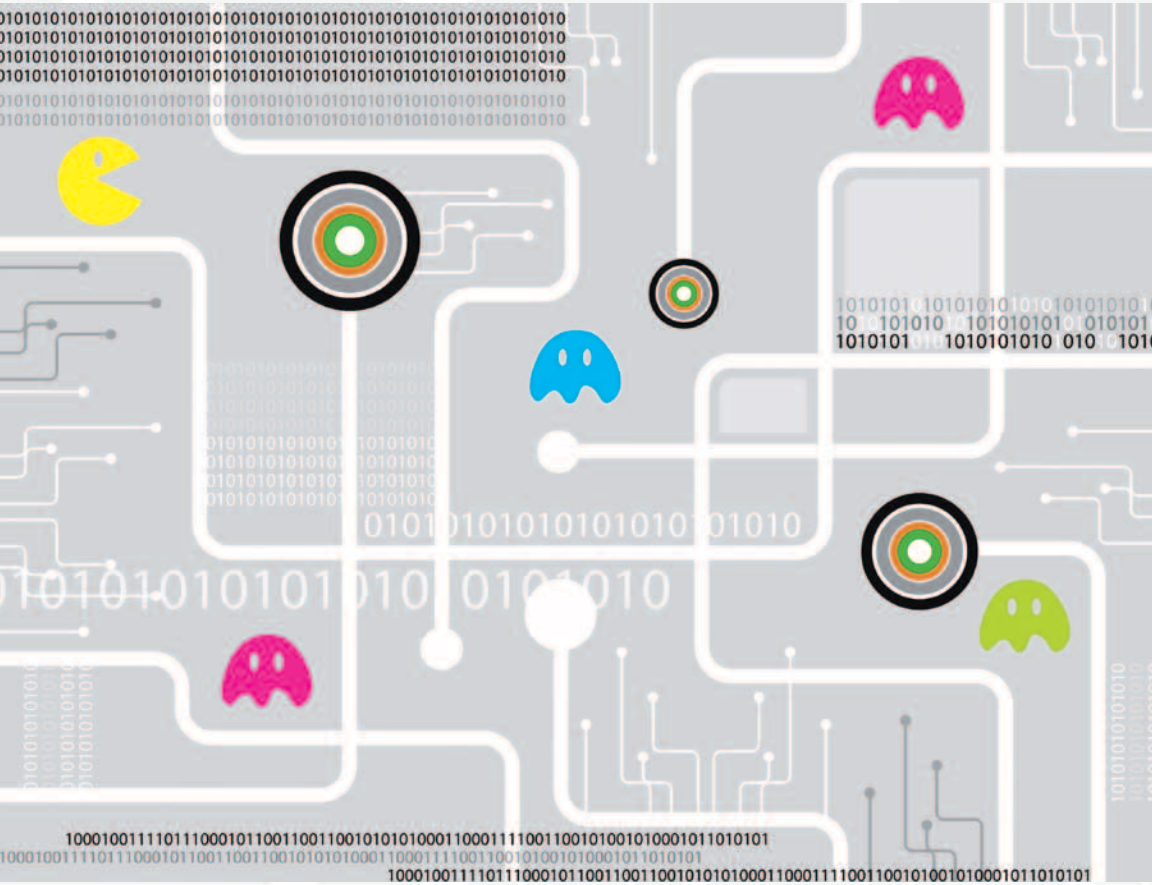
PLAYING the PAST

HISTORY AND NOSTALGIA IN VIDEO GAMES



EDITED BY

Zach Whalen
Laurie N. Taylor

Playing the Past

History and Nostalgia in Video Games

Edited by Zach Whalen and Laurie N. Taylor

Vanderbilt University Press • Nashville

Library of Congress Cataloging-in-Publication Data

Playing the past : history and nostalgia in video games /
edited by Zach Whalen and Laurie N. Taylor.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-8265-1600-8 (cloth : alk. paper)
ISBN 978-0-8265-1601-5 (pbk. : alk. paper)
1. Video games. 2. Video games—Psychological aspects.
3. Video games—Study and teaching. I. Whalen, Zach,
1979– II. Taylor, Laurie N., 1978–
GV1469.3.P483 2008
794.8—dc22
2007051878

Contents

5

Hacks, Mods, Easter Eggs, and Fossils

Intentionality and Digitalism in the Video Game

Wm. Ruffin Bailey

The term "interactive digital media" contains an often-overlooked adjective, *digital*. Espen Aarseth has given us a detailed study of the aesthetics of cybertext; Nick Montfort, the textuality of interactive fiction; and Mark J. P. Wolf, a strict review of the hardware requirements for a work to be labeled a video game. These authors provide useful, high-level work that introduces a young field, but only begin to provide an in-depth look at the digital underpinnings of gaming software and digitalism's undeniable influence on the creation of virtual realities.

The "digital" in digital media needs to be examined to gauge its characteristics' influence upon the creation of virtual spaces. Every web page lives on a digital host, forcing preschoolers to grandmothers to become familiar with the highly technical standards of Universal Resource Locators (URLs) and Hypertext Markup Language (HTML) that enable digital interpretation. Engines that drive video games are mediated by similar digital hosts. Games are limited in size by their ability to access their host's physical memory. They are limited in complexity by their authors' ability to construct decision-making algorithms that approximate the authors' visions of virtual realities. Theorists may be able to observe the results of these virtual realities, but in the absence of an interrogation of the fundamentals of computer science, they will not be able to effect a complete study of digital media and its creation.

This chapter seeks to establish a rhetorical method—of terminology and taxonomy—by which to explore what is unique to software-based digital media, starting with a nostalgic application of the method to the Atari 2600 (the first truly modular home gaming console) and watching how its workings continue to reflect and inform studies of games written nearly two decades later. Such a methodology currently requires a

multidisciplinary cultural approach and is instrumental in grasping how interactive digital media, here video games in particular, operate.

This study begins by reviewing Montfort's study of interactive fiction, which provides an accessible introduction to video game studies by focusing on the games' use of narrative and riddle—two conventional approaches to text—before launching into code. I then review two foundations of a digitalistic approach that are currently housed in the field of computer science, the concepts of Boolean logic and memory addressing. These prove crucial for studying the operation of video game engines and creating a technical taxonomy for video game content. I end with a consideration of several video games to demonstrate how a nostalgically informed study of code can be employed in practice to shape game studies, using examples as varied as programmers innocuously hiding graffiti of their names in Atari 2600 games to more modern artifacts like Rockstar Games' "accidental" inclusion of a hidden sex game in *Grand Theft Auto: San Andreas* (2004). In sum, because digital media is heavily influenced by technical characteristics, game studies will not achieve its full potential until considerations of computer science are more fully integrated into its methods.

The Bias of Accessibility or Limited Approaches

Montfort's enlightening study of interactive fiction (IF), *Twisty Little Passages*, recommends approaching IF, a subset of video games, in three ways: as narrative, as riddles, and as computer programs (14–15). The first two have received a great deal of discussion, largely through scholars like Montfort constructively framing games as texts. Purely by virtue of being accessible, however, these two approaches risk garnering inordinate attention within cultural studies at the unfortunate detriment of the third.

In the field of cybertext, six years prior to Montfort, Aarseth expressed his "wish to challenge the recurrent practice of applying the theories of literary criticism to a new empirical field" (14). This "recurrent practice" is one possible symptom of over privileging customary approaches—including narratives and riddles. At the same time, scholars' familiarity with these approaches is one reason that Montfort's book on interactive fiction (a style of game with a high affinity for the common codex) is a proven and popular introduction to the field.

Aarseth admits that "[traditional literary] approaches are useful for establishing the legitimacy of the field [of hypertext literary theory]" (76). *Twisty Little Passages* does a fine job establishing such legitimacy by dis-

secting IF-as-narrative and riddle. Still, both authors ultimately privilege the experience of IF's and cybertext's content without giving the same level of sustained attention to how that content was shaped by its digital hosts, a topic crucial for the study of both IF and cybertext.

Guarding against unfairly strong remediations of old media approaches into the study of the new proves a major difficulty in many studies, exemplified in part by John Muckelbauer's critique of a recent collection of essays regarding the rhetoric of film. Muckelbauer warns against exploiting that which is unique to a medium to support conclusions that were reached exclusively via historically more familiar means of compiling evidence: "This [narrative bias] is particularly striking insofar as discussion of things like images and sound—things that would seem to be important elements of film's distinctiveness—are relatively invisible in this collection" (905). In the case of video games, this warning translates directly, and it is this tendency to ignore games' "distinctiveness" that ludology and other multidisciplinary approaches must avoid.

The importance of truly digital approaches is evident in a section of *Twisty Little Passages* where Montfort heuristically recommends initially treating an IF work's engine, called a parser, "as a black box that accepts input and generates output." He goes as far as to proscribe "making reference to a program's specific data structures, functions, objects, and so forth" (23). However, to complete his analysis, Montfort (himself an author/programmer of several impressive works of IF) ultimately does move to his third approach: looking at IF compositions as computer programs. When he compares *Adventure* (1976) to a successor, *Zork* (1982), he lists a number of ways that *Zork*'s parser is superior to *Adventure*'s. Montfort states that *Adventure* "only accepts commands of one or two words" and that *Zork*'s parser understands longer directives. *Adventure* requires explicitly mentioning the object a player wants to use in commands, but *Zork*'s parser can "disambiguate" the item a user implicitly requested to employ with certain actions (e.g., a player's avatar might dig with its hands if another object was not mentioned). *Zork*'s parser can also understand prepositions, unlike *Adventure* (108–9).

Montfort's conclusions about IF engines—that one parser is limited to one or two words, that it never understands prepositions, or that its content is limited to a certain size —cannot be stated with full confidence without an in-depth, open-box understanding of the works' parsers.[1] Perhaps *Adventure* anachronistically understands the sentence, "Climb over the northern mountains, read Sam Lantinga's blog, and then summarize any advantages of using OpenGL over DirectX." If one were to take *Twisty Little Passages*' initial proposal to treat game engines as black boxes until

every possible sentence has been tried, a task of ridiculously irrational scope, its conclusions could not be stated in absolute terms and likewise could not be authoritatively woven into critical works. Determinations about maximum work lengths or the potential to display graphics could not be made. As Montfort's and Ian Bogost's forthcoming *Platform Studies* series' concentration on just these issues attests, it is crucial for scholars of digital artifacts to now move past accessible remediations and forward toward understanding the inner workings of these artifacts' platforms. To analyze these inner workings requires the explicit introduction of a number of concepts from computer science.

Digitalism

For digitalism, the first steps involve further pursuing Montfort's third approach to IF works: to treat them as computer programs. Commercial video games, thus far, are digital creations.[2] I argue that scholars studying works of digital media must, as an essential position for rhetorical and cultural studies of games, approach the content on an equal footing with those games' creator(s) or author(s).

Digital means a system of representation based on discrete digits rather than any sort of continuous spectrum. What this means in practice becomes especially evident in the way computer programs make decisions. The simplest flow chart that includes at least one decision step makes the point obvious.

In each decision step, there are one or more discrete choices for exit. Even in the most complicated piece of branching logic, entry into each potential branch is, at its core, mediated by a simple test of true/false (that is, Boolean) logic. Either the condition or conditions for taking the branch are all true statements and the branch is taken, or at least one condition is false and the branch is avoided. By convention, conditions that point towards taking a branch evaluate to a Boolean value of "true," usually represented by the digit "1." Conditions that would stop the branch's execution are given a value of "false," and are usually represented by the digit "0."[3]

In technical terms, Boolean data types that hold these values are often called *switches*, and these evaluations act like switches on a railroad track. If "on," the train of logic will travel down one track; if "off," the logic will proceed down another. For trains, taking a third choice is disastrous. In digital hardware, a third choice, much less an infinite spectrum of choices, is—by design—impossible. This is precisely why cybertext so

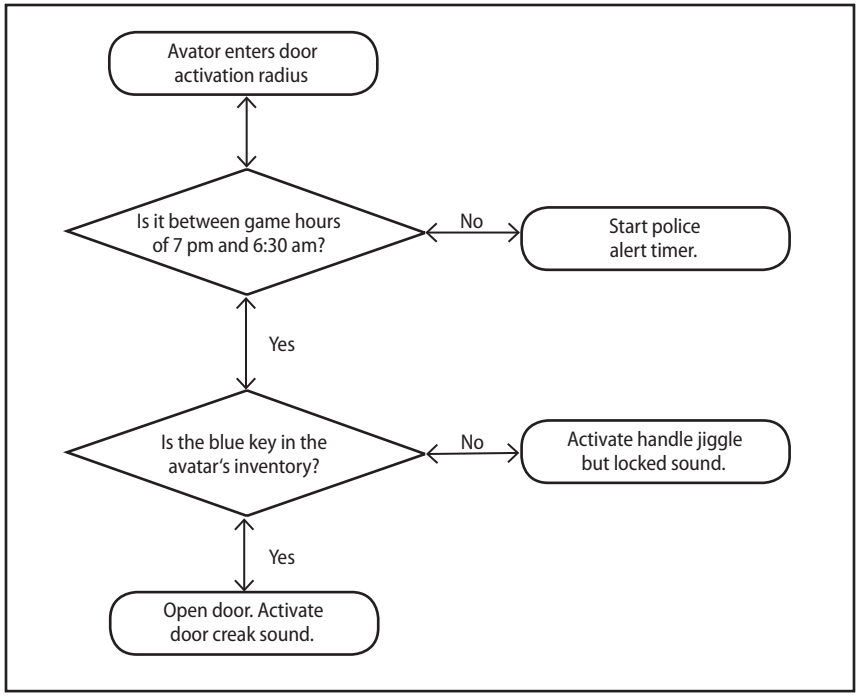


Figure 5.1. A simple flowchart

naturally gave rise to a system that, as Aarseth remarks, is "forcing the reader to pay attention to the strategic links" between portions of text (78). The reader is *being forced to act digitally*—to select one of a finite number of branches for their progression in a work. The operation of a cybertext interface can be reduced to nothing more than the most trivial of covers above the Boolean logic being evaluated beneath.

Exactly like they do in the thinly veiled interfaces of most hypertexts, Boolean evaluations or switches drive decision-making even in the most complicated representational systems created by video games. A player's avatar may approach a locked door that only operates during the night and if the avatar is carrying the appropriate key. Here there are at least two conditions to be evaluated. If the value of the night switch is "true" and a key inventory switch is also "true," the engine will allow the code to open the door to be executed. To the player, the checks in this example happen behind the scenes. Regardless, the digital switches are still there, evidenced by the consistent appearance of if-statements in the representation of Figure 5.1's logic shown in Figure 5.2's pseudo-code.

```
if (measurer.pointDist(p1,p2) < 20)      {
    if (    Game.currentTime() < Game.MORNING_TIME
            &&
            Game.currentTime() >= Game.EVENING_TIME
    ) {
            if (inventory.key(BLUE_KEY))  {
                    blueDoor.open();
            }       else    {
                    Game.sounds.
play(handleJiggle());
            }
    }       else    {
            police.soundAlarm();
    }
}
```

Figure 5.2. Pseudo-code for opening a door

This Boolean logic also introduces the method for creating "hacks." Hacks occur when an unsanctioned third-party or independent programmer changes a program's code to function in non-standard but not necessarily unintended ways. If a hacker could change the above example's logic to read, "If the night switch is 'true' and the key inventory flag is also 'true'," or "1 = 1" (that is, simply adding a trivial expression that makes the evaluation of the whole always true), the hacker could easily short-circuit the game's door to open regardless of the time of day within the game or whether the avatar is carrying an appropriate key.[4] This is precisely what happens in certain "god mode" cheats in games that, for instance, allow players' avatars to walk through walls. By locating the Boolean evaluation where the collision detection occurs, a hacker can essentially insert a switch that makes the game's engine believe that collisions never occur. The new switch permanently sets the collision state to "false" for every check. The avatar can then move through any wall in the game. The game is now full of "not-collisions."

Certain games establish a documented standard where modifications can be added by placing the homespun content at certain memory addresses on the computer. *Memory addressing* is the second important digital concept computer science lends to ludology.[5] Every instruction in a computer has a memory address of one sort or another. Much as

street addresses, hotel room numbers, or the line numbers in a BASIC program label potential locations for their users, digital computers use discrete numbers to reference specific hardware memory addresses that contain the code or data necessary to perform their tasks. Most, if not all, contemporary games have abstracted this process so that the machine's literal memory addresses do not have to be known. These games can situationally reference the correct address for, as an example, a folder as represented by a user's operating system rather than a specific spot on a stick of RAM or a hard disk.

Still, anyone who has experienced a "Blue Screen of Death" on a Windows computer has had every layer of abstraction ripped from her or his interface when the memory address describing the location of a deal-breaking error was displayed, usually as a particularly befuddling hexadecimal number similar to "71d633e5." Memory addressing is also why game console developers remain so intent on letting gamers know what was an 8-bit, 16-bit, 32-bit, or 64-bit system. Each increase indicates an exponential growth in the maximum number of addresses, meaning (at least indirectly) more resources with which the consoles could construct games.

Hacking Nostalgic

At this point, it is useful to revisit the term *game engine* to help establish how games are "hacked." Since the earliest days of home consoles, a game engine has been the part of a video game that interfaced with a game's content and evaluated Boolean logic. An engine, if run alone, is not able to provide a traditional gaming experience. In some ways, engines enforce the rules of their virtual worlds, but they are not worlds themselves. Engines are often created to be reusable, supporting more than one package of content, as with those displayed in Table 5.1.

It is sometimes difficult, especially in classic games, to know where an engine ends and content begins, as they may contain engines that were not designed for reuse and seem inextricably wedded to their content, like *Space Invaders* (1979) for the Atari 2600. Some contemporary "games," however, are hardly games at all; instead, they are little more than specialized engines with a minimum of content added before their release, begging third-party modification, like *Quake 3* (1999).

Atariage.com remarks that *Space Invaders* for the Atari 2600 home gaming console was the first arcade game licensed for home use ("Atari 2600—Space Invaders (Atari)"), and the ability to create the arcade experience in the home gave the 2600 its iconic stature. Yet *Space Invaders*, like

Table 5.1: Selected popular first-person shooter engines with implementations

Engine	Release	Implementations
Wolfenstein 3D	1992	*Wolfenstein 3D*, *Rise of the Triad*, *Spear of Destiny*
Doom	1993	*Doom, Doom 2, HeXen, Strife, HacX*
Quake 1	1996	*Quake, Half-Life,* HeXen 2, X-Men: Ravages of the Apocalypse*
Quake 2	1997	*Quake 2, Soldier of Fortune, Heretic 2, SiN, Kingpin, Daikatana*
Quake 3	1999	*Quake 3, Star Wars: Jedi Academy, American McGee's Alice*
Unreal 1	1998	*Unreal* (original), *Unreal Tournament, Deus Ex, Harry Potter and the Sorcerer's Stone, Star Trek: The Next Generation: Klingon Honor Guard*
Unreal 2	2002	*Unreal 2, America's Army, Thief: Deadly Shadows, UT2003, UT2004, Unreal Championship 2, Tom Clancy's Splinter Cell, Star Wars: Republic Commando, XIII*
Lithtech 1.0	1998	*Shogo, Blood II: The Chosen*
Lithtech 1.5	2000	*No One Lives Forever, Alien vs. Predator 2*
Lithtech 2.0	2002	*No One Lives Forever 2, Tron 2.0, The Matrix Online*
Halo	2001	*Halo, Stubbs the Zombie, Red vs. Blue* (machinema)

**Half-Life* contains a heavily modified *Quake 1* engine with some code from *Quake 2* as well.

a number of 2600 arcade ports,[6] was not nearly as true to the original as even the limited 2600 hardware would allow. In 1999, using new tools like modern emulators, Rob Kudla hacked *Space Invaders*' code in a doubly nostalgic attempt to re-create more faithfully the coin-operated arcade version of *Space Invaders* on the 2600 ("Atari 2600 Hacks"). Kudla made the invaders, the player's tank, sounds, and colors more closely reflect the original's. His improvements succeeded impressively.

The first step in hacking *Space Invaders*' graphics was to discover the memory address where graphic content (as opposed to the location of the game's *engine*) was kept in the machine language code. This is not nearly as hard a task as it may sound. Each invader is made up of ten "scanlines" of graphics, where each scanline matches one pass of a television's elec-

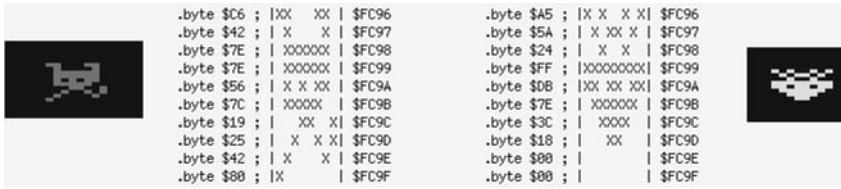


Figure 5.3. Original and hacked graphics, code, and memory addresses for *Space Invaders* and *Space Invaders Arcade*. Images © Taito and Atari, Inc.

tron gun across its screen. Each scanline of the 2600 invaders' graphics is eight bits, or switches, wide. It is possible to decompile the machine language code into a graphic representation of the program's switches' values, and the invaders' graphics' location in the code become obvious, in spite of appearing upside-down in the decompiled representation.

In Figure 5.3, the graphics (rotated for easy comparison with the code) for the original top-level, 2600 invader is displayed on the left next to the hacked, more arcade-faithful version by Kudla on the right. There are three columns in each section of disassembled code. The first column for each invader's disassembly shows the byte value of the eight-switch line of graphics in hexadecimal notation.[7] The second column shows the value of each individual switch in every eight-switch line, displaying an "X" for each switch that is "on" or set to "true." The last column shows the address in memory where the line is held, again in hexadecimal format.

Note that the addresses in the third column are the same for both invaders' listings ($FC96-$FC9F). The Atari 2600 does *not* rely on an operating system to abstract memory addresses like most present-day platforms, differing what happens here from the engines of "modern games" that Bogost notes are modularly "split up into software objects and frameworks" (55). These, then, are exact, static, unabstracted memory addresses in fairly monolithic code. That the addresses for the graphics are the same in the hack as the original suggests that when Kudla made his updates in *Space Invaders Arcade*, the logic of the game's original engine was not changed, and only a few switches solely related to graphics were hacked in place.

Kudla's reliance on a hack in place is also an indication of the degree to which the engine and content of this classic game are intertwined. In the byte—a collection of eight switches—located at memory address $FC96, two switches that were off were switched on and two that were on were switched off to change the appearance of the bottom of the invader. Kudla's creation of two blank lines of graphics at the top of the invader

ensured the memory addresses remained synchronized in the modified content. In other words, *Space Invaders Arcade* is still using the original 2600 *Space Invaders* engine to provide the enforcement of its world's rules.

A crucial characteristic of the definition of a hack is that the original author of the game did not intend for a third party to change the game's content. *Space Invaders Arcade* can be safely considered a hack through a number of pieces of evidence. First and foremost, *Space Invaders* was initially released on a cartridge. Its code was permanently burned into Read Only Memory (ROM) and this hardwiring could not be changed, similar to the music on a commercial compact disc. Second, the game continues to be under copyright, legally forbidding just this sort of modification. Third, the author of *Space Invaders* did not provide documentation identifying the memory address of the content that Kudla changed, and the tools that now enable a hobbyist to disassemble and reassemble Atari 2600 games easily did not exist at the time of *Space Invaders*' release in 1980. Bogost described a similar situation with *Tank* and *PONG*, as well as *Combat* for the 2600, a home version of the former arcade game. The shared codebase in *Tank* and *PONG*, like that in *Space Invaders* and *Spaces Invaders Arcade*, served as a proto-engine, tying the games together in a manner that could be discovered only by a digitalistic critical approach. "[T]heir common gameplay properties relied entirely on the same codebase . . . *Tank*, *PONG*, and *Combat*'s relation to one another is far stronger than interpretative notions like intertextuality or new media concepts like remediation allow" (58). Kudla's modification is different from the one Bogost describes in that Kudla's code-sharing was unauthorized. From the medium of the game's release to a complete lack of documentation on providing new or altering old content, it is safe to say *Space Invaders Arcade* is an unsanctioned hack.

There is another species of game alteration that works in a similar fashion, but where the hackers' addition of new content is anticipated and encouraged by the games' designers. Rather than a hacker finding a memory address through subversive code disassembly, in these cases designers expose a standardized memory address (usually abstracted by a folder location) where new content can be placed and read by a game's engine. This content can be as simple as a new skin for an avatar's frame that gives the player a different color or set of clothes or as complicated as a full modification, including new maps, textures, player models, and even rules for in-game scoring or physics.

Taken together, hacks and the addition of new code show that games are never done. From the very first games released on a home console to

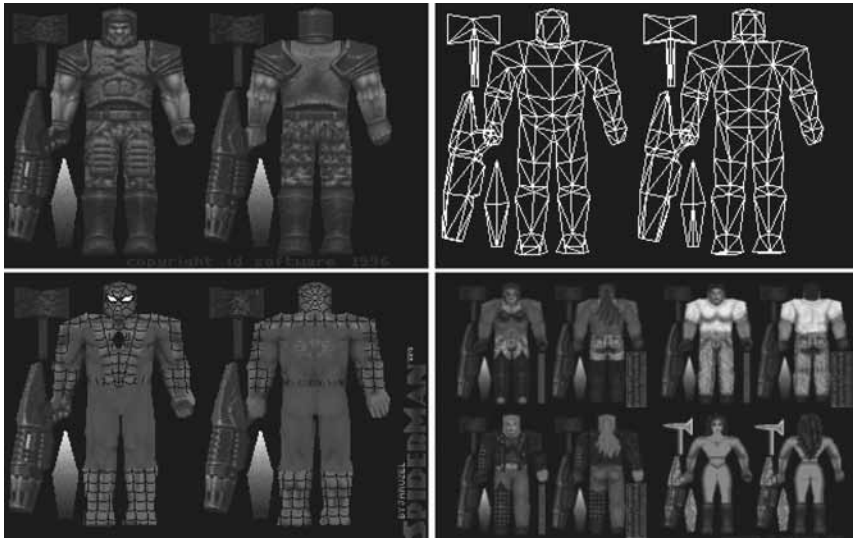


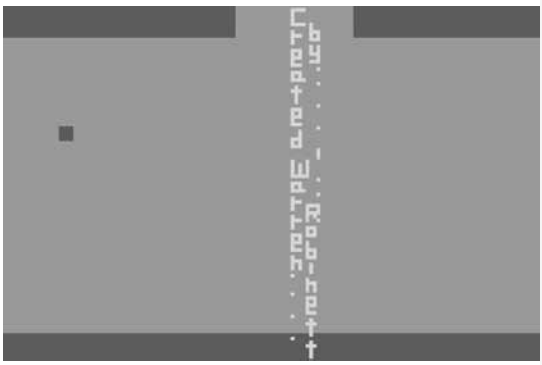
Figure 5.4. From *Quake 1*: (*top left*) default skin; (*top right*) model wire frame outline; (*bottom left*) *Spider-Man* skin; (*bottom right*) female skins, marine model. Images © id Software.

the latest games to hit market shelves, every one is open to alteration and additions in the digital age, with the Internet enabling mass consumption even of hobbyist releases.

Engines and Intentionality: Easter or Fossilized Egg?

Hacks, skins, and modifications are means for technically savvy gamers to coauthor the games they play. Another important concept in the video game that deserves close attention is the Easter egg, a portion of a game that is not added by savvy players but hidden by author(s) in the original. If found, an Easter egg provides a metalepsis similar to what Montfort describes in Infocom's *Planetfall* (30), but the effects are usually much more innocuous. Like literal Easter eggs, Easter eggs in video games *are meant to be found*. Determining whether a hidden portion of a game is a fossil (an unintended leftover from earlier development) or a true Easter egg waiting for discovery can be difficult. Distinguishing between the two and understanding how they affect the interpretation of digital artifacts is the goal of this portion of this chapter.

The most famous Easter egg in a video game can be found in Warren Robinett's Atari 2600 game, *Adventure* (1980). Game authors were



```
.byte $F0 ; |XXXX    | $FD88
.byte $80 ; |X       | $FD89
.byte $80 ; |X       | $FD8A
.byte $80 ; |X       | $FD8B
.byte $F4 ; |XXXX X   | $FD8C
.byte $04 ; |     X   | $FD8D
.byte $87 ; |X    XXX | $FD8E
.byte $E5 ; |XXX  X X | $FD8F
.byte $87 ; |X    XXX | $FD90
.byte $80 ; |X       | $FD91
.byte $05 ; |     X X | $FD92
.byte $E5 ; |XXX  X X | $FD93
.byte $A7 ; |X X  XXX | $FD94
.byte $E1 ; |XXX    X | $FD95
.byte $87 ; |X    XXX | $FD96
.byte $E0 ; |XXX      | $FD97
.byte $01 ; |       X | $FD98
.byte $E0 ; |XXX      | $FD99
.byte $A0 ; |X X      | $FD9A
.byte $F0 ; |XXXX     | $FD9B
.byte $01 ; |       X | $FD9C
.byte $40 ; | X       | $FD9D
.byte $E0 ; |XXX      | $FD9E
.byte $40 ; | X       | $FD9F
.byte $40 ; | X       | $FDA0
.byte $40 ; | X       | $FDA1
.byte $01 ; |       X | $FDA2
.byte $E0 ; |XXX      | $FDA3
.byte $A0 ; |X X      | $FDA4
.byte $E0 ; |XXX      | $FDA5
.byte $80 ; |X       | $FDA6
.byte $E0 ; |XXX      | $FDA7
.byte $01 ; |       X | $FDA8
.byte $20 ; |  X      | $FDA9
.byte $20 ; |  X      | $FDAA
.byte $E0 ; |XXX      | $FDAB
.byte $A0 ; |X X      | $FDAC
.byte $E0 ; |XXX      | $FDAD
```

Figure 5.5. *Adventure's* Easter egg with code disassembly. © Atari, Inc.

not credited in games released for the 2600 by Atari, and Robinett decided to circumvent this rule by burying his name in a secret room of the game. That Robinett intended players to find the secret room can be deduced from three clues. The first and most obvious is that the message is visible when playing the game. As will be shown, the game's code clearly holds his name, but Robinett provided a means for gamers to read the message on their television screens without sifting through its digital code. The second is that the graffiti reads, "Created by Warren Robinett." This message expects interpretation: it does not simply hold the author's name but unambiguously tells the gamer that the game was indeed made by the named individual. The last comes from the method by which the room is discovered. The room is accessed when the gamer's avatar places

a "dot" Robinett hid in the game near the wall of a specific, easily accessed room. Robinett had programmed the game's engine to flash objects when more were on the screen than the 2600's hardware could easily support. The 2600 will only easily display a maximum of two complex objects called "sprites" (called "player graphics" on the 2600) on the screen with each frame. When more needed to be present, Robinett circumvented the 2600's limitation by making *Adventure*'s frames flash quickly enough (with two different objects displayed per frame) to let the gamer understand that more objects were there, though with the side effect that the items would seem to strobe constantly. Robinett put an extra object into the room that contained the secret dot to ensure that the room would flash when the player entered.[8] A perceptive player would notice the flashing in spite of there apparently being fewer items than what caused flashing elsewhere and be curious enough to eventually discover the existence of the dot. Not only does *Adventure* allow a gamer to read the message on his or her screen, it also provides hints for the player to discover those means.

That the secret message was not discovered by Atari during the game's development testifies to the way games could be, in the classic era, created by a single programmer without much oversight. A simple disassembly today, as shown in Figure 5.5, displays Robinett's message as easily as the same method finds the invaders' location in *Space Invaders*. Yet *Atariage.com* lists five more games released by Atari that included some sort of Easter egg containing "Programmer Credit" ("Tips, Cheats, and Easter Eggs").

San Francisco Rush: Time-Delay Easter Eggs and the Sophisticated Gamer

Not all digital Easter eggs are created to resist a parent corporation's attempts to dehumanize programmers.[9] *San Francisco Rush: Extreme Racing*, created for one of Nintendo's home consoles, the N64, found itself in a particularly precarious position. The N64 had enough power to approach very closely the game play and experience of the coin-operated original,[10] released the previous year. Gamers were also expecting the home version to contain content from the more recent arcade update, *San Francisco Rush: The Rock*. The arcade sequel's title headlined a new track for the racing game, which went to great pains to intricately recreate the streets of San Francisco; the extra track was one created on the landscape of Alcatraz, "The Rock." When *San Francisco Rush: Extreme Racing*

was released, however, not only was there no Alcatraz, *Extreme Racing*'s project lead, Ed Logg, quashed any reports of plans to bring *The Rock* to the N64 (IGN Staff "Rush").

There was a rumor, propagated online, that claimed that *Extreme Racing* had the Alcatraz level hidden on the cartridge. A number of myths regarding how to access the track sprouted, including winning a race on each track in record time, using a manual transmission in those races, and even using a specific car while racing. None of these in-game methods unlocked the track (Stevefel et al.)

The Alcatraz track was unlocked no later than January of 1998, when a player posted his or her successful discovery of a code to hack the game to Usenet (bunivfan). The code worked with a device called a GameShark, which allows gamers to change values in their console's memory, much as Kudla did with the *Space Invaders*. Instead of changing bit values, the hack pointed the game's engine to the secret memory address that held Alcatraz, tricking the engine into loading and playing the hidden track. On March 19, *IGN.com* published a code from Atari Games that allowed gamers to access the track without a GameShark hack (IGN Staff "You're Going").

IGN.com interviewed Logg on April 1, 1998, regarding how the track came to be hidden on the cart. In brief, Atari Games' sales department did not want Alcatraz in the N64 version so that it would not, as they saw it, compete with the arcade game, and when the level was added it was hidden so well that, short of the GameShark, no one was able to access the track. The sales department was never contacted. The motivation for hiding the level was not strictly commercial, however; Logg admitted that the level was not likely to be as well tested as the other six in the game, which is apparent when the level is played (IGN Staff "Exclusive Interview").

San Francisco Rush: Extreme Racing challenges the archetypal concept of the Easter egg, like that found in *Adventure*, in two main ways. The first is that *Extreme Racing*'s track was embedded to create a sort of time-release content rather than exist as something gamers could hunt and locate within the game. Logg and his team did not provide a hint like Robinett's flashing objects. At the same time, it was content that the game's authors intended for the gaming public to experience if and only if they were in some way in contact with the circles that provided codes, be it Usenet, *IGN.com* and other Internet sites; gaming magazines; or friends who could access one of those outlets. In effect, they hid part of the game *outside* of the physical media of, in this case, the cartridge. Alcatraz was to be a multimedia Easter egg.

Multimedia Easter eggs have become the rule in video games, and the means of their distribution mark a sharp change from Robinett's in-joke for perceptive players. Lists of newly released codes that almost certainly could not have been guessed are standard in the back of magazines dealing with video games in what has become a somewhat codependent state of affairs. The publishing of video games' codes amounts to free game advertisements at the same time that the anticipation of new codes sells each new magazine issue, turning Montfort's metalepsis into an industry.

The second way that *Extreme Racing* helps to redefine the Easter egg is the way that it deprivileged the work's authors. Alcatraz was discovered without its authors' consent and contrary to the authors' design by GameShark hackers. Gamers are now savvy enough to search for hidden content that is not accessible solely using the game's traditional interface. Alcatraz was meant to be found. With its *early* discovery, however, gamers proved they had the means to find unintended "eggs" as well. GameSharks, disassemblers like that used by Kudla, model viewers like Pakrat (Naughton), and utility software that allows file inspection like hex editors allow gamers to break from the intended interfaces and creatively search for hidden content. These tools provide new directions for a player or "reader" to access digital works.

Fossils: Unintended Easter Eggs

If a programmer creates a subroutine that may not be used in a later version of their program, there is no technical requirement to remove it. If Warren Robinett had decided to hide his secret message from gamers, he could have deleted the extra room where the message was displayed and left the data with his message in *Adventure*'s code, only to be found with a thorough disassembly. Unlike text that has been struck through, unused code does not create any noticeable effect for its end users. It simply remains, fossilized.

In fact, there is often good reason to leave unused code in a program. If a later version needs a similar function, the code is already integrated with the codebase and is ready to be called. In some respects it can be an extra tool in the toolbox for an anticipated need, but one that is currently not required. There is also a potential advantage for fossilized code when testing an application. If the fossilized code has been cut out but not removed, perhaps for expediency's sake, and the program makes its way through user testing, completely removing the code later would require more testing to ensure a sloppy removal did not accidentally introduce new errors. Either reason can produce fossilized code in any digital

work. Fossilizing code is a uniquely digital process. One switch might be changed (a Boolean expression that, if "true," would have caused a subroutine to be called is set to always evaluate as "false," fossilizing the subroutine) and the end user's experience is, in theory, *exactly the same* as it would have been had the code never existed.

Tomb Raider: The Angel of Darkness (2003) is an example of a video game with fossilized content. Whereas the first five entries in the *Tomb Raider* series shipped like clockwork, one coming out before each year's holiday buying season, *Angel of Darkness* was released over three-and-a-half years after *Tomb Raider: Chronicles* (2000). One *Tomb Raider* fan, upset by *Angel of Darkness*' errors, created a list called the "AOD Bugs definitive list" in which sixty-two bugs were listed (Dragoncarer). Bug 1.1, "The Secret Garden," describes fossilized content: *Angel of Darkness* opens with a training level that allows users a chance to become familiar with the complicated controls of the game. Much of the level was scrapped, presumably to allow the game to be released more quickly. The fossilized content is still within the game, and parts of it can be seen from conventionally accessible portions of an extant level. Entering the fossilized content requires either hacking the game or loading a saved game where Lara Croft, the game's protagonist, begins at a location in the fossilized area. As Figure 5.6 shows, in the fossilized area, large parts of the surrounding cityscape are missing and training instructions are displayed for the player. One safe assumption from the archeological evidence is that the training section of *Angel of Darkness* was originally intended to be much more elaborate.

Hot Coffee and Possible Piltdowns

The skull planted in Piltdown, England, may be the greatest scientific hoax, and is certainly the greatest in physical anthropology.[11] No more than an orangutan jaw placed next to a human skull, the false fossil tricked anthropologists for more than four decades. The Piltdown concept is a useful one for video games, especially when trying to evaluate rhetorical intent with respect to apparently fossilized content. Are programmers cunning enough to disguise Easter eggs in their games as fossils to trick the gaming public into believing they were accidents?

Rockstar's *Grand Theft Auto: San Andreas* raises just this question. *San Andreas* contains what appeared to be a fossil: a portion of the game that is inaccessible without some sort of hack or download. Here, the ostensibly fossilized content allows a gamer to have limited control over relatively graphic sexual scenes between the player's avatar, named "CJ," and

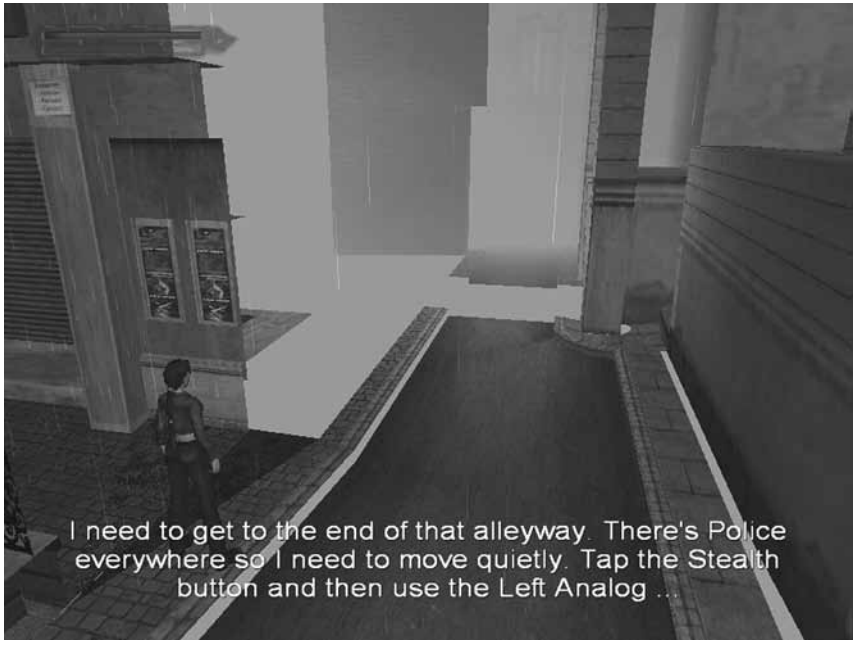


Figure 5.6. Fossilized training area in *Tomb Raider: Angel of Darkness*. Image © Eidos Interactive.

his girlfriend(s). After dating, CJ's current girlfriend may invite him into her house for coffee. For a player without the "Hot Coffee" modification, the camera shakes suggestively (like CJ's car when he picks up a prostitute) after he is invited into his girlfriend's home, but for a player with the modification, the camera follows CJ inside.

Patrick Wildenborg, who discovered the "fossil" and created a modification that allowed others to access it easily, says the following about the modification on his website:

> After reading various discussion [*sic*] about this mod around the internet, I would like to make the following statement:
> *All the contents of this mod was already available on the original disks. Therefor* [*sic*] *the scriptcode, the models, the animations and the dialogs by the original voice-actors were all created by RockStar. The only thing I had to do to enable the mini-games was toggling a single bit in the main.scm file.* (Of course it was not easy to find the correct bit). (Wildenborg, emphasis in the original)[12]

After Hot Coffee's discovery, Rockstar quickly became trapped in a controversy, and the Federal Trade Commission investigated how the game received a "Mature" rating, which *recommended* its sale to people seventeen years old and older, and not an "Adults Only" rating, which would have *restricted* its sale to those able to prove they were eighteen years old and up. An "Adults Only" rating was then given to *San Andreas* due to the apparent fossil discovery, and the game was pulled from mass retailers' shelves. A revised version without the Hot Coffee content was written and re-released, and an update was provided online to erase the content from personal computers with the game already installed.

Rodney Walker, a Rockstar spokesman, quickly attempted to fashion a metaphor for Hot Coffee's inclusion favorable for his company:

> An artist makes a painting, then doesn't like the first version and paints over the canvas with a new painting, right? . . . That's what happened here. Hackers on the Internet made a program that scratches the canvas to reveal an earlier draft of the game. (Schiesel)

While it is true that fossilized content can "reveal an earlier draft" of a game, the metaphor is flawed due to Walker's inattention to digitalism's influence on the rhetoric of the *San Andreas* minigame. A copy of a painting does not include an embedded history. Owners will not discover a sketch of Mary with her left hand to her breast in their copies of mass-produced prints of Leonardo da Vinci's *Virgin on the Rocks*. Every copy of *Tomb Raider: Angel of Darkness* has the unfinished portions of the training level, every copy of *San Andreas* has the hidden sex game, and infrared reflectography is not required to discover the content ("The Hidden Leonardo"). It is also possible to erase unused code from a digital product's final version, whereas it is nearly impossible to remove an unfinished draft from below a masterpiece's last layers of paint. Walker's metaphor may have been one of the best of the quick attempts to understand Hot Coffee's inclusion, yet it is one that unjustly favors the publisher in its depiction that ignores digitalism.

The Lessons of Digitalism

Studying virtual realities dependent on digital hosts requires an emphasis on digitalism, which in turn enables readings that would otherwise be missed. Conjoining what are now multidisciplinary concepts is required to question complex works, like *Grand Theft Auto: San Andreas*, on a footing equal with authors, whether those authors are sanctioned

or unauthorized. These approaches help remove the limitations of legacy methods, and instead of losing many of those tools, the integration reinvigorates them.

This chapter's quick survey, moving from nostalgic to contemporary digital artifacts, does little more than scratch the surface of a digital approach, as it concentrates on the contested fringe of the representation of virtual worlds. Articles like Carolyn Miller's "Writing in a Culture of Simulation: Ethos Online" have opened the door for analyzing the next wave of virtual reality, where great characters are created not simply by great writing, but by skillful writing combined with convincing artificial intelligence. The time and tools for integrating digitalism with cultural studies are here, and the field awaits.

Notes

1. Even more stark a break from the "black box" approach is Montfort's characterization of BASIC as a particularly difficult language with which to write IF and one he has tried using himself. Here, *Twisty Little Passages* has clearly moved from the accessible approaches of IF-as-narrative and riddle and on to the third—complex computer programs.
2. An interesting study would be the history of video games based on analog computers. I am only familiar with William Higinbotham's *Tennis for Two*, created in 1958 (US Dept. of Energy). Regardless, this claimant for the title of the first video game offers an intriguing alternative to the metaphors of digitalism.
3. There is always an exception that proves the rule. In Visual Basic 6.0, once arguably the most popular programming language for the Microsoft Windows operating system, "true" is represented by "-1." To make things even more convoluted, once the Visual Basic code is translated into machine language, the "-1" becomes a "1" again (Bailey et al.).
4. The is reminiscent of Captain Kirk's answer to the Kobayashi Maru simulation in cadet training, recounted in *Star Trek: The Wrath of Khan*. When presented with a "no-win"/always "false" trial in training, Kirk, after failing the simulation twice, reprograms/hacks the computer to create a winning solution.
5. Though the explanations given for these terms herein are arguably accurate, a much better—and extremely accessible—primer text is Richard Mansfield's *Machine Language for Beginners*, which explains assembly language programming and the operation of a series of chips found in the Atari 2600, Commodore 64, Nintendo Entertainment System, and the Apple II. These chips are simple enough to allow a beginner to achieve a reasonable understanding of their operation and complete enough in their design that the conceptual lessons learned easily extend to contemporary processors.
6. "Port" comes from "portable" and is something of a misnomer, as most early "ports," like *Space Invaders* for the 2600, were more precisely *rewrites* on alternate hardware. Other arcade ports that did not fully exploit the 2600's hardware

include the infamous *Pac-Man* (see *Ms. Pac-Man* on the 2600 as an example of what the game could have been), *Zaxxon*, and *Popeye*.

7. Hexadecimal is a base-16 numbering system that allows eight binary switches to be displayed in two digits. In the "ones" place, *A, B, C, D, E*, and *F* represent *10, 11, 12, 13, 14*, and *15*, respectively. An *A* in the "tens" place represents ten sixteens, or 160. For this study's purposes, all that is required is an understanding that these are numbers and that the "$" preceding each hexadecimal number is a signpost that we are using base-16, not base-10.
8. In this room, the "darkness" was technically an object. The darkness, the dot, and one extra item were enough to create the flash.
9. An engaging introduction to the topic of the dehumanization of the programmer is Edward G. Nilges's "PRACTICAL DECONSTRUCTIVE CODING."
10. *Extreme Racing*'s project lead mentions that tracks were shared between the arcade team and the N64 team, meaning that they were both using approximately the same format for their content, truly porting levels from platform to platform rather than rewriting the game. The levels from the original arcade game were going directly into *Extreme Racing* and three levels from *Extreme Racing* were borrowed and placed into a later arcade release (IGN Staff "Exclusive Interview").
11. This section can be found in an expanded form in "Inviting Subversion: Metalepses and Tmesis in Rockstar Games' *Grand Theft Auto* Series" in *The Meaning and Culture of Grand Theft Auto Critical Essays*, ed. Nate Garrelts (Jefferson, NC: McFarland Press, 2006), 210–25.
12. After a controversy appeared regarding the modification, Wildenborg appropriately hid this text and the locations of pictures of the modification in his webpages through the use of "html comments," a method usually used by html coders to leave messages for other coders viewing their pages' code. To read the quoted material, web users had to hack Wildenborg's page to access the trivially hidden content from within the web page's html code, a method not coincidentally similar to what Wildenborg did to hack *San Andreas*. He has, as of 13 January 2008, restored the text to his site.

Works Cited

Aarseth, Espen. *Cybertext*. Baltimore: Johns Hopkins University Press, 1997.

"Atari 2600 - Space Invaders (Atari)." *www.atariage.com* (6 November 2004). Accessed 7 August 2005.

"Atari 2600 Hacks - Space Invaders Arcade." *www.atariage.com* (19 November 2004). Accessed 7 August 2005. .

Bailey, Wm. Ruffin, Cor Ligthert, Herfried K. Wagner, et al. "Why is minus one (-1) equal to true in VB again?" *microsoft.public.dotnet.languages.vb* (Usenet, 21–23 June 2004). Accessed 13 August 2005

Barr, Roger, ed. "Hacked Rom Reviews!" www.*i-mockery.com*. Accessed 12 August 2005.

Blue Zircon. "Thompson 1928." *www.gtagaming.com* (26 July 2005). Accessed 12 August 2005.

Bogost, Ian. *Unit Operations*. Cambridge: MIT Press, 2006.
bunivfan. "SF Rush Hidden Track Code." *rec.games.video.nintendo* (Usenet, 13 January 1998). Accessed 10 August 2005.
DieselGT. "'Tumbler' Batmobile." *www.gtagaming.com* (17 July 2005). Accessed 12 August 2005.
Dragoncarer. "AOD Bugs definitive list." *db.gamefaqs.com* (9 December 2003). Accessed 7 August 2005.
IGN Staff. "Exclusive Interview: Ed Logg." *ign64.ign.com* (1 April 1998). Accessed 7 August 2005.
"The Hidden Leonardo." *www.nationalgallery.org.uk* (2001). Accessed 12 August 2005.
"History of the 11th Annual Interactive Fiction Competition." *ifcomp.org/comp05/history.html* (2005). Accessed 8 August 2005.
IGN Staff. "Exclusive Interview: Ed Logg." *ign64.ign.com* (1 April 1998). Accessed 7 August 2005.
___. "Rush: The Rock Not Coming to N64." *ign64.ign.com* (17 November 1997). Accessed 7 August 2005.
___. "You're Going to Alcatraz." *ign64.ign.com* (19 March 1998). Accessed 7 August 2005.
Mansfield, Richard. *Machine Language for Beginners*. Greensboro, NC: Compute! Publications, 1983.
Montfort, Nick. *Twisty Little Passages*. Cambridge: MIT Press, 2003.
Miller, Carolyn. "Writing in a Culture of Simulation: Ethos Online." In *The Semiotics of Writing: Transdisciplinary Perspectives on the Technology of Writing*. Ed. Patrick Coppock. Turnhout, Belgium: Brepols, 2001. 253–79.
Muckelbauer, John. Review of *The Terministic Screen: Rhetorical Perspectives on Film* by David Blakesley. *JAC* 23.4 (2003).
Naughton, Tom. *Pakrat* "About" page. *pakrat.fragland.net* (10 October 2001). Accessed 14 August 2005
Nilges, Edward G. "PRACTICAL DECONSTRUCTIVE CODING, an essay in the critical theory of computer science." *comp.lang.basic.visual.misc* (Usenet, 16 November 2001). Accessed 10 August 2005.
Pomaville, Leann, ed. *Quake Woman's Forum*. 1999. *planetquake.com*. Accessed 11 August 2005.
"San Francisco Rush–Cheat Codes & Secrets–GameFAQs." *gamefaqs.com*. Accessed 7 August 2005.
Schiesel, Seth. "Video Game Known for Violence Lands in Rating Trouble Over Sex." *New York Times* (21 July 2005).
Stevefel; Tal A. Funke-Bilu; bliss; et al. "SF Rush Alcatraz track: Relase [sic] the code Atari-Games !! (please)." *rec.games.video.nintendo* (Usenet, 18 January–18 February 1998). Accessed 10 August 2005.
"Tips, Cheats, and Easter Eggs." *www.atariage.com/hint_list.html* (10 October 2004). Accessed 7 August 2005.
US Dept. of Energy. "Brookhaven 1958 Video Game - DOE Research and Development (R&D) Accomplishments." *Office of Scientific and Technical*

Information (*www.osti.gov/accomplishments/videogame.html*). Accessed 7 May 2005.
Wildenborg, Patrick. "PatrickW GTA Modding | Home." *patrickw.gtagames.nl/index.html*. Accessed 10 March 2006.
Wolf, Mark J. P., ed. *The Medium of the Video Game*. Austin: University of Texas Press, 2001.

Games Cited

Adventure (Atari VCS/2600). Sunnydale, CA: Atari, 1978.
Descent (Windows). Champaign, IL: Parallax Software, 1995.
Grand Theft Auto: San Andreas (PlayStation 2). New York: Rockstar Games, 2004.
Grand Theft Auto III (PlayStation 2). New York: Rockstar Games, 2001.
Quake 1 (Macintosh). Santa Monica: id Software/MacPlay, 1996.
Quake 2 (Windows). Santa Monica: id Software/Activision, 1997.
Quake 3 (Macintosh). Santa Monica: id Software/Activision, 1999.
Space Invaders (Atari VCS/2600). Sunnyvale, CA: Atari, 1980.
Space Invaders Arcade (Atari VCS/2600). Atariage.com, 1999.
San Francisco Rush: Extreme Racing (N64). Milpitas, CA: Midway Games West, 1997.
Tank (Coin-Op). Kee Games, 1974.
Tomb Raider: Angel of Darkness (Macintosh). Austin: Aspyr, 2004.
Tomb Raider: Chronicles (Macintosh). Austin: Aspyr, 2000.
Zork (various platforms). Cambridge, MA: Infocom, 1979.