there are more of us participating in each instance of making/using than we might assume and where that 'us' is composed of, amongst others, non-human agents. In this respect a key interest of creative engagement with digital technology is the manner in which such relations can be rendered explicit.

June 2012, Edinburgh

Bibliography:

Aarseth, E. (1997) *Cybertext: Perspectives on Ergodic Literature*. Baltimore: John Hopkins University Press.

Biggs, S. & Leach, J. (2004) *Autopoiesis, Novelty, Meaning and Value*. London: Artwords.

Biggs, S. & Travlou, P. (2011) 'Distributed authorship and creative communities' in *Dichtung Digital* Nr. 41 (2012 ISSN 1617-6901), Basel: http://www.dichtung-digital.org/2012/41/biggs-travlou/biggs-travlou.htm [accessed 15.05.2012].

Block, F. W. (1999) *Beobachtung des 'ICH'*. Bielefeld: Aisthesis.

Bolter, J. & Grusin, R. (2000) *Remediation: Understanding New Media*. Cambridge MA: MIT Press.

Foucault, M. (1980) 'Confessions of the Flesh' in Gordon (ed.) *Power/Knowledge: Selected Interviews and Other Writings*. Hassocks, Sussex: Harvester Press.

Heidegger, M. (1977) *The Question Concerning Technology and other essays*. New York: Harper and Row.

Ingold, T. (2008) 'Bringing things to life: Creative entanglements in a world of materials', a paper presented at the *Material Worlds symposium*. Providence: Brown University, http://proteus.brown.edu/cogut-materialworlds/4080, [accessed February 26, 2010].

Logan, R. K. (2005) 'The Extended Mind Model of the Origin of Language and Culture' in Gonthier, N. van Bendegem, J. P. & Aerts, D. (eds.) *Evolutionary Epistemology, Language and Culture*. Dordrecht: Springer.

Maturana, H. & Varela, F. (1972) *Autopoiesis and Cognition: The Realization of the Living*, Dordrecht: D. Reidel Publishing.

McLuhan, M. (1964) *Understanding Media*, New York: McGraw Hill.

Simanowski, R. (2011) *Digital Art and Meaning: Reading Kinetic Poetry, Text Machines, Mapping Art, and Interactive Installations*. Minneapolis: University of Minnesota.

Vattimo, G. (1997) 'Beyond Interpretation: The Meaning of Hermeneutics' in *Postmodern Culture*. trans. by Webb, D. Stanford: Stanford University Press.

Winograd, T. (1991) Sheehan, J., & Sosna, M., eds., *The Boundaries of Humanity: Humans, Animals, Machines*, Berkeley: University of California Press.

PROGRAMMING FOR FUN, TOGETHER

Nick Montfort

Ever since computers have been programmed, people have programmed them together. From almost the first days of programming, people have also programmed them unofficially, for fun, to create literary and artistic works, games, and technically impressive feats that suggest new directions for computing.

In September 2010, at the first ELMCIP seminar in Bergen, I discussed the interactive fiction community, which includes programmer/authors as well as those focused mainly on programming; avid reviewers and critics; people who run contests, in-person events, and online community resources; players; and enthusiasts of other sorts. In this discussion that I have developed for the final ELMCIP conference in Edinburgh, my topic is in many ways broader, although in one respect it is more limited. Broader, because I am not restricting myself to the discussion of interactive fiction or even electronic literature – I am considering creative computing generally. Narrower, because I focus on one type of community participant and one way of engaging with creative computing – as a programmer.

I will present relevant scans, photos, and video to illustrate how programmers have worked together in the area of creative computing. I will also try to make my fourth point (below) by offering concrete examples of how anyone who is conversant with computers can begin programming. In this article, I provide a brief discussion of three types of creative programming practices.

Four Main Points

I have four main points to make about programming:

- Programming is a social as well as a 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.
- Programming communities are related to computational platforms, longstanding art and media practices, and communities of practice beyond programming itself.
- 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.

These points are interrelated, so I will argue for them by looking at the specific ways that programming has been done at different points in the past. Not exactly a cohesive history, not an archaeology, not a fully traced genealogy, I offer instead simply a few glimpses of how programmers have worked over the years in different contexts. To be clear, I am really considering not how they have worked, but how they have played. That is, I am considering how programmers have engaged in creative computing.

Human Moments in Programming

There are many examples of social programming from the earliest days of general-purpose electronic computing, when women worked as 'coders' (as they were initially called), programming the ENIAC. Whether it is the development of a new Data General computer (Kidder 1981) or the early work to define and enable the Internet (Hafner & Lyon 1996), work with computation is clearly not isolated from society, and programming, however wizardly it may seem, is not an abstract and hermetic activity. As many writers have explained, social programming is not restricted to creative, unofficial uses of the computer. Teams work together on scientific projects, military applications, and business systems. It is the creative and unofficial type of computing, however, that seems to connect to the development of electronic literature most directly.

‘Recreational Computing’ and Early Games Programming at MIT

In 1958 an experimental computer with some of its memory removed was effectively donated (more precisely, loaned) to the Research Laboratory of Electronics at MIT. This was the Transistorized Experimental Computer Zero, called TX-0 (and pronounced ticks-oh). Many of the students who descended on it to become the first hackers knew each other from their work in another technical community, the Tech Model Railroad Club or TMRC (tee-merk), which had an elaborate model railroad layout that used an extensive system of relays.

The TX-0 was one of the first systems where programmers could interactively write programs for fun, engaging in ‘recreational computing.’ Game and proto-game programs were developed including a tic-tac-toe game and ‘Mouse in the Maze.’ The latter game -like program let the user employ the light pen to place the mouse and the cheese that was its goal; there was also a mode in which in the mouse consumed not cheese but martinis, becoming less and less able to navigate the maze as it did so.

In 1961, MIT’s Electrical Engineering Department received a new and more powerful computer, the first minicomputer, from Digital Equipment Corporation. This PDP-1 became the new focus of hacker attention. Pattern-generating programs and ‘Expensive Typewriter,’ possibly the first word processor, were developed on it. The most famous program written by hackers on the PDP-1 was surely *Spacewar* (Graetz 1981). It was first imagined by Steve ‘Slug’ Russell, Martin ‘Shag’ Graetz, and Wayne Wiitanen, who all lived on Hingham Street in Cambridge, MA, in a residence that came to be known as The Hingham Institute Space Warfare Study Group. The game was augmented by Dan Edwards and Peter Samson and achieved wide fame thanks to a write-up in *Rolling Stone* (Brand 1972).

More than a decade later, the play in the system at MIT was still allowing programmers to code for fun (Montfort 2003). Some of the results in the 1970s included *Maze*, which Greg Thompson, Dave Lebling, and others developed into a sophisticated multiplayer game in 1974. *Maze*, the progenitor of *Maze War* and the first first-person shooter, ran on the Imlac PDS-1. This platform was used mainly a terminal in the Dynamic Modelling Group. Lebling and three others developed the famous interactive fiction *Zork* starting in 1977. Those that developed Zork and went on to found the successful company Infocom had a few things in common besides the general affiliation with MIT. One was the Dynamic Modelling Group, but another was the Lecture Student Committee, an organisation at MIT that arranged screenings of films.

Programming on Home Computers

The ability to program a computer, to use its general power in customised ways, was a core selling point for many home computers of the late 1970s and early 1980s. Home computers were often positioned against videogame systems in advertisements. Implicitly, this comparison reminded the prospective buyer that a computer could be used to play video games; explicitly, it pointed out that computers could be used with business and educational software – and that they could be programmed to do much more. This point was driven home in the many Commodore TV ads that compared the VIC-20 to game systems – including one in which William Shatner says ‘unlike games, it has a real computer keyboard.’ (Commodore Computer Club 2010).

That computers were programmable, and that they specifically could be programmed in BASIC, were hardly afterthoughts in their development or marketing. A Commodore 64 advertisement that was aired in Australia in 1985 provides evidence that BASIC was a central selling point (Holmes3000 2006). After the television spot showed bikini-clad women descending a waterslide (‘♪ In a world of fun and fantasy . . . ♪’) and cut to a woman happily using a Commodore 64 in a retail store (‘♪ . . . and ever-changing views . . . ♪’), it cut once again: to a screen full of BASIC, and then to depict a boy programming in BASIC (‘♪ . . . and computer terminology . . . Commodore and you! ♪’). The commercial clearly signals that programming was an obvious, important, and fun use of a home computer.

An early print ad for the Apple II that ran in *Scientific American* among other publications boasted, ‘It’s the first personal computer with a fast version of BASIC – the English-like programming language – permanently built in. That means you can begin running your

Apple II the first evening, entering your own instructions and watching them work, even if you’ve had no previous computer experience.’ It was very easy for home computers users to type in or modify a BASIC program, and the fact that the manufacturers encouraged such behaviour in mass media advertising primed users to partake of programming once they’d purchased a machine.

It isn’t necessary to head to YouTube to find evidence that ordinary users were supposed to start programming in BASIC in the late 1970s and early 1980s. The standard user manuals that came with such computers included sections on BASIC or instructions on how to program in BASIC throughout. Programmers had opportunities to collaborate when they gathered in schools, during the meetings of user groups, and in retail stores (which often allowed children to spend time there programming computers).

The Demoscene

The constellation of creative practice known as the demoscene (Tasajärvi 2004, Carlsson 2009) is concentrated in Northern Europe. People in the demoscene (sceners) create various computational visual and musical works, almost always non-interactive. The ‘demo’ or shorter ‘intro’ is the prototypical production, and is a small file that executes to produce a music video, rendered in real time. Sceners gather to program together and show their work in parties, sometimes immense ones: Summer Assembly, for instance, takes place in Helsinki’s Hartwall Arena and draws thousands.

The demoscene began in the confluence of a computer game industry and the impulse to enforce legal restrictions on copying with technical ones. Certain microcomputer disk drives working in certain modes could be used to read data slightly better than they can write data. Producers of video games exploited this, joining with early videogame publishers to implement so-called ‘copy protection’ for games delivered on floppy disk. Then, so that games could be copied and shared, programmers worked to alter what was on these disks and remove the copy protection – to *crack* the games.

This activity of cracking software led those who were removing copy protection to enhance the software they were dealing with in certain ways. It was possible to tidy programs up and compress them a bit for easier copying (This tradition is alive and well in many circles, including even electronic literature. At a reading at the Modern Language Association, Jim Andrews told the story of how, when one of his works was being translated into Finnish, Marko Niemi returned not only the translation but also a version of his program that was bug-fixed and tidied up.) If there was a little space on the disc, either to begin with or as a result of this compression, it was possible to add a sort of splash screen that credited those who did the cracking and that pilloried the crackers’ enemies. Of course, those who cracked and distributed software took the opportunity to do this, and the ‘intro’ was born – the first production. With it, although crackers of software may not have known at the time, was born the demoscene.

This sort of crack screen or ‘intro’ to the game had to fit in a small amount of space; it was initially sometimes a static image, sometimes slightly animated. The intros and longer demos that are shown nowadays, like the graphics and chip-tunes that are also featured at demo parties, are there for their own sake, not introducing games or demonstrating anything except aesthetic computation. Initially, they demonstrated one trick or a series of tricks tied together by very little – perhaps music, perhaps a certain graphical style. Now, in our current era, where design is highly valued, demos tend to offer unity rather than units and often treat a theme, portray subjects, evoke a situation or narrative. They remain closely tied to platforms, either ‘old school’ platforms such as the Commodore 64 or Amiga or current computers running recent versions of Windows.

The demoscene has its own values; demos can be dark and industrial but tend to project a rhythmic, utopian world with cities that rock out in unison. Demos are shown at parties to those in the know, where they are voted on by the attendees, who are essentially all programmers. They have their own traditions and obligatory segments, including shout-outs to other demo groups. They are programmed in groups and sometimes worked on at parties in larger collaborative settings. While demos are not truly mainstream in any way – not managing to be pure computer science productions, not accepted as art, not reaching the status of an Internet meme – they are one of the richest non-mainstream uses of the computer.

Revisiting those Four Points

Now, I will consider the four main points that I made about programming once again in light of these three glimpses.

Programming is a social as well as a cultural activity.

This seems worth reiterating, but it also seems by far the least controversial of these points. Are there any human activities that are not social, that do not occur within society and culture, relating to each in some way? What I mean to assert here is simply that the social and cultural dimensions of programming are significant. This would almost certainly be granted from the outset, but the glimpses of different programming practices in different contexts, and engaging with different dimensions of culture, and with different communities, should provide a clear warrant to this claim.

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

There is a fantasy, sometimes voiced, that the full power of the computer can be harnessed without programming, without a programmer. If one's goal is to develop a standard sort of computer production (a slide-based presentation, a spreadsheet, a text, a *LittleBigPlanet* level, or so on) then one of course does not need to program. Greetings cards, however, are no substitute for the ability to write and express one's self, no matter how well-designed they are. To make full use of the general-purpose computer, there is no substitute for a general-purpose programming language of some sort. To make full use of such a language, knowledge of programming is essential. The accomplishments of recreational programmers, of home computer programmers, and of demosceners could not have been made with point-and-click interfaces.

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

The glimpses shown have revealed connections between programming and communities of many other sorts, from model railroad hobbyists to film enthusiasts. Communities of programmers have also been closely associated with particular home computer platforms (during the era of home computer programming) and various other platforms from those up through contemporary platforms (as seen in the demoscene). Clearly, programming is not a pure activity that people rally around for its own sake, without any concern for their other engagements with media or for the computer platforms that they know and use.

Programming is not an activity restricted to professionals with years of training; some essentials of this activity can be undertaken by computer users after a few hours.

This idea, which may have been initiated with the populist Dartmouth BASIC, was a commonplace by the early 1980s, when home computers were pitched to the public as machines that were programmable by anyone. While purpose-built software has exploded since then and standardised systems (such as those for 'office' productivity) have become rich with features, popular programming has not kept pace. Nevertheless, systems such as Processing and to some extent HTML with JavaScript allow people to see each other's code, to learn from it, and to quickly try their hands at programming.

In this discussion, I haven't even started in on the question of how the contemporary free software movement may hold lessons for electronic literature and free culture. Or, for that matter, of how electronic literature and its work with innovative interfaces, careful translation of language and function, and the connection of literary and artistic work with critical perspectives might inform other areas of programming practice. All of this, and more, is important in continuing the conversation but must be left for the future.

Since the earliest days of computing, programming has been a social activity, undertaken at times for fun and for creative purposes. Programming may be, for some, a way into electronic literature – as it was for me; I started programming interactive fiction and poetry generators almost as soon as I started to write programs. For others, electronic literature may be a way into programming. To learn more about programming may enrich the electronic literature practice of certain authors, but it may also be a way to more broadly make

use of the computer, turning computation to practical purposes and to a variety of cultural interventions, literary and otherwise. Programming is not in every way like literacy, but it is similar in how it can be individually and socially empowering. It can extend the range of activity that is done on computers, showing new possibilities and directions. To realise the cultural potential of the computer, programs must be made by all.

Acknowledgements

'The Demoscene' is adapted from Montfort 2012. The first three paragraphs of 'Programming on Home Computers' are from Montfort et al. 2012 (with my collaborators' permission).

Works Cited

Brand, S. (1972) 'Spacewar: Fanatic Life and Symbolic Death among the Computer Bums' *Rolling Stone* 123: 50-58. December 7.

Carlsson, A. (2009) 'The Forgotten Pioneers of Creative Hacking and Social Networking – Introducing the Demoscene' *Re:live Media Art Histories*. http://www.mat.ucsb.edu/Publications/burbano_MAH2009.pdf

Graetz, J. (1981) 'The Origin of Spacewar' *Creative Computing* 7:8, 56-67. August.

Hafner, K. & Lyon, M. (1996) *Where Wizards Stay Up Late: The Origins of the Internet*. New York: Simon and Schuster.

Kidder, T. (1981) *Soul of a New Machine*. New York: Little, Brown and Company.

Montfort, N. (2003) *Twisty Little Passages: An Approach to Interactive Fiction*. Cambridge, MA: MIT Press.

Montfort, N. (2012) 'Gamer vs. Scener, or, Scener Theory' *Opening keynote, DiGRA Nordic 2012*, Tampere, Finland. June 7. http://nickm.com/post/2012/06/gamer-vs-scener-or-scener-theory/

Montfort, N., Baudoin P., Bell J., Bogost, I., Douglass, J., Marino, M., Mateas, M., Reas, C., Sample, M., & Vawter, N., (2013) *10 PRINT CHR$(205.5+RND(1));: GOTO 10*. Cambridge, MA: MIT Press.

Tasajärvi, L. (ed.) (2004) *Demoscene: The Art of Real-Time*. Helsinki: Even Lake Studios.