Playing with Materiality: An Agential-Realist Reading of SethBling's *Super Mario World* Code-Injection

C. Mckeown

University of Glasgow, UK

ARTICLE HISTORY
Compiled March 30, 2018

ABSTRACT
This article takes its inspiration from Youtuber and software developer "SethBling" and his 2016 "code-injection" (Bling, 2016), in which, using only a standard Super Nintendo Entertainment System controller and in-depth knowledge of the console, he wrote and executed the code of popular mobile game *Flappy Bird* (Nguyen, 2013) within a running instance of *Super Mario World* (Miyamoto, 1990), effectively transforming one game into another through play. Spurred on by his actions, I propose, a performative understanding of videogames (and software in general) to shed new light on discussions of software's materiality. Though we can contrast Wendy Chun's (2008) suggestion to view software as 'vaporous' with Friedrich Kittler's (1995) belief that 'there is no software', I propose a more holistic approach. I suggest that we see software as living a double-life: as simultaneously solid as it is (metaphorically) gaseous. We can then embrace software as performative examples of the entangled 'phenomena', suggested by Karen Barad (2007), that produce reality, from the entangled quantum realm to our everyday lives. I explore SethBling's code-injection suggesting that his actions clearly reveal software's existence as both tangible "thing", locatable on magnetic storage memory, and as a vaporous, non-entity. Accepting both of these propositions together, we can see software as something that, as evidenced by the "injection" of one game into another through play, continuously re-emerges through shared actions. Following Barad, I conclude that this quality is not unique to software, but software — and videogames above all — are a useful tool for understanding a vision of reality that favours activity over materiality as the basis of our existence.

KEYWORDS
Sections; lists; figures; tables; mathematics; fonts; references; appendices

1. Introduction

Increasingly, software is drawn on to frame philosophical problems that can otherwise seem inaccessible (see, for example, Yuk Hui's 2016 *On the Existence of Digital Objects*). This may be testament to the increasing ubiquitousness of technology in contemporary society. Frabetti argues this, attributing the resonance with software and philosophy 'to the pervasiveness of software and to the fact that media users are constantly being asked to make decisions about software-based technologies in everyday life (for instance, when having to decide whether to use commercial or free software, or whether to upgrade their computers)' (Frabetti, 2015, p. xii). The need to regu-

CONTACT C. Mckeown Email: cpmmckeown@gmail.com

larly evaluate the software that guides our lives, may be prompting users to remain in a more consciously critical state. Computer software, and its regular maintenance, it could therefore be argued, promotes a form of critical thinking. While this is an engaging possibiltiy that deserves further investigation, there may be other reasons ubiquitous software stimulates philosophical thought.

It is possible that the growing entanglement of software and philosophy is due to the disconnect between how much the average user *uses* software, and how much they understand it. As we become increasingly reliant upon software to shape society, there is an increasing need to try to fathom the implications of it in a variety of ways. Yet, for many average users, contemporary digital technology such a cell-phones and laptop computers, are tools to be used rather than understood. This distinciton has been conceptually framed as the difference beween 'mastering ideas, not keystrokes'(Knobel, 2008, p. 2). For many, although software is used and relied upon for everyday tasks, the elegant, underlying complexity therein, may be unimportant; what matters is that these devices can ease the completion of tasks. Given this distinction between how much average users do with software, and how much we understand that software, digital devices provide an interesting entry-point for analysing every day life. If we as users stop to consider our "tools" more deeply, there is a potential to experience profound feelings of (re)discovering the strange in the familiar. However I am not attempting to establish a dichotomy between those users who perceive software as functional without considering its construction (those, for whom, technology has become indistinguishable from "magic"(Clarke, 1973)) and those who possess an in-depth understanding of technical processes. Rather, I suggest that understanding software in depth — through a particular philosophical lens — actually yields a great deal more "magic", strangeness and unpredictability than when we do not consider it mechanically at all.

In this paper, I aim to reframe the — now, almost abandoned — discourse of software materiality, arguing that closer examinations of hardware and software do not bring us any closer to a kind of dominating understanding of machines, but rather allows a technologically engaged society to understand the ontological possibilities behind our actions. To accomplish this, I will invoke limited elements of the agential-realist (though sometimes dubbed, "new materialist") philosophy of Karen Barad as a means to engage with existing writings on software. Focusing on Barad's idea's of performative ontology, I will explore the implications of Youtuber SethBling's (who, in interviews, provides only his first name, "Seth" as a real name) 'code injection' of *Flappy Bird* into *Super Mario World*.

2. What is a Code Injection?

Before presenting the theoretical backdrop of this article, however, I will explain my choice of media-phenomenon for this case study. On March 27th, 2016, software developer/hacker Seth Bling completed what is known as a "code injection" into the popular Nintendo videogame *Super Mario World* (SMW). Playing the game on an unmodified Super Nintendo Entertainment System (SNES) console, Bling was able to "inject" 331 bytes of processor instructions (machine code) into the SNES' memory. These instructions were an interpretation of the source code for popular mobile game, *Flappy Bird*. Bling was then able to prompt the running game to execute the code, causing it to mutate into another game entirely. Bling documents this process in great detail in a video blog he posted to his Youtube channel the following day. To date,

this video has received close to two million views and many thousands of admiring comments including my personal favourite, "does this mean that *Super Mario World* is a programming language now?".

In the interests of clarity, short overviews of both games are required, as well as some explanation of the link between the two. In short, *SMW* is a 1990 side-scrolling platform game, in which the player takes control of a small representation of the Nintendo company's mascot, Mario. Using a series of simple commands, the player must run, jump and swim their way through a colourful land of musrooms, turtles and large green pipes. *SMW* is a particularly noteable entry in the franchise's history for its pleasing cartoon graphics and memorable soundtrack. By comparison, *Flappy Bird* is an incredibly simple game developed by lone Vietnamese programmer Don Nguyen. The game tasks players with controlling a small, pixellated bird character, that is moving from left to right interminably while plummeting towards the ground. Players can keep the bird aloft by pressing a single button but the nature of the elevation is often at odds with what would be ideal. The game is noted for its difficulty and addictive quality. Notably, players must also prevent the bird from colliding with green pipes, stylised in an obvious homage to *SMW*. For SethBling and fellow programmers involved in developing the exploit I am focusing on in this paper, the connections between the two would have been obvious while the simplicity of the game allows for a relatively easy job of reprogramming.

Given technological virtuosity of what Bling accomplished, transforming or reprogramming a videogame while playing it, it is also important to provide the clearest picture I can of how this was accomplished. SethBling used a normal SNES console for the exploit, the only modification being the use of extra controllers with taped down buttons. These were simply used to prompt the system into accepting code from addresses other than those on the game cartridge. The first part of the exploit was to trigger a glitch in the game using a small error in the game's original code, whereby if the player has collected certain items and completed certain tasks, standing on a specific spinning platform boosts Mario's power-up level. SethBling explains, players can receive three-stages of "power-up", by collecting the ubiquitousness Mario-mushrooms that are so emblematic of the Mario franchise and a symbol of gaming culture as a whole. Collecting an item changes Mario's appearance and abilities. The three standard stages are to grow in size, to don a cape, and to gain the ability to throw fireballs. These three states, with the addition of Mario's small, base-state, create a total of four possible states. However, the game, for unknown reasons, has a total of seven possible states in the source code. Although the player will not experience any changes, it is possible to collect additional power-ups in order to play the game in the undefined states. Doing so triggers a range of strange behaviours from the game, changing Mario's appearance or the behaviour of the level-timer. Behind the scenes, however, at the level of machine processes, collecting a power-up triggers the game console into attempting to load code for the corresponding states. This allowed SethBling and fellow designers to engineer their exploit of the program by providing code for the uncoded states, left in the source-code by accident.

SethBling and fellow hackers manipulated the "x-co-ordinate sprite table", a small algorithm based on a number of arrays and simple addition loops, that trigger on-screen graphics in relation to Mario's relative x-position (how far along the screen the avatar is). In other words, SethBling was able to move Mario a certain distance across the screen, then collect a mushroom, triggering the SNES to look for new code to execute, corresponding to how far along the screen the character was. Although this is simplified, we can understand that if Mario was, let's say, 10 pixels along in a given

level, the system would look in a memory-address corresponding to that number. It should be understood that this, on its own, is not of great consequence as although interesting, without new code, the game would simply run its own code either earlier or later than intended. Manipulating the game in this state allowed SethBling et. al. to make a more suitable environment for code manipulation (extending the game timer by some hours, changing the "coin" collector information to display Mario's x-co-ordinate rather than the number of coins collected) but the true genius of SethBling's exploit came from an ability to write new code to the unused system RAM, without executing it. This allowed SethBling to manually "inject" 331 bytes of processor instructions into unused system memory, only to execute it later. To be clear, this whole process took place through a system of highly specific and coordinated movements, only to be "executed" by the collecting of a power-up mushroom. Having injected the code and collected the final mushroom required, SethBling's screen went momentarily blank, only for an image to re-appear of Mario, flying through the air, attempting to avoid pipes emerging from the right-hand side of the screen, while a small number hangs in the left-hand corner, counting the score. In other words, the entire game was shape-shifted from *SMW* to *Flappy Bird*.

SethBling did not achieve this feat on his own and he is careful to acknowledge that what he accomplished was not, in itself, ground-breaking. In the first instance, several other hackers were involved in generating the assembly code necessary to create the exploit used here. What's more, others helped provide the knowledge necessary to reverse engineer the game in such a way that writing, storing, and executing code in the running game, turning *SMW* into a kind of code development environment, became possible. At the same time, SethBling acknowledges that code injections are not new and that he is not the first to do something of this kind. However, he does point out that he is one of very few humans to do anything of this sort. While inputting machine code into SMW had been achieved in the past using hacked versions of the game and by rigging up input systems so that commands could be entered as a series of values (a form of programming language in itself), SethBling's manual input of these incredibly specific actions is no small achievement.

I have chosen this particular code injection for a number of reasons some of which will only become significant in due course (by the end of this article, for instance, I will suggest that it can be beneficial to view videogames as specific performances rather than as texts in a manner familiar to literature or film studies; rather than looking at *SMW* gameplay as a whole, I am analysing a specific performance). Most importantly, however, is what SethBling's actions tell us about software and its materiality.

As stated in the brief introduction to this article, software studies' various approaches highlight the varied conceptualisations we have of software. Andrew Goffey sums this up well, discussing the importance of the algorithm in software design. He writes, 'computer scientists work with them as if they were purely formal beings of reason (with a little bit of basic mathematical notation, it is possible to reason about algorithms, their properties and so on, the way one can reason about other mathematical entities), algorithms bear a crucial, if problematic, relationship to material reality [...] while the Turing machine is an imaginative abstraction, its connotations of materiality are entirely real'(Goffey, 2008, p. 16). Taking the discourse of software's materiality on the whole we can split it into two large sections: those who view software as less-than-material ("vaporous", "ephemeral", "immaterial"), and those who are confident to assert its materiality in a number of ways. I will begin by elaborating on the first school of thought, focusing on the work of Wendy Chun for its appeal to common-sense use of technology, where the workings of complex machines seem almost

like magic. Chun appropriates, to an extent, the term 'vapor theory' taking inspiration from Peter Lunenfeld's denigration of a trend in 1990's media-studies he deemed 'a gaseous flapping of the gums about technologies, their effects and aesthetics, usually generated with little exposure, much less involvement with those self-same technologies and artworks'(Lovink, 2002, p. 235). For Chun, the issue is not vapor theory in itself, our lust for software's potential in the face of software's oft-underwhelming reality, but rather that it was abandoned so quickly. She writes, 'vaporiness is not accidental, but rather essential to, new media and, more broadly, to software. Software, after all, is ephemeral, information ghostly, and new media projects that have never, or barely, materialised are among the most valorised and cited'(Chun, 2008b, p. 301). Chun warns us of the pitfalls of placing too much faith in ability to understand the workings of computation; the faith we place in our knowledge of source code, like learning the workings of a magic spell, will only ever be a part of what it is to work with a machine. To an extent, a part of our relationship with technology is, 'to some extent imagined'(Chun, 2008b, p. 300). Although we may see source code in front of our faces, take the time to educate ourselves in computer science, learn the logic of a programming language, stay abreast of its updates and permutations, memorise the many types, methods, declarations or else the syntax of a low-level system like assembly language, at some point we must place our faith in the machine and accept what is produced as the nearest thing to what we hoped to achieve.

Reflecting on Bling's code injection, we can attempt to see how a vaporous understanding of software can be beneficial for understanding performances such as computer-hacking and, particularly, Bling's manual code-injection. Taking Lunenfeld's ideas in isolation, his belief that vaporous appreciations of software resulted from a lack of knowledge of the software itself, we can reverse engineer his claim to a certain extent. We can reason that, to Lunenfeld, greater knowledge of a software system should result in a more material, solid apprecation of the physical impact that software can have on the real-world. The aforementioned "real" impact on materiality that can be evidence from the outcomes of algorithms. While it is tempting to think this way, that greater knowledge should yield a greater grasp of software as something we can know, see, or even in some way, hold, Bling and fellow hackers provide an alternative take on this. Instead, encouraged by exploits like the code injection, we can see software as increasingly vaporous, the more knowledge of it we have. Though software may function largely as expected, it is still filled with a variety of points for potential entry and exploitation. The more we know about a digital artefact — though, by the end of this paper, I will challenge the notion that such a thing even exists — the kind of advanced knowledge that Bling and fellow hackers have of *Super Mario*, is what allowed them to illustrates its entire "vaporous" qualities.

Bling and associates have, in a sense, an "inside-scoop" on the SNES system; its RAM, its many processors, the games' source code, of SNES assembly language, and of programming as process in general. This knowledge did not make *SMW* appear more "solid" to them, more of a "thing" beyond question. Rather their insider knowledge allowed them to see how exactly how this piece of software was more mutable and more open to change or manipulation. For instance, SethBling explains that one key method for implementing the code injection relied upon a peculiar quirk of the game's programming: the "power-up incrementation" described earlier in this paper. To Bling's mind, *SMW* is not a solid entity but, rather, a series of processes, different ones triggered at different moments, entirely dependent upon specific configurations of various elements. Mario had to be positioned in a specific manner, controllers to be taped down, jumps and power-ups to collected at precise moments. Yet, all of these

precise actions hinged on the central notion that the "game" in itself is a series of electrical processes that can be varied and changed. In the end, the in-depth knowledge displayed by Bling and his co-conspirators, did not result in a deeper appreciation of the game as a fixed entity. Familiarity did not give rise to a deeper knowledge of *SMW* as a "text", in the manner we would expect of a book or film. Rather, it was precisely their knowledge of the various glitches produced by the game that allowed them to transform it so completely. Greater knowledge of "the game" revealed that it was simply a series of instructions and activities that could be manipulated and distorted. Knowledge did not give rise to solidity but vaporiness.

Conversely, it is possible to contend an alternative understanding of the code injection. We can develop a reading of Bling's intricate performance that is contrary to my understanding of this element of Chun's work. After all, Chun would be the first to accept that her celebration of vaporiness is a minority position. She establishes that for many scholars, such as Alexander Galloway, Geert Lovink, McKenzie Wark and Paul Virilio, the solidity of software is all too certain (Chun, 2008a). Software, in the form of global systems tied to magnetic memory banks that cannot be erased, that are constantly computing, they argue, has changed the world indelibly. Software's materiality is its impact and the certainty with which it divides spaces and time. In doing so, it challenges the fluidity of human experience and presents the potential of a 'total loss of the bearings of the individual'(Virilio, 2002). Particularly powerful examples of this thinking are Kittler's polemical writings on software-as-hardware. For Kittler, software is anything but imaginary, and its effects can be felt through its ramifications on society. For him, the creation of software came in the form of Alan Turing's theories of computational numbers being put to the test at Bletchley Park. Software is the collapse of the essential subjectivity of language and its replacement by symbols becoming actions. For Kittler, 'anyone who would like to see the phallus in the 5 volts of a logical 1, and the hole in the 0.7 volts of a 0, confuses industrial standards with fiction'(Kittler, 1999, p. 246). Kittler's theories hinge on the notion that digitality is a departure from the symbolic and an engagement with the enacted. Unfortunately, though we engage with this on a daily basis, we remain — according to Kittler — fundamentally unaware of this shift. Drawing on the daily use — but near universal lack of comprehension — of complex word processing software, he writes, 'what remains a problem is only the realisation of these layers which, just as modern media technologies in general, have been explicitly contrived in order to evade all perception. We simply do not know what our writing does.'(Kittler, 1995). After Kittler's death in 2011, his somewhat techno-determinist ideas live on through the loosely collected 'Berlin Brand'of media studies, spearheaded by the likes of Wolfgang Ernst(Ernst, 2013, p. 145). Through these scholars, Kittler's ideas relating to software-as-hardware remain largely intact with discussions of technology still, at times, coming down to questions of volts(Ernst, 2013, p. 159).

The alternative form of media studies discussed above can also be effectively related back to SethBling's artful manipulations of *SMW*. Although we may choose to envision the code injection as resultant from the uncertainties inherent in computing, the product of glitches and undesired program flaws, the fact remains that the program could be manipulated because certain elements could be targeted and adulterated with certainty. Telling of this is the code-injection's reliance upon SNES "assembly language", the form of low-level computer programming, one step above machine code. Although assembly is still readable by humans, it is tremendously time consuming to do so, particularly when attempting to create larger programs. A key element of assembly programming is writing, loading and storing variables at specific memory addresses.

Now, this still seems many levels of abstraction away from the operations of a complex machine, but in reality, when programming in assembly we are writing the journeys of electronic charges through a mechanical system. For example, when we look at the small sample of assembly code provided by Bling in his video, we can see a large portion of the instructions either read 'ld'or 'st'— in other words, load or store. I will attempt to make this as brief as possible to not necessitate a deep-dive into SNES assembly language or how to read hexadecimal numbers, but suffice it to say, in the program written for Bling to inject into the program, specific memory addresses, which is to say, specific, physical sectors of the SNES' onboard RAM were being selected and provided with varying levels of electric charge.

We can continue further in this vein, digging deeper into the relationship between what Bling was doing with the controller and what was occurring at a system level. Understanding Bling's actions as a specific performance, more akin to a piece of theatre or a protest event, something that took place at a specific time and place, we can view his performance as an attempt to break down the standard relationship between screen media as a representative form. Instead, we are presented with something closer to what the Berlin Brand discuss in their writings on technology. Rather than accepting Bling is playing *SMW*, the familiar videogame "text", we can accept the code injection as an act in itself. In doing so, we are free to move away from associations we may have from childhood, wherein we are familiar with the on-screen avatar, the Italian-American plumber, "Mario", who moves through the land of the "Mushroom Kingdom", and instead entertain a radically removed understanding of what is occurring. Embracing an approach to videogame studies with more in common to software studies or media archaeology, a "platform studies" approach can provide us with a different appreciation of Bling's actions.[1] Although we can see Bling moving Mario from one area of the screen to another, we can instead attempt to appreciate the different flows of information-as-energy coursing from one part of the system to another. Dominic Arsenault informs us that the SNES' processing speed, for instance, was determined by the 'architecture of the buses. The SFC [Super Famicon] featured two address buses responsible for directing the data traffic to the right memory locations: "bus A" (or the "main bus") was 24 bits wide and handled data transfers between the CPU, cartridge ROM, and the console RAM; "bus B" was 8 bits wide and had the specialised function of handling hardware registers for the PPU [Picture Processing Unit] and APU [Audio Processing Unit], responsible for the player's audiovisual experience'(Arsenault, 2017, p. 91). Rather than conceiving of Bling's actions as the manipulating of Mario for conventional means, we know that he is engaged with the transferral of 8-bit data, using memory bus B, passing registers between the PPU and system RAM, generating a series of charges, that will trigger a number of machine activities when eventually initiated. Like setting up a series of electronic dominoes, Bling's button presses are preparing a chain reaction. Although this will result in an apparent transformation of the game into something entirely other for a viewer, in "reality", to the SNES and its internal system, this is business as usual.

An interpretation of the phenomena of videogame play suggested by the above has many strengths. One major point in its favour is that the attention to detail, to the minutiae of computer processes, is so uncommon. Perhaps too often, we consider computers to be near-magical objects completing tasks while cloaked in mystery. Though we may be experts at using them, we never need to know how they do what they do. Yet, a Berlin-brand software/platform/media archaeological appreciation of software also detracts, in many ways, from the complex reality that allows the existence of what can be observed. Indeed, there is a temptation, particularly if — like me — you

have spent the majority of your education in humanities departments, to bow down to numbers and figures as in some way, "more real" than the social constructs that are familiar to us. When faced with code, particularly low-level code, or hardware details, the "code fetishism" suggested by Chun, takes over (Chun, 2008b). A confidence that computers can be understood completely ensues. Reassuring mathematical details, such as those provided by Kittler "5 volts" and "0.7 volts", establish an order to what, on the surface, appears to be chaos. Yet, without meaning to throw a cat amongst the pigeons, to put it very simply, we should not place too much confidence in this overly "material" engineering-influenced approach when attempting to understanding the philosophical weight of computation. After all, a binary "1" can range anywhere from 2.7 to 5 volts. And within a volt, we find a multiplicity of many trillions of electrons, ready to re-establish chaos at any time.

Instead of suggesting that we understand SethBling's code-injection using either Chun's understanding of code "sourcery" or else placing our faith in Kittlerian readings of computers as machines, I believe there is a way to bring these two understandings of computation together in a manner that allows the strengths of both to flourish. We can dig in to the depths of computer activities but instead of attempting to find certainty there, we find uncertainty. We find an undeniable chaos that allows computation and, so, videogame play to take place. Yet, in doing so, we also uncover a method through which an in-depth understanding of the functioning of videogames can shape our view of the material world. This is the potential provided by an agential-realist approach to videogame analysis that I will now unpack and explore.

3. An Agential-Realist Approach

Karen Barad, feminist philosopher and theoretical physicist has been praised as 'arguably the most prominent figure in the new materialism'(Hein, 2016, p. 1). Her fame comes with good reason. Barad's science-philosophy is not only a radical reworking of contemporary thinking relating to familiar objects of critical inquiry, not only a daring posthuman stance on well-trodden ground, but a bold rethinking of much of western-thought. At the centre of Barad's hypothesis is a concept of existence, of base, meta-physical ontology, contrary to definitions of place, space and time common western-thought. She writes, 'entities, space and time exist only within and through their specific intra-actions; this is not to say that they are mere transient and fleeting effects but rather that they are specifically materially constituted.'(**?**, p. 111). To Barad, "things" are only ever apparent, constantly, co-constituted through multiple phenomena.

Although Barad's philosophy can initially be dizzying, understanding its roots make it somewhat easier to grasp. As mentioned, Barad's background is in theoretical physics but a great inspiration for her work comes from the social-performative theories of Judith Butler. Barad notes of Butler's famous work on gender, 'Butler proposes that we understand identity not as an essence but as a doing.'(Barad, 2007, p. 62). Though we may be tempted to construe this comment in a manner that invokes a singular, powerful actant, in control of their self-identity, removed from social constraints, Barad reminds us that, 'Butler cautions that this claim — that gender is performed — is not to be understood as a kind of theatrical performance conducted by a willful subject who would choose its gender. Such a misreading ironically reintroduces the liberal humanist subject onto the scene, thereby undercutting poststructuralism's antihumanism, which refuses the presumed givenness of the subject and seeks to attend to its

production.'(Barad, 2007, p. 62). Rather, it is essential that we understand Butler's theories as commenting on how society as a milieu acts collectively to produce the subjects within it, subjects that may act with/against the constraints they face. Extending this idea outwards, not just to the construction of identity but all things, we are given a glimpse of a performative account of matter.

By the same token, however, Barad's knowledge of cutting-edge practices within the hard-sciences plays just as important a role in her work. Just as she notes the social-construction of identity she pairs this with the quantum instability that plays an important part in shaping matter. Barad notes important scientific phenomena such as super-position and uncertainty, drawing our attention to the flexibility of matter. Writing about a famous experiment on the nature of both light and (sub-atomic) matter, the two-slit experiment, Barad notes, 'the nature of the observed phenomenon changes with corresponding changes in the apparatus. But this is contrary both to the ontology assumed by classical physics, wherein each entity (e.g., the electron) is either a wave or a particle, independent of experimental circumstances, and to the epistemological assumption that experiments reveal the pre-existing determinate nature of the entity being measured'(Barad, 2007, p. 106). In all, we are presented with an idea of reality, founded on observations, inspired by social theory that speaks to the performative nature of creation.

Returning to Bling's code-injection then, it is important for us to understand his actions in the same manner that we can understand the configuration of a two-slit experiment. The set-up of the experiment plays a role in producing the results that can be observed. By moving Mario back and forth across the screen, collecting power-ups to trigger the all important machine actions that his hack relies on, we understand that matter is being manipulated in specific ways. However, the very mutability of this matter, speaks to the very performative nature of our reality. That Bling can become involved in reshaping *SMW* as it runs is testament to the malleability of our reality. We can observe through this digital text, the qualities of co-constituted creation that Barad speaks to. As such, it is important for us to understand that the solidity of software, the existence of voltages in specific memory addresses, and our ability to manipulate these voltages through specific performative actions — in SethBling's case, pressing a buttom for a long enough time, corresponding with moving an on-screen avatar to a specific location which, in-turn, prompts the movements of charges through a computer system — should not prompt us to consider software as something more akin to concrete. Rather, we should embrace that although we are moving voltages, those voltages are, in themselves, composed of trillions of electrons, which, in-turn, have only a loose "location" in existence (super-position). Rather, the mutability of software speaks to the constant dance of matter that constitutes things such as videogames, but also — potentially — all things. The cycle of loading information that SethBling is able to hijack and use to his advantage is a particularly materially-performative process. However, it relies on the performativity of the matter of which it is composed. The malleability of software is an echo of the malleability of the sub-atomic compounds it is forged from. The store and load loops that comprise the main program of *SMW*, the same processes of constant intra-activity that co-constitutes our reality.

4. Conclusion

In this essay, taking the lead from discourse within software studies, I have argued for a new way of looking at videogames. Embracing software's odd double-life as both magically vaporous and empirically solid, we are free to understand videogames as ontologically performative phenomena. Using the science-philosophy of Karen Barad, I have proposed a turn towards understanding videogame play as shared performances; not just between machines and human users, but specifically resultant from our material universe. In turn, I suggested that the performances of videogames are also invaluable tools for understanding reality as produced by a cooperative flow of prematerriality. Although this paper represents the early stages of suggesting videogames as tools for this specific school of philosophical thought, it is my hope that this approach can be built upon in future, to create a cohesive performative philosophy of videogame play.

Notes

[1] A substantial body of work has been produced within game studies journals or by academics associated with game studies on the role of code and construction in games. Of particular importance is the work of Nick Montfort, Ian Bogost and Mia Consalvo who, among others, have played an important role in forming the 'platform studies' offshoot from game studies, placing an emphasis on the role and importance of design practices such as coding and software engineering. This branch of game studies is growing in popularity every year, perhaps as increasing numbers of humanities scholars are gaining skills in computer design or perhaps as an impact of the growing interdisciplinarity of the field. While not entirely blind to concerns such as narrative and engagement, platform studies does place an impetus on exploring links between affective qualities and the mechanical properites of a programmed digital text. As such, the work of these scholars was a tremendous inspiration for this article but was excluded from the body of the article in the interests of readability.(Montfort, 2006) (Montfort & Bogost, 2009) (Montfort et al, 2012) (Montfort & Consalvo, 2012)

References

Arsenault, D. (2017). *Super Power, Spoony Bards, and Silverware: The Super Nintendo Entertainment System*. London, UK: MIT Press.

Barad, K. (2007). *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*. Durham, NC: Duke University Press.

Barad, K. (2017). No Small Matter: Mushroom Clouds, Ecologies of Nothingness, and Strange Topologies of Spacetimemattering. In A. Tsing, H. Swanson, E. Gan and N. Bubandt (Eds.), *Arts of Living on a Damaged Planet: Ghosts of the Anthropocene* (pp. 103–120). London, UK: University of Minnesota Press.

Anonymous [SethBling]. (2016, March 26). SNES Code Injection – Flappy Bird in SMW Retrieved from https://www.youtube.com/watch?v=hB6eY73sLV0

Chun, W. H. K. (2008) The Enduring Ephemeral, or the Future Is a Memory *Critical Inquiry*, *35*(1), 148–171.

Chun, W. H. K. (2008) On "Sourcery," or Code as Fetish *Configurations*, *16*(3), 299–324.

Clarke, A. C. (1973) Hazards of Prophecy: The Failure of Imagination, in *Profiles of the Future: An Enquiry into the Limits of the Possible*.

Ernst, W. (2013) *Digital Memory and the Archive* London, UK: University of Minnesota Press.

Frabetti, F. (2015). *Software Theory: A Cultural and Philosophical Study*. New York: Rowman and Littlefield.

Goffey, A. (2008). Algorithm. In M. Fuller (Ed.), *Software Studies: A Lexicon* (pp. 15–20). Cambridge, MA: MIT Press.

Hein, S. (2016). The New Materialism in Qualitative Inquiry: How Compatible Are the Philosophies of Barad and Deleuze? *Cultural Studies ¡-¿ Critical Methodologies 16*(2), 1–9.
Kittler, F. (1995). There is No Software *CTheory*. Online Publication. www.ctheory.net/articles.aspx?id=74
Kittler, F. (1999). *Gramophone, Film, Typewriter* Stanford, CA: Stanford University Press.
Knobel, M. (2008) *Digital Literacies: Concepts, Policies and Practices* New York: Peter Lang.
Lovink, G. (2002). *Uncanny Networks: Dialogs with the Virtual Intelligentsia*. Cambridge, MA: MIT Press.
Miyamoto, S. (1990). *Super Mario World*. Kyoto, Japan: Nintendo Entertainment Analysis & Development.
Montfort, N. (2006). Combat in Context *Game Studies 6*(1).
Montfort, N. & Ian Bogost (2009). *Racing the Beam: The Atari Video Computer System*, London: MIT Press.
Montfort, N., Ian Bogost, Patsy Baudoin, John Bell, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample & Noah Vawter (2012). *10PRINT CHR$(205.5+RND(1)); : GOTO 10;* , London: MIT Press.
Montfort, N. & Mia Consalvo (2012). The Dreamcast, Console of the Avant-Garde *Loading... 6* (9).
Nguyen, D. (2013). *Flappy Bird* Hanoi, Vietnam: dotGEARS.
Virilio, P. (2002). Speed and Information: Cyberspace Alarm! *CTheory*. Online Publication. www.ctheory.net/articles.aspx?id=72