DHQ: Digital Humanities Quarterly
2023
Volume 17 Number 2

# “Any Means Necessary to Refuse Erasure by Algorithm:” Lillian-Yvonne Bertram’s Travesty Generator

Zach Whalen <zwhalen_at_umw_dot_edu>, University of Mary Washington

## Abstract

Lillian-Yvonne Bertram's 2019 book of poetry is titled *Travesty Generator* in reference to Hugh Kenner and Joseph O'Rourke's Pascal program to “fabricate pseudo-text” by producing text such that each n-length string of characters in the output occurs at the same frequency as in the source text. Whereas for Kenner and O'Rourke, labeling their work a “travesty” is a hyperbolic tease or a literary burlesque, for Bertram, the travesty is the political reality of racism in America. For each of the works *Travesty Generator*, Bertram uses the generators of computer poetry to critique, resist, and replace narratives of oppression and to make explicit and specific what is elsewhere algorithmically insidious and ambivalent.

In “Counternarratives”, Bertram presents sentences, fragments, and ellipses that begin ambiguously but gradually resolve point clearly to the moment of Trayvon Martin's killing. The poem that opens the book, “three_last_words”, is at a functional level a near-echo of the program in Nick Montfort's “I AM THAT I AM”, which is itself a version or adaptation of Brion Gysin's permutation poem of the same title. But Bertram’s poem has one important functional difference in that Bertram's version retains and concatenates the entire working result. With this modification, the memory required to produce all permutations of the phrase, “I can’t breathe”, is sufficiently greater than the storage available most computers, so the poem will end in a crashed runtime or a frozen computer--metaphorically reenacting and memorializing Eric Garner’s death. Lillian-Yvonne Bertram's *Travesty Generator* is a challenging, haunting, and important achievement of computational literature, and in this essay, I expand my reading of this book to dig more broadly and deeply into how specific poems work to better appreciate the collection's contribution to the field of digital poetry.

1

```
A NATION OF SPECIFIC CHAINS
                ON SHIPS CROSSING AN OCEAN
        SUMMONING
                  ANY MEANS NECESSARY
TO REFUSE ERASURE BY ALGORITHM
A NATION OF REGULATED FRAY
                RIOTING
        DIGGING UP
                  ANY MEANS NECESSARY
TO REFUSE ERASURE BY ALGORITHM
A NATION OF OLD DISPOSSESSIONS
                IN THE SHIP’S HOLD
        UNEARTHING
                  ANY MEANS NECESSARY
TO REFUSE ERASURE BY ALGORITHM
```

[Bertram 2019a]

The passage quoted here is from Lillian-Yvonne Bertram’s poem, “A NEW SERMON ON THE WARPLAND”, which is included in their 2019 collection, *Travesty Generator*. All told, the poem includes 26 of these cinquains, and because the poem bears the subtitle “a poem by algorithm,” an attentive reader can deduce the function of that implied code. One can infer that it makes use of a template similar to the following: 2

> A NATION OF <NOUN> <PREPOSITION> <VERB> ANY MEANS NECESSARY TO REFUSE ERASURE BY ALGORITHM 3

Bertram notes that some of the words in this poem come from Gwendolyn Brooks “sermon on the warpland”. In addition, the cinquain template and the lists that supply words into its frame is based on *A HOUSE OF DUST* by Alison Knowles and James Tenney. And yet, even though this poem is oriented by these two contingencies, it is still Bertram's voice and vision that assembles these words into ideas and their meanings. The lineage that informs this poem is an important part of its meaning, and the software that creates the text is not mere technology but an ideologically-committed framework that operates on legacies of systemic racism: algorithms that erase; specific chains. 4

Critical Code Studies is a practical method of inquiry that applies literary and critical hermeneutics to computer code. In many cases, as in reading the source code of an application or a webpage, one may be tempted to read computational media and programmed objects as inherently dual -- a legible, playable, visible surface and the normally invisible code that directs the machine to create that surface. Mark Marino and others have disrupted the neat division of surfaces and the depths they obscure by turning a critical, hermeneutial lens onto code as yet another textual surface where meaning is inscribed and can be interpreted. As Marino discusses at length, this proposition is not necessarily intuitive, nor is it without its critics. [Marino 2020] Code, after all, must be read both to the software that compiles it and the human that shares it, but clearly these are two different meanings of "reading." Once a compiler transforms the text of a computer script into actionable assembly code, the idiomatic vaguaries of idiosyncratic, human reading can have no effect on the executation of that process. This gap between what a human means and what a computer understands is the challenge that novice programmers learn to struggle against: just because a line of code makes sense to me does not mean that the computer will do with it what I think it will. 5

Because so many areas of our world impacted by or accessed through programmed interfaces, opportunities abound for interpretations of code to bring critical attention to latent power dynamics and cultural assumptions. Because of this, much of the published work following a critical code studies orientation reads code in spite of what it does. For instance, Mark Sample discussing how offensive mysoginistic language discovered in a debug file for the videogame *Dead Island* reveals mysoginistic assumptions baked into the games design [Sample 2011]; Winnie Soon and Eric Snodgrass disentangling the social and political relations implied by the design of application programming interfaces (APIs) [Snodgrass and Soon 2019]; Outi Laiti explaining how programming languages based on languages other than English helps to foreground and critique the hegemony of English within computing culture. [Laiti 2016] 6

In examples of poetry generated by code, poetry written as code, and code that generates poetry, it is less useful to think of a tension between the code as text and the code as preamble to its processes and more productive to consider the imbrication of those surfaces. John Cayley's influential essay, “The code is not the Text (unless it is the text)” [Cayley 2002] disentangles two incommensurable ways of understanding language. As Cayley insists in a later essay reviewing Nick Montfort's book of computer poetry, *#!*, “a potential for human reading...is required to produce an event of language” [Cayley 2015]. And for works where the code is finite and legible but what the code generates is practically or literally infinite (or infinitesimally brief), the code *must* be the poem; that is the only option. Herein lies the counterintuitive aporia for critical code studies when code poetry is the object of critique: whereas a close reading of a videogame's source code offers a peak behind the curtain, reading a poem's source code requires a more subtle recalibration of hermeneutic attention. Either way, as Marino observes, “Regardless of where one sees the poetry, the procedure or the possibility, both exist, in hibernation or as seeds, in the code.” [Marino 2020] 7

For poet Lillian-Yvonne Bertram, those seeds of possible meaning saturate all available surfaces, and whereas code poetry of the sort practiced by Montfort and Cayley can be esoteric and conceptual, the ideas in Bertram's poems are familiar, situated, and ideologically specific. In other words, Bertram's poems in *Travesty Generator* are already doing for 8 digital poetry what critical code studies seeks for everyday systems and software when that software is bound up with regimes of oppression. My goal in this essay is to explore how Lillian-Yvonne Bertram uses code in their poetry to perform that critique, specifically by bearing witness to the pain and violence inflicted on Black Americans.

In writing about the way Bertram's poems address these topics with specificity and familiarity, I am aware that I approach this with the benefit of several privileges: I am a white, able-bodied, cis-gendered man. I am a tenured faculty member. These privileges necessarily mean that the experiences remembered in some of these poems -- some of which are traumatic -- will carry deeper and more personal meanings to those still oppressed and disproportionately harmed by police violence. Bertram, who identifies as biracial African-American, does not need my interpretation of their work for that work to be haunting, resonant, and beautiful, and the recognition this work has received indicates how well it speaks for itself. In this essay, I hope to use my voice and privilege to bring attention these poems and invite others to appreciate their nuances, complexities, and their lineage within digital poetics. 9

Computer code executed in the memory of a machine is characterized by its speed, ephemerality, and volatility, and digital poetry involving combinatorics can easily create a scenario such that any individual poem has a vanishingly small probability of existing. Thus, a poet’s decision to inscribe the output of a computer program into the pages of a book is an act of curation as much as composition. To select one result among many, or to capture a system at the point of its failing to continue, are as essential to digital poetics as are the tasks of defining a frame and creating the database of language to fit into that frame. This characterization of digital poetry consisting of a “frame” is a formulation Carole Spearin McCauley defined in her 1974 book *Computers and Creativity* [McCauley 1974], and it still illuminates contemporary digital poetry. Although poets may innovate in the types of frames to create and in the complexity of dictionaries to mine for language, the agenda of contemporary digital poetry is diminished if its meaning is circumscribed by mere novelty. 10

Lillian-Yvone Bertram’s book of poetry, *Travesty Generator*, is among the most salient and compelling recent works of digital poetry because its poems address the contemporary realities of racism through the situated specificity of a computer generating poetry and the material specificity of print. In this essay, I offer close readings of several poems from this award-winning collection [Dennigan 2020] to emphasize Bertram’s contribution to digital poetics and explicate the ways in which their poems operate in, on and against other systems of code and code poetry. 11

The book bears the title *Travesty Generator* in reference to Hugh Kenner and Joseph O'Rourke's Pascal program designed to “fabricate pseudo-text” [Kenner and O'Rourke 1984] by printing out language such that each n-length string of characters in the output occurs at the same frequency as it does in the source text. According to its writers, by combining different authors through this algorithm, one might stumble across a travesty of “haunting plausibility” wherein James Joyce's writing becomes muddled with Henry James's. [Kenner and O'Rourke 1984] The pseudo-text's nonsensibility is simultaneously its act of transgression and its literary horizon, but for Kenner and O'Rourke’s program and its many progeny, those possibilities extend no farther than a novelty, a formal curiosity or at best an insight into a specific author's writing style. 12

When Charles O. Hartman uses Travesty in his *Virtual Muse*, he finds that the nonsense it generates helps Hartman's composition process by disrupting the habits he found himself following as a writer. Of the generator itself, Hartman noted with some awe that “here is language creating itself out of nothing, out of mere statistical noise” [Hartman 1996]. 13

Bertram’s title invokes the legacy of this program in the history of electronic literature, but more importantly, the name, “travesty generator,” invokes other resonances that raise the stakes in computer-generated poetry. Whereas for Kenner and O'Rourke, labeling their work a “travesty” is a hyperbolic tease, a literary burlesque, for Bertram, the word is invigorated with its political reality. In 2013, attorney Don West, responding to the not guilty verdict for his client George Zimmerman in the shooting death of Trayvon Martin, congratulated the jury for keeping “this tragedy from becoming a travesty” [BBC 2013]. West is implying that the media attention on the case risked creating what he and Zimmerman would have called a farcical miscarriage of justice. 14

## “Counternarratives”

For others, perhaps most observers, the event was always a travesty: a “grotesque parody or imitation” [OED 1989] of criminal justice enacted by a civilian playing at being a law enforcement officer that resulted in the extra judicial killing of an unarmed high-school student. 15

Central to any response to that verdict is how one understands the narrative of events on February 26, 2012, and this is the question Bertam takes up in their poem “Counternarratives” which presents sentences, fragments, and ellipses beginning ambiguously but gradually hinting at a series of specific implications and images. These narratives are “counter” for several reasons, most importantly for the fact that they include Martin's point of view. Each stanza presents several sentences or fragments separated by ellipses, and with each page more of these phrases accumulate to present a gradually more complete image of what happened on February 26, 2012. 16

> [4] … He never told anyone, but he always wanted to go to space camp.
> [5] … He rides from station to station until he can rest at home.
> [6] … Sometimes he wakes feeling gone and doesn't know why. [Bertram 2019a] 17

The poem includes these sentences with others in 14 successive stanzas, each printed on its own face of a page, each growing in size until the blank 13th stanza breaks the pattern of expansion. One can see the inertia of automation and the inevitability of processes at work in the consistency of this pattern and in the way the poem intersects other perspectives that include the algorithmic prompts of search engines (e.g. “People also ask: what really happened?”) and the horticultural (e.g. “Only the flowering catalpa trees are on watch”) -- all echoes of the machinic “generator” in the book's title. But because this is literally generated from computer code, the “counter” in the poem’s title works in at least two other significant ways. 18

In Nick Montfort’s 2014 collection, *#!*, Montfort includes the source code of each poem alongside a representation of a poem produced by running that code.[Montfort 2014a] For Montfort’s poems, the poetics of the code is as much (and sometimes more than) any semantic meaning in the resulting output it generates, and his terse coding style, usually presented without comments, can be admired for its elegance as well as used to validate Montfort’s work. Harkening back to the days when programmers like Kenner and O’Rourke’s shared programs like Travesty Generator for Micros by publishing them in print magazines, Montfort’s poems invite readers to type them into an interpreter and see what happens. 19

Bertram does not include the source code for “Counternarratives” alongside its text, but in an appendix they acknowledge that the poem is adapted from the Python version of Montfort's poem, "Through The Park," published in *#!* and also available on his website[Montfort 2008]. By analyzing how Montfort’s poem operates, we can speculate about Bertram’s compositional and computational processes and thus explore how it situates Trayvon Martin’s death. 20

Montfort's poem is framed by a `for` loop, or a “counter”, `i`, iterating through 8 sequences, each of which prints a numbered stanza: 21

```
for i in range(8):
```

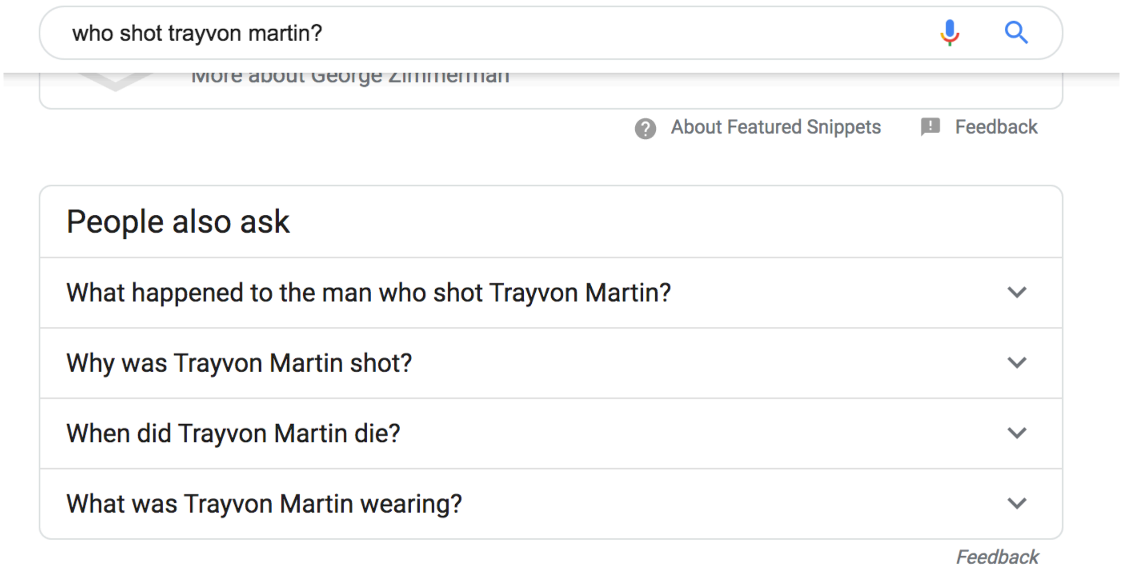
For each loop and each stanza, the program selects a number between 7–11 (`phrases = 7 + random.randint(0,4)`) and randomly deletes lines from the initial 25 until the lines that remain reach the selected number of `phrases`, joining those remaining lines with ellipses. The resulting poems achieve their meaning through omission, elision, and innuendo, relying on the ambiguity of language and the way readers respond to that ambiguity by imagining a context that creates the poem's implied narrative. What that narrative is will depend on which lines end up being printed, and in *#!*, the 8 stanzas that Montfort selects show how widely those contexts may diverge. These could range from a casually flirtatious encounter with a stranger to a sexual assault. The latter reading is activated by whether two or three key phrases remain in the resulting poem. The suggestive phrases “The girl puts on a slutty dress”, “The man makes a fist behind his back”, “The man's breathing quickens”, and “The man dashes, leaving pretense behind” makes “The girl's bag lies open” metonymic and lets the question of what their glances know (“The man and girl exchange a knowing glance”) hinge the meaning of the moment toward violence.[Montfort 2008] 22

> The girl puts on a slutty dress. ... A wolf whistle sounds. ... The muscular man paces the girl. ... A wildflower nods, tightly gripped. ... Laughter booms. ... A lamp above fails to come on. ... The girl's bag lies open. ... A patrol car's siren chirps. ... 23

This violence is implied, but generic, and the poem's meaning is about the way language gives shape to violence. The victim blaming in the "slutty dress" line, for example, could be Montfort reminding us of the ways that the framing of language creates an inertia of belief or predisposition that precedes and undermines knowing. Ultimately, however, this implication about language appears to be all that is at stake for the poem. 24

Bertram's "Counternarratives" is more than an adaptation of Montfort's poem because it is also a counter to "Through the Park". Whereas the context for Montfort's poem is hypothetical, Bertram's is real and specific. While we cannot consult Bertam's source code, it is possible to imagine the process whereby Montfort's "A wildflower nods, tightly gripped" becomes Bertram's "The frangipani swans in the moonlight" and Montfort's "She puts on a slutty dress" becomes "People also ask what he was wearing" or perhaps "No mention of his clothing". The code for "Counternarratives" may not be visible, but the source is eminently and tragically knowable.[1] 25

What Trayvon Martin wore -- a hoodie -- became a symbol for that event and a meme for Black Lives Matter because of its power in calling out racial profiling. The assumption that "a black man with a hoodie" is inherently threatening is conveyed in Bertram's "People also ask..." phrasing, which quotes a google search suggestion. In the context created by this poem, this question recalls the victim blaming in Montfort's poem's "slutty dress" line, but following Bertram's move from the generic to the particular: this search suggestion and others like it are drawn from reality. 26


who shot trayvon martin?

More about George Zimmerman

About Featured Snippets Feedback

People also ask

What happened to the man who shot Trayvon Martin?

Why was Trayvon Martin shot?

When did Trayvon Martin die?

What was Trayvon Martin wearing?

Feedback

**Figure 1.** A screenshot of a Google search and suggestions for related searches.

After a few trials, I discovered the suggestion "What was Trayvon Martin wearing?" in response to the search query "Who shot Trayvon Martin?" (see Figure 1). I also found another recurring line, and the final line in Bertram's poem, "People also search for: Emmett Till" came up in several similar searches. Safiya Noble's 2018 *Algorithms of Opression* studies the ways that search engines are among the systems of oppression that commodify black male criminality and sustain social, political, and racial tension in America. In a 2014 article, Noble contrasts two sets of Google's autocomplete engine. "Trayvon Martin was..." completes with phrases like "a thug", "no angel", and "a drug dealer"; "George Zimmerman is ..." completes with phrases like "a hero", "innocent", and "not white." [Noble 2014] As Noble argues, the ideas that Google suggests conform to a normative viewpoint on the violent events of that night actually extend from media narratives developed and promoted as part of the media spectacle that emerged during and after the trial.[Noble 2014] Notably, Bertram's implied query is different from: using the pronoun "he" in place of Martin's name, so the antecedent in "what he was wearing" could be Zimmerman, especially since neither are named outright in the poem. 27

This final opening out of meaning is another significant way in which the poem reflects on its constructedness because it goes a step further than Montfort's implied invitation to run his code. Instead, Bertram invites us to speculate about their implied code and simultaneously confronts us with the opacity of Google's algorithmic suggestions.

28 In *Travesty Generator*'s appendix, Bertram notes that the output of their program has been edited and arranged, and this rewriting may be evident in the way that certain of the phrases evolve as the poem progresses [Bertram 2019a]. In some cases, Bertram's revisions evolve to follow a trajectory implied to begin in Montfort's version with Montfort's phrase, "A patrol car's siren chirps." [Montfort 2008]

29 Bertram's correlating lines (though never the last for any of their 14 stanzas) evolves from there into

> [3] ... A patrol car's siren sings several streets away...
> [10] ... Several weeks away, a patrol siren sings...
> [11] ... A patrol car's siren sings several streets way...
> [12] ... A patrol car's siren swats bugs and halos away...
> [14] ... A siren signs
> several streets away.
> [Bertram 2019a]

30 The most striking movement in the poem is that way that "Counternarratives" uses ellipses. Whereas Montfort's poem is suggestive through the innuendo created by stochastic omission, Bertram's is subversive through the implications of its elisions, and those elisions and ellipses gradually resolve into prosodic syntax as the text of the poem gradually replaces its poetic mechanism. The phrase that first appears in stanza 4 as "Real gaps spread in the tropic of paradise" is stark in its proximity with its initial following line "Forty-two miles from Disney" [Bertram 2019a], where proximity here is both the poetic associations as well as the geopolitical identity of this Orlando suburb.

31 After the blank penultimate stanza -- a final elision bearing the number 13 -- this line has become "Gaps split open the tropic of paradise". A blank line splits the "siren signs" sentence, followed by the now damningly unambiguous "Cause of death: It was a gated community" [Bertram 2019a]

32 By editing and arranging the output of their program, Bertram is declining to let the machinery of language or Python control the narrative and opening up paths to resistance by insisting that we bear witness to the tragedy without retreating to the algorithmic distance of a travesty. Two other poems in *Travesty Generator* offer similar insights on the relationship between the mechanization of computing and the use those processes to interrogate the structures that make them possible.

## "three_last_words"

33 The poem that opens the book, "three_last_words", is at a functional level a restatement of the program in Nick Montfort's "I AM THAT I AM", which is itself a version or adaptation of Brion Gysin's permutation poem of the same title. That phrase does not appear in Montfort's version. Rather, Montfort's program defines a `permutations` function and then executes that function with the string `'AEIOU'` as its argument, yielding the 120 possible rearrangements of AEIOU as its outcome. Montfort's code accomplishes this succinctly through an elegant recursion as the `permutations` generator works through a list of `elements`, rearranging all of the following elements (`elements[1:]`) by passing them back until no more next `elements` remain.

```
"I AM THAT I AM" (#! 18)
def permutations(elements):
 if len(elements) == 0:
   yield elements
 else:
   for result in permutations(elements[1:]):
```

```
            for i in range(len(elements)):
              yield result[:i] + elements[0:1] + result[i:]
```

[Montfort 2014a]

Bertram's generator is mostly the same code but the commentary they add -- and more importantly the uses they put it to -- changes the meaning entirely. That new meaning is jarring and, as in “Counternarratives”, tragic. The three last words in reference here are Eric Garner’s and, more recently, George Floyd’s: “I can’t breathe.” Montfort’s permutations create a beginning; Bertram’s memorialize endings. 34

Bertram’s poem creates meaning at least two distinct levels. It can be read first as lines of poetry and second it can be executed as a program that generates poetry. The typographic presentation of the poem blends those two levels by alternating lines of code with comments and statements that execute the `permutations()` function defined at the outset -- statements that accumulate toward something unavoidable. In the excerpt below, lines preceded with the character, “#”, are comments and not processed as code. 35

```
def permutations(elements):
#the
 if len(elements) == 0:
#the knife
   yield elements
  #the knife they
 else:
#the knife they hung
```

[Bertram 2019a]

The sequence in comments reads, “`the`”, “`the knife`”. “`the knife they`”, “`the knife they hung`”, adding one word at a time over the course of the code until the completed comment observes, “`#the knife they hung him on / #was a legal trinket`”[Bertram 2019a]. When he was killed by Daniel Pantaleo while being placed under arrest, Eric Garner was selling loose cigarettes -- a minor violation of cigarette tax law. “Legal trinket” here brings to mind the way that Garner's alleged crime of selling cigarettes became a meme for racist backlash to Black Lives Matter -- responses that included an Indiana police officer selling t-shirts with the mocking slogan, “Breathe easy, don't break the law” [Ortiz 2014]. Bertram’s comment characterizes this as a “trinket” because the relatively trivial infraction was used to justify police violence. 36

The word "trinket" does something else here as well, perhaps coincidentally, by naming a service, Trinket.io, that allows users to create snippets of Python code that they can embed and run in a webpage. Intentionally or not, this possible reference hints at a specific, practical, and materially-situated environment for running this code, which is something that is implied to be unnecessary for Montfort's ontologically denotative and self-evidently tautological symmetry. In other words, the fact that Montfort prints the code with its output precludes a need to execute the code for ourselves. 37

Bertram's poem instead invites us to experience the consequences of their program as a response to its explicit invitation a page later: 38

```
#run the code
   #in this cell
       #away
```

[Bertram 2019a]

poem, the “cell” here may connote a jail cell, but more likely it describes a Python Notebook cell. Python Notebooks, like the embeddable Trinket widgets, are a common way to experiment with Python and to invite others to review, execute, and bear witness to the results of a program. In “three_last_words”, the samples of output and the poem's culminating error message are typographically consistent with what one would see while running this code in a notebook.

Montfort's program runs one permutation and prints the output: 120 variations on the five vowels arranged from “AEIOU” to “UOIEA” in 8 tidy, monospaced columns. Bertram's program runs three times, and whereas Montfort's final line of Python 2 code prints the permutated sequences as a string (`print''.join(list(permutations('AEIOU')))`[Montfort 2014a]), Bertram's Python 3 simply prints the list as it is represented in Python's memory. The first iteration permutes a single-character: 39

```
print (list(permutations("I")))

['I']
```

[Bertram 2019a]

In addition to this invitation to experience the practical implementation of the program, this difference in printing the yield of the `permutations()` function pulls us toward the subjective point-of-view of the Python runtime that experiences the code, and the chaotic typography of the output -- in contrast to Montfort's neat rows and columns -- performs the messy complications of computation that will eventually culminate in the computer’s failure.

There is one more subtle variation between Montfort and Bertram's code in the final line of the constructor. As Python iterates through the `elements` list (a string of text characters, in this case) the newly-permutated line `yields` a + `result[i:]` for Montfort or just + `result` for Bertram. 40

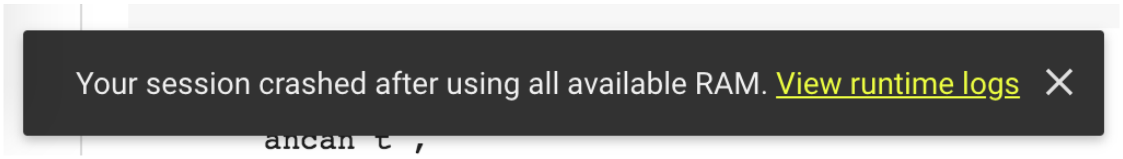
The functional difference is that Montfort's line restores to the target string only those parts of the input string that come after the character operating for the current iteration. The `[i:]` slice captures only those elements after `i`. Bertram's version retains and concatenates the entire working result, so the practical difference is that while Bertram's code will generate the same number of permutations, the strings it generates gradually become longer. The last element in the `permutations()` of the string, “`can't" is "t'nact'nat'nt't`”. Whereas the 120 permutations of “AEIOU” are a complete and homogenous set, `permutations("can't")` is asymmetrical and grows geometrically. The articulation of 41

```
print(list(permutations("breathe")))
```

[Bertram 2019a]

runs past the right and bottom edges of the page.

The asymmetry culminates finally and dramatically when the full phrase "I can't breathe" is passed in to the `permutations()` function. Bertram's code concludes with the printout out of a Python MemoryError because the number of possible sequences of the 15 letters in that phrase (1,307,674,368,000) exceeds the available memory on the computer hosting the Python runtime, and like the 15 seconds that Officer Pantaleo held Eric Garner in a chokehold, those 15 characters result in the death of the Python Notebook or the computer hosting it. 42


Your session crashed after using all available RAM. View runtime logs

ancan t ,

**Figure 2.** A screenshot displaying the error message of a Colab Notebook that crashed trying to execute Bertram's final permutation in "three_last_words".

I attempted to run Bertram's code two different ways: first in a Google-hosted Colab Notebook and later in a Jupyter Notebook running on my laptop. The Colab Notebook ran for some time with the RAM usage indicator creeping slowly to the right and becoming first green, then yellow, and finally orange before the runtime crashed and disconnected, as shown in Figure 2. 43

On my Macbook Pro, the results were similar, but as Python continued to take up more and more RAM, my computer gradually stopped working. The cursor slowed down, the trackpad clicked more slowly, and keyboard shortcuts stopped working. I couldn't take a screenshot, so I had to use another device -- my phone -- to capture the final moments before I resorted to a hard reboot and power cycle. 44

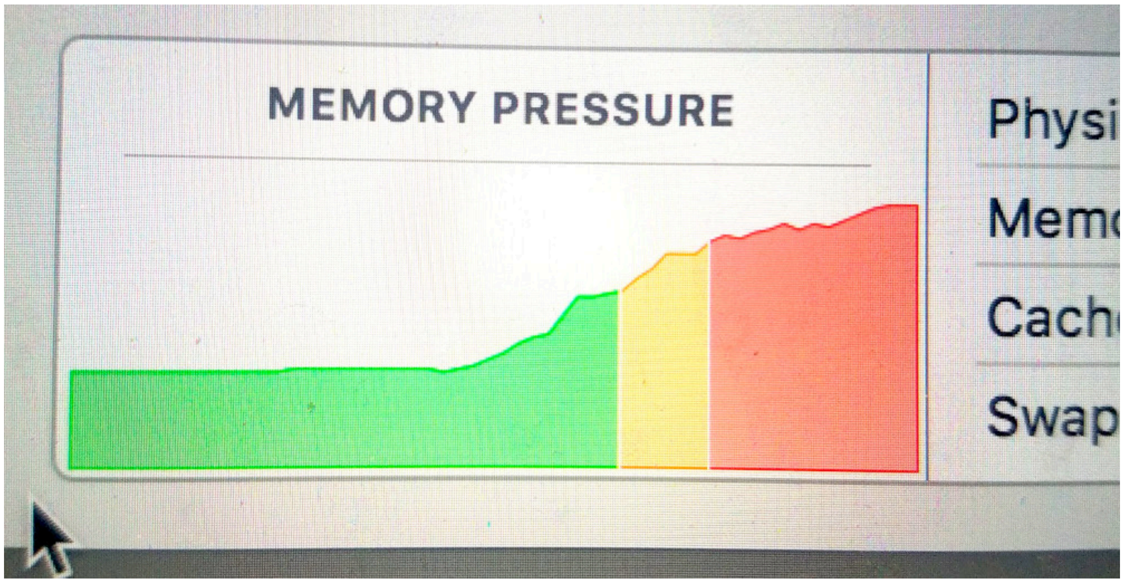


**Figure 3.** A photo of my Macbook Pro's memory pressure monitor as it filled with the final permutations in "three_last_words".

It seems trivial or maudlin to compare a computer running out of memory to the tragedy of Eric Garner's asphyxiation and death, but the way the computer's memory usage creeped inevitably upward conveyed a vivid anxiety and sense of panic in a way that was hauntingly effective. This poem is directly addressing the physical environment of computing in a manner reminiscent of Montfort's poem "Round" [Montfort 2013], or Sam Lavigne's parodic "Slow Hot Computer" project that "lavigne makes your computer run slow, and hot, so you can be less productive" by running "processor-intensive calculations" in a web browser. [Lavigne 2015] 45

This kind of metaleptical irruption of attention -- what Terry Harpold has called "recapture" -- is typically accomplished in the service of play, both in the satiric sense of Lavigne's playfully critical projects and, for Harpold, in both the literal sense of playing videogames and the semiotic sense of expressive freedom within constraints. In Montfort's examples, recapture moderates the conflicts of entanglement when technical limitations are expressed through the terms of the gameworld, as when a text-based game attempts to parse an unknown word of user input and it replies, "I didn't quite understand that", instead of reporting an error code or just crashing. Recapture is a fundamental operation of videogames, which means that "recapture happens during play, in the complex digressions and feedback loops that are activated in actual play". [Harpold 2007] 46

Bertram works with the same entanglement of technical process and expressive semiotics, but poetry is not a videogame. And in this case, the poem denies its reader the recapture of memory, challenging us instead to bear witness to the trauma it symbolizes. 47

As Wendy Hui Kyong Chun has discussed, computer memory is an “enduring ephemeral”, always-already conflating memory with storage and, through metaphors and pretenses of permanence. They note that the instability of this term, memory, follows the volatility of RAM, which is “based on flip-flop circuits and transistors and capacitors and which require a steady electrical current”. [Chun 2011] Like human memory, software “memory is not a static but rather an active process. A memory must be held in order to keep it from moving”. [Chun 2011] Bertram's software presents the volatility of memory as the entanglement of poetic representation, police brutality, and technical limitation, and by setting it up to fail, Bertram invites us to reenact the trauma of that brutality and interpret its technical and political consequences. 48

When Bertram disrupts the symmetry of Montfort's slices (result[:i] and result[i:] becomes result[:i] and result so that "ehtaerb" potentially becomes the drawn-out, Joycean susurration "ehtaerbehtaerehtaeehtaehtehe", that is, `print(list(permutations("breathe"))[-1]))`, but both versions of the generator choke on Eric Garner's three last words. Memory fails, but the poem preserves its failure even as its overflow suggests a world uncontemplated by code. This overflow is not infinite -- hinting that someday, when it becomes possible to run this program with the multiple terabytes of RAM necessary to compute and convey one trillion three-hundred seven billion six-hundred seventy-four million three-hundred sixty-eight thousand lines, the program will complete, but until then we are compelled to follow Bertram's invitation to bear witness and 49

```
#return
   #this articulation
#the exhaustion
   #we can't stop hearing
```

[Bertram 2019a]

## “A NEW SERMON ON THE WARPLAND”

Like “three_last_words” and “Counternarratives”, “A NEW SERMON ON THE WARPLAND: a poem by algorithm” is a work with a genealogy, and Bertram directs us to the source codes that ground it. This is another poem where Bertram gives credit to an example by Nick Montfort, but since his code is a straightforward implementation of Alison Knowles' “A House of Dust”, Knowles is a more appropriate counterpoint to consider for Bertram's work.[Knowles 1968] Additionally and more significantly, Bertram acknowledges “that some words and phrases are taken directly from Gwendolyn Brooks' corpus” in their notes [Bertram 2019a], so understanding those origins allows these phrases to act hypertextually as a bridge into Brooks' poems. One could see “NEW SERMON” as the poem Brooks might have written with access, inspiration and context parallel to Knowles. Bertram encourages us to explore that hypothetical bridge between Brooks and Knowles by recommending Brooks' 1987 collection Blacks, and specifically calling attention to “The Sermon on the Warpland” and “In the Mecca” as especially important sources. Brooks’ “Boy Breaking Glass” and “THE WALL” also seem to be the sources for several phrases.[Brooks 1987] 50

Alison Knowles and James Tenney produced *A House of Dust* in 1968, the same year that Gwendolyn Brooks published “In the Mecca.” The source code for *House of Dust* is not publicly available, but its function is easy to deduce by examining the output published in three different venues. Knowles created the structure and four lists of words and phrases, and Tenney wrote those into FORTRAN code that creates quatrains by selecting and printing one item from each list, indenting each line a few more spaces before returning to left justification as each new quatrain begins. 51

```
A HOUSE OF ROOTS
    AMONG HIGH MOUNTAINS
        USING NATURAL LIGHT
            INHABITED BY VARIOUS BIRDS AND FISHES
A HOUSE OF WOOD
    BY A RIVER
        USING ALL AVAILABLE LIGHTING
            INHABITED BY HORSES AND BIRDS
A HOUSE OF DISCARDED CLOTHING
    UNDERWATER
        USING ELECTRICITY
            INHABITED BY FRENCH AND GERMAN SPEAKING PEOPLE
```

[Knowles 1968]

Bertram credits Montfort's simulation of this poem as the source of the program they manipulated, but around the same time Montfort created his web version [Montfort 2014b], I also created a JavaScript simulation of the poem capable of running in a web browser.[Whalen 2014] The results of each of our implementations are largely the same (save for some typographic and kinetic nuances), but while I named the variables holding each list: `materials`, `places`, `lights` and `inhabitants`. Montfort used the names `material`, `location`, `light_source` and `inhabitants`. The differences in variable names have no impact on the operations of our scripts, but our choices when programming those operations -- my evident preference for plural labels, for example -- might extend from differences in our readings of the poem. Those differences are only possible if one begins by paying attention to the operations of code separately from the inscriptions of that code. 52

Bertram doesn't share the source code for “NEW SERMON” in *Travesty Generator*, but observing its patterns reveals it to have a similar structure of lists randomly sampled. Also, two earlier versions of Bertram's “A NEW SERMON ON THE WARPLAND” are available online, one in JavaScript (originally available at https://ruby-buffet.glitch.me, the source code of this version notes it was based on Laurel Schwulst JavaScript implementation of Montfort's Python version of Knowles and Tenney's FORTRAN) and another in Python 2.[Bertram n.d.] In the Javascript version, each list simply bears a number (`one`,`two`, etc.), but the Python 2 version assigns names that -- like Montfort's and my simulations of *House of Dust* -- signpost the semantic position of each list. There are evidently some differences between the source code shared on Bertram's website and the code that created the poem published in *Travesty Generator*, but the five names suggest their content and reveal Bertram's primary departure from Knowles: `materials`, `locations`, `verb`, `means`, and `outcome`. 53

54

```
A NEW SERMON ON THE WARPLAND (excerpt)
A NATION OF GRIEF
            IN THE CUT
    SUMMONING
                ANY MEANS NECESSARY
TO REFUSE ERASURE BY ALGORITHM
A NATION OF OLD DISPOSSESSIONS
            STILL FIGHTING
    CONJURING
                ANY MEANS NECESSARY
TO REVISE ERASURE BY ALGORITHM
```

[Bertram 2019a]

Comparing *House of Dust* with “NEW SERMON” reveals more differences than the addition of a fifth line. Knowles' locations are diverse but more stable -- “BY THE SEA”, “IN MICHIGAN”, “IN A HOT CLIMATE”, “ON AN ISLAND”,“IN SOUTHERN FRANCE”,“AMONG OTHER HOUSE” -- while Bertram's are active, violent, or disrupted, and some are not conventionally “locations” but more so describe liminal states of being: 55

```
...IN THE CUT...
...STILL FIGHTING...
...IN FRONT OF A WINDOW ABOUT TO BE BROKEN...
...IN TRANSLATION...
...BETWEEN SCYLLA AND CHARYBDIS...
...IN THE SHIP'S HOLD...
```

[Bertram 2019a]

Instead of a house, Bertram addresses their sermon to a “NATION”. Allusions to the slave trade are perhaps most striking in comparing the subtle differences in phrases between Knowles' “ON THE SEA” and Bertram's “ON SHIPS”, but the shifts in Bertram's poem do more than invert the stasis in Knowles'. While *A House of Dust* dwells, a “NEW SERMON” moves, acts, and resists. The verbs and means lists propel each cinquain through the same `means` -- “ANY MEANS NECESSARY” -- to the same `outcome`: “TO REFUSE ERASURE BY THE ALGORITHM”. 56

The Python 2 version includes multiple possible outcomes, but the verbs list is somewhat shorter. Because the printed version of the poem includes the same outcome in each cinquain, it is reasonable to infer that its source code is closer to the Javascript version where list `five` only contains one element: `to refuse erasure by the algorithm`. The `means` list works the same way, demonstrating that the only option is Malcom X's ANY MEANS NECESSARY. 57

For both of these one-choice lists, the program is still selecting that choice as the result of a process. The Python 2 draft uses the common `random.choice()` function, and the JavaScript version takes a custom function with a comment remarking on its conventionality: 58

```
// This is a very common randomizing function.
// It takes a list (array) and returns one at random.
function select_random(x){
   y = x[Math.floor(Math.random()*x.length)];
   return y;
}
```

[Bertram 2019a]

The fact that this program is making a choice with no freedom echos the slavery imagery within the poem, but revisiting *A House of Dust* opens another way of understanding the significance of this choice without a choice. 59

Benjamin Buchloh argues that Knowle's project in creating *House* was to find a way to deconstruct the “prison house of language”. [Buchloh 2012] Noting the irony of an avante-garde artist like Knowles using a house, that most conventional vehicle for subject formation, as their central figure, Buchloh contends that the aleatory construction method of the 60 poem demonstrates the infinite fluidity of the process of subject formation.

Channeling Nietzsche, Buchloh goes on to observe that the constructedness of *House of Dust* resists the subject's being at “home” in language. “Knowles's The House of Dust conceives of the process of subject formation as a perpetual process of construction and undoing, precisely to prevent it from becoming an inhabitant of the prisonhouse of language, a merely substitutional system of fraudulent and aggressive convictions...the formation of of the subject at this point in history has become a more complex and, by necessity, a more open process, since the subject's intersections with language...are certainly no longer the primary ... foundations.” [Buchloh 2012] Buchloh's “convictions” are a play on words, a synonym for "belief" in the mode of the metaphoric correctional system. But it also opens up another angle on "NEW SERMON," which is filled with punishment and captivity but conspicuously without conviction. 61

Although the JavaScript version of the poem does include “of parolees” and “of prisoners” as possible materials, possibly drawn from Brooks' “The Wall”, the printed poem's materials and locations invoke punishment without the framework of judicial justification: 62

> ...A NATION OF SPECIFIC CHAINS... ...IN THE SHIP'S HOLD... ...A NATION OF MEDGAR EVERS... ...IN A STRING-DRAWN BAG... [Bertram 2019a]

In this way, Bertram's poem comes back to the prison house of language that Knowles' is playfully resisting. Captivity and imprisonment are not just metonymic critiques for the post-structuralist decentering of lexical epistemology; instead, the prison house of “NEW SERMON” is the literal kidnapping, enslavement, and genocide of Africans whose exploitation built the foundations of this nation. 63

Finally, by naming one of the lists `verbs`, Bertram follows a schema one often finds in generative works that use context-free grammar: patterns or templates where words are given a syntactical place based on their part of speech. In “NEW SERMON”, the verbs list does the most work to take the poem away from its origins in *House of Dust*. The seven verbs that appear in the 22 printed cinquains all suggest movement with resistance or a sense of bringing something out from below: 64

> DIGGING UP
> SUMMONING
> CONJURING
> DIVINING
> UNEARTHING
> STRIKING MATCHES AGAINST
> HOLLERING DOWN [Bertram 2019a]

Each of these verbs immediately precedes “ANY MEANS NECESSARY” [Bertram 2019a]. So, by omitting the preposition “by” from Malcom X's well-known phrase, the call to action in the poem asks its reader to dig up, to unearth, strike matches against their own means of refusing erasure. 65

In an appendix on the book, Bertram quotes Harryette Mullen's essay, “Imagining the Unimagined Reader”: “When I read words never meant for me, anyone like me ... then I feel simultaneously my exclusion and my inclusion as a literate black women, the unimagined reader of the text”.[Mullen 1999] 66

Bertram likewise considers themself an “unimagined coder”. 67

> I use codes and algorithms in an attempt to create work that reconfigures and challenges oppressive narratives for Black people and to imagine new ones. I consider this an intervention into a set of literary practices that have historically excluded women and minorities. [Bertram 2019a]

For each of the works I have analyzed here and for many others of the 10 poems in *Travesty Generator*, Bertram uses the generators of computer-generated poetry to critique, resist, and replace narratives of oppression and to make explicit and specific what is elsewhere algorithmically insidious and ambivalent. 68

## Conclusion

Writing in a blog entry for the Poetry Foundation, Bertram opens their reflection on Kenneth Goldsmith's controversial performance of "Michael Brown's Body" by qualifying their response through the lens of an individual, bodily experience of race. "I would like to make it clear that my writing about race, like myself, is a collection of incomplete moments in time". [Bertram 2019b] Crucially, Bertram makes a connection here between time and identity. Time is a fundamental characteristic of computing where each event, action or change is associated with a specific "moment[] of time," and most high-level languages include built-in functions to call up that time in various ways. In Perl, `localtime` converts a timestamp into chunks that are more easily processed into a human-readable format. Bertram's poetry in *Travesty Generator* is likewise thinking about race through specific, recent moments of time where the suffering of Black men and women has brought America's identity as a nation built on white supremacy into clearer focus. 69

As I type this paragraph in the year 2021, it is a few weeks after a jury in Minnesota found a white police officer guilty of murdering George Floyd [Xiong et al. 2021] -- a crime recorded on camera that sparked an intense summer of protests in 2020. A few days after that guilty verdict, a young man in my suburban Virginia community was shot 10 times by a sheriff's deputy after calling 911 for help. [Carey 2021] It is nearly six years since Eric Garner's death, nine years since Travyon Martin died, and sixty-five years since Emmett Till was lynched in Mississippi. 70

Bertram's "@Code_Switching" includes the following Perl code (based on Nick Montfort's "PPG256-6"): 71

72

```
#!usr/bin/perl@d=split/_/,
switch_gods_switch_black_codes_you_when_you__where_god_belong_bl
ack_hills_to; {$_=localtime;/(..):(.)(.):(.)(.)/;print"\"x$5."
$d[$1] $d[$2] $d[3] $d[$4] $d[$5] $d[$8]\n";sleep 1; while $d >
0}{print"\"x$5." $d[$3] $d[$6]\n";}}redo
```

[Bertram 2019a]

There is much to unpack here, but briefly, a pattern match filter captures integers from the `localtime` string and uses those to select specific words from an underscore-separated list of words. Each line of the poem is therefore determined by the current system time, which Bertram invites us to consider by helpfully including the initial `localtime` for the three runs of the poem printed in the book. In other words, the poem is not randomly generated, but is instead a product of its time. This method recalls the technique of rjs in his *Energy Crisis Poems* whose similarly time-based method prompts that writer to reflect, "the poet need not change his vision; he need only to move forward or backward in time to achieve innovation". [rjs 1974] And yet, a close reading of Bertram's code reveals that whenever one runs the poem, some things will always be true. The expression `$d[3]` will always select the word "black" because the index is the integer, `3`, and not the regular expression backreference `$3`, and `$d[$8]` and `$d[$6]` will always select "switch" because there are only 5 possible capture groups and a null backreference operates like a `0` in referencing the first element in a list. 73

Therefore, in this poem, the second word of each shorter line will always be "switch", the third word of each longer line will be "black", and the final word of each longer line will be "switch". 74

> [$_] you switch \ black codes black switch gods switch \ you switch \ black codes black switch switch switch \ you switch \\ black codes black switch black switch \\ you switch \\ black codes black switch codes switch [Bertram 2019a]

In this poem and throughout the collection, Bertram provides an alternative to rjs. Simply moving forward in time is not necessarily innovation, just as it is a mistake to assume that America, with its history built on Black suffering, is naturally less racist as time passes. "@Code_switching" is dedicated to Frantz Fanon whose *Black Skin, White Masks* emphasizes the linguistic mechanisms of colonization. As a product of time, "@Code_switching" reminds readers that to be Black and of the moment within that mechanism is to be always switching: codes switch, you switch, when switch, you switch. Always "black" and always "switch". 75

Perhaps the only way to stop the switch is to kill the process, which Bertram portrays in a metaleptic gesture similar to the crashing halt of “three_last_words”. Bertram concludes “@Code_switching” with a series of `^C` -- the characters one sees after typing `ctrl+c` in a terminal to abort the currently-running process. We can imagine the poet at their keyboard spamming `ctrl+c` until the poem crashes. 76

This is one of the many poignant moments in *Travesty Generator* where a computational process gives way (switches) from a formal concept to a lived, individual experience of a reality shaped by the consequences of colonization, oppression, and injustice. As I noted at the outset of this essay, my experience of race is very different; as a white man working in academia, I participate in a system that was designed by and for people like me. By working with the tools of similarly-monolithic computer programs, Lillian-Yvonne Bertram uses their poetry to think outside of the ideologies and paradigms those systems take for granted. Rather than letting language speak for itself (human or computer), Bertram speaks past the filter of those generators to share their lived, individual, and sometimes traumatic experiences of race and identity in 21st century America. 77

## Acknowledgement

This essay is based on a forum post I shared on the Critical Code Studies Working Group forum in March, 2020. I am grateful for feedback from Mark Marino, patricia_s, Milton Laufer, and especially Lillian-Yvonne Bertram as I transformed that post into this longer essay. 78

### Notes

[1] The lineage that connects "Counternarratives" to "Through the Park" is similar to the network of remixes created in response to Nick Montfort's poem, "Taroko Gorge," many of which have been collected in Volume 3 of the *Electronic Literature Collection*. As both Cayley and Marino have discussed, the fact that many of the derivative works leave the algorithm of that poem intact -- and thus its form -- but make meaningful changes to the data means that the original work Montfort created is more like a poetic form than simply a poem. J.R. Carpenter's remix of that poem, "Gorge," replaces Montfort's variables with words related to gluttonous consumption. While an analysis of Carpenter's work is beyond the scope of this essay, it is interesting that her collection, *Generation[s]* which includes "Gorge," also features several poems generated by remixing and modifying "Through the Park." Reading those alongside Bertram's "Counternarratives" as would likely highlight different perspectives on Montfort's original poem.

### Works Cited

**BBC 2013** BBC. (2013) “George Zimmerman not guilty of Trayvon Martin murder”. 14 July 2013. *BBC News*. BBC.com. https://www.bbc.com/news/world-us-canada-23304198

**Bertram 2019a** Bertram, Lillian-Yvonne. 2019. *Travesty Generator*. Noemi Press, Blacksburg VA.

**Bertram 2019b** Bertram, Lillian-Yvonne. 2019. “The Whitest Boy Alive: Witnessing Kenneth Goldsmith by Lillian-Yvonne Bertram” *Poetry Foundation*. https://www.poetryfoundation.org/harriet/2015/05/the-whitest-boy-alive-witnessing-kenneth-goldsmith

**Bertram n.d.** Bertram, Lillian-Yvonne. n.d. “New Sermon Code”. Python 2. https://www.lybetc.tech/new-sermon-code

**Brooks 1987** Brooks, Gwendolyn. 1987. *Blacks*. David Co, Chicago.

**Buchloh 2012** Buchloh, Benjamin. 2012. “The Book of the Future: Alison Knowles's The House of Dust”. *Mainframe Experimentalism: Early Computing and the Foundations of the Digital Arts*. Ed. Hannah Higgins and Douglas Kahn. University of California Press.

**Carey 2021** Carey, Julie. 23 April 2021. “Virginia Man Shot by Sheriff's Deputy After Calling 911 for Help”. *NBC4 Washington*. https://www.nbcwashington.com/news/local/northern-virginia/virginia-man-isaiah-brown-shot-by-sheriffs-deputy-after-calling-911-for-help/2649178/

**Carpenter 2011** Carpenter, J.R. 2011. *Generation[s]*. Trauma Wien.

**Cayley 2002** Cayley, John. 10 September 2002. “The Code is not the Text (Unless It Is the Text)” *electronic book review*. https://electronicbookreview.com/essay/the-code-is-not-the-text-unless-it-is-the-text/

**Cayley 2015** Cayley, John. 31 January 2015. “Poetry and Stuff: A Review of #!” *electronic book review*.

http://electronicbookreview.com/essay/poetry-and-stuff-a-review-of/

**Chun 2011** Chun, Wendy Hui Kyong. 2011. *Programmed Visions: Software and Memory*. Boston, MIT Press.

**Dennigan 2020** Dennigan, Darcie. 2020. “2020 Anna Rabinowitz Prize.” *Poetry Society of America*. https://poetrysociety.org/award-winners/2020-anna-rabinowitz-prize

**Harpold 2007** Harpold, Terry. 2007. “Screw the Grue: Mediality, Metalepsis, Recapture”. *Game Studies*. 7:1.

**Hartman 1996** Hartman, Charles O. (1996) *Virtual Muse: Experiments in Computer Poetry*. Middletown: Wesleyan University Press.

**Kenner and O'Rourke 1984** Kenner, Hugh and Joseph O'Rourke. 1984. *Byte Magazine*, 9:12.

**Knowles 1968** Knowles, Alison and James Tenney. 1968. *A House of Dust*.

**Laiti 2016** Laiti, Outi. 2016. *Ethnoprogramming : an indigenous approach to computer programming : a case study in Ohcejohka area comprehensive schools*. University of Lapland, Faculty of Education. https://lauda.ulapland.fi/handle/10024/62624

**Lavigne 2015** Lavigne, Sam. 1 April 2015. “Slow Hot Computer” Javascript. http://slowhotcomputer.com

**Marino 2020** Marino, Mark. 2020. *Critical Code Studies*. MIT Press, Cambridge, MA.

**McCauley 1974** McCauley, Carole S. 1974. *Computers and Creativity*. Praeger, New York.

**Montfort 2008** Montfort, Nick. 2008. “Through the Park” Python. https://nickm.com/poems/through_the_park.py

**Montfort 2013** Montfort, Nick. 2013. “Round” Javascript. *New Binary Press*. http://newbinarypress.com/publications/

**Montfort 2014a** Montfort, Nick. 2014. *#!*. Counterpath Press.

**Montfort 2014b** Montfort, Nick. 2014. “A House of Dust reimplementation”. HTML. https://nickm.com/memslam/a_house_of_dust.html

**Mullen 1999** Mullen, Harryette. 1999. “Imagining the Unimagined Reader: Writing to the Unborn and Including the Excluded”. *boundary 2*. 26:1.

**Noble 2014** Noble, Safiya. 2014. “Teaching Trayvon”. *The Black Scholar*. 44.1 (2014): 12-29.

**OED 1989** Oxford English Dictionary. (1989) “travesty, v”. *Oxford English Dictionary*. {2nd edn.} OED.com. https://www.oed.com/oed2/00256847

**Ortiz 2014** Ortiz, Eric. 19 December 2014. “Indiana Cop Told to Stop Selling 'Breathe Easy' T-shirts” *NBC News*. NBCnews.com. https://www.nbcnews.com/news/us-news/indiana-cop-told-stop-selling-breathe-easy-t-shirts-n271581

**Sample 2011** Sample, Mark. 13 September 2011. “Zombie Code and Extra-Functional Significance”. *Play the Past*. https://www.playthepast.org/?p=1989

**Snodgrass and Soon 2019** Snograss, Eric and Winnie Soon. 1 February 2019. “API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange”. *First Monday*. 24:2.

**Whalen 2014** Whalen, Zach. 7 February 2014. “House of Dust by Alison Knowles and James Tenney” HTML. http://zachwhalen.net/pg/dust/

**Xiong et al. 2021** Xiong, Chao, Paul Walsh, and Rochelle Olsen. 21 April 2021. “Derek Chauvin cuffed after murder, manslaughter convictions in death of George Floyd” *StarTribune*. https://www.startribune.com/derek-chauvin-cuffed-after-murder-manslaughter-convictions-in-death-of-george-floyd/600047825/

**rjs 1974** rjs. (1974) *Energy Crisis Poems: Poetry by Program*. Cleveland: Ground Zero.