(en)coded poetry:
read, write, execute

(en)coded poetry: read, write, execute

http://encoded.codetext.net

Benjamin Laird

Bachelor of Engineering (Robotics and Mechatronics)
Bachelor of Applied Science (Computer Science and Software Engineering)
Diploma of Professional Writing and Editing

Submitted in partial fulfilment for the requirements for the degree of Bachelor of Media and Communication (Honours)

October 2012

Dr Jessica Wilkinson

School of Media and Communications
RMIT University

Contents

Abstract

As a programmer and a poet who writes in both print and programmable media I wanted to understand how the code could be read and written in programmable poetry. I recognised writing the code as a separate activity yet necessary to the poetic expression of my programmable works. How do I code the poetic? To answer this question I first investigated the reading of the expression of executed code in programmable works. I then explored the ways in which code was discussed in regards to digital works. With these understandings of code I applied them to case studies of three poets working in different forms. To explore these modes in my own work, I developed ten constrained poetic pieces: five print-based poems and five HTML-based poems.

Statement of Authorship

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the exegesis is the result of the work which has been carried out since the official research program; and any editorial work, paid or unpaid carried out by a third party is acknowledged.

Benjamin Laird

Acknowledgements

I would like to thank my supervisor Jessica Wilkinson for always having the time to discuss even the smallest parts of the project, the conversations about poetry, and the invaluable advice.

Thanks to Honours Program Director Adrian Miles for all the research strategies and for those few crucial discussions about the project. To the Nonfiction Lab tutor David Carlin for guaranteeing that the Monday lab would always be fascinating.

To the other honours students, thank you for being around and creating an ever-enjoyable research environment.

Thank you to Albert for always being there.

Most importantly, thank you to Jacinda for the boundless support, limitless encouragement, and for listening to every new turn in the project.

Introduction

Code is poetry

WordPress

Code is akin to innovative poetry in the physicality of its interweaving of text, metatext, and erased text.

Loss Pequeño Glazier, *Digital Poetics*

Early in 2012, I was on a panel at the Sydney Writers' Festival titled "Is code poetry?" It was a discussion that examined the possibility of "coding" as a creative form, and questioned whether code could, or indeed did, fit the definition of "poetry". Code, I argued, was not poetry. There is a complicated relationship between the two, however, as code may not be by definition poetry, but both are made by "writing". Formally, expressions of written code share qualities with poetry. Both are commonly measured by the line, structured to produce meaning or meaningful execution, and, as another of the panellists[1] remarked, code, like poetry, can be beautiful and evocative. Code also suggests or creates a world outside of itself, and it produces, in its execution, a second form, which is what most people commonly experience, running software. In a similar way, poetic language stands in for more concrete expressions

1 Mark Pesce, co-creator of VRML, described a job he had poring through the source code of various games and every so often coming across a beautifully written piece of code.

of language; it evokes through metaphor, wordplay, textual experimentation, rhythm and shape.

"Code is poetry" is then a useful metaphor that allows those who understand code to understand something about poetry and for those who understand poetry to understand something about code. While metaphors generally help to provide an outline of what is being referenced, they simultaneously obfuscate the differences. A webpage, for instance, is a convenient conceptual metaphor that uses the print page as way to understand a visual rendering at a specific URL, but a webpage is not a page. The material differences between webpages and pages extend to how they are read and how they are written: readers do not turn webpages and they do not write webpages at their interface level. Additionally, inherent to a webpage is the ability to view the written text, or, in other words, the unrendered content. In keeping with the metaphor, there would be no reason to view the source as readers do not view the source of a page (as much as they might want to). Likewise, if the metaphoric direction is reversed and instead the printed page is understood as a webpage, it may never be turned.

Poetry in programmable media challenges the context of code in poetic forms. The code in programmable media is most often realised not at the inscription of the code but at the expression; that is, at the interface. Code, if considered unique, cannot merely be an extension of the poetic. Its significance to producing the poetic is important, however. How then can we

read code in poetry? Is it only read in its expression, its executed form?

Importantly for my own practice of writing both print and programmable poetry, if code in poetry is unique, what is the relationship between the code and the poetry? I recognise coding as a separate activity but in what ways do they differ, and, more significantly, how could they be similar? In an attempt to answer these questions, chapters one and two will look at the ways poetry in programmable media can be read and then discusses code both in programmable poetry and in codeworks. In chapters three to five, these readings will be applied to case studies of Patrick Jones, Mez (Mary-Anne Breeze) and Jason Nelson, and which will be followed by a discussion of my own writing in print and programmable media.

CHAPTER 1

The Reading Event and the Extranoematic Event

Before reading this poem,
read another poem.

Read another and another.
Then tell me what you think the difference is.

Michael Leong, "Poem"

In *The Act of Reading* Wolfgang Iser describes the literary work as having two poles. On one side he places the author's text (the artistic) and on the other side the reader (the aesthetic realisation of the text). According to Iser, the literary work "cannot be identical with the text or with the concretization, but must be situated between the two" (21).

Reading, for Iser, is the interaction between text and reader, as it is for all reader-response theorists[2]. For these theorists, meaning, in the reading process is not a one-way transmission from text to reader, but instead is produced through "a dynamic

2 Reader-response theorists such as Louise Rosenblatt and Han Robert Jaus.

interaction between text and reader" (107). A problem of the process of reading, however, is that not all the text is perceived at once; this is what distinguishes a text from actual objects as those objects can be "viewed or at least conceived as a whole" (108). From this observation, Iser concludes that we stand outside given objects but the reader's viewpoint moves within the literary text. When reading, the reader can only perceive a part of the text at any given time. The difference, Iser writes, is that the aesthetic object cannot, therefore, be solely identified with the way it appears at a given moment of reading. The fragmentary development of the process of reading must then be synthesised by the reader. Initially the synthesis of these manifestations is to build consistency for the reader in the illusion-making. This is a consistency held together by techniques such as narration.

For the text to operate as an event, Iser suggests that the elements that the reader pushes to the side in the consistency-building of the text are brought with it; as a result, the reader holds the elements in her mind while reading, rather than discarding them along the way. The disruptive virtual possibilities come to the fore generating a "conflict" that creates an oscillation that for Iser constitutes "the event" (128). In addition to this process meaning is evoked as a result of the "gaps" in the text where the reader fills in what she perceives as "missing". Consequently, what is not written creates meaning for the reader alongside what is written; in dialogue, Iser theorises, "[w]hat *is* said only appears to take on significance as a reference to what is not said" (168).

As Aarseth writes in *Cybertext* many theorists have suggested that these gaps imply a form of participation with the text that can be seen to materialise when readers must make physical choices in a text (110). For example, when reading a hypertextual sequence

the act of making a choice builds on the narrative. However, Aarseth argues that this is a misreading of logical techniques as strategic filters for narrative development. Expanding on poststructuralism and reader-oriented literary theories, Aarseth builds a framework and typology for forms of literature he calls *ergodic*, where "nontrivial effort is required to allow the reader to traverse the text" (1). For the nontrivial to make sense, Aarseth writes, there must be a nonergodic literature in which the only extranoematic activities required are those that we expect from traditional forms of book reading, such as eye movement and page turning. This view of reading, Aarseth states, suggests a further integrated reader than even that referred to in traditional reader-response theory as, "[t]he performance of the reader [the reader-response reader] takes place all in his head, while the user of cybertext performs in an extranoematic sense" (1).

Aarseth contends that a cybertext is a form of ergodic literature that "involve[s] calculations in their production of scriptons", where scriptons "are what an "ideal reader" reads by strictly following the linear structure of the output" and are formed through combinations of textons (62). In Montfort's "The Purpling" each HTML page acts as a texton, while clicking on a line within individual pages introduces a new HTML page and so forms a path through the work; a scripton. The various combinations of textons constitute each separate scripton. For programmable cybertexts in computational and networked environments the material properties of those environments change the storage and presentation of those texts. Furthermore, programmable media potentially extends the reader's relationship with the text through techniques natural to the media such as user functions (clicking a mouse, using the keyboard) or kinetics (movement of text, images and other visual

artefacts within the work) as occurs in both of Nelson's versions of *Dreamaphage*.

While Iser poses the reading experience as an event, Hayles argues that the material properties of programmable and networked works are themselves a process and an event (in this sense, they could be considered an extranoematic event). Hayles describes the poem in digital media as having "a distributed existence spread among data files and commands, software that executes the commands, and hardware on which the software runs" ("The Time of Digital Poetry: From Object to Event" 181). Hayles proposes that the poem in programmable and networked media is no longer an object but a process and, as such, becomes an event when expressed. "The poem is 'eventilized,' made more an event and less a discrete, self-contained object with clear boundaries of space and time" ("The Time of Digital Poetry: From Object to Event" 182). This transformation from object to event occurs prior to the reader's interaction with the text.

Understanding a poem in programmable media as "eventilised" implies a change in reading. No longer is the text an object to be transformed into an event when read; more accurately, it is now another kind of event. Cayley contends in "Screen Writing: A Practice-based, EuroRelative Introduction to Digital Literature and Poetics" that temporal and spatial aspects are part of and realised in the form and structure of poetic engagements with language. In print-mediated literature it is "in the special attention we pay to the time and space of the poem" that dictates how it is arranged: words, lines and spacing. These arrangements are then "silent-implicitly or oral-actually realized in the temporal rhythms". Cayley argues that what programmable media allows us to do is to materialise the spatial and temporal

qualities of poetry that programmable poetry performs "in real passing time and space rather than in the imaginary space-time of the silently reading mind" (183). This contrasts an event of reading to the process (extranoematic) event of programmable media.

When interaction becomes part of the process of reading, what happens to reading? That is, what are the implications for reading when the reader "uses" the poem or loses control of the poem, as with kinetic works? In *Cybertext*, when addressing forms of hypertext poetry directly, Aarseth asks "[i]s a hypertext poem a poem?" He continues:

> *[i]t may be argued that clickable words and menus subvert the lyrical genre by inviting the user to play an (imagined) personal role in the production of a reading path. The 'poeticness' of a poem would be challenged by the readers' awareness of their own subjective action. (86)*

Beyond the subjective action is the reader's/user's behaviour toward the interaction with the text. The reader in many works in programmable media must move a mouse, click a button or, in the case of installations, move their body. Massumi notes in *Semblance and Event* that, in regards to interactive art, there is a risk that the interactivity overpowers the artistic dimension: "You often feel there's a trick you need to find and master, and once you've done that, you lose interest because you've got the feel of it and know how it 'works'" (46). Simanowski raises a similar issue in relation to interactive art and the relationship between the cognitive and the physical. Specifically he questions "whether the interactor can attempt to understand—decode—the work with which she has been interacting" (126). In answering the question he suggests that "[t]he shift from the

field of objects to the field of events and behaviour includes a change from providing a specific message to providing a specific space for interaction" (126). It is in interactive art that meaning is created, but it is created by the interaction, not by a specific message of objects.

These questions are not unique to poetry in programmable media; rather, they are questions that have long being asked in relation to print-mediated poetry. In his essay "Artifice of Absorption", Bernstein compares what he describes as absorptive and antiabsorptive writing. An absorptive writing is one that supposedly favours a reading of a text as content, separate from its materiality. In reading such a work, the reader is expected to shed the text for the experience of reading and absorbing content-as-meaning. By contrast antiabsorptive writing draws attention to the material or artificial nature of the writing. It is not a binary relationship; rather, in all poems "[t]he artificiality of a poem may be more or less foregrounded" (10). Regardless of the degree to which the artificiality is apparent it is all a necessary part of a "poetic" reading. He reasons that if the artifice is conspicuous there is a tendency to expect the text to have no meaning, while if the artifice is hidden then the work is expected to be about the content and therefore meaning. However, Bernstein reminds us, "[c]ontent never equals meaning" (10). The form of the poem is in a relationship with its content and requires a multilevel approach to meaning. Bernstein writes that absorptive texts hide their artifice while antiabsorptive texts flaunt it. It is for this reason that programmable poetics can be read as antiabsorptive texts when they foreground their artifice. Readerly absorption can be disrupted by a direct address to the reader, which then forces the reader to recognise the artifice of the reading experience (32).

Bernstein outlines antiabsorptive traditions in poetry including visual and textual play, collage, fragmentation, errors, found material, typographic inventions and references to things outside the text (56). This includes the writing methods and writings of the Language poets, Surrealists, Dadaists, John Cage's mesostics, and Jackson Mac Low's diastic methods. It is however, Bernstein claims, specific to the social, political and historical period: what is antiabsorptive now may not be in the future as it is incorporated into the expected reading experience. While early modernists encountered resistance, for example, it would be unthinkable to leave those same poets out of modern-day anthologies.

Contemporary antiabsorptive qualities point to techniques that have been extended to programmable media. Contrary to Aarseth's questioning of the "poeticness" of the reader's awareness, this reading of poetics places the reader's awareness within a tradition. An aesthetics that, as Simanowski says, "promote a focus on the intensity of the present moment and on the materiality of the signifiers rather than their meaning" (127).

As with the oscillation between the illusion-making and illusion-breaking of the text as event, absorptive and antiabsorptive methods interact while reading. Although both are predicated on reader experience, the relationship between Iser's illusion-making and illusion-breaking are not the same as the relationship between absorptive and antiabsorptive methods. While the former is due to the reader's own attempt to reconcile the experience of the narrative, absorptive and antiabsorptive properties relate to the materiality of text. Even though "reading a text as an object" or "reading a poem in programmable media as an event" are different, they still produce a reading event in

the reader. While the expression of poetry in programmable media distinguishes itself in its expression, the cause of that expression must also be addressed; the cause I am alluding to here is the code.

CHAPTER 2

(De)coding Code

rocked up to the address entered the
site i popped the hood & just started
pervin' on your code (it's clean & oh
so elegantly compliant) tags all in a

row <!-- only you & I see this bit -->

David Prater, "Code Pervin'"

According to Hayles, "[t]he importance of active code to the production of digital texts cannot be overemphasized; it is one of the distinctive ways in which electronic literature differs from print" " ("The Time of Digital Poetry: From Object to Event" 181). Hayles distinguishes "active code" from "passive code", which act as instructions to the reader like italics or indentations. This implies even in the domain of programmable media a usage of the word "code" that contains multiple meanings. That is, there must be at least two ways in which we can understand code either computationally or as additional codes that work upon the text, like italics, which we can read as emphasis or as a reference.

For Cayley, "code" is discussed in many more ways within discussions surrounding digital poetry, poetry in programmable media and codeworks. He provides five categories of code

to clarify the usage of "code" in discussions of codework ("Time Code Language: New Media Poetics and Programmed Signification" 311). The first category is code as language; that is, treating code as language on its own terms. As an example he gives Glazier's treatment and discussion of code for its own poetic potentiality. In *Digital Poetics*, Glazier examines the materiality of HTML both in terms of the code and also the context in which the code runs (the paratext or, in this case, the paracode).

Cayley's second category describes code where the language works but the code is broken (it does not execute). This is seen in works like those by Mez, whose work is discussed in more detail in chapter four, where natural language is broken up by codelike structures that disrupt the reading and the process of reading, all of which occurs in the interface text. An example is Mez's "_trEm[d]o[lls]r_" where the codelike structure, which resembles XML, frames the poem. In this work, the XML-like elements that wrap the poetic lines that suggest structuring an identity are non-functional. It provides a context for the poetic and lends meaning to the poem through associations that are both structural and technological. Cayley argues that the second category of code demonstrates "the extension of the long-standing enrichment of natural language that occurs whenever history or sociology produces an encounter between linguistic cultures and subcultures" ("Time Code Language: New Media Poetics and Programmed Signification" 312). In that sense, this second category of code can be comfortably situated within the field of experimental poetics.

The third form is when code is presented as a natural language to non-specialist readers. This is executable code written to be read at the code level, even though the computational execution

may be meaningless. This code, as Cayley argues, operates as a heavily constrained natural language form. Therefore, the poem is written in a programming language only in the sense that is executes and that it appears as if it were poetry. This is one of the modes of "Perl poetry". Perl is a programming language, and Perl poetry includes poems written in Perl, Perl programs that writes poems, and, sometimes, poems about Perl. As Cox, McLean and Ward point out, often this is merely the result of "porting": the act of moving a computer program from one platform to another, such as rewriting a program written for Microsoft Windows to operate in Mac OS X. In this way the poetry is ported from a natural language to a programming language, "[i]t produces poetry in a conventional sense, possibly expressing some clever word order and grammatical changes, but does little to articulate the language of perl in itself" (par. 13). Mateas and Montfort describe this kind of coding as part of a broader *double-coding* or *multiple-coding*: the "words" are the same but the meaning is different. It is a process that they also observe in natural languages, when, for instance, a sentence may be grammatically correct and sensible in two different languages, say English and French, but in doing so ends up having two different meanings (sect. 5 par. 1).

Cayley distinguishes these first three categories as usages of "code" that produce codeworks that are "interface texts subject to interpretation by readers". The code here is "not running to generate the text" at the location of its reading. Even if the code here is executable, Cayley states that read in this context, the code is not "significantly present in the text in a way that might alter or inflect the manner of reading" ("Time Code Language: New Media Poetics and Programmed Signification" 313).

The fourth category of code for Cayley is where code is a "system of correspondences"; that is, code as encoding. This concept incorporates the materiality of code, recognising that digital media has to be encoded in order to be stored and displayed. That is, what appears on screen is a decoded, reorganised representation of how the code is stored on the machine. The signifier then has never been fixed, but in the digital space the code acquires additional resonance from the materiality of the medium. Encoding, however, is not a new concept unique to digital media. Citing Barthes, Cayley writes that the text evokes simultaneously corresponding codes. However, Hayles's digital encoding, as Cayley remarks, is mostly sublinguistic, indicating, for example, the storing and displaying of the digital representation of a letter ("Time Code Language: New Media Poetics and Programmed Signification" 313).

Cayley's fifth and last category is code as programming. This category is not limited to computationally processed code but extends to all programmatic or directed text. It includes, then, instances in which the text operates programmatically. Cayley argues that code as programming is part of all textuality and programs are "a necessary aspect of the materiality of language" ("Time Code Language: New Media Poetics and Programmed Signification" 314). When the text itself operates to produce writing Cayley refers to it as paratextual programming. Cayley defines paratextual programming as when "the (integral) aspects of inscription that frame or infect or undermine or position the text to be read, that is, the interface text" ("Time Code Language: New Media Poetics and Programmed Signification" 315). For example "(mis)read" can be understood as both "misread" and "read".

Referring again to the codeworks belonging to category two, (such as the works of Mez, wherein the interface text is "infected" by code), Cayley maintains that writing can also be considered paratextual programmed when it uses "postmodern punctuation". He writes that "[a]ny text in which codes and the codes of punctuation are integrated with the interface text … can be unpacked and analyzed in these terms as inflected and driven by paratextual programming" ("Time Code Language: New Media Poetics and Programmed Signification" 315). Hence, Mez's title "_trEm[d]o[lls]r_" can be read as "tremor", "dolls", "tremor dolls", "Emo dolls" and so on. Cayley sees paratextual programming as being in continuity with the programming of programmable media. In addition, he views hypertext as positioned between paratextual programming and the textuality created from programs. In its simplest form, computational hypertext is an arrangement of documents that are organised and navigable. Whereas the paratextually programmed operates on the word or the line, hypertext operates formally at the node. Computationally operable, it thus bridges more complex uses of programmable language production ("Time Code Language: New Media Poetics and Programmed Signification" 317).

Common to discussions of code is a sense that it functions in hierarchy. Raley writes "[w]hat the façade of the code surface masks is the deep structure of code, the tower of programming languages that descend from software to hardware" (sect. 1 par. 5). At the lower levels are machine and assembler (hardware specific implementations) and further up there are higher-level languages, like Python and C++. This structure has an effect on the implementation and writing of code. Hayles notes in *My Mother was a Computer*, that, at the lowest levels, code and computational processing is intolerant (ch. 2). That is, the code

closest to the hardware leaves less room for variation in the way the function intended to be performed is written. As the intolerance decreases with each level of programming language, ambiguities enter the system. This hierarchy of languages might imply a value structure in which the code closest to the hardware is seen as more fundamental than higher-level languages. Alexander Galloway, however, argues against this, instead suggesting that value divisions between levels are "perhaps misguided", as the same program compiled or uncompiled is logically equivalent (167). Galloway also views code as language: "Code is a language, but a very special kind of language. *Code is the only language that is executable*" (165). Further, in comparing code with natural language, Galloway argues that while natural languages have a legible state, "code has both a legible state and an executable state" (166). Thus, code for Galloway is language plus an executable metalayer.

What, though, does it mean for code to be executable? Galloway states "code is the first language that actually does what it says" (166). Does this not mean, however, that code can be considered to be like laws or directives? As represented by the "tower of languages" or hierarchy of languages, high-level programming languages are removed from the specificities of the hardware, and incorporating as they progress more elements of natural language. In "The Code is Not the Text (Unless it is the Text)", Cayley argues that in order for a codework to express all the qualities of code it must not only be contextual, it must also be executable.

Marino, in disagreeing with Cayley's position, refers to Mateas and Montfort, reasoning that as code can be "written for programs that will never be executed", then execution must not

be a criteria (par. 32). Yet even when never executed, code is written to be executed. Writing code is writing with the intent that it will perform some behaviour, and, in the end, the measure of a program's success is if it functions as intended. Execution as such is an aspirational relationship between the code and the executing machine. A book, while intended to be read, does not cease being a book when it is not read, but the fact that it can be read is crucial to understanding it as a material object. The execution of the code should be understood in similarly abstract terms. Pseudocode, written to prototype functional code, for example, sits above the tower of languages. As the name implies, it is codelike in that it follows the logical constraints of code but is not computationally executable (and therefore, as far as computing is concerned, "pseudo"). The aim, however, is to produce a logical structure in a language that can be tested against. As such, it needs to conform to the logic of code through an unambiguous expression of a natural language. There cannot be linguistic slippage in pseudocode otherwise it fails to test the constraints of the program that it prototypes. Pseudocode is a programming language without machine hardware to execute it; consequently, it always needs to be "ported". While pseudocode cannot be computationally executed, it is written with the aim that an executable form is produced. It is aspirationally executable and so, although removed from a machine relationship, is a form of programmable code. A similar logic works in relations to Cayley's paratextual programming: it too has an abstract execution. While computational programming relies on a compiler, browser or other form of mediation in which the rules for reading are built in, paratextual programming by contrast requires the rules to exist within the reading practices of the reader. Generative making of language in programmable

media occurs at the same level in paratextual programming at the stage when meaning-making is occurring.

As higher-level languages come to resemble constrained natural language, the distinctions between code and pseudocode becomes purely contextual. For instance, Python, a high-level programming language, is described has having syntax that "resembles executable pseudocode" (Lutz 5). Code as execution and code intended for programmers invites an aesthetic that is dependent on programmatic execution. Cox, McLean, and Ward argue that the reading or hearing of poetry is the execution of the poem, which is realised as it is experienced. They continue, claiming that "like poetry" the aesthetics of code are in its written form and its execution, and that the experience of written code should be in parallel with its execution, or the realisation of the code. This hypothesis, however, reduces all reading or hearing to execution rather than considering execution as a mode of reading.

Reading code, then, has multiple audiences, including an informed expert audience. Hayles writes, "Like esoteric theoretical writing, code is intelligible only to a specialized community of experts who understand its complexities and can read and write it with fluency" (*My Mother was a Computer* ch. 2). Computationally executable code for an expert audience (programmers) still involves a subjective aesthetics. Numerous factors (including those that vary between programming languages) such as validity, structure, efficiency, verbosity, and clarity are qualities that are enjoyed differently depending on the individual programmer. Some value clarity, while others value tightly written, efficient scripts. Testing a programming language or exploiting a language's idiosyncrasies (easily translatable to

poetic readings) can also be seen as an aesthetics of code. Mateas and Montfort write of programming competitions and challenges that aim at hiding the performance of a program in confusingly written code:

> *This play, which can be called naming obfuscation, shows one very wide range of choices that programmers have. Such play refutes the idea that the programmer's task is automatic, value-neutral, and disconnected from the meanings of words in the world. (sect. 11 par. 4)*

Since the publication of Glazier's *Digital Poetics* there have been ten years of changes to the HTML standards and web programming cultures. Currently, emphasis on separation of content (textual), structure (HTML), display (CSS) and behaviour (JavaScript) is seen as industry best practice. Glazier writes that "[i]t is informative to consider an approach to writing code that treats the source code as a fundamental part of the meaning-making structure, not as secondary to another 'purpose'" (103). This meaning-making is currently adopted as the "semantics" of the webpage. That is, the structure of the HTML itself is expected to explicitly denote the type of content contained (in this regard, it is frowned upon to use table tags for design layout). This is extended in HTML5 to include tags such as "article" for group content and "nav" for navigation. In the development of the latest standard of HTML5 a discussion evolved about the fact that "HTML5 lacks explicit semantic mark-up to express poetic forms" (W3C "Issue: Explicit Markup to Semantically Express Poetic Forms"). While no poetry specific mark-up was included, this illustration reveals a debate within the industry about the semantics of poetry. In the semantic view of the web (including both semantic and Semantic, as seen in

standards such as Resource Description Framework), the code and content are not only readable by machines to produce a rendered version of the code, but also to present an unambiguous representation of what that content is. As Galloway notes in *Protocol*, "the word 'Galloway' is meaningless to a machine... But wrapped inside a descriptive protocol it can be effectively parsed: '<surname>Galloway</surname>.' Now the machine knows that Galloway is a surname" (139). In poetics, however, do we want to enforce that level of precision? Galloway is also a place and a type of cattle, both adding complexity and layers to the noun "Galloway".

In HTML-based programmed or marked-up poetic works, as in all codeworks, the code does not exist solely in the work. Ambiguities are introduced through the range of different implementations of the HTML standard(s). The interpretation of HTML changes between web browsers (layout engines) and versions. The standards outpace the adoption within the browsers, even if at times the standards in flux are experimentally included for a specific browser. The default displays will still differ, though, as the underlying layout engines produce slightly different results even when the same elements are supported. Further to this, programmable works in Flash, Java applets, and Silverlight require plug-ins to run.

There are many ways that code works in codeworks just as, Cayley points out, there are many ways in which "code" is used. While the second category Cayley describes only uses code contextually, all other modes reveal a behaviour of code: it operates at some level. In reading code as language, in infecting code with language, in acknowledging the encoded depths, the nature of programming or reading the visible execution of code,

users, readers and programmers accept that code performs at a metalayer.

The presentation of code, both materially and contextually, invites multiple methods of reading. The numerous interpretations and position of code allow this. Marino proposes a reading of code that incorporates all aspects for interpretation: "Everything. The code, the documentation, the comments, the structures—all will be open to interpretation" (par. 32). Simanowski argues that code requires close reading that "combines expertise in code and coding, as well as in the interpretations of the representations that are generated by code on the screen or at the site of installation" (219). Code-as-writing extends this argument to a place of writing. Glazier writes: "Code is a scene of poesis"; the nature of HTML, he contends, forces an engagement with its materiality. Code, then, can be written and read in as many ways in codework as the word "code" is used. The multiplicity of different interpretations of "code" reveal the multiplicity of ways code performs.

Given that code can be understood as occurring as paratextual programming and code as a scene of poesis, modes of experimental poetics could potentially be read as ante-coded. In the next chapter I look at Patrick Jones's "Step by Step". Jones is an artist and poet who uses experimental techniques that can be considered as at the boundaries of programmable works.

CHAPTER 3

Patrick Jones: Hypertextual Free-dragging Mesostics

How did we go from
meeting our needs to
excess and waste?

History of plastic
History of capitalism

Stephen Collis, "The History of Plastic"

The reading event is tied to the physicality of the presentation of poetry. This materiality of language allows for poetry to produce complex readings through formal and experimental techniques. A poetry that forces a reader to make choices foregrounds the unconscious decision-making that we do as readers.

Patrick Jones's "A Free-dragging Manifesto", published in *[How To Do Words With Things]*, begins with a quote from Joan Retallack's "What is Experimental Poetry & Why Do We Need It?": "Page becomes stage transfigured into time-bracketed instances of a continuous present; written language becomes a surprising performance of its charged materiality". The concept

of the "stage transfigured" underlies much of Jones's poetic work ("A Free-dragging Manifesto" 47).

Fig. 1. Images from *A Free-dragging Manifesto*

Jones, an artist and poet living in rural Victoria, creates poetry that consciously disrupts the reading experience. His "free-dragging" poetry initially consisted of non-dance performance which involved Jones and a fellow artist dressing in women's skirts, shirts, stockings and shoes—"drag"— and arranging themselves in various positions around urban locations, as seen in Fig. 1 from "A Free-dragging Manifesto" ("An Interview with Patrick Jones" 149). Jones considered these works a form of poetry written with the body, with the text a non-lyrical, disruptive and political statement. The act of free-dragging itself is the result of a predetermined method to perform the act. Returning to print after these experiments led Jones to move

from the physical performance to a "slow-text" form inspired by John Cage's mesostics[3].

Jones contends that he uses "a decentralised Cagean mesostic procedure, to create an example of what I want to call *slow text* – a text where the once streamlined words become a little disobedient on the page" ("A Free-dragging Manifesto" 24). The slowing of reading through textual disruption is an example of what Bernstein calls antiabsorptive writing, as noted above, Jones's methods reduce the sensation of being "transported" and what Jones deems easy-to-consume text. This theory reveals Cage's influence, who saw making language un-understandable as political and artistic action: "[S]o what we're doing when we make language un-understandable is we're demilitarizing it, so that we can do our living... It's a transition from language to music certainly. It's bewildering at first, but it's extremely pleasurable as time goes on".

Unlike Cage's generative mesostics, created by applying a process to a text, Jones's poetry is written specifically for the form. Jones, in drawing a parallel to his physically performative work, states his work is "creating a physicality for the reader's eye" ("An interview with Patrick Jones" 149). The rendering of the text on the page is important to Jones who sees his poetry practice as "being very much focussed on the materiality of language" (*[How To Do Words With Things]* 14). In "Step by Step" Jones draws focus to his techniques as political, but, at the same time, his practice does not eschew the politics of his poetic content.

3 Mesostics is a form of poetry in which a letter within a line aligns vertically to produce an additional line.

His poem "Step by Step" begins:

when populationS swell an$_{d}$ ^{g}et sucke$_{d}$ into cities
we $_{r}$ely on $_{r}$esources T$_{r}$ucke$_{d}$ f$_{r}$om somewhe$_{r}$e else
and in$_{d}$oin^{g} so fo$_{r}$fEit the ecolo^{g}ical intelli^{g}ence
that ou$_{r}$ unsPecialise$_{d}$ ancesto$_{r}$s kept so close

"Step by Step" is formally striking, consisting of eighteen four-line stanzas, with each stanza acting as a separate mesostic spelling out the word "STEP". The pattern of the words, with superscripted letters *g* and *b* and subscripted letters *r* and *d*, reduce the likelihood of a skimming intake of the words and lines. The reading is, as Jones intends, slowed. A slow text opposes easy absorption precisely because you have to "notice" the text as you "consume" it. As an antiabsorptive technique, it slows the reading event. The conversion of mark to meaning requires additional decoding as the words are defamiliarised. Each "step" in the poem shadows its stanza and hints at the ordered progression of a ticking clock, a progression ominous and irreversible. Reading each line, however, requires actively disengaging from the step and resisting its pull down the page.

The step as a separate reading event attempts to speed up the poem in what initially seems to be inevitable catastrophe, as implied in the second stanza: "we ha^{b}iTually wa$_{r}$ and $_{r}$ationalise its ^{g}enoci$_{d}$e / an$_{d}$ wE leave ou$_{r}$ foo$_{d}$ p$_{r}$o$_{d}$uction to faceless co$_{r}$po$_{r}$ations". Jones's "A Free-dragging Manifesto" is an argument for a poetics that embodies a sustainable way of life. Recounting a discussion with graphic designer Ian Robertson, Patrick Jones writes, "We talked about perceiving the poem; that once you ascertained for yourself what a poem is or what

is poetic, the potential form of the poem becomes infinite, or at least specific to its subject" (*Words and Things* ii). It is in this potential form that Jones's mesostics are realised.

Jones's work creates a sense of the ordered organic that reflects his politics. It is a sustainable natural response and what he calls a *permapoesis*, modelled on the ethics of permaculture for a poetic sensibility. Consumption is slowed in his work as a statement against consumer society. Formally, the mesostics function hypertextually, creating a secondary reading in a downwards direction on the page.

In "Step by Step" the stanzas move into endings of rhyme and half-rhyme. The third stanza, for example, rhymes civilisation with hyper-separation, which refers to both an ecological and social result of "industrial civilisation". The hyper-separation also marks a point where Jones uses a footnote (the only one in the poem), which acts hypertextually, visually drawing out the reader and creating the third antiabsorptive method used in the poem. "Step by Step" is both a warning of the approaching collapse of industrial society (Jones takes Jensen's quote "industrial civilization is not and can never be sustainable" as one of the starting points for his introductory essay in *[How To Do Words With Things]*) and a movement toward building environmentally conscious communities (24).

The poem, if extracted from the textual interference, is lyrically traditional. The slowing down of the reading emphasises the experience of reading the text. Jones views reading and listening to poetry quite differently, and the lyrical structure of his poems reflect this; he writes, "The reader of these slow-text mesostics can hear quite conventional poems often with rhymes so that on

the page there is a sort of physical difficulty but when heard … they appear like folk songs" ("An Interview with Patrick Jones" 150). The aural quality, once decoded, reduces the complexity of the reading event; the event here is the relationship between the reader and the text.[4]

Jones provokes the reader's oscillation between involvement and observation through, as Iser describes, the attempted resolution of the illusion-forming and illusion-breaking. The antiabsorptive techniques push toward observation but it is in the oscillation that the text becomes an event. The oscillation is what is significant, but is also, of course, contextual to the reader. The reader can experience an antiabsorptive effect by being hostile or bored with a device. Bernstein writes that:

> *devices, whether absorptively*
> *or antiabsorptively employed, are in themselves conventionalizing*
> *& readers can be expected*
> *to enjoy a device that ruptures the 'commodification'*
> *or reading insofar as this fulfills*
> *their desire for such a work &, likewise, to*
> *be bored to irritation by a device meant to soothe*
> *or entertain (65)*

Jones, through the use of experimental techniques and "folk lyricism", is successful in producing and representing Iser's

4 It is beyond the scope of this study, which is concerned with the reader-text engagement, but it raises interesting questions concerning the aural/oral qualities of poetry. The decoding, for Jones, is a textual experience, which actually suggests two *different* poems. How could the encoded complexity of the slow text manifest aurally? An additional question arises for the materiality: if Jones's slow-text is aurally realised, is this also required in a reading of the extranomatic event in programmable works?

oscillating effect of the living event. While Jones is able to do this expertly, it is by no means the only way to achieve synthesis between antiabsorptive techniques and absorptive results in the reading event. He achieves his experimental techniques by using layout to affect the reading. His methods suggest a programmable or hypertextual mode but do not employ code in execution or content.

Following this trajectory of experimental modes of poetry that employ textual techniques to affect reading, in the next chapter I will look at Mez's *_cross.ova.ing][4rm.blog.2.log][_*.

CHAPTER 4

Mez: Paratextually Programmed Mezangelle

< autonomy / >

or

< if yes, goto phase X / >

< if no, go … / >

or

> Maged Zaher, "Rented luxuries (made out of collapsing thoughts)"

Mez (Mary-Anne Breeze), a NSW-based artist and poet, has amassed numerous works using her own "digital creole" mezangelle. Mezangelle mixes punctuation, abbreviated language associated with online and SMS communication, and technology-informed or codelike structures.

Her works, as Cayley argues, can be read as codework in the context of code-informed subcultures or in some cases read as "code" because they are paratextually programmed text. In discussing electronic works, Stephanie Strickland extends Hayles "flickering signifier" to include an additional form of oscillation: that which exists between "processing alphabetic text and the

processing of image in works that use both" (185). For text-only works, this is a flickering between the text-to-be-read and the text-to-be-viewed-as-image. The reading of Mez's work, for Strickland, requires moving between positions to "unpack" the text, triggering a change from reading to scanning, which is "a perceptual act more often associated with image". As such, Strickland regards this process as allowing "for multiple, plural, and contradictory readings" of Mez's texts (186). Funkhouser, on the other hand, suggests that the verbal "plasticity inherent in mezangelle enables many interpretations" (161).

cross.ova.ing][4rm.blog.2.log]/, anthologised in *Electronic Literature Collection Volume 2*, is a 258-line collection of mezangelle work presented as a text file. As it is a text file, its interface text is identical to its viewed source; this means that although it is represented online (and delivered across a network), it can be read on a single interface layer. It is divided into ten sections, each numbered and timestamped. The ten sections work as stand-alone pieces, having been previously published in various locations on the web or as emails, but they also form a single work. The gathering of individual pieces into a single work is implied by the title, *_cross.ova.ing][4rm.blog.2.log]/_*; the title itself is representative of mezengelle. The title mixes phonetics and punctuation to produce multiple readings; for example, "cross.ova.ing" can be read as "cross over" or "crossing over", and "4rm.blog.2.log" as "from blog to log". The additional subtitle of the poem indicates an anthologised form: "Codewurk [actual work]". Due to its text-file form, the text is additionally restricted to a monospaced format and simple ASCII text presentation.

The first section (or poem) is "SocialConnectionAccessProtocol[-SCAP -]", which imitates an exchange, login and checkout of

CVS (Concurrent Versioning System) a version control system, though the piece also suggests a "ControlVersioningSystem". Version control systems are software that allow the storing and tracking of projects as they progress. They are a way for multiple programmers to work on the same project at the same time and to release the project incrementally. The "SCAP" in Mez's title also technologises social communication via the multiple meanings of "protocol". "SCAP" is, like many of Mez's works, understood best through Cayley's second category of code. That is to say, code and coding subcultures as context for the codework.

The lines in "SCAP" are not operational, or even non-operational, code, but are instead lines that mimic a command line connection. Reading the CVS "commands" give the poetic lines additional context. For example, "codependentserver" is in the position of the CVS access method, which is the method for connecting to the repository. The "codependentserver" here could be read as "codependent server", or, with a double-reading of the "co", "code dependent server". While the reader knowing that the position of "codependentserver" is where the access method is located provides an additional meaning to the line, particularly in the context of the "SocialConnectionAccessProtocol" as co-dependency for social access, the co-dependency of the server also implies a similar relationship. In the same line, "internaltripwiring" is positioned as the user for logging in. This "internal trip wiring" can be read both as travel, as in "internal trip", and as a note to take caution when reading the pieces. There is enough code in this work to expose its technology/code informed creation, and yet the code is sufficiently broken to trigger a reaction from readers who realise the fact. Similarly, those who are not familiar with

the technology-informed fragments are likely to trip on them. "SCAP" continues with the SCAP program being checked out:

> *cvs server: Updating abortive/directory/SCAP*
> *Ur abortive/directory/SCAP/NO.pls*
> *Ur abortive/directory/SCAP/YES.dmg*
> *Ur abortive/directory/SCAP/ChangeRealityLog*

The conclusion to "SCAP" emulates release information that is presented as if different versions are available with commentary in mezengelle. Within this list, suggestive of many of the forms in *_cross.ova.ing][4rm.blog.2.log]/_*, is the line "#unre[a|]eling unstable_conversation_w[g]r[e|]app[l]ing".

The second "bet[t]a[living.thru.brutal_ness]", fourth "In this album", eighth "#.Pls. .Select. .ur. .Char[r(i)ed.H]Ac(k)tor.#" and ninth "The 10 Best Synapse.Skys of the Web" pieces all use mezengelle to various effects as a core mode of presentation. The works are technologically contextual with "bet[t]a[living. thru.brutal_ness]" a beta access invitation to try a new service that ends with the damning, "www.Trickling.D(CL)o(P)wn(ed). Ur.Marketing.Facex .com", revealing that the invitation was in fact a marketing manoeuvre. "In this album", on the other hand, establishes a set of names and albums describing "(photos)", while "#.Pls. .Select. .ur. .Char[r(i)ed.H]Ac(k)tor.#" presents two character profiles. The final of these pieces, "The 10 Best Synapse.Skys of the Web", is a list of ten mezengelle. The mezengelle act as lines of language generation. Whereas Jones's slow text slowed the readers reading, Mez's mezengelle generates language during the act of reading. The more familiar the reader is with the context, the more language generated. Words and lines are encoded within other words and lines. As a form of generating language they exhibit, as Cayley maintains, a form of

paratextual programming. In a certain sense, the lines in these works "execute" as they are being read.

In "#dn[p]a[per.cut here.]bird#" mezengelle merge with the A, C, G and T DNA-bases. Here the code is genetic and contextual to the work: it is non-functioning DNA code. In a similar way, this contextual code use is apparent in "_trEm[d] o[lls]r_", wherein the code is XML-like. The structure the pseudo-XML compositionally implies a meaning that oscillates between the fracturing caused by a plastic presentation, for instance "var='user' val='YourDollUserName'", and a tremor. The first fracture posts to the character inscription, itself a cutting into the surface, while the second is a post to the skin with "YourPolyannaUserName", hinting at a surface that is constantly upbeat. The "polyanna" also shares "poly" seemingly referring to "polymer", plastic reference, and "polygon", an SVG reference. If treated as XML, however, it is not well formed. The top-most "fracture" opens with "fracture" (the start-tag), but closes with "fractures" (the end-tag), hence the XML itself is broken and consequently fractured. The code, therefore, does not "execute" in any meaningful way.

The third codelike piece contains allusions to the Perl programming language. "531 - (ch) . amber (ed) k (h) e (a) r (t) nels" joins variable like non-variables "$stiff", "$limb", "$swelt" with strings "ening", "less" with what seems to function as a string-joining-dot operator. However, after the variable-like elements are joined, the lines avoid the code structure and fall into mezengelle. The code, then, functions like the rest of the mezengelle: as a paratextually programmed writing. Beyond the associated code are the additional meanings being generated by the placement of the associated code.

Although disparate, each of Mez's pieces within *_cross.ova.ing] [4rm.blog.2.log]/_* form narratives that are more meaningful than simple parataxis. The associated codelike contexts aid in building the narrative and while the lines themselves are meaningfully dynamic, producing multiple readings, the pieces do not read randomly or without construction.

A reading of Mez's work as that uses code to structure a narrative presentation of poetry, as well as recognising the way the lines can be paratextually programmed, reveal methods of reading code that operate programmably. As a poetic reading is an approach to reading Mez's codework, the reading of Mez's works suggests a method of reading code. As such, this also provides a method of writing code in operational programmable poetry; that is, using the context of the code as meaningful layer to structure a poem at the code-layer. Lastly, Mez's paratextually programmed works produce language when the encoded lines "execute" in reading. This is not a computational execution, however. What, then, can occur when code is executed computationally and how can that help to understand code in programmable media? In the next chapter I will look at both versions of Jason Nelson's Flash work *Dreamaphage*.

CHAPTER 5

Jason Nelson: 3-Dimensional Page in *Dreamaphage*

From this hospital bed
I can hear an engine
breathing—somewhere
in the night

William Carlos Williams, "The Injury"

Jason Nelson is a Queensland-based new media artist and poet who works in Flash. Due to the fact that the code within Flash-based works are not accessible to the reader of the work without additional or specific software, this reading will focus primarily on the codes' execution or realisation. Dreamaphage versions 1 and 2, first published on Nelson's website secret technology and later collected in Electronic Literature Collection Volume 1, are Flash poems presented as medical reports of a Dreamaphage, a term that hints at an infection of dreams or, perhaps, a dream of infections.

The second version was created a year after first using the same poetry, narrative and images to produce a layered work of mixed textual forms and media. Nelson states in relation to this work, "I love the 3-dimensionality of the different dreams and the layering of stories, poetry, science and multimedia playthings". Interestingly, the first version of the work creates more depth, while the second rendering revisits the textual material and imagery from the earlier work, but more drastically alters the interaction mode. Nelson writes, as an introduction to the second version:

> *Unfortunately the first version of Dreamaphage suffered from usability problems. The main interface was unwieldy (but pretty) and the books hard to find (plus the occasional computer crash). I redesigned the main interface, playing off the 3D feel of version one, but placing it within two dimensions. ("This Is Almost Everything I've Created")*

Fig. 2. *Dreamaphage* version 2 title page

Both versions of the work begins with a similar title screen. Motion-filled squares foreshadow the instability in the work, while a quote from Dr Bomar Felt insinuates the piece contains books of dreams that contain a pattern to a cure. Dr Felt's quote ends with the ominous question, "[h]ow long before I become another lost?". It hints at a disorienting horror to come, complete with a haunted hospital.

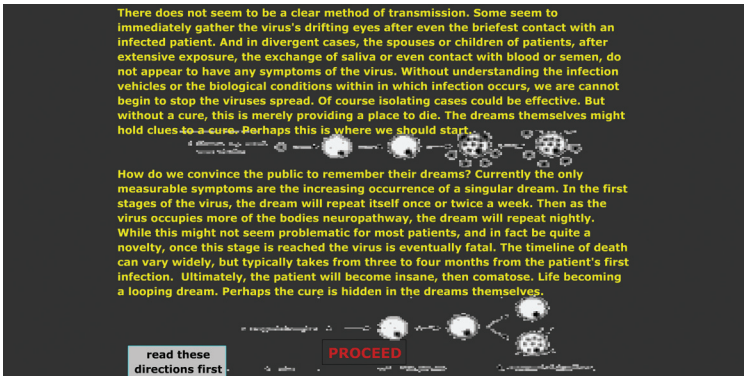

Fig. 3. *Dreamaphage* version 1 introduction

After clicking to begin the first version, the reader is presented with an introductory screen. It is a single screen entitled "diseaseinterface"; with bright yellow text that breaks apart, before transforming into blurred virus-like objects, it evokes the aesthetics of early '90s computer games, such as *Doom*. The *Dreamaphage* itself, we learn during the introduction, is a disease with no cure that begins by occurring once a week until, finally, it repeats nightly, overwhelming the neuropathway and causing death.

Upon commencing the reading, the depth in the piece is immediately apparent. There is a sensation of falling, or moving down a corridor, as the user/reader moves, via a dragging action with the mouse, through the many layers. Each layer of the text contains a patient case file. That case file can then produce a "book"; within these books are links that can open additional spaces. Hence, the *Dreamaphage* reading is like peeling away layer upon layer, while simultaneously adding to the collage of sound, image and text contained within the work. While moving between layers, the reader is presented with patient records. These files are anonymous and only associated with a patient number, but they contain an analysis, virus cure date, treatment and doctor. Selecting the patient record opens a "book". The "book" is an interaction method within the work. To "use" the book, a pulling motion is required, which drags the pages from right-to-left (or left-to-right to move backwards in the book). It is a style similar to the navigating forward and backward through the layers, through holding the mouse button and moving the mouse. Even though the interaction styles are similar extranoematic activities (holding down the mouse button and moving the mouse), the sensation is altered by the responses from the work. That is, the book navigation evokes a feeling of flipping through a book, while the movement navigation induces a sense that the reader is moving.

The second *Dreamaphage* changes the interaction style due to difficulties readers had finding and reading certain elements within the first. In the second version, the patient records sit over moving backgrounds that indicate a networked infection. The introductory is replaced with the book-style interaction mode previously seen in the body of the piece. The change to the interface and interaction mode in the two versions provide

competing readings of what is nominally the same work. If the extranoematic event within the text is too opaque, then the reading event suffers. The methods of reading compete for attention to disrupt the reading experience. For example, version 1 requires the reader to fight the motion within the work to even read the patient record.

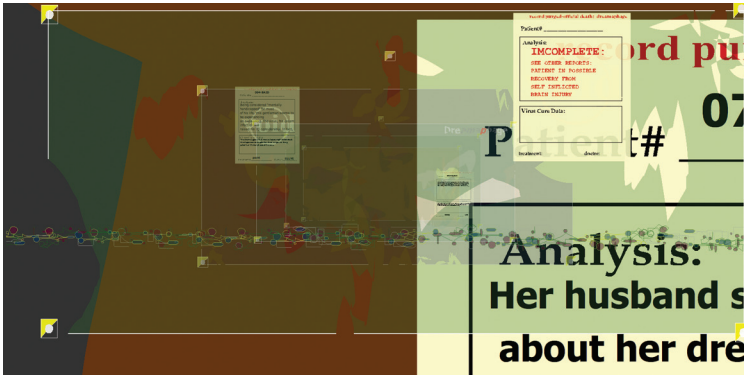

Fig. 4. *Dreamaphage* version 1

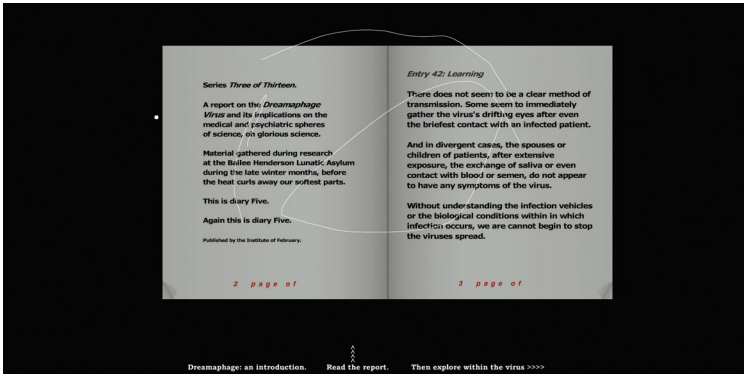

Fig. 5. *Dreamaphage* version 2 introductory "book"

The Dreamaphage virus's dual computational-human nature is foregrounded in the reading when a countdown is triggered within the work. The countdown begins stating "'Dreamaphage will be downloaded into the machine'", with the download success being noted as "'0000000111100011virusloaded11110001010100110'".[5] It suggests acts as a computational disruption of the text. Though Despite the fact that this is the most explicit element in *Dreamaphage* that presents self-reflexivity regarding code, it actually describes an event that does not occur. It, like contextual codework, is code that merely appears in the text as the functional code here displays it

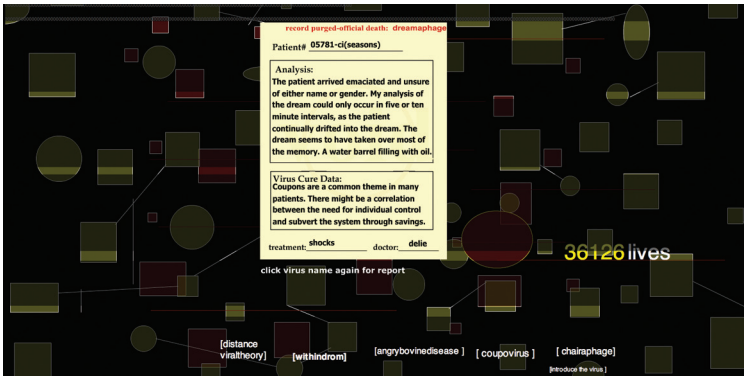

Fig. 6. *Dreamaphage* version 2

The multiple *Dreamaphage* versions also add to the sense that the work is layered. In producing it Nelson writes, "I wish more artists revisited older works, adjusting for changes in tech and poetic sensibility" ("This Is Almost Everything I've Created").

5 Dreamaphage is of course already loaded as it needed to be loaded in order to execute.

This aesthetic preference suggests he believes poesis is not fixed, but is instead realised in the dynamic nature of works.

In both versions, links within the books open fragments that increase the chaotic interaction with text, and an initially readable section will become overwhelmed with text, making it difficult to read. The HTML that frames the Flash piece, however, contains exactly the same text, so even as the "executable layer" becomes difficult to read, the more difficult to access source layer includes the text of the piece written as a whole. This reading of the paracode also extends to a reading of the directory composition. Fragments of the work's creation leak into the file structure. The directory that holds the second version of the work contains Flash source files (.fla) and Flash movie reports that list frames, images, file, bytes, scenes, ActionScripts and fonts. The listing of each of these HTML pages and Flash files adds to the collaged nature of the work.

Hayles, writing about Nelson, claims that as a practitioner he is among those "who think of themselves as primarily graphic artists and programmers writ[ing] texts to incorporate their works" (*Electronic Literature: New Horizons for the Literary* 22). The texts, however, seem incorporated in the collage much like the other elements involved. Yet, as with collage, the looseness covers a careful artifice. In a case study on Nelson's work *I made this. you play this. we are enemies.*, Funkhouser writes "Nelson uses and digitally expands appropriative techniques established by writers who have used cultural refuse as compositional elements in order to question the status quo and enliven human experience" (172). Those appropriative techniques suggest previous constructions within works like collage. This method of collaging translates to

the way that Nelson writes his code. In an interview with Carmel Hagen, Nelson states that:

> *When you are entirely self-taught, and rarely collaborate, you spend heaps of time scouring the net or code answers and possibilities. So what you would see if you cracked open my works is a mess of loosely tethered actionscript, sometimes in various different languages and versions of flash. ("Spear Talks: Jason Nelson")*

The program here then also reflects the collaged nature of Nelson's style. A reading of the mess of scripts may well give a parallel experience to the reading of the final *Dreamaphage* version.

Dreamaphage, it follows then, occurs as a sensory collage including aural, visual and interactive styles. This is an aesthetic that appears both in the production of the work and also in the expression of the executed work. Hayles writes that Dreamaphage, through its use of animation, sound and design, "testifies through its very existence to the extent to which code has become indispensible for linguistic expression" ("Traumas of Code" 39). The code acts both symbolically and actually in the realisation of this expression, whether contextually in the phrase suggesting a downloaded virus or even directly through the creation of the extranomatic event in the different versions. The work is impossible without code and capable of being realised without being read.

How then to write the ways in which code exists in poetic works? In the next chapter I discuss the works I created to attempt to help answer this question.

CHAPTER 6

Programmable Poetry and Writing Code ~~That~~ (Works)

> *Text is indeed "the web's primary and foundational media" and the artists of text are poets.*
>
> John Cayley, "Time Code Language: New Media Poetics and Programmed Signification"

For this exploration of code as containing multiple methods of reading and writing, I wrote ten pieces. The first five are print-based works that attempt to test the boundaries of code in a flat-material medium (without going beyond the page). The last five are HTML-based works that use the two surfaces inherent in that medium. All ten works are constrained works even within their respective medium, but each attempts to engage with code in the different ways that have been discussed throughout this exegesis.

Print-based works

Code can be used to produce, develop and extend the poetic. The use of code provides an opportunity to create poetic works in print either as a cultural reference, using its social and functional attributes symbolically, or to programmably generate poetry with a fixed end result. These two methods of creating poetry from "code" distance the work from an actively executing code. The code has been run either purely contextually (with an awareness of the social value of code), or run as a computational method of generating a work. A third form of code(d) poetry, as described by Cayley, uses punctuation experimentally on the page as a form of programming.

I have used these three methods as a way to explore the poetic in code. The print-based works were written with poetry generation software, privately written scripts or in the context of programmable relations, as were also an attempt to explore the paratexually programmable nature inherent to experimental poetics. The poetry is constrained to only use the flat surface suggested by print, rather than the dynamic possibilities of the material properties of paper.

Sognare la tromba / Suonare la tromba

"Sognare la tromba / Suonare la tromba" is only possible because of contemporary society's relationship to the web. The loss in translation intimated at in the poem, the translation of Eco's *Foucault's Pendulum* from Italian to English, can be symbolic associated with loss of meaning when code is executed, the translation from text to function.

The difference between code as writing and code as programming is like the difference between the "sognare la tromba" and "suonare la tromba". More specifically, they are alike in appearance but not in enactment. Playing and dreaming are two different acts, even if they sound similar in Italian.

The cultural context for the work is a networked coded environment. Both the Amazon review and Google Translate exist because of very large amounts of code and networked systems. These are popular software systems wherein the execution of the underlying code is visible. This does not place it necessarily within the realms of codework, however, as the poem is not referring to coding subculture or technically specific esoterica. There is no subculture here: both Amazon and Google are monoliths in contemporary internet-aware cultures. Moreover, it is the expression of code in the poem that is naturalised and detached from its form. The experience of software, web browsers and websites are now a normal and commonplace mode of interacting with computers.

elemental positioning

The work "elemental positioning" is an attempt to express the conceptual contextual nature of code within a print-based piece. As a consequence, it operates like Cayley's second category of codeworks, reflecting a work that uses code as a context.

XML's base node, the root node, offers an intersection of terms between the meaning of root as computational term and root as a natural term. Building from the root node extends the XML outward like a tree. The root moves onto the trunk, branches (terms also used in software, such as "version control systems") and then finally to a bird. This is structured nature. Setting

the XML namespace as “nature://tree” puts “nature” in the protocol position and “tree” in the identity position; this is then read as, “tree as understood through nature”.

The work illustrates the boundaries in which certain codeworks can be placed. Like antiabsorptive techniques, the dominant cultural relationship to the form and content challenge their definitions. If the content or form is code-informed poetry (as Cayley says of the second category of codeworks), then when the subcultural becomes cultural and everyone starts writing XML, the work’s designation as codework will be challenged. Engineer and poet Maged Zaher also uses XML-like structures in his poetry, but within the context of experimental poetics as opposed to codeworks. Thus, code as a base of execution is a material property of the work as opposed to a contextual property.

rendering

A form of paratextually programmed writing, “rendering” works on the word and the line to generate additional and ambiguous language. The coding here directly appears on the line by using non-software code and punctuation, breaking apart the words. The multiple readings then arise from rereading or shifting perspective along the words and lines. Rendering is an act of interpretation, a translation, and the generation of computer graphics. I attempted to have the poem enact its contents as it writes it, with the reading itself a process of generating meaning from the poem. A reading event in “rendering” metaphorically follows the generation of meaning by (re)reading the text and the generated text.

The need to reread causes a jitter between the text that presents its multiple meanings linearly, such as, “daedalus amazed by the

bullish market . / his son by the bear" refers to the Labyrinth (a metaphor for the text itself), the Minotaur and Icarus's fall through a metaphor of the market. Then the line, "trading futures froze- 'n fry's combinatorial post simp- le . ic . / (hth|ar) us broadcast on ev| angel . i . ca . l . end . ars gratia art /.is", nominally begins with a reference to the market but also produces "icarus / ichthus", "angel / evangelical / calendar / ars gratia artis / art is". The "icarus / ichthus" is then encoded once again, this time to stand in for "sons" to produce a cultural reference to *Futurama* "future's frozen Fry's post Simpsons".

Producing language at the line through word-encoding parallels a constrained coding in more forgiving high-level languages. The writing is constrained by the natural language (in this case English) and the function that acts on the text. The execution is, however, culturally contextual, rather than technically contextual, as it for the case with code. Natural languages fail to execute when the language changes or loses context. Programming languages fail when they cease to be compatible with newer versions or are no longer supported by hardware.

I sing the sound electric

"I sing the sound electric" was generated using Gnoetry v0.2 from Linux HOWTOs, the screenplay for *Blade Runner*, and the lyrics to Gary Numan's "Metal". In writing this work, I took what could be called a curatorial position to produce the writing. The method of creating the work itself was left to the program. Instead I chose, found and formatted the input texts and also the method of output. Gnoetry produces new texts based on the properties of the source texts, such as the statistical distribution of words.

The source texts were chosen because of their exploration of ideas of construction, as depicted in *Blade Runner*'s replicants or Numan's line, "We're in the building where they make us grow". The Linux HOWTOs, part of the Gnoetry base texts, provided the third text. Given the use of the properties of the source text as inputs to produce the final text, Gnoetry recasts the texts as coded inputs. The structure of these source texts also acts like a code, which helps to shape the final text.

The code, then, is distributed between Gnoetry and the source texts. The fragments of the code that persist in the print piece are those that existed as statistical properties in the original texts.

hangman

To produce "hangman" I wrote a Python script that reads in a text and breaks it into strings (of mostly words). The script is designed to filter or remove characters, exclude words with certain characters or require the words in the output to contain specific characters. The last word of the work selects the number of lines and the last word of the line selects the number of words in the line. This results in a method that only links to the source text through the use of words, and not with the structure of the original text (unlike in the last poem).

Using Mary Shelley's *Frankenstein* as a source text, I generated hangman by selecting "ang". This makes the "ang" common to all the words in the work. As with its namesake, the game Hangman, it centres on selecting the correct letters to find the final game. Rather than using the format produced by the Python script, I reworked the text into a single line and justified its formatting. The change in structure emphasised the

"ang" and made the piece seem more complete. The language, however, was not edited and led to the fortuitous phrasing "stranger strangled languages", which left me to ponder whether I am the stranger strangling the language or whether it is the machine.

This piece also distinguishes the code in this work as external from the work, highlighting it as a method, as opposed to a part of the work. The code here has been used to automatise a process I could have manually performed; thus the code reifies the method of producing the text but is clearly not a part of the final text. Hence, the aesthetics of the code in this work are as separate from that of the work as the source text it was generated from.

HTML-based works

Using HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript gives the advantage of presenting both the code and the representation of the processed code in the finished work. Artificially, I have sacrificed the networked and interlinked nature of the medium. While the code required to process the works is present in the browser, the works themselves are self-contained. I have eschewed the use of image and audio files, as well as external libraries such as jQuery or prototype, in order to keep the code contained to a single file. The result is a constrained process that requires all effects to be generated by the visible code in the work. In this way, I aimed to mimic print-based single poetic works, while exploiting the differences that code brings to those texts. Libraries of code do have a

symbolic value in representing the way we bring our knowledge, as readers, to a text. This, however, extends the readable text to the libraries, which if the code is to be read in its entirety, should also be read.

The code in the pieces attempts to characterise the ways in which code challenges the methods of writing and reading. The HTML works combine and use various forms of languages in order to produce the effect. It is important to note that the natural language used within all the works is English. Natural languages in poetics are manipulated to produce meaning in wordplay, rhyme, visual rhyme and so on. In HTML, using English as the natural language is significant because HTML, like most popular programming languages, uses a programmable vocabulary based on English. The HTML is used structurally to scaffold the work, and, as Glazier points out, is still part of the meaning-making and cannot be separated from the natural language, English, present in the work. More specifically, the HTML used is HTML 4.01 and so the content type (MIME type) is "text/html". The style level, the level describing the presentation of the HTML, is defined by CSS, a separate style sheet language. JavaScript, the fourth language used, is a scripting language for programming the behaviour of the works. All the languages used "execute" on the client-side: in the browser. The HTML works have a defined encoding. Text computationally has multiple methods of encoding that define how the characters are encoded; in other words, they are read by the computer in ways that can be stored and displayed. For all the works listed below, UTF-8 is used.

In addition, the following pieces attempt to explore the way code affects reading in terms of reading as an event, and the

representation of "active" code in the presentation of the extranoematic event.

the representation of self

"the representation of self" is a fixed non-interactive poem. The first layer of interface text only casually hints at deeper layers of the work. The single line stands broken on the surface, a shape suggesting a roof, or a logo for a real estate agent, or the tip of an iceberg (the white against blue). Reading only the surface text opens multiple readings of the piece through the ambiguity of both the line and the line's shape. The break on the line occurs mid-word giving an angular balance but additionally breaking the language in the text. The code at the surface layer could be presented as merely a typeset concrete poem. The poem, situated as it is in HTML, suggests a representation of the self is a re-presentation.

At the HTML code layer, the line "the representation of self" is still broken but now it appears as straight line split by code.

> * the representation of self *

The line is translated by the browser, which uses styling to present the angled text. The self is then split on the surface and in the code: language fractures the self. Disabling the style leaves an unbroken single line in the top-left displayed by the default styles within the text. Removing the programmatic transformation removes any meaning signified by the line's shape.

The line is above the rest of the poem, suppressed from the interface text. The two modes of suppressing text include hiding

the text stylistically, so the content is transformed to be invisible (such as with the styling "display:none"), or preventing the text from being rendered using HTML comments (between "<!--" and "-->"). The text under the line is set as comments, meant for reading but not for rendering. Glazier writes explicitly in his poem "Mouseover" that "[a] document source is writing, too", an invitation to readers to read the work's source code. As demonstrated, the source is always present for HTML works and always performing, even in the simplest sense of translation.

The fact that there is another layer acting upon the surface works metaphorically here for how the self is typically presented. We only present our outer selves to the world while the inner self remains hidden. The code hidden in the comments, however, is not acting on the surface. While it exists in the piece, it is never represented. Equally, while the real exists as surface, the line "the real self" only exists in the work's code.

subsurface

"subsurface" is a layered encoding: literally encoded characters at the HTML layer. In the surface text the work is fragmented, while the initially presented interface text is what appears to be a natural language text infected by a typographical (en)coding. For this reason, it could fit Cayley's second category of codework in the way that it is written within the context of typographical coding, and the third category at the HTML layer, whereby it uses the letters of the HTML4-named entities to create words. The letter shares its place as the smallest unit of composition with the entities that required defined patterns in order to be presented.

The oscillation between surface and subsurface flips between readings and codings. The "sub" of the title is encoded leaving only a visible title of "surface" (the sub being the symbol for "subset of", allowing another reading of the title as "subset of surface"). The code pushes against the surface stylised by the CSS to appear as protruding. Yet, the surface codes do not clearly translate on the first interface text (the surface text) as expected in English. The "&" read as "and" exists in the code as "&". The text within the poem plays on this form, paralleling the entities as ingredients. Code is a recipe for the realisation of an anticipated executed form.

In regards to HTML entities, Glazier classifies them as situations "where resistance of coding, or its disruptive penetration into text, give code a more arcane look" (109). In treating the code as "[c]ode as text to be read as (if it were) natural language", the entities have been abused, separated from both their meaning and verbose forms ("Time Code Language: New Media Poetics and Programmed Signification" 317). At the code level, "subsurface" presents itself as a work in the mode of more common print-mediated works but still broken by "&" and ";", which are both required for the named entity to be rendered. To read this layer as a more common form of poetry, the characters would need to be pushed out of the reading, with the letter and word forms foregrounded, while the intrusive characters were pushed to the background.

Re(U)topia

"Re(U)topia" is an infinitely generating poem. On scrolling, more text is generated so that new stanzas are created from a list of unique words that occur in Gilbert Burnet's translation

of Thomas Mores's *Utopia*. Each stanza is comprised of six lines, with each line consisting of six words. "Re(U)topia" does not, however, use the word frequencies or word combinations within the original text. As such, the lines are random artificial combinations of words producing meaning only in the reading. A sense of unity is created by the slightly archaic nature of the translation and the thematic word listing.

This work is, like "Machine State" and "TERMINAL", a combination of HTML, CSS and JavaScript. "Re(U)topia" is another example of the hidden code within the work, as the text of Utopia was processed by a Python script to obtain the word listing. This code is entirely functional, a shortcut to what I could have done manually, but still shows the grey areas where other "codes" intrude into these works.

The attempt to reach or find a utopia is given over to the computations of the poem. The poem, however, will not end until the software or hardware it contains is unable to continue processing the stanzas. Thus, the utopia is unattainable; it cannot be reached, and yet each line promises more possibilities for meaning.

TERMINAL

"TERMINAL" plays on and with the multiple meanings of the word "terminal". It is an end point but rarely acts like one: in computing it is an interface for entering data, in illness a death, and in transport an end of a journey although more often a transition point (no-one permanently sleeps at an airport terminal). In this way it both interacts with computational ideas of the terminal literally and symbolically because the work is in

constant motion. The work is an end point when experienced on screen, yet is never still.

The letter-board below the title "TERMINAL" was created as two sets of letters that are then switched in time by JavaScript. Its functions operate as a separate portion of the whole work. The functions calculate the list of characters between the current letter and next letter and then swap each letter along the list until it arrives at a new letter. The timing difference between the top and bottom set are intended to create a rotating sense, as is the offset between the sets of letters. Below the faux flight board, the list of phrases move up and down, depending on the location of the arrow. They are able to create new longer lines even as they exist as separate fragments.

The terminal displays the code, but the code is also used to transform the stationary code into motion. Within this piece, code is able to perform in a way beyond the merely metaphorical layered sense, engaging in the temporal and the spatial, through its hiding and ordering of letters and numbers. The expression of the code is the end point ("terminal") realised in an extranoematic event. However, the extranoematic event is not the end of the piece, but rather a transition to the reading of the work. The real terminal is the reading of the work, which can never be terminal, because the work will resurface while read and even after being read. It is a transition that cannot stay still.

The code is necessary to the work as programming but also as encoding. The rotation of the letters in the board suggests the old flight information boards but like the boards that have now replaced them there is no need to show the transitional letters. The transition mimics the material constraints that existed in those boards but that are now entirely artificial.

The code in this piece is necessary to the work as programming but also as encoding. The rotation of the letters on the board is reminiscent of the old flight information boards, but as with the boards that have now replaced them, there is no longer a need to show the transitional letters. The transition mimics the material constraints that existed in those boards but that are now entirely artificial.

The act of reading "TERMINAL" below the static title is disrupted by the constant movement of the text. The removal and rearrangement of text is anchored thematically. While narrative meaning is almost entirely erased by the movement, the movement itself is an attempt to initiate a meaning from the actualisation of the temporal and spatial. Terminals are places of constant movement; the language of terminal, the movement back and forth, become representative of that movement.

Machine State

This poem presents the reader with a set of "windows". The windows are meant to resemble an old Macintosh-like interface but, as with the nostalgia it may evoke, there are differences, such as the colour, which is not quite the same, nor is the interaction method. It is a (mis)remembered experience.

The HTML code in "Machine State" structures the content of the poem. Each poem fragment is placed within blocks of code identified by their class as "cards". The cards are marked up as a list. At a code level it suggests a set of index cards, thus an informational design. The CSS applied to the HTML is used to display the "cards" as windows, while JavaScript is used to define the window's behaviour. The poem can only exist in the

presented form when all the code is active. It remains "readable" however when the code is disabled.

There are multiple methods of reading "Machine State", and the state of each window is transitory at the interface layer. Each of the windows allows the text to present three states. The first state displayed is the mid-state. From this option the window can be enlarged or closed. In the larger state, the text is transformed. In the closed state, the text is unreadable. The text, though, is always present in the source code; in fact, the code in this example is artificially layering the text. The different windows within "Machine State" also present different texts and modes of text. The "State", for instance, signifies both a condition and the nation state. Each card has thesis and anti-thesis and a conflict between its own state. The dialectic presents a metaphor for reading the code as well as reading the poetic. As a result, code can be read as an executable form or a literary form, with the synthesis of these providing meaning from the work.

This is to show that code both acts as a symbolic layer and also acts as a virtual layer. Like reading process, the window of interest, what the reader reads, is foregrounded while the presence of the other windows lurk at the edges. No reading order is favoured between the windows. The reading event occurs in the juxtaposition of the windows both internally in and across. The extranoematic event requires reader participation to realise the spatial dimensions of the work and is essential to the reading that occurs.

"Machine State" transformed the most during its development. Initial iterations of the piece divided a lyrical content from the framework the code built. The poetry in each card was similar to each other and while they addressed ideas of state they acted as

separate works. Generated and textually experimental windows were added when the idea of the state and the programmable was broadened in this piece.

Conclusion

Poetry is code

Cordite Poetry Review

Code is, naturally, significant to the understanding of works in programmable media. The reading of code has presented the opportunity to view poetic works as capable of being read not just in execution but also in context. While broken codeworks do not work, they realise the potential to read code culturally as well as functionally. Cayley's categorisations of code, as discussed here with regards to literary works, provide a means of defining the way that code is understood.

Through the case studies and my own work, it seems code in poetic works can have three forms of being read: in the surface text due to execution (also referred to as the extranoematic event), in the code against a literary aesthetic and in the code against a computational aesthetic.

Writing code in a variety of methods in print and digital works changed the way I thought about my practice. Although I had written both print poetry and poetry in programmable media prior to this project, and recognised that they stemmed from similar traditions, I always treated both as separate practices.

I saw the programmable works as challenging my functional programming practice and the print works as a poetic linguistic practice. While I appreciated the linguistic and literary play in programmable works, I did not grasp the potential of the programmable in print works.

The notion of the programmable as a method applicable to textual manipulation acting on inscribed texts seems to make it even more specific. By producing these ten texts, I understood that my work in both print and programmable media are on a continuum. More than that, I realised that the programmable was crossing over into my print works as a method of writing the text. The code, when written to perform a function when it is read, changed my perspective of it as aspirationally executable. The code pieces I wrote that used coding as context or a non-functional method, such as "elemental positioning", structured the text rather than working programmably.

As an extension, investigating the programmable acting on the literal as it does, that is, abstractly from the medium, would be a fascinating topic.

Works Cited

Aarseth, Espen J. *Cybertext: Perspectives on Ergodic Literature*. Baltimore [u.a.: Johns Hopkins Univ. Press, 1997. Print.

Bernstein, Charles. *A Poetics*. London England; Cambridge Mass.: Harvard University Press, 1992. Print.

Cage, John. "John Cage on 'Empty Words' and the Demilitarization of Language, in a Radio Interview, August 8, 1974." Alan Filreis n.d. Web. 17 Oct. 2012.

Cayley, John. "Screen Writing: A Practice-based , EuroRelative Introduction to Digital Literature and Poetics." *Literary Art in Digital Performance: Case Studies in New Media Art and Criticism*. Ed. Francisco J Ricardo. New York: Continuum International Pub., 2009. Print.

---. "The Code Is Not the Text (Unless It Is the Text)." *Electronic Book Review*. Electronic Book Review 10 Sept. 2002. Web. 3 Mar. 2012.

---. "Time Code Language: New media Poetics and Programmed Signification." *New Media Poetics : Contexts, Technotexts, and Theories*. Ed. Adalaide Kirby Morris & Thomas Swiss. Cambridge, Mass.; London: MIT Press, 2006. 307-333 Print.

Collis, Stephen. *On the Material*. Vancouver: Talonbooks, 2010. Print.

Cox, Geoff, Alex McLean, and Adrian Ward. "The Aesthetics of Generative Code." Web. 17 Jun. 2012.

Funkhouser, Chris. *New Directions in Digital Poetry*. London: Continuum, 2012. Print.

Glazier, Loss Pequeño. *Digital Poetics: The Making of E-poetries*. Tuscaloosa: University of Alabama Press, 2002. Print.

Hayles, N. Katherine. *Electronic Literature: New Horizons for the Literary*. Notre Dame, Ind.: University of Notre Dame, 2008. Print.

---. *My Mother Was a Computer: Digital Subjects and Literary Texts*. University Of Chicago Press, 2005. AZW.

---. "The Time of Digital Poetry: From Object to Event." *New Media Poetics: Contexts, Technotexts, and Theories*. Ed. Adalaide Kirby Morris & Thomas Swiss. Cambridge, Mass.; London: MIT Press, 2009. 181-209 Print.

---. "Traumas of Code." *Critical Digital Studies : A Reader*. Ed. Arthur Kroker & Marilouise Kroker. Toronto; Buffalo [N.Y.]: University of Toronto Press, 2008. 25-44 Print.

Galloway, Alexander R. *Protocol : How Control Exists After Decentralization*. Cambridge, Mass.: MIT Press, 2004. Print.

Iser, Wolfgang. *The Act of Reading : A Theory of Aesthetic Response*. Baltimore: Johns Hopkins University Press, 1980. Print.

"Issue: Explicit Markup to Semantically Express Poetic Forms." HTML WG Wiki. n.d. Web. 11 Aug. 2012.

Jones, Patrick. *[How To Do Words With Things]*. Daylesford, Vic.: treeElbow, 2008. Print.

---. "Interview with Patrick Jones." Ed. Jessica L Wilkinson. *Rabbit number 2* (2011): 132-154. Print.

---. "Step by Step." *Overland*. Overland Winter 2012. Web. 17 Oct. 2012.

---. *Words and Things : Concrete Poetry Supersigns Multiple Language*. Daylesford, Vic.: Reverie Press Publications, 2004. Web. 17 Oct. 2012.

Laird, Benjamin. "elemental positioning." PDF.

---. "hangman." PDF.

---. "I sing the sound electric." PDF.

---. "Machine State." Web. 17 Oct. 2012.

---. "rendering." PDF

---. "the representation of self." Web. 17 Oct. 2012.

---. "Re(U)topia." Web. 17 Oct. 2012.

---. "Sognare La Tromba / Suonare La Tromba." PDF.

---. "subsurface." Web. 17 Oct. 2012.

---. "TERMINAL." Web. 17 Oct. 2012.

Leong, Michael. *e.s.p. : poems*. Columbus, OH: Silenced Press, 2009. Print.

Lutz, Mark. *Programming Python*. Sebastopol, CA: O'Reilly, 2006. Print.

Marino, Mark. "Critical Code Studies." *Electronic Book Review*. 4 Dec. 2006. Web. 17 Oct. 2012.

Massumi, Brian. *Semblance and Event: Activist Philosophy and the Occurrent Arts*. The MIT Press, 2011. Print.

Mateas, Michael and Nick Montfort. "A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics" Digital Arts and Culture. Copenhagen, 2005.

Mez. "_cross.ova.ing 4rm.blog.2.log 07/08 XXtracts_." *Electronic Literature Collection Volume 2*. February 2011 Web. 22 Jun. 2012.

Nelson, Jason. "Dreamaphage." *Electronic Literature Collection Volume 1*. October 2006 Web. 22 Jun. 2012.

---. "Spear Talks: Jason Nelson." *The Josh Spear Blog* 2 Feb. 2010 Web. 3 Oct. 2012.

---. "This Is Almost Everything I've Created." *secret techology* n.d. Web. 22 Jun. 2012.

Raley, Rita. "Code.surface || Code.depth." *Dichtung Digital*. 2006. Web. 17 Aug. 2012.

Simanowski, Roberto. *Digital Art and Meaning: Reading Kinetic Poetry, Text Machines, Mapping Art, and Interactive Installations*. Minneapolis, MN: University of Minnesota Press, 2011. Print.

Strickland, Stephanie. "Moving Through Me as I Move: A Paradigm for Interaction." *First Person : New Media as Story, Performance, and Game*. Ed. Noah Wardrip-Fruin & Pat Harrigan. Cambridge, Mass.: MIT Press, 2004. 183-191 Print.

Williams, William Carlos. *The Collected Poems of William Carlos Williams. Vol. 2. 1939-1962*. Ed. Christopher J MacGowan. New York: New Directions, 2001. Print.

Zaher, Maged. *Portrait of the Poet as an Engineer.* Boston, Mass.: Pressed Wafer, 2009. Print.

01010000
11001011
11011100
01010011
11000111
11011111
11001000
11001011
11001000
01110100
00011011
00010100
11100101
11001011
11000011
11001000
01011000
01000000
11101111
11100101
11010011
11101000
11001011
01011000
01000000
11001011
11110000
11001011
11000111
11101011
11101000
11001011