorge, to explore the structure of the program and see its variables. Without asking Nick's permission, or really informing him at all, I set about rewriting it, first by substituting what I perceived as a rather limited vocabulary with one that is considerably more verbose. I was interested in seeing if I could take the basic structure of Nick's poem and invert its meaning yet again, taking a minimalist poem about ordered nature in which humans are virtually absent, and turning it into a maximalist poem about the chaotic city in which humans are everywhere in all their imaginative messiness. I wanted to take Nick's rather reflective, somewhat serious tone, and explode in a more comical, rather absurdist direction. And so overnight "Taroko Gorge" became "Tokyo Garage." I changed all the variables, and thus changed all the language of the poem, tweaked a few aspects of the program so that it would have a slightly different color scheme and speed of rendition, crossed out Nick's name on the right hand side of the page and inserted my own above it, and then uploaded the poem on my own website. Finally, I emailed Nick the URL of the new poetry generator, and waited a couple hours to see how he, on viewing it from the other side of the ocean, would respond.

That last part is important. During the hours in which I was rewriting Nick's poem, I of course knew that there was at least one other person in the world who would be amused by and appreciate this act of overwriting, of hacking, of remix. I had an audience in mind. That audience was Nick.

This was, in essence, an elaborate but not terribly involved joke, performed in public but intended for shared amusement with a friend. I think that neither Nick or I could have imagined what happened in the months and years thereafter. Several months after "Tokyo Garage" fellow electronic literature author J.R. Carpenter produced another iteration, titled simply "Gorge" -- in this case the new program produced poems that describe in some obsessional detail processes of eating, of foods exotic and mundane working their way through the human gustation system. J.R. would go on to produce several more remixes, and soon this was joined by Talan Memmott's "Toy Garbage", Eric Snodgrass's "Yoko Engorged", Mark Samples "Takei, George", Maria Engberg's "Alone Engaged", Flourish Klink's "Fred & George", Andrew Plotkin's "Argot, Ogre, OK!" and many others. At this point, it may be more difficult to find a writer working in the e-lit community who has not hacked and rewritten "Taroko Gorge." And it is remarkable every time. Without a great deal of programming knowledge or even a great deal of work, each author is able to work from the original code base and produce a system that in turn produces poetry centered on radically different themes and tropes than the last. This bit of kludgery has become a genre of poetry.

I may have slightly diverged here from the points made during Nick's talk but I hope that both these examples offer some ways of thinking about how Nick's ideas of programming, and might insert more generally collaborative creative writing in the networked context, can be understood as social acts of generative play.

Montfort sums up four of his main points. Let's come back to those and consider their potential wisdom, just as we might consider the potential that could be programmed within a poetry generation system. I'll also try to phrase a few questions here that I hope can open up a conversation with Nick.

• Programming is a social as well as a cultural activity.

This can be the case, but maybe something to consider is whether the opposite might also be true. Might programming sometimes also be an anti-social or or anti-cultural activity?

• Programming is a deep engagement with computation that can connect the power of the computer to creative purposes in ways that other practices cannot.

A question which is often asked from the perspective or theorists of critics of electronic literature is the level to which critics of e-lit should also be deeply engaged with programming and computation. Does a critic need to be able to parse the code of a poetry generator in order to form a coherent response to it?

Also following on that—some years back Michael Mateas famously made the argument that "writers must be programmers"—that is essentially to say that those of us who are interested in creating narrative or poetic works for digital media must first tackle our computational literacy. To what extent do you think this is essential or true?

I have always been prompted to reply: "Yes . . . and programmers must be writers." That is to say that neglecting engagement with the apparatus and structure of stories, or ignoring the rhythms, tonal qualities, diction, language, etc. of the poem can be equally destructive for electronic literature. How do we, for example, in creative writing programs for digital media, balance training and engagement with computational literacy, with engagement with the process and craft of literary writing in itself?

• Programming communities are related to computational platforms, longstanding art and media practices, and communities of practice beyond programming itself.

While I obviously agree with this one, I wonder to what extent those relationships are understood, and how we can facilitate better communication between for example communities of art practice that have structural and cultural similarities to programming communities? How can productive outreach take place?

• Programming is not an activity restricted to professionals with years of training; some essentials of this activity can be undertaken (and have been undertaken) by ordinary computer users after a few hours.

I think this is a very important point and I wholeheartedly agree, but of course there are going to be different levels of hacking mastery here, right? For instance, I am not a master of JavaScript, or Ruby, or even Processing, and I sort of need to relearn programming every time I create a new piece that involves programming. It is easy, but also not *that* easy.

What can be done, Nick, to provide avenues of entry to programming for non-programmers? How can we as a field make the learning curve less steep, more iterative, more engaging, and more playful?