DHQ: Digital Humanities Quarterly
2013
Volume 7 Number 1

Criminal Code: Procedural Logic and Rhetorical Excess in Videogames

Mark L. Sample <msample1_at_gmu_dot_edu>, George Mason University

Abstract

"Criminal Code: Procedural Logic and Rhetorical Excess in Videogames" explores the code of two videogames, suggesting that reading game code is a fruitful way to enrich our understanding of videogames and the culture they represent. In particular, I show how the code of the open source version of *SimCity* and the controversial first person shooter *JFK: Reloaded* reveals elements of the games unavailable to the player and unaccounted for by other critical readings of those games.

Introduction

We understand games by playing them. Card games, board games, videogames — their idiosyncrasies and dynamics become clear not when we read the instructions, but when we play them. Play is, as Katie Salen and Eric Zimmerman describe it, "free movement within a more rigid structure" [Salen & Zimmerman 2004, 304]. We bump up against this rigid structure — rules, boundaries, mechanics — and reveal the contours of the game, slowly uncovering what Lev Manovich calls its "hidden logic" [Manovich 2001, 222]. This is play. This is how we come to know games, by experiencing them. Like all cultural activities, however, games can be approached through means apart from the experience itself of playing them. Games can be played, but they can also be interpreted. The early years of videogames studies were defined by this tension. Were games mainly about rules, structure, and play? Or did games tell stories and contain allegories? Self-proclaimed ludologists argued for the former, while many others defended the latter. The debate played out in conferences, blogs, and scholarly journals such as *Game Studies* and *Electronic Book Review* (e.g. [Eskelinen 2001], [Aarseth 2004], [Jenkins 2004]). In the past few years the debate has largely dissipated, with most scholars in the field recognizing that no single approach can adequately explore the cultural significance of videogames. Yet I want to advance an approach to videogames that, in light of this reconciliation, might at first appear to be retrograde, a reactionary doubling down on the position that videogames are texts. I don't mean metaphorical texts, using that word haphazardly the way literary scholars like to describe everything as texts. I mean texts in a literal way, comprised of words, or at least, of numbers and letters. I am referring to the code of the game. Videogames are pieces of software, made of code, and I argue that thinking about game code as a signifying text is a fruitful way to enrich our understanding of videogames. To illustrate this methodology I will consider the code of two games — *Micropolis* (2008), which is the open source version of the legendary simulation *SimCity* (1994), and the notorious first-person shooter *JFK: Reloaded* (2004). In particular I will focus on an aspect of the games in which the signifying excess of code is palpable: the procedural logic of crime and the shock of history found in programmer comments in code.

1

My insistence on the importance of code follows a recognition that for too long new media studies has suffered from the symptoms of what Nick Montfort calls screen essentialism [Montfort 2004]. Screen essentialism occurs when the "digital event on the screen," as Matthew Kirschenbaum puts it, becomes the sole object of study at the expense of the underlying software, hardware, storage devices, and even non-digital inputs and outputs that make the digital screen event possible in the first place [Kirschenbaum 2008, 4]. One response to this essentialism is found in Katherine Hayles' call for media-specific analysis of creative works. A media-specific analysis attends to a work's materiality, which Hayles formulates as "the interplay between a text's physical characteristics and its signifying strategies" [Hayles 2004, 74]. A media-specific analysis is an important intervention into screen essentialism, but it privileges the manifestly present

2

elements of media. What about digital works with characteristics beyond the physical — or at least, beyond the visible — that nonetheless influence how that work circulates within culture?

3 Nick Montfort and Ian Bogost's development of platform studies presents one answer to this question. Unlike typical screen-centric approaches to born-digital works, platform studies considers both the software and hardware of systems, exploring the relationship between these hidden foundations and the more visible creative and expressive acts that appear on the surface of the machine. Every platform is a composite of chips, circuits, controllers, software, storage devices, and so on. And each of these individual components offers affordances — capabilities that are both enabling and limiting. A platform studies approach reveals, for example, that the Atari VCS's unique Television Interface Adapter constrained playfield graphics in a way that shaped the look and playability of games themselves, the Atari 2600 adaptation of *Pac-Man* being a dramatic example of this dynamic [Montfort & Bogost 2009, 67–75]. Chips and other hardware constraints also affect what kind of software can be executed on a computer, making computer code a critical part of platform studies.[1] Without code, a computer is inoperable, inert. But the significance of code goes beyond its purely computational power. In a recent examination of the parallel configurations between the development of UNIX and changing race relations in 1960s America, Tara McPherson hones in on code as an especially relevant site of cultural engagement for scholars. Using the UNIX pipe command to dramatize the modularity found in many social fields since 1968 — including urban segregation and academic departments — McPherson demonstrates that code is deeply intertwined with culture. The difficulty is that with its emphasis on modularity and interfaces that distance the programmer from the kernel of the machine, "the structures of code work to disavow these very connections" [McPherson 2012, 36].

4 Textual studies, with its awareness of the social and material history of texts, perhaps comes closest in traditional humanities scholarship to the spirit of both media-specific analysis and platform studies. The concerns of textual studies — attentiveness to physical forms, recognition that hands other than the author's shape the text, an almost forensic desire to trace the history of marginalia, errata, and variants — have their analogs in emerging strains of new media studies. It should not be surprising, then, that I will revisit much earlier notions of what McGann calls "the textual condition" as I examine computer code in a gaming context [McGann 1991, 3]. The late German media theorist Friedrich Kittler has argued that, as Alexander Galloway phrases it, "code is the only language that does what it says" [Galloway 2006, 5]. But my close reading of code insists that code not only does what it says, it says things it does not do. Staking out the territory of a field he calls critical code studies, Mark Marino describes code "as a text, as a sign system" [Marino 2006]. Code may speak to a machine, but it also speaks to us. It is a rich textual object, layered with meaning. "Code may in a general sense be opaque and legible only to specialists," as Rita Raley notes, "…but it has been inscribed, programmed, written. It is conditioned and concretely historical" [Raley 2006]. Here, then, are the two sides of code I will explore as I seek to understand these two videogames — *Micropolis* and *JFK: Reloaded* — as extensions of our textual condition. First, code's performative power, or what may be more accurately called the procedural power of code. Second, code's evocative power, rife with gaps, idiosyncrasies, and suggestive traces of its historical context. Approaching code in this twofold fashion, it becomes clear that the more a programming language emphasizes human legibility, the greater the chance there's some slippage in the code that is readable by the machine one way and readable by scholars and critics in another.

Criminal Code in *Micropolis*

5 Consider the case of *Micropolis*, the open-source version of *SimCity* that was included on the Linux-based XO computers in the One Laptop per Child program. Designed by Will Wright, *SimCity* was released by Maxis in 1989 on the popular (though nearing the end of its life cycle) Commodore 64. It was the first of many popular Sim games, such as *SimAnt* and *SimFarm*, not to mention the enduring *SimCity* series of games — that have been ported (the programming equivalent of translation) to dozens of platforms, from DOS to the iPad. The videogame behemoth Electronic Arts now owns the rights to the *SimCity* brand, and in 2008, EA released the source code of the original game under a GNU GPL License — a General Public License that stresses the freedom of users to share and modify the program's source code. EA prohibited any resulting branch of the game from using the *SimCity* name. The developers, led by Don Hopkins — who had worked on the first Unix port of *SimCity* in 1992 — called it *Micropolis*, which was

Wright's original name for his city simulation [Hopkins 2007a].

6 According to Hopkins, Will Wright was once asked by a journalist how much urban planning theory, demography, criminology, and sociology went into *SimCity.* Wright replied simply, "I just kind of optimized for game play" [Hopkins 2007b]. Indeed, from the beginning, *SimCity* was criticized for presenting a naïve vision of urban planning, if not an altogether egregious one [Bleecker 1995], [Friedman 1995]. As recently as 2007, the legendary computer scientist Alan Kay called *SimCity* a "pernicious…black box," full of assumptions and "somewhat arbitrary knowledge" that cannot be questioned or challenged [Kay 2007]. Kay's and others' critiques of *SimCity* focus on what Ian Bogost calls the "procedural rhetoric" of the game. By procedural rhetoric, Bogost simply means the implicit or explicit argument a computer model makes. Rather than using words like a book, or images like a film, a game "makes a claim about how something works by modeling its processes" [Bogost 2009]. In the case of *SimCity*, I want to explore a particularly rich site of embedded procedural rhetoric — the procedural rhetoric of crime.

7 In fact, Kay illustrates his point about the black box nature of *SimCity* by describing how crime operates in the game. *SimCity*, Kay argues, "gets the players to discover that the way to counter rising crime is to put in more police stations" [Kay 2007]. Of all the possible options in the real world — increasing funding for education, reducing overcrowded housing, building mixed use developments, creating employment opportunities, and so on — it's the presence of the police that lowers crime in *SimCity*. This is the argument that game makes, its procedural rhetoric. Naïve though it may be, the game has staked out a position on urban planning from which it cannot deviate. It doesn't take long for players to figure out the position. Indeed, the original manual itself tells the player that "Police Departments lower the crime rate in the surrounding area. This in turn raises property values" [Bremer 1993, 5]. It's one thing for the manual to propose a relationship between crime, property values, and law enforcement, but quite another for the player to see that relationship enacted within the simulation. Players have to get a feel for it on their own as they play the game. Recall Manovich's comment about the hidden logic of a game. A player's success in a simulation hinges upon discovering the algorithm underlying the game. But if the manual describes the model and players can discover it for themselves through gameplay, what's the value of looking at the code of the game? What can the code reveal that playing the game cannot?

8 To answer this question we must examine the code itself. Because the code for *Micropolis* — *SimCity*'s open source twin — is freely available, this close reading of code is feasible in a way that a similar reading of a commercial game is not.[2]To wit, below are lines 413–424 of span.cpp, one of the many sub-programs called by the core *Micropolis* engine. It's written in C++, a common middle-level programming language; Firefox is written in C++, for example, as well as Photoshop, and nearly every Microsoft product. By paying attention to variable names in span.cpp, even a non-programmer might be able to discern that this code scans the player's city map and calculates a number of critical statistics: population density, the likelihood of fire, pollution, land value, and the function that drew Alan Kay's attention to *Micropolis*, a neighborhood's crime rate.

```
413     for (int x = 0; x < WORLD_W; x += crimeRateMap.MAP_BLOCKSIZE) {
414             for (int y = 0; y < WORLD_H; y += crimeRateMap.MAP_BLOCKSIZE) {
415             int z = landValueMap.worldGet(x, y);
416                     if (z > 0) {
417                             ++numz;
418                             z = 128 - z;
419                             z += populationDensityMap.worldGet(x, y);
420                             z = min(z, 300);
421                             z -= policeStationMap.worldGet(x, y);
422                             z = clamp(z, 0, 250);
423                             crimeRateMap.worldSet(x, y, (Byte)z);
424                             totz += z;
```

9 The manual tells us what this code does. But what does it mean? In the most general sense, these lines generate information for game engine about the game's own state. The first several lines provide the scan mechanism, as the existing crime rate map of the player's city (crimeRateMap.MAP_BLOCKSIZE) is methodically checked along the X and

Y axes. This process alone highlights the quantitative over qualitative nature of computing. The map — and therefore, crime rate — can never be understood by the computer holistically, absorbed and interpreted all at once as a player might see it. Instead, it is checked granularly, numerically, point by point. The computer sees the city as a grid of discrete spaces, even though cities certainly aren't experienced that way by their occupants. The scan mechanism raises a question that we may never have thought to ask before, namely, what is it like to be a city from a computer's perspective?

10 The implications of understanding a cityscape as a dataset become clearer when other aspects of span.cpp are examined. In line 418, the crime rate variable Z starts off at a baseline of 128. This at first seems to be an arbitrary choice, but it is not random at all. The number 128 is (obviously) exactly half of 256, which is (less obviously) the highest 8-bit binary value (2^8) available on the original *SimCity* platform, the Commodore 64. Line 418 is a historical clue, providing a glimpse into the computers of the era. The heart of the Commodore 64 was its MOS Technologies processor, the 8-bit 6502 chip. The 6502 is now legendary for its role in home computing in the 1980s. Released in 1975, the 6502 or some tweaked version of it powered the Atari VCS, the Apple I and Apple II, the Nintendo Entertainment System (NES), and of course, the Commodore 64 [Bogost 2009, 12]. By developing *SimCity* as an 8-bit program tailored to the 6502 CPU (even though 16-bit computers and videogame consoles were beginning to arrive on the scene), Will Wright guaranteed that ports of the game would be compatible with the majority of home computers of the time. But why choose 128 as the baseline crime rate, rather than 0? From a baseline of 128 the crime rate can either go up or down; 128 allows the crime rate to fluctuate negatively or positively without having to deal with negative numbers. But the rate can only ever fluctuate within a predefined limit. Line 422 makes sure of this, using the clamp function of C++ to define the ultimate range of Z (from 0 to 250). In effect, *SimCity* specifies a maximum crime rate. Crime in *SimCity* is, like the discrete coordinates of the city, a rigidly defined quantity.

11 Immediately following the establishment of 128 as baseline in line 418, the land value variable (previously defined in line 415) is subtracted from Z, creating a direct causation between land value and the crime rate. This formula is complicated in the next line, as the population density is added to Z. Finally, with line 421, the number of police stations lowers Z. Track these additions and subtractions from Z, and it's exactly as the manual explained: crime is a function of population density, land value, and police stations, and a strict function at that. But the code makes visible nuances that are absent from the manual's pithy description of crime rates. For example, land that has no value — land that hasn't been built upon or utilized in the player's city — has no crime rate. This shows up in lines 433–434:

```
} else {
   crimeRateMap.worldSet(x, y, 0);
```

12 Because of this strict algorithm, there is no chance of a neighborhood existing outside of this model. The algorithm is totalizing and deterministic, absolutely so. A populous neighborhood with little police presence can never be crime-free. Crime is endemic, epidemic.

13 Land value is likewise reduced to set formula, seen in this equation in lines 264–271:

```
if (landValueFlag) {   /* LandValue Equation */
        dis = 34 - getCityCenterDistance(worldX, worldY) / 2;
        dis = dis <<2;
        dis += terrainDensityMap.get(x >>1, y >>1);
        dis -= pollutionDensityMap.get(x, y);
        if (crimeRateMap.get(x, y) > 190) {
                dis -= 20;
        }
```

14 These lines stipulate that land value is a function of the property's distance from the city center, the type of terrain, the nearby pollution, and the crime rate. Again, though, players will likely discover this algorithm for themselves, or else read about it in the manual, which spells out the formula, explicitly stating that "the land value of an area is based on terrain, accessibility, pollution, and distance to downtown" [Bremer 1993].

Simulation Fever and a Textual Cure

15 Beyond highlighting how the computer reads the city as a set of data points, looking at code of *Micropolis* serves as exit point from the seemingly totalizing power of simulation, which is reinforced by both the manual and gameplay itself. Recall Sherry Turkle's now classic work, *Life on the Screen*, about the relationship between identity formation and what we would now call social media. Turkle spends a great deal of time discussing what she calls the "seduction of the simulation" [Turkle 1995, 71]. Turkle has in mind exactly the kind of simulation whose code appears here, the Maxis games such as *SimCity*, *SimLife*, and *SimAnt* that were so popular 15 or so years ago as well as MUDs and MOOs, text-based precursors to virtual worlds like *Second Life*. Turkle suggests that faced with engaging, immersive simulations, players can respond in several ways. On the one hand, players can surrender themselves totally to the simulation, accepting whatever processes are modeled within. Turkle calls this "simulation resignation." On the other hand, players can reject the world of the simulation entirely — what Turkle calls "simulation denial." These are stark opposites, and our reaction to simulations obviously need not be entirely one or the other. Turkle proposes a third, ideal response: "understanding the assumptions that underlie simulation" and demanding "greater transparency" in simulations, whether they are popular videogames or real world planning simulations [Turkle 1995, 71].

16 Years after *Life on the Screen*, one would be hard-pressed to argue that simulations have become more transparent. If anything, they are more prevalent and yet more invisible, a situation Bogost diagnoses — riffing on Derrida — as simulation fever, a compulsion to render any and all real world processes as a simulation. Bogost explains that a simulation is "a representation of a source system via a less complex system that informs the user's understanding of the source system in a subjective way" [Bogost 2006, 98]. Just as the most detailed map can never match the territory it represents, so too is the simulation bound to be irreconcilably impoverished compared to the system it simulates. Moreover, note Bogost's use of the word "subjective," which suggests a kind of crack in the exterior of the simulation. Awareness of the subjective nature of simulations can expose what Bogost calls the simulation gap, his own version of Turkle's third response to simulations. With every simulation, Bogost argues, there is a "gap between the rule-based representation of a source system and a user's subjectivity" [Bogost 2006, 107]. Exploring this gap is, for Bogost, the most productive way to overcome simulation fever. Bogost observes that simulations are "not exactly like textual or electronic archives" [Bogost 2006, 108], so the simulation gap must be understood by means different than those used to interpret literature or poetry. He imagines that "working through simulation fever means learning how to express what simulations choose to embed and to exclude" as well as learning how "to relate their rules to their subjective experiences and configurations" [Bogost 2006, 109]. But this is what a close reading of code reveals: simulations are indeed texts. The code itself expresses what "simulations choose to embed and to exclude." The close reading of code by non-coders can act as a critical intervention into the simulation gap. Code exposes the guts of the simulation. Reading code exposes the gaps of the simulation. While the manual may explain processes and playing may set them in motion, neither provides the kind of traction gained by seeing the code. Reading the code turns the simulation into a textual object. And as a textual object, it can be taken apart, even tweaked and recompiled with our own algorithms.

Comments in Code

17 When we crack open the code like this, we may well find surprises that playing the game or reading the manual will not tell us. Remember that though code does what it says, it also says things it does not do. Marino calls this characteristic of code its "extra-functional significance" — meaning-making that goes beyond the purely utilitarian commands in the code [Marino 2006]. Extra-functional significance means the code participates in a system of signs beyond those executable by the machine. This extra-functional significance may arise in the layout of the code, which forms meaningful patterns recognizable to the human eye but which are irrelevant to the program's interpreter. Or perhaps in the developers' choice of variable names, such as the "FeministWhore" variable that was discovered in a piece of code in the Steam version of the zombie survival horror game *Dead Island* [John 2011]. Or, quite likely, in the comments developers leave in code.

18 Returning to the example of *SimCity* and *Micropolis*, consider the code for disasters.cpp. Anyone with a passing familiarity with *SimCity* might be able to guess what disasters.cpp does. It's the routine that determines which random disasters will strike a player's city. The entire 408 line routine is worth looking at, but to illustrate the extra-functional

significance of comments in code, examine the section that begins at line 109, where the probability of the different possible disasters is defined:

```
109     if (!getRandom(DisChance[x])) {
110             switch (getRandom(8)) {
111                     case 0:
112                             case 1:
113                             setFire(); // 2/9 chance a fire breaks out
114                             break;
115                     case 2:
116                             case 3:
117                             makeFlood(); // 2/9 chance for a flood
118                             break;
119                     case 4:
120                             // 1/9 chance nothing happens (was airplane
crash,
121                             // which EA removed after 9/11, and requested it
be
122                             // removed from this code)
123                             break;
124                     case 5:
125                                     makeTornado(); // 1/9 chance tornado
126                             break;
127                     case 6:
128                             makeEarthquake(); // 1/9 chance earthquake
129                             break;
130                     case 7:
131                     case 8:
132                             // 2/9 chance a scary monster arrives in a dirty
town
133                             if (pollutionAverage > /* 80 */ 60) {
134                                     makeMonster();
135                             }
136                             break;
```

In the midst of rather generic biblical disasters (22% chance of fire in line 113 and 22% chance of flood in line 117), there is a startling excision of code, the trace of which is only visible in the programmer's comments. These full-line comments in lines 119–123 can be identified by the double slashes that precede them: 19

```
case 4:
      // 1/9 chance nothing happens (was airplane crash,
      // which EA removed after 9/11, and requested it be
      // removed from this code)
      break;
```

These comments reveal that in the original *SimCity* there was a 1 out of 9 chance that an airplane would crash into the city. After 9/11 this disaster was removed from the code at the request of Electronic Arts. 20

Playing *Micropolis*, say, perhaps as one of the children in the OLPC program, this erasure would likely escape notice. It escapes notice because the machine doesn't notice — the comment stands outside the algorithms of the game. It's only visible when we read the code, a clear argument for treating code as a textual document rife with non-performative functions. We could raise any number of questions about this decision to elide 9/11 from *Micropolis*. There are questions, for example, about the way the code is commented. None of the other disasters have any kind of contextual, historically-rooted comment, the effect of which is that the other disasters are naturalized, even the human-made disasters like the Godzilla-like monster that terrorizes an over-polluted city in "case 8" (line 131). There are questions about the One Laptop Per Child program and the white-washing of American history for global audiences. There are questions about the relationship between simulation, disaster, and history that call to mind Don DeLillo's *White Noise*, where one character tells another, "The more we rehearse disaster, the safer we'll be from the real thing….There is no substitute for a planned simulation" [DeLillo 1985, 196]. 21

And finally, there are questions about corporate influence and censorship — was EA's request to remove the airplane crash really a request, or more of a condition? How does this relate to EA's more recent decision in October of 2010 to remove the Taliban from the latest version of *Medal of Honor*? In this case, a controversy erupted when word leaked out that *Medal of Honor* players would be able to assume the role of the Taliban in the multiplayer game. After weeks of holding out, EA ended up changing all references to the Taliban to the generic and unimaginative "Opposing Force" [Bogost 2010]. At least twice, then, EA — and by proxy, the videogame industry in general — has erased history, making it more palatable, or as a cynic might see it, more marketable. 22

Approaching Comments in Code

While these social and historical questions are exactly the kind that critical code studies seeks to ask, it's important to step back and consider the epistemological status of comments in code as well. Comments are ignored by the machine interpreter and readable by humans, but not exactly legible. They are visible only if one is able to view the source code. They are not intended for the end-user, but with the right tools, the end-user might find them. Comments in code thus exemplify McGann's notion of the social private text — and although McGann is primarily discussing poetry, his definition of a text as a "laced network of linguistic and bibliographic codes" would certainly include computer code [McGann 1991, 13]. "Texts are produced and reproduced under specific social and institutional conditions," McGann explains, "and hence that every text, including those that may appear to be purely private, is a social text" [McGann 1991, 21]. Code is certainly produced in a specific social and institutional context, one might even say regime — for example, as a developer for Electronic Arts, coding eight hours a day, six days a week [Wark 2007, 44]. 23

Code is a social text, even the comments that no one other than another developer is meant to see. Questions of *voice* (who speaks), *address* (to whom), and *intention* (and why) all arise, and because these questions have similarly occupied literary scholars, it's tempting to transpose lessons from narratology onto the study of code and comments in code. Even early considerations by computer scientists of the role of comments in code acknowledged the centrality of these narratological questions. A 1976 study of the practice of commenting in code observes that comments require the programmer to approach the program from at least two simultaneous points of view — the coder and the documentor. This study divides comments into two categories of intention: "functional comments" that describe what a piece of code does and "operational comments" that explain how the code performs the function [Sachs 1976]. 24

More recently Jeremy Douglass has convincingly characterized comments in code as a type of paratext, "continuous with and yet set apart from the source" code. Douglass likens code that is commented to a "parenthesis for a different reader" [Douglass 2010] — a typographical metaphor that underscores the essential textuality of comments. Summarizing the way software developers have tended to think about comments in code, Douglass notes that comments serve as either documentation, specification, or metadata. These are all ways of Taylorizing the process of software engineering. Yet with *Micropolis*, it is clear that code comments can serve as either entry or exit points to the game, connecting it with social and historical contexts in a way that has nothing to do with software engineering. 25

It is tempting to think of code comments as a kind of textual marginalia — the notes, corrections, and even doodles that authors and readers add in the margins of texts. Patrick Murray-John, a software developer who also holds a doctorate in Anglo-Saxon literature, has suggested that comments in computer code might have imperfect analogs in the medieval division of marginalia into separate categories. As Murray-John notes, medieval scholars distinguished between *lectio* and *enarratio* [Murray-John 2011]. *Lectio* refers to aids for reading at the level of comprehension — notes and marks that help a reader make the text legible (originally for the purposes of reading aloud). *Enarratio* refers to marginalia that actually help readers interpret the text on a rhetorical and symbolic level — or, to extrapolate to the world of software engineering, on a procedural level. To *lectio* and *enarratio* we might also add, as Whitney Trettien suggests [Trettien 2011], *emendatio*, comments that correct or even offer improvements to the existing text (such as might be found in an open-source project, with one developer making suggestions on another developer's code). These medieval categories of textual annotation do not map perfectly onto categories of comments in code, of course. More than a one-to-one correspondence, these categories can provide a starting point to begin to differentiate between kinds of comments in code. All comments in code are extra-functional, but not all comments in code are extra-functional in the same way. Categories such as *lectio*, *enarratio*, and *emendatio* can highlight the range of extra-functional significance 26

found in code comments. This range is exemplified in the code of *JFK: Reloaded*, a game that perfectly embodies the contradictions between the playable algorithms of a game and the internal and usually invisible signifying structures of code.

Rhetorical Excess in *JFK: Reloaded*

27 Traffic's 2004 *JFK: Reloaded* is premised upon a player's ready resignation (to use Turkle's terminology) to the simulated world, in this case Dallas, 1963. The game is rooted soundly in an identifiable — and for many of its outraged detractors — experienced event: the assassination of President Kennedy. The videogame is a first-person shooter in which that first-person happens to be Lee Harvey Oswald. The goal of *JFK: Reloaded* is to reenact the shooting at Dealey Plaza with as much fidelity as possible to the findings of the Warren Commission Report (i.e. Oswald acted alone, firing three bullets from a single rifle, from the sixth floor of the Texas School Book Depository).

28 Upon its release in 2004, Traffic labeled *JFK: Reloaded* a "docugame" and most critical readings of the game continue to see it as some form of interactive documentary. Tracy Fullerton places the game squarely within Michael Renov's classic definition of documentary media, as the game "interrogates" the past [Fullerton 2008]. More recently, Bogost, Ferrari, and Schweizer suggest in *Newsgames* that *JFK: Reloaded* is a very specific kind of documentary. Bogost et al. describe three different "playable realities" that documentary videogames can simulate in the name of experiencing or understanding the past: a spatial reality, in which players explore the physical environment of a historical event (a recent example would be Osama Bin Laden's Abbottabad compound, modeled in *Counter Strike: Source*); an operational reality, which recreates specific events, hewing to the historical record; and a procedural reality, which models "the behaviors underlying a situation, rather than merely telling stories of their effects" [Bogost et al. 2010, 69].

29 According to this framework, *JFK: Reloaded* presents an operational reality. Players recreate in a structured, limited way (and guided by their own knowledge of the assassination) the essential operations of the assassination: waiting for the motorcade from a hidden perch, sighting the rifle onto Dealey Plaza, and firing into the presidential motorcade. However, as Bogost notes in *Persuasive Games*, a work that precedes *Newsgames*, there is more to *JFK: Reloaded* than the simple attempted recreation of a historical event. It is nearly impossible to "win" the game, by which I mean match the Warren Commission findings. This impossibility, Bogost suggests, contributes to the game's procedural rhetoric, a kind of procedural reality layered upon (or beneath) the more explicit operational reality. As Bogost puts it, "the developer's stated goal [of reaffirming the Warren Commission Report] was a ruse" [Bogost 2007, 132–133]. And in fact, I would add, the game highlights its proclaimed goal's exact opposite — the improbability of the Warren Commission's findings.

30 Whether or not the procedural rhetoric of *JFK: Reloaded* supports or refutes the Warren Commission Report, it's possible to find traces of another narrative by examining the code of the game. To be more precise, consider the comments that appear in one of the two WAD files that comprise *JFK: Reloaded*'s game assets. An acronym for Where's All the Data?, a WAD file is a collection of individual sounds, sprites, level information, NPC (non-playable character) behavior, and other often customizable game data. Originally used in id Software's *Doom* (1993), WADs or similar composite files are now commonplace in many PC games.

31 In the case of *JFK: Reloaded*, opening up the core000.wad in a text editor reveals mostly binary codes that look like junk (because they ought to be opened in a hexadecimal editor rather than a plain text editor). But beginning with line 224,070, there is a chunk of plain text code that resembles XML structured data, accompanied by full-line programmer comments.[3] For example, the following lines specify the actions that should occur when a generic character is fatally hit by Oswald's rifle:

```
// Generic character's killed action
// ————————————————————————
// This is the action that a character takes when they should die if
// they've got no special animation for it
// ————————————————————————
[ACTION]
<NAME>
```

```
PersonKilled
<DIE>
0
0
<RAGDOLL>
```

The code comments — the marginalia — in *JFK: Reloaded*'s WAD file at first blush resemble *enarratio*, not only helping readers to make sense of the individual lines of code that follow the comment, but also offering an interpretative gloss on the code. But quickly the comments move from *enarratio* into a perverse reworking of the fourth category of medieval annotation — *judicium*, or judgments upon the "aesthetic qualities or the moral and philosophical value of the text" [Parkes 1999, 90]. For example, in the following snippet, Jackie Kennedy's actions are defined: 32

```
// ——————————————————————
// Jackie cradling JFK before the money shot
//——————————————————————
[ACTION]
<NAME>
JackieCradleJFK
<CONCERNED>
0
0
```

The comment provides the context for these lines of code (*enarratio*) but the casual use of the overtly sexual phrase "money shot" is an implicit judgment (*judicium*) not on the code, but on the historical event itself. A phrase inexorably linked to pornography, "the money shot" comment is sudden, unsettling, and, like all comments in code, extra-functional. While it's tempting to suggest that the comment sheds some light on the developers' attitude toward the subject matter (what we'd call the author's "tone" in literary studies), it is not necessary to do so. The comment is structurally unnecessary, but that doesn't mean it doesn't *mean*. Its rhetorical excess spills over, making this supposedly private text palpably social. A money shot commonly refers to the climactic scene of male ejaculation in a pornographic film, so named by the pornography industry because it's typically the most expensive scene to shoot. It is also, as Linda Williams argues, the "most representative instance of phallic power and pleasure" in a hard-core film [Williams 1999, 95]. Williams positions the money shot as a dense collision point between commodity fetishism, visual desire, and anxiety over women's "invisible and unquantifiable pleasure" [Williams 1999, 113]. In more ways than one, the money shot is "a substitute for what cannot be seen" [Williams 1999, 95]. 33

The money shot comment in *JFK: Reloaded* reenacts this substitution and reverses it. Consider the comment closely: "Jackie cradling JFK before the money shot." While Kennedy is nominally the target in *JFK: Reloaded*, describing the fatal gunshot as a money shot substitutes Jackie as the intended target. In heteronormative pornography, it is the man who delivers the money shot and a woman who receives it. Jackie thus becomes an unwilling participant in a death scene that is pornographic in nature, while the player is given license to fulfill the role of the male performer in a pornographic film. In other words, imagining Oswald's bull's-eye as a money shot makes it okay to *try* to hit Jackie. Jackie Kennedy is not collateral damage but in fact *was the target all along*. Furthermore, like a money shot in a hard-core film, the event is commodified and repeatable. Yet the programmer's comment also reverses the usual dynamic of a money shot, which makes visible (ejaculation) what would otherwise be hidden in intercourse. In the case of *JFK: Reloaded*, though, it is the money shot comment itself that is meant to be hidden, off limits to the player. But it's still there. It means something. It cannot not mean something. 34

A textual imagination must reckon with the comment. Once it is known, it cannot be unknown. To ignore the comment means overlooking the kind of evidence that historians and textual scholars have long used to create a more complex understanding of our cultural heritage. At the very least — the very, very least — the strictly objective perspective that we falsely associate with documentary media dissolves here. The misogynistic tone of the comments becomes even more troubling in the following snippet of code, which defines Nellie Connally's actions when her husband, Texas Governor John Connally, is shot: 35

```
// ——————————————————————
// Nelly shoving Connally's bonce down into her minge,
```

```
// in a last desparate attempt to get oral sex out of
// him before he croaks
//_____________________________
[ACTION]
<NAME>
NellyShoveConnally
<CONCERNED>
0
0
```

In three lines of code commentary, the developers at Traffic absolutely undermine the entire stated pedagogical project of their docu-game. Their outwardly respectful “interactive reconstruction of John F. Kennedy’s assassination” is undone by inaccuracies (the suggestion that Connally “croaks”) and misspellings (Nelly for Nellie, “desparate”) but even more so by the explicitly pornographic and misogynistic reframing of this traumatic event. This comment marks the second time a grieving woman protecting her injured husband is portrayed in *JFK: Reloaded* as transgressively sexual in a moment of pain and suffering. While the <concerned> variable initiates an in-game routine appropriate for the situation, the hidden code retells the story almost as a snuff film. The governor’s critical wound is met with arousal, and Nellie’s desperation is sexual, not emotional. Simulating the direst of situations — a national tragedy but also a personal one — this first-person shooter collapses sex onto violence. They are indistinguishable from each other, but only in the comments. 36

Code and Paracode

Given the rhetorical excess of these comments, it’s worth thinking about them as more than paratext. I want to suggest the idea of *paracode*. In textual studies, of course, para-, as in paratext, is what Genette calls the “threshold” of the text, the “zone between text and off-text” [Genette 1997]. Paratext includes all the textual apparatus at the edge of a book — indices, acknowledgments, and so on. *Paracode* likewise includes the apparatus at the edge of code, the comments chief among them. But there’s another meaning of “para” I want to evoke. It comes from the idea of paracinema, introduced by the film theorist Jeffrey Sconce. Paracinema is a kind of “reading protocol” that valorizes what most audiences would otherwise consider to be cinematic trash [Sconce 1995]. The paracinematic aesthetic redeems films that are so bad that they actually become worth watching. Following Sconce’s work, the videogame theorist Jesper Juul has wondered if there can be such a thing as paragames — illogical, improbable, and unreasonably bad games. Such games, Juul suggests, might teach us about our tastes and playing habits, and what the limits of those tastes are [Juul 2009]. Along the same lines, *paracode* is code so excessive or remarkable that it becomes productive to fully engage with it. The example of the missing airline disaster in *Micropolis* is a noteworthy example. The algorithm for the disaster is there, present in the code, but commented out, in a kind of Derridean erasure. The paracode leaks out from the code, and it is up to cultural critics to make sense of it. 37

The paracode of *JFK: Reloaded* is even more startling. In addition to undermining Traffic’s official rationale for the game, the paracode complicates the arguments of the critics who defended the game. Access to the code allows us to write a revisionist history of *JFK: Reloaded*. Some might argue that the marginalia of the game, which was never intended to be available to the player, should not guide our interpretation of the game. But this is precisely why code must become a site of engagement for humanists. Recall McPherson’s argument that not only is code a hidden marker of social relations, code hides the means by which itself operates. As I have demonstrated here, approaching code on a textual level as opposed to a procedural level exposes some of these transparent dynamics. In the final analysis, my exploration of this particular pair of games — *Micropolis* and *JFK: Reloaded* — is secondary to the broader questions of critical code studies. What does an attentiveness to code, or even comments in code, mean for literary scholars and cultural historians who study digital artifacts? Code is too important to be left to coders, and it will increasingly be necessary for humanists to develop some degree of what Michael Mateas has called “procedural literacy.” An accomplished new media designer himself, Mateas describes procedural literacy as “the ability to read and write processes, to engage procedural representation and aesthetics, to understand the interplay between the culturally-embedded practices of human meaning-making and technically-mediated processes” [Mateas 2008, 1]. Like any reading literacy, procedural literacy has different thresholds, each requiring greater proficiency. I have argued elsewhere 38

for the notion of computational competency over literacy [Sample 2012], but the point remains: even the most modest efforts to "engage procedural representation" can yield rewarding results. In the examples of *Micropolis* and *JFK: Reloaded* we encounter programmer comments that in no way help us to understand what Bogost would call the procedural rhetoric of the games. Yet, they do help us to develop an understanding of the games as cultural objects and of coding itself as a cultural practice. Especially in the case of *JFK: Reloaded*, the comments are an expression of the male-dominated, conquest-driven *milieu* of contemporary gaming and coding. At the very least, they show that the "inside" of software does not always match the "outside." And ultimately, a thorough engagement with the textual condition of software reveals that procedural literacy must not be strictly limited to reading or writing code, but must also extend outward to language and cultural practices.

Notes

[1] A common criticism of platform studies is that the methodology is overly technologically deterministic: the chip determines the code, the code determines the program, and the program determines what its users do with it. Bogost and Montfort are staunchly opposed to such "hard" technological determinism, arguing that in fact "people make negotiations with technologies as they develop cultural ideas and artifacts, and people themselves create technologies in response to myriad social, cultural, material, and historical issues" [Bogost 2009, 2]. Indeed, platform studies is deeply concerned with the historical and cultural context of any platform. Consider any one of the five layers Bogost and Montfort include in platform studies (platform, code, form and function, interface, and reception and operation), and it's clear that each layer is not only dependent on the layers below, but also on the social context that presses in from every side. Every platform is a product of its times, quite literally. And every piece of software too is a historical document.

[2] Free software operates in a gift economy. Users and software developers may be the immediate beneficiaries, but in a larger sense, the broader public benefits, as free software dramatically reconfigures existing and entrenched knowledge, cultural, and economic power structures [Kelty 2008]. More to the point here, open software is a gift to scholars who can bring their own disciplinary tools to the investigation of what would otherwise be occult knowledge, accessible only to a privileged few individuals.

[3] Credit for discovering the chunks of legible code in the *JFK: Reloaded* WAD files goes to an internet forum user known by the name BrooksMarlin, who first noticed them in 2004. Nobody to my knowledge has ever read these comments against the game itself and against the existing scholarship on the game. See BrooksMarlin 2004.

Works Cited

Aarseth 2004 Aarseth, Espen. "Genre Trouble: Narrativism and the Art of Simulation". In Noah Wardrip-Fruin and Pat Harrigan, eds., *First Person: New Media as Story, Performance, and Game*. Cambridge, MA: MIT Press, 2004. pp. 45-55.

Bleecker 1995 Bleecker, J. "Urban Crisis: Past, Present, and Virtual". *Socialist Review* 24 (1995), pp. 189-221.

Bogost & Montfort 2009 Bogost, Ian, and N. Montfort. "Platform Studies: Frequently Questioned Answers". Presented at *Digital Arts and Culture, UC Irvine* (2009). http://escholarship.org/uc/item/01r0k9br.

Bogost 2006 Bogost, Ian. *Unit Operations: An Approach to Videogame Criticism*. Cambridge: MIT Press, 2006.

Bogost 2007 Bogost, Ian. *Persuasive Games: The Expressive Power of Videogames*. Cambridge: MIT Press, 2007.

Bogost 2009 Bogost, Ian. *The Proceduralist Style*. *Gamasutra*. 2009. http://www.gamasutra.com/view/feature/3909/persuasive_games_the_.php?print=1.

Bogost 2010 Bogost, Ian. *Free Speech is Not a Marketing Plan*. *Gamasutra*. 2010. http://www.gamasutra.com/view/feature/6158/persuasive_games_free_speech_is_.php.

Bogost et al. 2010 Bogost, Ian, S. Ferrari and B. Schweizer. *Newsgames: Journalism at Play*. Cambridge: MIT Press, 2010.

Bremer 1993 Bremer, M. *SimCity User Manual*. Orinda: Maxis, 1993.

BrooksMarlin 2004 BrooksMarlin. *JFK Reloaded (thing)*. *Everything2.* November 23 2004. http://everything2.com/user/BrooksMarlin/writeups/JFK+Reloaded.

DeLillo 1985 DeLillo, D. *White Noise*. New York: Penguin, 1985.

Douglass 2010 Douglass, Jeremy. “Comments on Comments in Code”. Presented at *Critical Code Studies*, sponsored by University of Southern California (2010). http://thoughtmesh.net/publish/369.php.

Eskelinen 2001 Eskelinen, Markku. “The Gaming Situation”. *Game Studies* 1: 1 (2001).

Friedman 1995 Friedman, T. “Making Sense of Software: Computer Games and Interactive Textuality”. In S.G. Jones, ed., *CyberSociety: Computer-Mediated Communication and Community* . Thousand Oaks: Sage Publications, 1995. pp. 73-89. http://www.duke.edu/~tlove/simcity.htm.

Fullerton 2008 Fullerton, T. “Documentary Games: Putting the Player in the Path of History”. In Z. Whalen and L.N. Taylor, eds., *Playing the Past: Nostalgia in Video Games and Electronic Literature* . Nashville: Vanderbilt University Press, 2008.

Galloway 2006 Galloway, Alexander R. *Gaming: Essays on Algorithmic Culture*. Minneapolis: University of Minnesota Press, 2006.

Genette 1997 Genette, Gerard. *Paratexts: Thresholds of Interpretation*. Translated by Jane E. Lewin. Cambridge: Cambridge University Press, 1997.

Hayles 2004 Hayles, N.K. “Print is Flat, Code is Deep: The Importance of Media-Specific Analysis”. *Poetics Today* 25: 1 (2004), pp. 67-90.

Hopkins 2007a Hopkins, D. *History and Future of OLPC SimCity / Micropolis*. *Don Hopkins.* 2007. http://www.donhopkins.com/drupal/taxonomy_menu/4/49/66.

Hopkins 2007b Hopkins, D. *SimCity Rules*. *Don Hopkins.* 2007. http://www.donhopkins.com/drupal/node/145.

Jenkins 2004 Jenkins, Henry. “Game Design as Narrative Architecture”. In Noah Wardrip-Fruin and Pat Harrigan, eds., *First Person: New Media as Story, Performance, and Game*. Cambridge, MA: MIT Press, 2004.

John 2011 John, T. *Misogyny in code is still misogyny*. *Tracey Writes Stuff.* 2011. http://traceyjohn.blogspot.com/2011/09/misogyny-in-code-is-still-misogyny.html.

Juul 2009 Juul, J. *Paragaming: Good Fun with Bad Games*. *The Ludologist*. 2009. http://www.jesperjuul.net/ludologist/?p=732.

Kay 2007 Kay, A. *Kay, A. Email to Don Hopkins*. 2007. http://www.donhopkins.com/drupal/node/134.

Kelty 2008 Kelty, C.M. *Two Bits: The Cultural Significance of Free Software*. Durham: Duke University Press, 2008. http://twobits.net/.

Kirschenbaum 2008 Kirschenbaum, Matthew. *Mechanisms: New Media and the Forensic Imagination*. Cambridge: MIT Press, 2008.

Manovich 2001 Manovich, Lev. *The Language of New Media*. Cambridge: MIT Press, 2001.

Marino 2006 Marino, Mark. “Critical Code Studies”. *Electronic Book Review* 4 (December 4 2006). http://www.electronicbookreview.com/thread/electropoetics/codology/.

Mateas 2008 Mateas, M. “Procedural Literacy: Educating the New Media Practitioner”. In D. Davidson, ed., *Beyond Fun: Serious Games and Media*. Pittsburgh: ETC Press, 2008. pp. 67-83.

McGann 1991 McGann, Jerome. *The Textual Condition*. Princeton: Princeton University Press, 1991.

McPherson 2012 McPherson, Tara. “U.S. Operating Systems at Mid-Century: The Intertwining of Race and UNIX”. In Lisa Nakamura and P. Chow-White, eds., *Race After the Internet*. New York: Routledge, 2012. pp. 21-37.

Montfort & Bogost 2009 Montfort, Nick, and Ian Bogost. *Racing the Beam: The Atari Video Computer System*. Cambridge: MIT Press, 2009.

Montfort 2004 Montfort, Nick. “Continuous Paper: The Early Materiality and Workings of Electronic Literature”. Presented at *Modern Language Association Convention*, sponsored by Modern Language Association (December 28 2004). http://nickm.com/writing/essays/continuous_paper_mla.html.

Murray-John 2011 Murray-John, P. *Twitter reply to @samplereality*. *Twitter*. 2011. https://twitter.com/#!/patrick_mj/status/71568034640306176.

Parkes 1999 Parkes, M.B. “Reading, Copy, and Interpreting a Text in the Early Middle Ages”. In G. Cavallo and R. Chartier,

eds., *A History of Reading in the West* . Amherst: University of Massachusetts Press, 1999. pp. 90-102.

Raley 2006 Raley, Rita. “Code.surface || Code.depth”. *Dichtung-Digital* 36 (2006). http://www.dichtung-digital.org/2006/1-Raley.htm.

Sachs 1976 Sachs, J. “Some comments on comments”. *ACM SIGDOC Asterisk Journal of Computer Documentation* 3: 7 (1976), pp. 7-14.

Salen & Zimmerman 2004 Salen, Katie, and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. Cambridge, MA: MIT Press, 2004.

Sample 2012 Sample, Mark. *5 BASIC Statements on Computational Literacy*. *SAMPLE REALITY*. 2012. http://www.samplereality.com/2012/05/19/5-basic-statements-on-computational-literacy/.

Sconce 1995 Sconce, J. “‘Trashing’ the academy: taste, excess, and an emerging politics of cinematic style”. *Screen* 36: 4 (1995), pp. 371-393.

Trettien 2011 Trettien, W. *Reply to @samplereality*. *Twitter*. 2011. https://twitter.com/#!/whitneytrettien/status/71575703077916673.

Turkle 1995 Turkle, S. *Life on the Screen: Identity in the Age of the Internet*. New York: Simon and Schuster, 1995.

Wark 2007 Wark, M. *Gamer Theory*. Cambridge: Harvard University Press, 2007.

Williams 1999 Williams, L. *Hard Core: Power, Pleasure, and the Frenzy of the Visible*. Berkeley: University of California Press, 1999.