Game Histories
edited by Henry Lowood and Raiford Guins


Zones of Control: Perspectives on Wargaming, edited by Pat Harrigan and Matthew Kirschenbaum, 2016

The MIT Press
Cambridge, Massachusetts
London, England
Copyright 2016
Code is a set of instructions to a computer. Code is more or less readable by both humans and machines under certain conditions and for different purposes. Programmers write code; machines enact it. In between these two stages, the original code—called the source code—often needs to be compiled. The compiler itself is a computer program, and it translates the lines of legible commands, conditionals, variables, functions, and loops into machine language, a sequence of numbers in hexadecimal or binary notation that the computer can understand. These numbers essentially tell the computer when to add and by how much. This is what even the most sophisticated software comes down to—combining ones and zeroes to manipulate the logic of a circuit. Video games are no different in this regard from word processors, spreadsheets, email programs, or web browsers. All start out as source code, and they eventually become executable programs. If the program is a video game, it runs on a gaming platform, which might be anything from a handheld Tamagotchi to the latest console system in the living room. Or a smartphone or a personal computer or even a calculator.

Code may appear arcane to nonprogrammers or be utterly inescrutable in its compiled form. Because of these occult properties, code is often privileged over other aspects of software design. As Wendy Hui Kyong Chun (2008) puts it, there is a tendency to fetishize source code. The attention that game designer Jordan Mechner received when he rediscovered and then released his original Apple II assembly code for *Prince of Persia* (Braderbund, 1989) is a testament to this fetishization (Mastrapa 2012). The availability of the source code for Gabriele Cirulli’s game 2048 (2014) has likewise spawned dozens of so-called clones (Ballard 2014), as if code alone—and not gameplay, mechanics, and a compelling balance of ease and difficulty—were all that is required for a hit video game. The term clone itself suggests there is something genetically essential to game code, the software equivalent of DNA. Understood this way, game code implies a vast, hidden universe under the surface of the game, difficult to crack, a cryptic mystery. When it comes to games, it’s code all the way down.
Indeed, one of the earliest video games was about code. Darwin (1961) was a "game between computer programs as programs" (Aleh Null 1972, 93). Created by Victor A. Vyssotsky, Robert Morris Sr., and M. Douglas McIlroy at Bell Telephone Laboratories, Darwin set small computer programs in competition to take over each other's virtual environment. Each program attempted to PROBE and CLAIM memory blocks in an "arena" on an IBM 7090. Memory blocks occupied by an opponent could be eliminated with a KILL command. The first program to fill up the available memory in the arena won the game, a ludic inversion of the troublesome memory-leak problem that can plague sloppy code. Darwin was only the first in a long tradition of code-driven games and more subtly, as in the procedurally based game Minecraft (Mojang, 2011), of games that thematized code either overtly, as in the cult classic Codex War (D. G. Jones and A. Dewdney, 1984) or, more subtly, as in the procedurally based game Minecraft (Mojang, 2011), of games that thematized code either overtly, as in the cult classic Codex War (D. G. Jones and A. Dewdney, 1984) or, more subtly, as in the procedurally based game Minecraft (Mojang, 2011).

A more recent though similarly obscure video game also appears to insist on the primacy of code. Yet a close look at the game challenges the very idea of code as a stable property of the surface of a game. The Naked Game appears to expose the ActionScript code that powers the game. The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1).

In a rare kind of evasion, making available to the player what is usually kept below the surface of a game, The Naked Game appears to expose the ActionScript code that powers the game. The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1). The game's initial state variables appear to the right of the retro black game (see figure 7.1).

For example, clicking the line of code Ball._x_+=ball1dx; removes that line from the program's algorithm, resulting in a ball that no longer moves along the x axis. The ball will now move only along the y axis, up and down in a straight line on the screen. Clicking the line again reinstates the code, normalizing gameplay. Turning off lines of code leads to similar unsettling effects. Eliminate line 18, the line that begins if (Ball._y < PLAYFIELD_MINY), and suddenly the top and bottom "walls" of the game no longer matter and the ball ricochets out of the playing field, right down into the code itself. Toggling some code a few lines down transforms either player's paddle into a phantom, which the ball simply passes through.

![Figure 7.1](http://www.retrogdev.com/MiscGames/NakedGame/TheNakedGame.html)

The Naked Game is what Ian Bogost might call a "throwaway," a game meant to be played once and then tossed aside (2011, 96). After all, it takes only a few minutes to explore the possibility space of The Naked Game. The game is also a throwaway in the sense that it is no longer readily available because the developer's website has long since disappeared—though a playable version of the game lives on in the Internet Archive. But even discarded or forgotten games can reveal a great deal about video games and in this case about video game code. The Naked Game clearly emphasizes the role of computer code in making video games what they are: rule-based interactive systems. Like Darwin, The Naked Game is in some sense a game about...
programming, turning the kind of glitch that can mar the seamless play of a video game into the game itself.

The Naked Game is also instructive in determining what counts or does not count as code. When a player toggles a line of code on the screen, has the code inside the game actually changed? Is the code on the surface of the game in fact the code inside the game? Decompiling NakedGame.swf—that is, unpacking the executable Flash file into lines of source code—provides an answer. Table 7.1 compares the first lines of playable code to the corresponding ActionScript code in NakedGame.swf.

The program operates by scanning the 17 lines of playable code (the LinesActive array in the ActionScript), determining whether the line is active or not. If the matching line is active—that is, not clicked by the player—then the program updates the variable in that line (for example, Ball._x = Ball._x + balldx). In other words, toggling off a line of code does not change the code of the game so much as it causes a variable to default to its previous value. The playable code is not code; rather, it is an interface to the variables of the game.

An important conclusion about code can be drawn from the example of The Naked Game: code is not a monolithic entity. What is commonly referred to as code includes a number of software—not to mention epistemological—dimensions:

1. Code is frequently interpreted to mean the rules of the game (Galloway 2006, 35; Manovich 2001, 222). This conflation crystallizes in The Naked Game, for when the player changes the "code," what is being changed is really the rules.

2. Code can be a loaded signifier aimed at human readers or players instead of at a machine. The onscreen code of The Naked Game is evocative of code, not literally code. Mez Breeze’s or John Cayley’s codework provides provocative examples of nonfunctioning pseudo-code outside the realm of video games (Raley 2002).

3. Functional code might also include what Mark Marino calls "extrafunctional significance" (2006). Extrafunctional significance means the code participates in a system of signs beyond those executable by the machine. This extrafunctional significance might be found in a developer’s choice of a variable name, such as BAT_WIDTH, which hints at The Naked Game’s British origins, likely referencing a cricket bat, or the more troubling “FeministWhore,” discovered in the code of the zombie survival horror game Dead Island (Teschland, 2011) (John 2011). Extrafunctional significance might also occur in the comments in the code, meant to be read only by other programmers on the development team (Sample 2013).

4. Code can also include nonexecutable digital assets such as plain text, images, movies, texture maps, avatars, sounds, and music. Decompiling NakedGame.swf reveals a host of assets (see figure 7.2) that have been bundled in the package, including sprites of the paddles and ball as well as the audio file. Flash’s packaging of assets in an executable file is relatively straightforward. Other video games may use other methods, including compressing assets in a separate file, such as WAD (“works as designed”) or VPK (Valve Package) files, or as stand-alone files referenced by the main program. In all of these cases, the media assets are not strictly speaking instructions to the computer, but they are composed of code—ones and zeroes.

5. Finally, code may also refer to reconstructions of the game the player makes outside of gameplay. These reconstructions include modifications, hacks, and cheat codes (Galloway 2006, 13). The famous Konami Code on the Nintendo Entertainment System game controller (up, up, down, down, left, right, left, right, B, A) is not code on the level of source code, but it nevertheless plays on the occult mystique of code, changing the properties of the game through an arcane process (Consalvo 2007, 29).

Taken together, these five supplements to the definition of code teach an important lesson: it is a mistake to think about code as always and only instructions to a computer.

This situation is complicated by an exigency of modern software development: software rarely speaks directly to hardware; it speaks instead to a container piece of software within which it resides, which is known as a virtual machine. For example, Flash functions as a virtual machine within a browser. The case of The Naked Game raises an existential question that applies to nearly all modern video games: What counts as the code of a game? If code isn’t in the playable code on the screen, is it the NakedGame.swf file, the web browser, the Flash extension needed to run the game? Or is it in the operating system of the computer itself? Faced with these propositions, one might readily see The Naked Game as circumstantial evidence that games are indeed code all the way down.

Yet this is not the case, and The Naked Game helps to explain why. Recall that The Naked Game is, as the developers put it, a “primitive version of ‘Pong.’” It’s unclear whether “primitive” is

---

Table 7.1

<table>
<thead>
<tr>
<th>Playable Code</th>
<th>ActionScript Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>function update()</td>
<td>function update()</td>
</tr>
<tr>
<td></td>
<td>if (LinesActive[1])</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>Ball._x = Ball._x + balldx;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>if (LinesActive[2])</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>Ball._y = Ball._y + balldy;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
There are at least three answers to this question, and they all stem from the way code arises from and yields to certain physical exteriorities and material conditions. To begin with, code occupies space, and not merely space as a computer understands it—in RAM (random-access memory) and ROM (read-only memory). Code is usually displayed on a screen, but it can be printed, and sometime it must be. The prototypical video game Spacewar! (Russell/MIT, 1962) was punched into the equivalent of twenty-seven pages of paper tape, the storage medium for the Programmed Data Processor-1 computer (Levy 2010, 63). The 80-column IBM punch card was another version of printed code, in use well past the heyday of batch processing (Lubar 1992). Early handheld electronic games powered by repurposed calculator chips, such as a Mattel’s Football (1977), relied on instructions punched into these IBM cards and assembled on a mainframe computer (Lesser 2007).

Code of course occupies digital space as well. The final code for Mattel’s Football occupied 512 bytes of ROM, or half a kilobyte. That’s roughly one-eighth the size of Mattel’s contemporaneous games on the Atari Video Computer System (VCS), which used cartridges with 4 kilobytes of ROM (Montfort 2006). As video games became more data and process intensive, their underlying code grew as well. Modern video games are such massive projects that large chunks of code are frequently reused (Bogost 2006, 62). These shared codebases are called “game engines,” such as id Software’s Quake engine (1996) and Epic’s Unreal Engine (1998). Consisting of millions of lines of code themselves, these game engines manage the rendering and physics of a game’s virtual world, ostensibly freeing the developers to focus on gameplay and narrative.

Code often costs money. Sometimes it is given away for free (to wit, the Quake engine is open source and free, whereas the development of commercial games with Epic’s Unreal Engine requires a monthly subscription). Either way, code counts as intellectual property and is copyrightable. Copyright provides the copyright holder with exclusive legal rights pertaining to the copying or distribution of the copyrighted object, a status conferred on software by US copyright law in 1980 (Kelty 2008, 183). Software is also frequently subject to End User License Agreements (EULAs), legal contracts players agree to when they open a software package or install a game. A typical EULA, such as the license agreement for Grand Theft Auto 5 (2013) from Rockstar Games, requires the player to agree not to “reverse engineer, decompile, disassemble, display, perform, prepare derivative works based on, or otherwise modify the Software, in whole or in part” (Rockstar Games 2013). Game developers occasionally release the source code of their games under alternative copyright licenses, which allow players to do precisely what Rockstar disallows. Quake, for example, is now under a GNU Public License meant to be ironic, but it surely must be taken as so, for here is a mostly forgotten detail about Pong, a legendary game that by all accounts launched the video game industry: Pong had no code. Charles Babbage’s theoretical Analytical Engine had more of a code base than Pong. In code, Charles Babbage’s theoretical Analytical Engine had more of a code base than Pong. In fact, Pong was so “primitive” that it had no microprocessor. Atari engineer Al Alcorn designed Pong using transistor-to-transistor logic, in which hardwired transistors completed the logic, performing the calculations necessary for the game (Lowood 2009). Pong is just one of many games that required no code; the game was preceded by a half-century’s worth of electromechanical arcade games that required no code. Pinball machines, shooting games, racing games—all without code (Wolf 2008, 35). If video games can exist without code, then what more ought to be said about code and video games?
Finally, one more dimension of code must be acknowledged. It is the greatest material. A code is made by people. Development teams range from a single person to massive multinational companies such as Ubisoft and EA that dominate the video game industry, most of these people are male, and there is roughly a 9,000 pay gap between men and women programmers (Dyer-Witheford and De Peuter 2009). During the crunch time before a game's release, the developers may work 50 to 60 hours a week, a working condition made possible by the relatively young age of the workers. The average worker in the industry is 31 years old, and fewer than 1 in 4 have children. Burnout is common in the industry; on average, programmers those surveyed have children. The work of the men and women who write the code that powers the games recapitulates the work of the men and women who do.

The work of the men and women who write the code that powers the games recapitulates the work of the men and women who do. The code is invisible, as are the people one of the defining features of code. A few exceptions aside, code is invisible, as are the people who write it. The legendary computer scientist Alan Kay once called SimCity (Maxis, 1989) the "Black Box." Full of assumptions and "Somewhat Arbitrary Knowledge" that a "PEXKS Free Flash DeComposer" makes decompilation a relatively trivial task. See http://www.free-decomposer.com/flash/.

For an overview of the concepts "Data intensive" and "Process intensive," see Crawford (2003), 89–90.

This EULA suggests an alternative way of defining code. For example, the code can be reverse engineered, decompiled, disassembled, displayed, performed, used as the base for derivative works, or otherwise modified.


Wolf, Mark J. P. 2008. The Video Game Explosion: A History from PONG to PlayStation and Beyond. Westport, CT: Greenwood Press.