"I hope you can read this"

Uncovering messages with Critical Code Studies

Mark C. Marino

"[I hope you can read this.]"

That is the first line of the source code of the work of interactive fiction called The Gay Science by Capricorn Van Knapp. The title of the work comes from Nietzsche's *Die fröhliche Wissenschaft*, and the piece offers a meditation on loss and the eternal return (*Ewige Wiederkunft*).[1] Let us read this code together as a way of pursuing the question: what does our interpretive exploration offer? The code goes on from there:

"[I hope this works.]

[I hope.]

'The Gay Science' by Someone Who Loves You.

[This is the truth, as I understand it.]"

The lines in brackets are commented out in the interactive fiction programming language Inform 7. The first line that is functional code (as opposed to comments) is the title and author credit, although note that

1 Christian Benne/Jutta Georg (Hg.): *Friedrich Nietzsche: die fröhliche Wissenschaft.* Berlin/Boston: De Gruyter 2015. *uosc.primo.exlibrisgroup.com*, https://doi.org/10.1515/9783110440300.

another name (Someone Who Loves You) stands in for the game's author (Van Knapp, which is also a pseudonym for Dan Ravipinto), so it would not be right to call that line merely a functional label, like the title of a document. It is more than merely the title of the piece and its author, it is part of the narrative of an interactive fiction framed as a work by a fictional Someone for a fictional You, reminding us that when we are reading the code we are still in the narrative framework of the story. In interactive fiction written in Inform 7, since the code shapes text that displays the story, it is difficult to draw a strong division between code that produces the story and code that tells the story, and it is especially difficult in a project written for a contest for beautiful code. More on that momentarily.

As we move sequentially through the code, after a chapter heading ("Chapter 1 – Alethiology," which I will gloss as "the study of truth"), an organizational feature of Inform 7, the code continues with more comments:

> "[There is truth here. I have to believe that. For I may only make statements of truth here. For only the truth can save you.]
>
> The story headline is 'An Interactive Truth'.
>
> [Only the truth can set you free.]"

The "story headline" is another feature of Inform 7, which will appear as a subheading after the title, so that the start of the story will read, "The Gay Science" followed by the line "An Interactive Truth by Someone Who Loves You." The code continues.

> "Section 1 – Us (You and I)
>
> You are a man.
>
> Someone Who Loves You is a man."

These lines create characters, whose genders may inordinately focus our attention on the homoerotic meaning of a work that is deeply concerned with Nietzsche's text. Nonetheless, this game is ultimately a love story or an infinity of love stories about two men, a "You" who is lost and a "Someone Who Loves You" who is trying to reach out to them. The names of the two men are interchangeable, as this sad romance generator randomly serves up characters coupled in the positions of You and Someone Who Loves You. The You character in each iteration is lost in some way. The Someone Who Loves You, who, as it seems, is the avatar or proxy for the fictional character author of the code, wants to rescue or at least reconnect with the You. We, readers of the code, are invited to identify with the You character, as the section header indicates, for we are part of the "us."

In generated story after story, the two characters take their position in randomly chosen (or pseudo-randomly chosen) scenarios, from lovers separated by death to a police detective romantically entangled with a suspect. That fungibility of the reiterated stories and characters play out Nietzsche's eternal recurrence, for these characters can be given randomly assigned names and fictional positions, but their pairing in this just-out-of-reach love story is the same. The code itself becomes a self-declared representation of "eternal recurrence" and our reading of the code exposes that existential pattern, exploring its repetition with randomized variation. Life is a recurrence of the same story in a randomly chosen change of clothes.

However, randomness, in this code, is not always so random.

In a gesture of love and loss, Someone, the fictional author of the code, decides to rig the odds of the game. The following code determines whether the program will display the "happy ending" or the "bad ending."

```
"To say where this story ends:
        if a random chance of 1 in 1 succeeds:
            now the happiness of the end is true;
            say '[the happy ending]';
```

otherwise:
say '[the bad ending]'.

[And so because I love you, I cheat. And give you only happy endings.]"

As it turns out, this game will only deliver happy endings, for the "random chance of 1 in 1 succeeds" every time. The program has been written to stifle the randomness, and that act is meaningful, that choice in the code is the sign of the programmer, Someone, placing their hand on the scale all for the sake of You.

But who is this "You" of the comments? Who is being addressed? One reading is that the You is us, anyone who reads the code of this piece, which was entered into a competition for a work of beautiful code. The first event of the Second Quadrennial Ryan Veeder Exposition for Good Interactive Fiction challenged participants to write a great game with "beautiful source code text." And yet, like a poem, the speaker or encoder seems to be a character who is other than (but possibly a stand-in for) the code's author, and the addressee seems to be someone that character loves or loved before they were lost to him. We are reading someone else's love letters that take the form of a work of interactive fiction. Or we are reading perhaps the letter a lover wished he had sent, wished he could send, a letter made out of code.

As I have argued elsewhere,[2] the code of Inform 7 is deceptively English-like, or perhaps better said, the code looks like English sentences which may obfuscate its status as code, with natural language reserved words, tokens that are arranged in a sentence-like syntax. For example, the line "now the happiness of the end is true" seems to be telling us that there is true happiness at the end of the story. But what that code is literally doing is setting the value of "the happiness of the end" to true. In the English sentence the assertion offers a judgment. In the Inform 7, the statement makes an assignment. That difference marks a central tension

2 Mark C. Marino: *Critical Code Studies*. Cambridge MA: The MIT Press 2020, 150.

of The Gay Science, where the work presents love stories with happy endings but only as programmed fictions, as maquettes under the control of the programmer, who presumably, like the rest of us, has very little control over real life, and hence ultimately experiences loss. This code then presents the fantasy, the love offering, of one who longs to reconnect, who longs to program for themselves a guaranteed happy ending. We would not know any of this backstory without reading the code, for the game itself offers little sense of what is at play or even how to play it. For the most part, you have to read the code to know what to do in the game, and "reading the code" comes to stand in for a kind of philosophical awakening, but also taking a peek at the private subtext and encoded messages of the code's fictional author.

Only the code indicates how to reach the game's end. In the game, the player can eventually move through successive spaces outward that represent moving sequentially through the story, from the beginning to each "consequence" and on to the end, including a section marked the Eternal Recurrence. However, to finally reach an end, the player must reverse course and move "inside." That is how they find their way to the ending called "Home," just how the You of the story could find their way to the heart of the Someone if they only read the code. They must move, in a sense, "inside" the work of interactive fiction.

The Gay Science serves as a powerful and literalized illustration of the kinds of discovery that awaits readers who explore any source code. In any code, there are messages and meaning that a (typically) unintended audience can find.

Code is a message overheard, full of remnants of the design process and sometimes indications of future directions. These signs are not merely in the comments of the code but the code itself. Though we name what intention lay behind the choices at our peril, the question "why" may lead us to a fuller understanding of how this code came to be. The code certainly offers a sense of how this system operates, and there are further signs of "when" it came to be and "who" is behind this code. As to the "what," the code does not so much offer us the underpinnings of the meaningful software object to be interpreted, but is in itself a complementary object to be perused.

1. Why code?

Computer code exists for human readers. Computers do not need code (assuming they need anything at all). They could run, as Kittler has suggested, on electrical signals (i.e., "There is No Software").[3] Code exists so that humans can see their own inputs to the computer in the form of programs and data, a distinction that becomes blurry on further inspection.

The more obvious examples, or rather the most easily accessible examples of code analysis for readers steeped in the humanities, involve the code for aesthetic objects, such as interactive fiction and other electronic literature. Code poet and critic John Cayley recently made the pronouncement,

> "We are right, critical code studies is key for the hermeneutic understanding of eliterary work. And we must keep on telling our colleagues in literary studies that reading the code of such work is required; or that if code reading is not done, the omission should at least be acknowledged. For true close reading, this work *must* be done."[4]

His stance is particularly pertinent, since the original Critical Code Studies manifesto[5] was written in response to his comments about code. Nonetheless, if we take Cayley's words to confine critical code studies to the literary sphere, we miss a major contention of these practices, the contention that all code offers material for interpretation

3 Friedrich A. Kittler: "There Is No Software." In: *CTheory* (1995): http://www.ctheory.net/articles.aspx?id=74 (accessed September 24, 2023).

4 John Cayley: *Bridging Electronic Literature & Critical Code Studies (a Digital Humanities Quarterly Discussion)*. https://ucpages.uc.pt/events/overcoming-divides-electronic-literature-and-social-change/programa/ (accessed September 24, 2023). Electronic Literature Organization Conference, University of Coimbra, Portugal.

5 Marino, C. "Critical Code Studies." *Electronic Book Review*, Dec. 2006, http://www.electronicbookreview.com/thread/electropoetics/codology (accessed September 24, 2023).

because code is a cultural text. More importantly, code is a cultural text made out of synthetic languages with a highly unusual double status as messages to humans to read and messages for machines to process.

Nonetheless, I do not want to make too much of this work of interactive fiction, which was created in the context where human readers (the contest judge and audience) are expected to read the code. I do not want to make it seem that the only pieces of code worth reading are either the code of art objects or code written as art. However, critical code studies posits that all code can and should be interpreted and offers opportunities to explore and produce meaning.

The Gay Science makes that invitation to read code literal, but I contend that it has a lesson about every piece of code. First of all, while code is for computers, code also addresses a reader, who may be the same programmer later on or maybe others. Code reveals an orientation toward the world through a problem space and decisions that have been made. Priorities. Models. Those decisions are not merely in constructing the code but also in the choice of the language and architecture, and those choices have been made within material and historical constraints. The reader of the code has the opportunity to find a message in the code, underlying logic, as well as signs of possibility for what they may never have encountered in the program, just as reading the code may not give them a full sense of what the code does. Thus, the process of reading the code and watching it operate is dialogic. Think of the practice of putting breakpoints and flags in code. I feel invited by software to read its code though I am not the one to whom it is addressed.

The digital world is made out of code. Often invisible, computer source code constructs and transforms our world, touching almost every facet of our lives. Under the metaphor of transparency, glossy and spare interfaces hide the code from us until it becomes as invisible as the wiring in our machines and the blood vessels in our body, or perhaps more accurately, the laws that govern our nations or the social norms that control our behavior. Think of the iPhone, a sealed, sleek brick with inaccessible circuitry, appearing to be only interface. Think of the Google search page, bare white except for a single text-entry box and two buttons. However, transparency, in this context, signifies its opposite

as software functions through an opaque interface designed to create the illusion of a clean window, a bare desk, or a blank page. Only in the moments of fissure or error does any sign of code emerge. And when we do encounter code, most cannot read it, despite increasing calls for programming literacy.[6] Actually, even the programmer working on the enterprise-level software package who inherits a project from another programmer may find that code inscrutable, as it has been constructed using just-in-time solutions with after-the-fact patches, all in desperate need of refactoring and often documentation. Yet, since code touches so many of our systems and has such a definitive effect on the way we live, not reading the code means we do not know what decisions have been made for and about us and why. Coming from a country that once rebelled against "taxation without representation," it is ironic that we now live in an age of "computation without comprehension," especially when you add the new affordances of LLMs to allow us to produce code without a definitive knowledge of how it works. Of course, the pragmatic understanding of code (knowing how our software works) is only the beginning of the fruits of close reading code. But placing all code into a kind of black box creates a crisis of literacy (Vee). To respond to that crisis, I offer Critical Code Studies (CCS), a collection of methods and interpretive practices that have been growing for nearly 20 years, developing tools for understanding code and exploring what it means for technoculture.

Critical code studies is the interpretation of the extra-functional significance of computer source code using the hermeneutics of philosophy, or, as it is known in the humanities, critical theory. "Extra" here means not in addition to but growing out of. In other words, the meaning is not outside of the functioning of the code but something that takes its computational effects as a basis but only a starting point.[7] A CCS reading is built on the foundation of an understanding of what the code does, the systems with which it interoperates, as well as its

6 Annette Vee: *Coding Literacy: How Computer Programming Is Changing Writing*. Cambridge MA: The MIT Press 2017.

7 Marino: *Critical Code Studies*, 39.

history and evolution. Not so much a method as a developing collection of approaches, critical code studies sees code not as the ends of the analysis but the entrypoint into discussions of the culture in which the code was written and operates. Critical code studies takes code as a "text," a "cultural text," meaning that it is an artifact that carries meaning in cultures and can be read, not that it is made out of linguistic signifiers and can be read as static. The object of study is not merely the static text of the code but the effects that it has on the system and the many states it creates when a system processes it.

From the moment I first proposed critical code studies in 2007 at the MLA convention in Philadelphia, PA, CCS created controversy. To be honest, that controversy caught me off guard. Audience members asked: Why should literature majors be studying code? Why should readings of code be discussed at a literary convention? What could we possibly say about code? It's just math, just algebra, after all, right? The field caused controversy not just in literary circles but in computer programming circles as well. Perhaps writing an article about a queer programming language, Transcoder, was not the mildest of beginnings.[8] The computer language theorists of the discussion board *Lamda the Ultimate* were furious that the literature majors were coming to plant their flag on the shores of code with their ridiculous language of deconstruction and left-wing politics of Queer Theory and Feminisms (see Why We Must Read Code).[9] Would they now have to suffer through discussions of their code the way they had to sit through discussions of Rilke and Shakespeare? If neither the literary scholars nor the programmers could support the endeavor, who could?

---

8 Mark C. Marino: Of Sex, Cylons, and Worms: A Critical Code Study of Heteronormativity. In: *Leonardo Electronic Almanac* Vol. 17/2 (2012): http://www.leoalmanac.org/vol17-no2-of-sex-cylons-and-worms/?utm_source=rss&utm_medium=rss&utm_campaign=vol17-no2-of-sex-cylons-and-worms (accessed September 24, 2023).

9 Mark C. Marino: Why We Must Read the Code: The Science Wars, Episode IV. In: Matthew K. Gold/Lauren F. Klein (ed.): *Debates in the Digital Humanities*. Minneapolis/London: University of Minnesota Press 2016, http://dhdebates.gc.cuny.edu/debates/text/64 (accessed September 24, 2023).

So there it was. The Berlin Wall between the programmers on the one side and the literary scholars was reinforced and the division between the sides reified. The Two Worlds of C. P. Snow's formulation. In response, the early code studies scholars knew we had to build some coalitions and, I suppose, tunnel through the wall.

Thus began the Critical Code Studies Working Groups, biennial online gatherings of scholars and artists from around the world, drawn from professional programming and academia, hobbyist artists, and newcomers to the world of code. The goal was to see what could be said about code. Early readings from that group found meaning in natural language parts of code, such as method names or comments. For example, when I found that the developers of the *Transborder Immigrant Tools*, a satellite-driven GPS project named a method after water witching or water divining, I discovered cultural subterfuge that *Electronic Disturbance Theater 2* had tucked into their code.[10] It was hard to ignore the reverberations of function names such as "witchingEvent," marking when the system found nearby water. Suddenly a modern digital system seemed to be framing itself within the context of a folk practice. How could that not shape the meaning of the code and the software itself?

However, CCS theorist Evan Buswell has warned that if we only ever interpreted names of methods and variables, we would easily be dismissed as interested only in the ornamentation of code, not the tokens and methods themselves.[11] That is why an invitation by code poet and Platform Studies co-founder Nick Montfort to interpret a single line of code in BASIC was so provocative. In our first book-length publication

10 Mark C. Marino: Code as Ritualized Poetry: The Tactics of the Transborder Immigrant Tool. In: *Digital Humanities Quarterly* No. 1 (2013): http://www.digitalhumanities.org/dhq/vol/7/1/000157/000157.html (accessed September 24, 2023).

11 Evan Buswell: Comment, In Pursuit of Natural Language: FLOW-MATIC -- WORKBENCH. In: *CCS Working Group 2014* (2014), http://wg14.criticalcodestudies.com/discussion/comment/474#Comment_474 (accessed September 24, 2023).

of CCS, 10 authors explored *10 PRINT CHR$ (205.5+RND (1)); : GOTO 10*.[12] In the exploration of that eponymous line of code, artists, scholars, and programmers approached the code from every angle they could find, from the appeal of mazes to experimental dance, from the culture of early home computing to the practices of textile design. We read the tokens and wrote programs to understand this simple yet generative BASIC one-liner. Bringing our diverse disciplinary backgrounds to the interpretation, we examined the code through the lenses of art, sociology, anthropology, as well as creative computing. That endeavor was probably the clearest early demonstration of the fruitfulness of extended code exegesis.

What is code exegesis? I recently had a conversation with some seasoned computer programmers with whom I'm analyzing the code to Joseph Weizenbaum's ELIZA system on which ran one of the most discussed programs, DOCTOR. Published in 1966, DOCTOR (which is often conflated with ELIZA) plays the part of a Rogerian psychotherapist in conversational exchange.[13] During our collective live code reading, we trudged line-by-line through the code written in a fairly niche language named MAD-SLIP. As we read on, one of the programmers, who possessed a deep knowledge of the MAD-SLIP code asked, "What are you trying to note?" In other words, what were we hunting for? I replied that we were engaged in exegesis, which I glossed as to walk out and about the code, starting, of course, from what we might call its denotative meaning and moving out to the connotative meaning. Such terms might be seen as basic practice for those in the interpretive arts, but for a programmer whose chief responsibility is to develop an almost intuitive fluency with the denotative meanings, so the process of signification becomes almost transparent, such a venture was not only unusual but almost antithetical to their daily encounter with code. If one stopped to

12 Nick Montfort et al. (ed.): *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. Cambridge MA: The MIT Press 2013.

13 Joseph Weizenbaum: ELIZA – a Computer Program for the Study of Natural Language Communication Between Man and Machine. In: *Commun. ACM* 9/1 (1966), 36–45. *ACM Digital Library*, https://doi.org/10.1145/365153.365168.

consider the significance of every token they used, they would not get very far. To partake in a leisurely exegesis would seem antithetical and impractical. Where might such an impractical stroll lead?

Our discoveries in ELIZA are still in their early stages, but allow me to offer two initial findings. One part of the code turns ELIZA into an editor of scripts. The CHANGE method, which Weizenbaum alludes to in his article, extends ELIZA from a platform for having conversations with a bot to a platform for changing the bot's script (for example, the DOCTOR script). Programming becomes conversational through this code. Another discovery led us to conclude that the ELIZA code Weizenbaum describes in the article is not the code that he used to generate the oft-quoted dialogue with DOCTOR. Additional discoveries will be forthcoming in our book manuscript, but I do want to mention that pouring through the ELIZA and DOCTOR code with this team of experts, with very diverse backgrounds and intellectual approaches, an exploration that began during a CCS Working Group, has once again reinforced my sense that critical code studies is best done in groups.

2. Mermaids and Pits

Critical code studies may have most to offer when supplementing other approaches. Jessica Pressman, Jeremy Douglass, and I demonstrated that in our book *Reading Project*, a collaborative exploration of one digital object.[14] In that collective reading, critical code studies proved to be a useful tool to complement the other approaches such as close reading the text, historical research into the references of the art work, and visualizations drawing on the methods of cultural analytics. First, our exploration of the code of this distracting flashing piece helped us find the entirety of the story in the form of a very readable text file. However, in the same file, we found the series of subliminal words that flash

14 Jessica Pressman/Mark C. Marino/Jeremy Douglass: *Reading Project: A Collaborative Analysis of William Poundstone's Project for Tachistoscope {Bottomless Pit}*. Iowa: University of Iowa Press 2015.

upon the screen in a variable named "spam," which transformed how we interpreted them from the subliminal to the suboptimal, not that which alludes our notice, but that which tries to slip past our filters. As we examined the ActionScript code further we likewise discovered that the call to these show these spam words alternate with every story word and actually precede the call to the story words, giving them priority, showing their centrality or even primacy in the piece. Such explorations show not only the fruitfulness of CCS in explorations of works of art, but also how combining CCS with other practices can lead to stronger overall readings of works.

We demonstrated the usefulness of CCS again in our collective reading of *FISHNETSTOCKINGS*, an immersive work created by Joellyn Rock and a team of collaborators.[15] In "Entanglements," co-written with Pressman and Diana Leong, we dove into the work looking at the legacy of mermaids, the aesthetics of silhouette art (featured throughout the piece), and of course the nature of the code, which led us not so much to fish as birds, since the flocking algorithm of the mermaids drew upon the code for bird flocking. As in our exploration of Poundstone's piece in *Reading Project*, this collaborative reading was enriched by the exploration of code, which showed us not only the way the artwork functioned but also revealed notes on its development and thematic tropes in names of objects and processes, revealing and resonating with themes in the work, such as hybridity. CCS does not have to be the sole focus in a reading, but now, as these methods have become more developed, they can become part of the toolset of the reader of digital objects in the digital humanities.

To read code is not merely to examine aesthetic objects. In Critical Code Studies Working Groups and in other collaborations, we have examined the Apollo Lunar Lander Code only to find "Burn Baby Burn" in the comments, a reference that tied the extraterrestrial moon shot to the very Earth-bound race riots of the 1960s in Watts, California. A group

15 Marc C. Marino/Diana Leong/Jessica Pressman: Entanglements. In: *The Digital Review* 2 (2022): https://thedigitalreview.com/issue02/marino_entanglements/index.html (accessed September 24, 2023).

based out of the Humanities and Critical Code Studies Lab, including Jeremy Douglass, Sarah Ciston, and Zach Mann and I have examined the code for predictive policing software to trace how it reinscribes racial and ethnic bias. Our work on racial justice, follows in the footsteps of Safiya Noble,[16] Ruha Benjamin,[17] and Joy Buolomwini (Algorithmic Justice League) as we formed an Anti-Racist Critical Code Studies Reading Group.

Recently, we have collected and published a group of early scholars in two special issues of *Digital Humanities Quarterly*. The first set of essays involve close readings of code for Alexa,[18] an early BASIC game FTBALL,[19] two artworks by Daniel Howe,[20] and another piece from the Ryan Veeder competition, a sonnet written in Inform 7.[21] Scholars also attempt to apply CCS methods to artificial intelligence as in the work of Rita Raley and Minh Hua[22] and of David Berry.[23] Examining the edges of pro-

16 Safiya Umoja Noble: *Algorithms of Oppression: How Search Engines Reinforce Racism*. New York: NYU Press 2018.

17 Ruha Benjamin: *Race after Technology: Abolitionist Tools for the New Jim Code*. Cambridge: Polity 2019.

18 Lai-Tze Fan: Reverse Engineering the Gendered Design of Amazon's Alexa: Methods in Testing Closed-Source Code in Grey and Black Box Systems. In: *Digital Humanities Quarterly* 17/2 (2023).

19 Annette Vee: BASIC FTBALL and Computer Programming for All. In: *Digital Humanities Quarterly* 17/2 (2023).

20 John Cayley: Computational Art Explorations of Linguistic Possibility Spaces: Comparative Translingual Close Readings of Daniel C. Howe's Automatype and Radical of the Vertical Heart 忄 . In: *Digital Humanities Quarterly* 17/2 (2023).

21 Jason Boyd: Poetry as Code as Interactive Fiction: Engaging Multiple Text-Based Literacies in Scarlet Portrait Parlor. In: *Digital Humanities Quarterly* 17/2 (2023).

22 Minh Hua/Rita Raley: How to Do Things with Deep Learning Code. In: *Digital Humanities Quarterly* 17/2 (2023).

23 David M. Berry: Tracing Toxicity Through Code: Towards a Method of Explainability and Interpretability in Software. In: *Digital Humanities Quarterly* 17/2 (2023).

gramming, Daniel Temkin takes up esoteric programming languages[24] and Bary Rountree and William Condee examine the use of "nonsense code" to create serious hardware effects.[25] Probably the most paradigm shifting work, however, is Jon Corbett's specification of the Cree# programming language, drawing on the culture and language of the indigenous Metis people.[26] The second set of essays includes works that develop the scope of CCS to examine the state of the code[27] and naming strategies,[28] historical work on the contributions of Ada Lovelace,[29] pedagogical concerns[30] and work on the first game console.[31] The collection also includes new approaches to reading code, including applications of

24 Daniel Temkin: The Less Humble Programmer. In: *Digital Humanities Quarterly* 17/2 (2023).

25 Rountree, Barry, and William Condee. "Nonsense Code: A Nonmaterial Performance." *Digital Humanities Quarterly*, vol. 017, no. 2, July 2023.

26 Jon Corbett ᐊᒪᐢᑭᐯᐢᐃᑲᓇ ᒪᒋᑐᓂᔨᐢᐃᑎᑲᓂᐢᑲᓂˣ | Acahkipehikana Mâmitoneyihicikanihkânihk | Programming with Cree# and Ancestral Code: Nehiyawewin Spirit Markings in an Artificial Brain. In: *Digital Humanities Quarterly* 17/2 (2023).

27 Evan Buswell: Contextless Text, Textless Context -- Programming Languages and Knowledge in the Age of Machine Learning. In: *Digital Humanities Quarterly* 17/3 (2023).

28 Kevin Brock: Practices of Naming in Code as Rhetorical Strategies. In: *Digital Humanities Quarterly* 17/3 (2023).

29 Zachary Mann: Reading Note G: Ada Lovelace and the Secretarial Labor of Codework. In: *Digital Humanities Quarterly* 17/3 (2023).

30 Briana Bettin: On the Polysemy of Programming Language Syntax in Java. In: *Digital Humanities Quarterly* 17/3 (2023).

31 Zachary Horton: Beyond Input: Decoding the First Video Game Console. In: *Digital Humanities Quarterly* 17/3 (2023).

NLP and Graph Theory,[32] affect theory and sentiment analysis,[33] defactoring[34] and even reading code aloud.[35]

The scholarship on code has also looked for new connections, epitomized by the new community Knit&Perl. Since the early days of critical code studies, we noticed a connection between coding and stitchcraft. It wasn't just the number of programmers who also engaged in some form of fibre art, such as sewing or crocheting, but in the nature of the two practices. Both involved iteration, emergent design patterns, and processes of creation. While the same might be said of many practices, the grid-like format of many stitching-related activities had so much in common with the grid of the computer screen, it was hard to ignore. Early on in our examination of 10 PRINT, I asked who else might have the knowledge of the kind of emergent pattern-making of this algorithm. The answer: textile workers. In primers on sewing, I found the algorithmic instructions that produced similar patterns, only minus the randomness.[36] In conversation with prominent digital humanists who also sewed, Anne Sullivan, Anastasia Salter, and I decided to form a community to discuss the intersections between stitchcraft and coding. So was born Knit&Perl, an online community dedicated to the discussion of programming and stitchcraft.

Much of the community has appeared in the Critical Code Studies Working Group, two special issues of *Digital Humanities Quarterly*, and strings of articles in *electronic book review*. These published readings present only the most preliminary of findings and demonstrations in a

32 Chris Tanasescu/Raluca Tanasescu: NLP and Graph-Theory-Based Coding for Text (Corpus) Analysis. A Comparative Poetry and Philosophy Implementation Case Study. In: *Digital Humanities Quarterly* 17/3 (2023).

33 Jeffrey Moro: "Machine Reading for Atmosphere: Affect Theory and Sentiment Analysis in TextBlob. In: *Digital Humanities Quarterly* 17/3 (2023).

34 Matthew Burton/Joris Van Zundert: Defactoring the Pace of Change. In: *Digital Humanities Quarterly* 17/3 (2023).

35 Mace Ojala/Katrine Kjær: Reading Code Aloud. In: *Digital Humanities Quarterly* 17/3 (2023).

36 Mark C. Marino: The ppg256 Perl Primer: The Poetry of Techneculture. In: *Emerging Language Practices* 1 (2012).

field too new to have fixed methods. They are developed in every new code reading.

> "The end of 'The Gay Science' offers three plaintive lines in the comments:
>
> [I hope you can read this.]
>
> [I hope this works.]
>
> [I hope.]"

Every programmer can relate to the hope that their code functions properly; however, the fictional programming persona writing the code for The Gay Science seems to hang their hopes on a reader who receives the hidden message placed inside the code. The first years of critical code studies suggest that there are many messages to be gleaned, messages full of meaning that we discover and create by looking deeply into the source code. In code, we find intercepted messages to a machine that have even more meaning for us.