I AM ERROR

The Nintendo Family Computer / Entertainment System Platform

Nathan Altice

The MIT Press Cambridge, Massachusetts London, England

This book was set in Filosofia by Toppan Best-set Premedia Limited. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Altice, Nathan.
I am error : the Nintendo family computer/entertainment system platform / Nathan Altice.
pages cm.—(Platform studies)
Includes bibliographical references and index.
ISBN 978-0-262-02877-6 (hardcover : alk. paper)
1. Nintendo video games. 2. Video games—Design. 3. Nintendo of America Inc. I. Title.
GV1469.32.A55 2015
2014034284

10 9 8 7 6 5 4 3 2 1

I AM ERROR

0

During his quest to find the elusive Island Palace, Link, protagonist from the 1987 Nintendo Entertainment System (NES) videogame *Zelda II: The Adventure of Link*, visits a small house in the Town of Ruto. When Link approaches its sole resident, a portly, bearded fellow in purple attire, the man says, "I AM ERROR." Until Link speaks to another character further along in his quest, any interaction with Error yields the same curious result.

For many players, the cryptic message appeared to be a programming flaw, as if the game's code mistakenly found its way to the graphical surface, replacing the character's name with a diagnostic message. The real cause was less mysterious. In the game's original Japanese script, the man said:

オレノナハ
エラー　ダ・・・

"I am Error" is a passable literal translation, but a more natural read is simply, "My name is Error…"[1] An unlikely character name, perhaps, but that too was part of a poorly translated programmer joke. Link later meets a man named Bagu—a Japanese romanization of "Bug"—who looks identical to Error save for his red tunic. Together, Bug and Error were meant to form a pair of sly computer malfunction references, but the fumbled translation killed the joke and left a generation of NES players thinking the latter's name was more literal than intended.

Error's dialogue has since become an infamous part of the NES's legacy. Despite the accessibility of online FAQs, wikis, and detailed translation notes for the *Zelda* series, articles regularly appear dispelling the myths of the Error error for the players apparently still perplexed by his introduction.[2] Modern games knowingly parody the dialogue. The *Legend of Zelda*–inspired PC game *The Binding of Isaac*, for example, has a hidden room—purposefully framed as a glitch—whose bearded occupant utters the famous line in a cartoon speech balloon.

The NES era was rife with mistranslations. Link's prior adventure in *The Legend of Zelda* was famous for its cryptic and frequently misleading character dialogue. When the old man presented Link with an upgraded sword and said, "MASTER USING IT AND YOU CAN HAVE THIS," many players assumed they could take the blade and practice. Instead, they were meant to return when they had passed a specific heart container threshold. A misplaced letter in *Metal Gear*'s English translation created a comical temporal paradox when the enemy soldiers exclaimed, "I FEEL ASLEEP!!" *after* waking up. Winning a match in *Pro Wrestling* yielded the congratulatory text, "A WINNER IS YOU," proving the translator's heart was in the right place even if their grammar was not. Congratulations in general were a failing point for many NES games, as the lack of an "L" equivalent in the Japanese language led to many "congraturations," "conglaturations," and all variations in between. In some cases, errors became canon, as in *Metroid's* "barrier suit," which was mistranslated as Varia in the U.S. instruction manual and remained as such ever since.[3]

The circumstances of localization—the process wherein games are translated, linguistically and culturally—were much different in the 1980s. Development teams of ten or fewer people worked for a few weeks (in the worst case) or months (in the best case) to produce what we would now call "AAA titles," the big-budget videogames released by the industry's leading publishers.[4] Such was the case with *Super Mario Bros.* and *The Legend of Zelda*, developed concurrently by the same small team for two different media (cartridge and disk, respectively) across a span of months between 1985 and 1986. Compare this to the teams of a hundred or more programmers, sound designers, producers, artists, actors, and animators who work for several years on a single videogame. Of course, the scope of games and their underlying architectures were less complex in the 1980s, but Nintendo's creators also had less time and resources to devote to localization, presuming a game was even slated for release outside Japan.

Yet the same technical concerns that kept development schedules short also limited the quality and content of translation. Even with a completed translation in hand, text replacement was not a trivial

cut-and-paste job. The alphanumeric characters onscreen were literally characters, like any other graphic. The letter "A" and the upper half of Mario's head were cut from the same digital cloth; they were both tiles composed of bits that occupied the same memory in ROM. Translation was not just a symbolic act but also a material act: text had to be uncompressed in memory, its tiles redrawn, its characters substituted, its memory limitations reconsidered. If the Japanese text fit cleanly in a hardcoded dialogue box, but the English translation did not, meaning would often be sacrificed for the sake of economy. Why say "My name is" when "I am" gets the same basic point across with six fewer characters?

Any study devoted to Nintendo's first videogame console must necessarily be about translation—not only in a linguistic sense, manifested in Error's dialogue, but in a material sense as well. Translation has real social, economic, and cultural consequences beyond simple misinterpretation; translation takes place between circuits, cartridges, code, and cathode rays just as it does between human actors; and translation is inexorably and inevitably riddled with errors. As Derrida wrote in his "Letter to a Japanese Friend," translation is not "a secondary and derived event in relation to an original language"—in other words, not merely a supplement.[5] Applied to the production of technological objects that must enter cultures, markets, and domestic spaces, that must be made by bodies and touched by bodies, that must be made from rare earths and precious resources, translation does not simply derive meaning from prior sources—translation produces new meanings, new expressions, new bodies, and new objects.

I AM ERROR explores the complex material histories of the Nintendo Entertainment System—and its elder sibling, the Family Computer—that characterized its cultural reception, expressive output, and hardware design. In the 1980s, few NES players knew that their boxy gray videogame console was a cosmetic re-imagining of an older machine known in Japan as the Family Computer. Fewer still understood the machinations necessary to introduce a new console—or an Entertainment System, as Nintendo's marketing team chose to call it—to an American market that had apparently exhausted the videogame fad, ready to move on to the "superior" experience of personal computers. Nintendo, a century-old Japanese company that until a few years prior was primarily known in the United States as the maker of *Donkey Kong* and the Game & Watch, would be the unlikely savior of the dedicated videogame console. Riding the crest of a wave propelled by "next-generation" hardware and superior software like *Super Mario Bros.*, Nintendo would seize the global videogame market with unprecedented force.

But Nintendo's success was equally unlikely based on their console's flaws. The NES's distinctive front-loading cartridge slot, for instance, partly caused the console's infamous blinking screen, leading millions of players to blow into game cartridges as a quick "fix" that inevitably exacerbated the problem. The other cause was a proprietary "lockout chip" exclusive to the NES, meant to wall off piracy and unlicensed developers alike, a hardware tactic that Nintendo developed after such threats had damaged the Family Computer's Disk System peripheral's success in Japan. Such game-disrupting imperfections regularly spell commercial failure for consumer electronics, but players, developers, and software partners alike absorbed the Famicom's/NES's shortcomings into the fabric of gaming culture. Hardware limitations that governed the complexity of graphics and the number of digitized sound channels—or worse, caused sprites to flicker or slowed on-screen action to a crawl—are now part of the living legacy of videogames, referenced by fans and recycled in game design. Contemporary games that aim for nostalgic or "retro" appeal still mimic the console's shortcomings, since they provide quick visual and aural cues to a past era of gaming.

Purposeful malfunctions are a peculiar aesthetic decision in the software industry, where errors and bugs are commonly the bane of programmers. Neither videogames nor word processors nor operating systems nor ATM software nor drone guidance systems should have them, but all inevitably do. In each case, the stakes are successively higher, but programmers' attitudes to them are generally the same. To borrow from Derrida again, glitches operate as a "dangerous supplement" threatening to usurp the integrity of the work as an outside force that originates from within. But users' attitudes toward glitches cover a wide spectrum, dependent upon their nature and severity. A glitch in ATM software that makes the machine inoperable is an inconvenience; one that erringly deducts money from a customer's account is harmful. No game glitch can ruin a household. At their worst, they can make a game unplayable. At their best, they are entry points to new modes of play, exploration, and creative expression.

As Philip Sandifer wrote in his online *Nintendo Project*, "Far from being an aberrant error, the glitch is a central part of the experience of the NES, an era where the games frequently existed on a spectrum between function and breakdown."[6] The minus world does not ruin *Super Mario Bros.* It broadens the play experience, adds mystery, makes code mythic. But the glitch is not just about player experience. It is also a part of the complex function of the machine itself, a means for it to assert its material obstinance. This is the philosophical irony of Error's quote. He is not only

erroneously naming himself, but surfacing a paradoxical ontological statement. The NES speaks through Error, naming itself as error. But how can an object be both itself and not itself?

The question is not an empty rhetorical gesture. As we will see throughout the book, a platform is defined less by its positive aspects than by its limits and negations—by what it isn't—situated in the weird liminal spaces that bridge one computational architecture to another. I do not mean that one cannot hold an NES in their hands without risk of its disintegration or that we cannot consult Wikipedia for a list of the console's technical specifications. Indeed, the word platform has an important practical use for this and other studies of its kind. But the deeper we probe into the myriad variations and permutations of Nintendo's console, the less it coheres as a single definable object.

Methodology

It is hard to overstate the NES's importance, both to videogame history and to culture at large. At the peak of their market dominance, the NES and Famicom were in one out of every three homes in the United States and Japan.[7] Even thirty years later, the legacy of the Nintendo's first cartridge-based console looms large in gaming, art, music, graphic design, fashion, literature, and popular culture. But economic landmarks and market superiority are not the focus of this book. There are numerous popular and scholarly texts that focus on the NES and its influence, which tend to fall into a few broad categories: Nintendo's role in the larger history of videogames;[8] Nintendo's corporate history;[9] Nintendo's individual game designers;[10] Nintendo's impact on game players;[11] and Nintendo's iconic videogame characters.[12] However, the NES's importance as a computational platform is widely overlooked.

The NES, of course, was neither the first nor the most technologically advanced home console, but it did mark a transition point in the types of videogames that they could proffer. Its early games were either direct ports of or callbacks to arcade games, designed for short-burst, single-screen play. But within two years, platforming—pioneered in part by Nintendo's own arcade hit *Donkey Kong*—emerged as the dominant genre of the "third generation" of consoles. "Platform" was a catch-all term for any obstacle or structure the player-character had to traverse in order to reach a goal, like the girders that Jumpman scaled to save Pauline from Donkey Kong or the pits and alligators Pitfall Harry had to swing across to reach the hidden gold bars. Later platformers built upon these early prototypes, expanding traversal beyond single screens to elaborate

scrolling spaces. Platformers encouraged progressive, long-term play, coherent world design, and narrative development, characteristics antithetical to the arcade's quarter-consuming economy.

Nintendo's console was primed to capitalize on this transition in gameplay style—the Famicom was engineered with hardware-based scrolling, plentiful on-screen sprites, dedicated VRAM, and ample cartridge program ROM. None of these individual technical specs were revolutionary, but in combination they served as the architectural foundation for tile-based worlds tailored to character-based platforming. Even the Famicom controller, with its patented plus pad and dual action buttons, was geared for cardinal movement through 2D space. *Super Mario Bros.*, a landmark in videogame history, became the archetype of the genre, featuring a distinctive world (The Mushroom Kingdom), a memorable protagonist (Mario, the plumber), and a clear narrative goal (rescue the Princess from Bowser)—all novel features for console games at the time. Successors like *Castlevania*, *Mega Man*, *Metroid*, *Contra*, and *Ninja Gaiden* underscored the sophistication of console gaming and paved the way for a new Famicom era.

While Nintendo's console shared architectural similarities with several other machines, including the Atari VCS and the Commodore 64, its expressive capabilities were radically different. And those differences were more significant than sprite sizes or color counts. Historically, media scholars have overemphasized the visual aspect of digital media, a bias that Montfort,[13] Kirschenbaum,[14] and others have called "screen essentialism." Its prevailing assumption is that videogames' primary object of study is what players see onscreen, negating the role of graphics processors, joypads, sound circuitry, and other material concerns in shaping the expressive possibilities of software. *I AM ERROR*, in line with the platform studies methodology first developed in Montfort and Bogost's *Racing the Beam*, adopts a "bottom-up" approach to digital media, unearthing the code- and hardware-level decisions that fundamentally shaped the platform's creative affordances, cultural reception, and styles of play.

I AM ERROR considers videogames and their platforms to be important objects of cultural, material, and personal expression, alongside cinema, dance, painting, theater and other media. It joins the discussion happening in similar burgeoning disciplines—code studies, game studies, computational theory—that engage digital media with critical rigor and descriptive depth. But platform studies is not simply a technical discussion—it also keeps a keen eye on the cultural, social, and economic forces that influence videogames. No platform exists in a vacuum: circuits, code,

and console alike are shaped by the currents of history, economics, and culture—just as those currents are shaped in kind.

The book argues for the Famicom's material importance along three interconnected trajectories: first, as a platform directly informed by prevailing trends in arcade, console, and PC design, which in turn influenced the software the platform would support; second, as a pivotal platform in the evolution and popularization of the platformer genre, for which the Famicom's hardware was distinctly suited; and third, as a platform ideally positioned to catalyze console emulation in the 1990s, when both PCs and the Internet reached the necessary maturity to support an emulation ecosystem. With these three trajectories in mind, *I AM ERROR* offers a sustained technical analysis of how the platform was programmed and engineered, from code to silicon, and how those design decisions shaped not only its expressive possibilities, but also the perception of videogames in general. The book also defines the platform not only as a single console, but as a holistic network of objects and texts, including cartridges, controllers, peripherals, marketing materials, play environments, and emulators.

In this light, *I AM ERROR* diverges from the platform studies model by expanding and critiquing the notion of a platform as a stable configuration of hardware and software. As a Japanese product that was later exported to the United States, the Famicom was born in a vastly different cultural context than predecessors like the Atari VCS or Fairchild Channel F, shaped by considerations ranging from the size of Japanese households to the legacy of suspicion that Americans felt toward Japan post–World War II. Thus hardware and software alike underwent a number of translations that prior consoles never did, from the shape and color of the console to the censorship of potentially sensitive religious or political imagery that might offend international audiences. The Famicom's hardware and software were under constant revision, mutating to adapt to new cultures, new play practices, new markets, and new genres.

The NES hardware also reached its commercial obsolescence at a pivotal moment in the history of personal computing. Console emulation became feasible for PCs in the mid-1990s, allowing another important translation to take place—physical hardware, rendered in silicon and plastic, became virtual hardware, rendered in code. The NES was uniquely positioned to make this transition and led the way for the deluge of emulation that took place in the late 1990s. The features built into NES emulators spawned new forms of play, performance, and videogame archiving. Suddenly players could record gameplay movies, save games at any point, play online, alter graphics, load translation patches, and more.

The NES platform blossomed beyond the bounds of hardware, expanding its reach and capabilities past what Nintendo's engineers ever thought possible. There is no single configuration of hardware and software, no single processor, no sound or graphic that defines the Famicom as a platform. It is all and none of them. It is error.

Plan of the Book

I AM ERROR is neither a chronological review of the Famicom's hardware and games nor a history of Nintendo as a corporation. Larger surveys of videogame history are better equipped to furnish comprehensive lists of names, dates, and events.[15] The book is instead structured as a series of "deep dives" into specific hardware and software topics that will illuminate how the platform works at a base technological level. And while *I AM ERROR* covers the full scope of the Famicom's lifespan, it does so to track key developments in its material history along with the broader cultural and technological contexts from which those developments arose. Since Japan, the United States, Europe, and the rest of the world experienced concurrent but staggered trajectories of Famicom/NES development, this hardware focus often demands chronological backtracking.

Similarly, hardware and software examples are not chosen based on their gameplay merits or review metrics, but according to their relevance to the platform-specific topic. Fortunately, some of the best and most critically acclaimed games are also the most interesting to study. *Super Mario Bros.*, for instance, is not only one of the most lauded and bestselling videogames in history, it is also the consummate example of the Famicom's affordances. And despite the game's influence, there has never been a close analysis of how its successful design is tied intimately to the Famicom's Picture Processing Unit and cartridge ROM. Others, like *Gyromite*, *Wild Gunman*, or *Devil World*, while less lauded, offer fascinating glimpses at the Famicom's architectural design.

Chapter 1 ("Family Computer") introduces Nintendo's first cartridge-based console, the Family Computer (or Famicom), along with a few of its key predecessors. The chapter's first half tracks the development of the Famicom hardware, spearheaded by Nintendo engineer Masayuki Uemura, and its trademark industrial design, from case and controller to cartridge and circuits. The chapter's second half provides a detailed overview of the Famicom's computational architecture, including its custom microprocessor and graphical processing capabilities.

Chapter 2 ("Ports") rewinds to Nintendo's first forays into the U.S. arcade market via an ambitious but mistimed failure and the unlikely hit

that arose from its ashes. *Donkey Kong* is used as a case study to examine the broader cultural context of Japan's entrance into the Western videogame industry and specifically their software's categorization into the catch-all "novelty games" genre. The chapter also provides a comprehensive technical comparison of arcade *Donkey Kong* to its subsequent Famicom port, a crucial building block in its console future.

Chapter 3 ("Entertainment System") covers the challenging launch of the Nintendo Entertainment System, a hardware translation of the Family Computer suited to the demands of a troubled U.S. videogame market. Two of the console's initial marketing gimmicks—the Robotic Operating Buddy and the Zapper—are profiled in depth alongside the software they supported. Despite their limited use, both peripherals were important links to Nintendo's rich gaming legacy. The chapter concludes with a survey of the inconsistent, and sometimes inexplicable, translations used to transition content from a Japanese to a worldwide audience, ranging from the design and marketing of box artwork to the censorship of "offensive" in-game content.

Chapter 4 ("Platforming") is devoted to a technical exegesis of the seminal Famicom game, *Super Mario Bros.* The chapter delves into the game's source code and analyzes how the Famicom's architecture guided the game's design. Many of the hardware programming concepts introduced in prior chapters—scrolling, metatiles, data compression, attribute tables, palette swaps, sprite 0 hit—are expanded and explicated through *Super Mario Bros.*'s remarkable object-based software engine. That careful code work is followed by an analysis of the game's unique and sometimes unintended innovations, including player movement beyond world boundaries and the famous "minus world" exploit.

Chapter 5 ("Quick Disk") discusses the Japanese launch of the Family Computer Disk System (FDS), the introduction of Nintendo's proprietary disk format, and their combined effect on the design of the landmark adventure game *The Legend of Zelda*. The chapter examines the FDS's design, media, features, and flaws, and the peripheral's eventual demise in the face of software piracy. The chapter also explores the "miniature gardens" cultivated by *Zelda* and *Super Bros.*' designers as reflections of a tragic ecological crisis facing modern-day Japan.

Chapter 6 ("Expansions") examines the breadth and depth of innovations Famicom developers explored after their "exhaustion" of the stock hardware, the demise of the FDS, and the influx of competitors' "next generation" platforms. The chapter covers the emergence of new genres, hardware mappers meant to expand the Famicom's capabilities, unlicensed cheat devices, and the development and debut of *Dragon Quest*, one

of Japan's most influential series and a game whose scope and style were impossible to realize in a standard Famicom cartridge.

Chapter 7 ("2A03") dissects the Famicom's dedicated sound hardware, the Audio Processing Unit (APU), explores its five channels in detail, explains their use in software, and outlines the APU's role in the larger context of synthesis. It also surveys the sound enhancements provided by hardware expansions, including the Famicom Disk System and various extended mapper hardware. Finally, the chapter describes the APU's lasting importance to chiptunes, a music genre dedicated to exploring the sonic boundaries of vintage sound processors.

Chapter 8 ("Tool-Assisted") tracks the Famicom's rebirth through emulation via an analysis of tool-assisted speedruns, a specialized play style devoted to completing games as quickly as possible using software assistance. An overview of emulation's emergence in the 1960s is followed by a history of early NES emulators and their eventual evolution into the modern forms used in tool-assisted play. The chapter concludes with a look at the surprising new forms of play afforded through human/software collaboration and their explicit challenge to the notion of platforms as stable objects of study.

The book's afterword ("Famicom Remix") speculates on Nintendo's future based on its platform past; appendix A ("Famicom/NES Bibliographic Descriptions") issues a practical call for more rigorous enumerative bibliographies for videogames (and digital objects in general) and provides practical models for scholarly and critical use; and appendix B is a glossary of technical terms used throughout the book.

A list of sources concludes the text. I mention it explicitly since, in lieu of citing each Famicom/NES videogame inline or bloating the endnotes unnecessarily, all cited cartridges, disks, and ROMs are instead found here, using the model enumerative format detailed in appendix A.

Family Computer

1

In October 1981, encouraged by the dual successes of their breakout arcade hit *Donkey Kong* and the Game & Watch LCD handheld games, Nintendo president Hiroshi Yamauchi approached Masayuki Uemura, head of the hardware-focused Nintendo Research & Development 2 (R&D2), to begin work on a home videogame console.[1] Yamauchi knew that arcade games and cheap portables were excellent for short-term profits, but an inexpensive console with interchangeable cartridges could generate profits for years.[2] Atari had proven the model with their long-standing Video Computer System (VCS), the wood-paneled wonder that had dominated the U.S. home console market despite a slew of capable competitors. Yamauchi reasoned that Nintendo could manufacture their own home system, leveraging their popular arcade titles to entice consumers.

Uemura, alongside young engineer Katsuya Nakakawa, researched the feasibility of the console's technical requirements at the budget price Yamauchi demanded: ¥9800, (roughly $40 in 1983).[3] Thanks to Nintendo's recent experience with *Donkey Kong*, it was:

> The conclusion [Nakakawa] came up with was that a domestic game console looked to be a possibility if they IC'd [packaged as an integrated circuit] the Donkey Kong arcade machine's circuits and used them as a base. In the spring of 1982, a concrete development project had begun. The code name of the game console they set out to develop was the GAMECOM.[4]

For nearly a year, GAMECOM was the internal name for Nintendo's first cartridge-based home console.[5] But when Uemura mentioned the name to his wife the following spring, she suggested that if the console was meant to be a "domestic computer that's neither a home computer nor a personal computer," perhaps they should call it a "family computer."[6] And since the Japanese commonly shortened "personal computer" (パーソナル・コンピュータ) to *pasokon* (パソコン), Nintendo's family computer should have a similar nickname. Uemura loved the idea, as did the rest of his team. The Family Computer (ファミリーコンピュータ) released in Japan on July 21, 1983, and was soon after known by its affectionate abbreviation: ファミコン, or Famicom.[7]

As Ms. Uemura intuited, the portmanteau of "family" and "computer" described how Nintendo envisioned the machine to fit into the lives of those who purchased it. "Family" designated the console's range of social functions: Nintendo was bringing its popular arcade titles into the home, to be shared with the family, to become part of the family, and to be played in the family's social space. It would be a "domestic computer" in the most familiar sense. But the console would be more than a simple machine that played variations of ball-and-paddle electronic games—it would also be a powerful computing device, rivaling the *pasokon* that Japanese players were already accustomed to using for home videogame play.

Of course, the Family Computer's idealistic nomenclature was not solely motivated by Nintendo's domestic goodwill. Nintendo was selling a game console, and game consoles were seen as toys, meaning that parents would need to be part of the purchasing decision. Marketing the Family Computer to children would find limited success without the entire family's economic input, as Katayama's 1996 profile of Nintendo explains:

> As the Japanese name, "family computer," shows, the designers had the family market in mind. The product had to be priced so that parents would buy it. No matter how great the games, if mothers thought they were too expensive the machine would never take off. Nintendo therefore aimed for prices that children themselves could afford or at least would be able to convince their parents to lay out.[8]

Juggling Yamauchi's demands of affordability, approachability, and power posed major challenges for the R&D2 team. Not only was Uemura expected to produce a console at bargain prices, but it had to be future-proofed against potential rivals for three years.[9] Moreover, the Family Computer's spec software was *Donkey Kong*, an arcade hit built atop bleeding-edge

hardware that cost hundreds of thousands of yen to produce. A cartridge-based machine would not only have to be cheaper, but also more flexible, serving as a platform for *Donkey Kong* and a host of ports to come.

An Unconventional Stone

Uemura grappled with the problem for many months, consulting with the company's arcade engineers to figure out how they might transition *Donkey Kong*'s powerful hardware to an inexpensive home console.[10] Nakakawa's solution to "IC the arcade machine's circuits" was not a simple plug-and-play operation. At the heart of any modern arcade game was a microprocessor. And the choice of that microprocessor had important ramifications in cost and capabilities. An underpowered CPU might hamstring the number of sprites available onscreen, lower the game's palette options, or restrict the audio channel count. Alternately, if a CPU was too complex, it would be costly to manufacture and difficult to program.

In the early 1980s, there were two major players vying for the low-cost microprocessor market: MOS Technology's 6502 and Zilog's Z80. Both 8-bit processors were cheap but powerful, capable of driving a range of videogame consoles, PCs, and arcade games. Japan's arcade industry leaned heavily toward Zilog's microprocessor. The massive hits of the era—*Pac-Man, Galaxian, Galaga*—were all powered by the Z80.

The Family Computer nearly had a Z80 too. In fact, prior to Nintendo's decision to forge ahead with their own console, the Family Computer was nearly a ColecoVision.[11] In the U.S., Atari had maintained a near-deadlock on the emerging home videogame industry with the Atari VCS. However, the unlikely Connecticut Leather Company (Coleco for short), who had previously dabbled in derivative *Pong* clones and electronic handheld games, forged a short-term exclusive licensing agreement with Nintendo to bring *Donkey Kong* to their new console, a Z80-based machine that technologically trumped the elder VCS. The *Kong* partnership benefited both parties, spurring the ColecoVision to impressive first year sales and expanding Nintendo's market reach beyond the arcades.

Nintendo had great admiration for Coleco's port of *Donkey Kong*. Licensing the ColecoVision in Japan would allow Nintendo to develop console versions of its arcade games without a massive upfront investment in research, development, and manufacturing. In *The Golden Age of Videogames*, Roberto Dillon indicates that the two companies were close to a console licensing deal but could not settle on economic terms. In the end, he writes, "negotiations were abandoned when Nintendo declared it would design its own system instead."[12]

Coding for the ColecoVision's Z80 core certainly would have smoothed software conversions. Nintendo's *Radar Scope, Donkey Kong, Donkey Kong Jr., Popeye, Mario Bros.*, and *Donkey Kong 3* cabinets were all Z80-based, so porting to ColecoVision would have been simpler than translating code to a dissimilar microprocessor architecture. There was also the inherent risk of introducing new proprietary hardware to a fledgling videogame market. The less friction there was for third parties (i.e., publishers not directly affiliated with the console manufacturer) to port their games across multiple systems, the better. Without a steady stream of software, a new console was dead in the water. But when negotiations with Coleco broke down, Nintendo decided not only to design their own technologically superior console, but to forgo the Z80 altogether in favor of the 6502. This surprising decision ultimately came down to a mixture of corporate politics, managerial mandate, hardware licensing, manufacturer supply, and competitive strategy.

President Yamauchi had a long-standing reputation as a shrewd but imperious executive, possessing, according to his employees' accounts, an incisive but opaque business sense. Prior to the Famicom's development, for instance, Yamauchi unexpectedly forbade any collaboration between Sharp and Nintendo related to the new console. The order was a jolt for Uemura—Sharp was both his former employer and Nintendo's hardware partner for the Game & Watch. Nonetheless, Yamauchi insisted that Sharp's attention would be detrimentally split if they had to juggle between their handheld line and new console development.[13] With Sharp out of the picture, Uemura found little support from other electronics suppliers. Officially, vendors told him that parts were scarce due to a recent surge in demand for PCs and word processors, but he suspected that they were either reticent to wager on a risky console product—and Nintendo themselves—or had no idea how to produce the machine that Nintendo required.[14]

Uemura and semiconductor manufacturer Ricoh found one another at a fortuitous time. Ricoh had the advanced facilities Nintendo required and were currently only producing at ten percent capacity, an unsustainable shortfall for a large manufacturing operation. Uemura, along with engineers Nakakawa and Masahiro Ohtake, visited the semiconductor factory, where they were met with enthusiasm about a potential partnership. Hiromitsu Yagi, now a Ricoh supervisor, had worked at Mitsubishi in the late 1970s when they had partnered with Nintendo and overseen the chip design for the Color TV Game 6, one of Nintendo's early all-in-one consoles.[15] Uemura pitched the idea of a console made for *Donkey Kong* and, thanks in part to their employees' desire to "take the game

home," Ricoh agreed to take on the challenge.[16] However, the per-chip cost necessary to comply with Yamauchi's target price was not feasible at videogame console production numbers. Chip prices drove down at volumes of millions, not tens of thousands, so Nintendo made an extraordinary gamble in guaranteeing Ricoh a three-million chip order over two years.[17] Ricoh agreed to the deal, but feared that Nintendo was headed for an economic catastrophe. If the Famicom flopped, Nintendo would be stuck with millions of unused CPUs.

Chip prices aside, there was a larger roadblock to overcome: Ricoh lacked a manufacturing license for the Z80. As an alternative, they suggested the MOS 6502, a chip that was popular in U.S. and European consoles and PCs. The chip was similar in specs, affordable, and already licensed—a worthwhile replacement, with one caveat: it was virtually unknown among Japan's engineers. Surprisingly, both Uemura and Yamauchi saw this as an advantage, since the tradeoff in engineering complexity would pay off in hardware obfuscation and give Nintendo the competitive lead time they desired. In a 2011 interview with Nintendo president (and former Famicom programmer) Satoru Iwata, Uemura noted that Nintendo's peculiar choice "turned out lucky" for them, since other companies "wouldn't be able to make sense of it."[18]

Uemura's team initially balked at the change, since they too could not make sense of the 6502. Forgoing the familiar Z80 architecture posed major engineering challenges. For one, R&D2 had to build development tools from scratch. And instead of reusing prior source code, they had to tediously reconstruct their arcade games through observation. As Uemura explained, "the work required a lot of patience, including tasks such as watching the game screen and measuring the timing of animations with a stop watch."[19] In April 1983, R&D2 eased their burden when they hired Shuhei Kato, a young engineer who specialized in the 6502. The "Living 6502 Manual," as they called him, spurred on the final surge of software development and sealed Nintendo's microprocessor future.[20]

In the end, the Ricoh partnership satisfied all of Yamauchi's stipulations. Thanks to an inexpensive processor, the console would be cheap (though not on target—the Famicom debuted at ¥14,800, around $60 in 1983). And thanks to an unconventional choice of "stones," as semiconductors were called in Japan, Nintendo had a competitive edge. Reverse engineering the Famicom would prove as troublesome to competitors' engineers as it had for Nintendo's. From the outset, it was clear that Nintendo designed their console with proprietary control in mind. And despite the risk of alienating third parties, they wanted to be the ultimate arbiters of who could and could not develop software for the Famicom—a

predilection that would intensify as Nintendo took control of the worldwide videogame market (chapter 3).

The 6502 had one other material advantage: its die was one-quarter the size of the Z80. As a result, Nintendo's and Ricoh's designers were able to shrink the Famicom's plastic body and further slash the cost of the machine. Compared to contemporaries that used the Z80—ColecoVision, Sega's SG-1000, MSX PCs—the Famicom was more compact and less expensive. The design had both marketing and cultural advantages. The smaller Famicom, garbed in bright red and white plastic, had a toy-like appearance, sure to grab the attention of young children. Moreover, the diminutive size fit the tastes of Japanese consumers and the size of their domestic spaces,[21] making good on the Family Computer name.

With the microprocessor question settled, Nintendo began to work in earnest on the Famicom prototype in late 1982 and soon began courting another U.S. licensing partner. Ever since their entry into the arcade business, Yamauchi had had his sights set on the American videogame market. *Donkey Kong*, its arcade successors, and the Game & Watch were strong starts, but in America, Nintendo was still a minor Japanese player with a "weird foreign name." In a reversal of their aborted negotiations with Coleco, Nintendo decided to license the Famicom to a U.S. partner—one who ironically had their own weird foreign name.

In April 1983, Atari executives flew to Kyoto to inspect demo versions of *Donkey Kong Jr.* and *Popeye* running on prototype hardware.[22] According to Atari's Don Teiser, Nintendo's machine ran the arcade ports with "only minor display glitches."[23] Teiser's memo indicated Atari's interest in the console along with, true to form, a sizable list of President Yamauchi's stipulations (e.g., a minimum two million console order, Atari's limited access to Nintendo's hardware specifications, etc.). But the memo also indicated that he and the other executives present were withholding their true intentions from Nintendo. Atari was shopping for a successor to the aging VCS and its disastrous follow-up, the Atari 5200. While they openly courted Nintendo, internally they were weighing a competing prototype, codenamed MARIA, developed by General Computer Company in Cambridge, MA. This alternative looked to be the "superior machine," according to the memo, but uncertainty regarding the chip's large-scale manufacturing costs kept Atari in talks with Nintendo. In short, they were purposely delaying their decision so they could pick the better machine.

Once again, an international partnership was not meant to be. Although Atari ultimately opted to use MARIA in the Atari 7800 (due in part to Nintendo's impatience with Atari's waffling), grander mitigating

circumstances intervened on Nintendo's behalf. By the end of 1983, the U.S. videogame industry collapsed catastrophically (chapter 3). Had Nintendo partnered with Atari, the U.S. Famicom would have likely been collateral damage. Even the promising ColecoVision was among the casualties. Thanks to an economic twist of fate, Nintendo would have to forge ahead alone.

Red, White, and Gold

Readers unfamiliar with the Family Computer might be surprised by its size: it measures 22cm long, 15cm wide, 6cm deep, and weighs approximately 620g. Its curious profile looks more like the torso of a plastic robot than a powerful computing platform. In short, the Famicom looks nothing like the personal computers with which it shared a name.

Nintendo's early consoles—all simple variations of tennis, tabletop, racing, and brick-breaking games—had already experimented with novel, colorful designs. The deep oranges, reds, and yellows of the Color TV series were far afield from the faux wood-paneled furniture style of U.S. consoles, thanks in part to the influence of a young designer named Shigeru Miyamoto. The cherubic Miyamoto is now one of Nintendo's most prominent representatives, widely hailed as one of the greatest innovators in videogame history. He has designed, produced, or directed the lion's share of Nintendo's most prized franchises, from *Donkey Kong* and *Super Mario Bros.* to *Nintendogs* and *Pikmin*. But prior to Miyamoto's industry ascension, he worked as an industrial artist. His first jobs at Nintendo included designs for mahjong labels, playing card stencils (*hanafuda*, or Japanese playing cards, were Nintendo's original gaming industry), arcade cabinet exteriors, and the 1979 console all-in-one Color TV Game Block Breaker (カラーテレビゲームブロック崩し).[24] Miyamoto brought a playful sensibility to his industrial designs, emphasizing simplicity, accessibility, and fun. The bold colors and compact form factors of Nintendo's earlier consoles would carry over to the Family Computer.

While Miyamoto was not directly involved with the Famicom's external design, Uemura was, and the latter established a list of specification guidelines to help shape the console's look. These included the necessity of two controllers, the ability to store them on the console (another legacy of the Color TV consoles), the number of controller buttons, the various ports and power connectors, and the desire to have the cartridge dimensions "be about the same as an analog cassette tape." Curiously, in spite of the console's name, Uemura wanted the Famicom to look like neither

a computer nor a toy, but something wholly different.[25] Ricoh's designers concurred. The console should not be judged on looks alone:

> If the system's exterior resembled an audio device, for example, consumers would make judgments on the product's price and value based on preconceptions. [Ricoh] instructed the team to design the exterior in such a way that people wouldn't be able to make snap judgments about it.[26]

While Uemura later admitted that he had failed to realize his design goals, there is no question that the final console stands out from its consumer electronics peers. The Famicom body is replete with ridges and angles. Numerous recessed surfaces, buttons, levers, hinges, and vents combine to form a unique plastic topography (figure 1.1).

The front of the console slopes forward slightly to display a slender aluminum plate printed with the console's name (in English) and the Nintendo logo. Behind the angled surface there are three mechanical switches: reset, power on/off, and a large slider to help children lever the cartridges out of the console's interior.[27] Cartridges are inserted vertically into a narrow slot behind the lever. Since the slot exposes the cartridge card edge connector (and the console's interior), a hinged plastic flap covers the hole when no game is present. Again, Uemura hoped to match the cartridges' dimensions with cassettes so they could be stored in standard tape cases. Cartridges' affinity with cassettes had an important cultural and marketing resonance, due to the recent worldwide success of

1.1 The Nintendo Family Computer. (Source: Evan Amos, Wikimedia Commons)

Sony's Walkman, which introduced the compact, high-technology Japanese aesthetic to much of the world.[28] "Cassettes" were already a common moniker for game cartridges in Japan, and Nintendo wanted to maintain that material affinity for their software.[29]

As Uemura's guidelines specified, the sides of the Famicom have recessed edges cut to house the console's wired controllers. Both controllers are rounded along the edges but have an additional raised molding around their perimeters that allow them to nest within their respective cradles without falling out. Cords emerge from the controllers on either side—from the upper left on controller I, upper right from controller II—rather than the top. Though the placement looks sleek when the controllers are stored, as there are no cords sprouting from the top, it makes the controllers awkward to hold, since the cords emerge where the hand naturally grips the joypad.

The sole feature of an otherwise barren front edge—a 15-pin expansion port—is evidence of the Famicom's future-proof design.[30] Yamauchi originally requested that the Famicom support a number of computer peripherals, including a cassette storage drive, a keyboard, and a modem, but eventually told his engineers to nix these add-ons in the interest of cost reduction. Furthermore, fewer peripherals made the Family Computer appear less intimidating to new users.[31] Nonetheless, Nintendo had the forethought to leave the expansion port. And indeed, as the Famicom gained popularity, the peripherals excluded from its initial launch were eventually added both by Nintendo and third-party manufacturers. The expansion port would support keyboards, all manner of controllers and joysticks, light guns, 3D glasses, an inflatable motorcycle and punching bag, a drawing tablet, a karaoke microphone, and even a modem.[32]

The Famicom's final distinguishing trademark is its color. While the bulk of its body is white, the switches, logo plate, cartridge slot cover, expansion port plug, controllers, and bottom plate are all painted a rich maroon.[33] The controllers feature two additional accent colors: each of the buttons, their labels, and two thin horizontal decorative lines are painted black and surrounded by a brushed gold face place. The red, white, and gold triumvirate is as iconic in Japan as the gray, black, and red of the NES are in the U.S. and Europe.[34]

Close Playing

The physical forms of computational devices, from mobile phones to room-size server racks, are not benign; they participate in and structure social, personal, cultural, and economic spaces. Upright arcade cabinets,

for instance, were played while standing. Their form facilitated fluid movement between machines—players were meant to insert a quarter, play for a short period of time, then move along to the next game. Screens were commonly angled backward to allow players to lean into the cabinet, shielding them from any exterior distractions. If players sat at upright cabinets, they perched on high stools, the common furniture of the pubs and taverns that initially hosted such machines. Cocktail arcade cabinets likewise reflected their social milieu. Their flat, squared surfaces and low profiles resembled tables. Players sat on either side of the machine and looked down at the monitor, which was mounted with the screen facing the ceiling. Unlike the sheltering hood of upright cabinets, cocktail cabinets encouraged social play. Accordingly, cocktail games were commonly programmed either for cooperative play or to rotate between multiple competitors in turns. Players could sit comfortably, rest their drinks on the plexiglass tabletop, and enjoy a videogame together.

When videogames moved into the home, consumers had to be taught how, where, and with whom to play them. In single-television homes, the TV was usually located in the living room, where it could be shared by the family. Since the videogame console required a television, it resided there too. The earliest home consoles featured simple variations on the ball-and-paddle play pioneered by *Tennis for Two*, *Pong*, *Breakout*, and their imitators. Due to both limited technology (i.e., programming a capable artificial opponent) and their arcade heritage, these games normally required two players. The living room, already a site of family gathering, was conducive to social play.

Early commercials for the Magnavox Odyssey, Fairchild Channel F, Coleco Telstar, and other contemporary consoles showed variations on the same themes: this is how the console connects to the television, this is how you select or insert different games, this is how you hold the joystick or paddle, this is how you and your friends and family gather around the television to play.[35] None of these practices were taken for granted. The instructions for play, including the arrangement of bodies and machines, were built into videogame advertisements, manuals, and even product packaging.

The Family Computer, as its name implied, was a console designed for domestic spaces. Two people playing simultaneously were close to both one another and the console, since the wired controllers kept players in close proximity.[36] Smaller televisions and shorter cables meant that videogames were played close to the screen—an ideal spatial configuration for most Japanese homes, which tended to be much smaller than their American counterparts. But in spite of its name, the Famicom was designed

to rest on the floor, a table, or a low shelf. PCs in the 1980s comprised several bulky components: a monitor, a case to house the internal processors and memory, a keyboard, disk drive, etc. Computers took up a lot of space. A desktop and chair were best suited for both the size of the machines and long-term computer use, especially typing. The linguistic legacy of the "desktop computer" and the metaphor of the desktop as the default state of the graphical operating system indicate as much.

Had the Famicom ended up looking more like a conventional PC as originally planned, it would have been better suited to the "vertical" orientation of desks and chairs. Uemura considered such a configuration, but he soon realized that most players would not want to use their consoles on desks. A "horizontal" orientation better suited the tastes of Japanese consumers, whom he thought "would probably be lying on the floor or snuggled up inside the *kotatsu* (a foot warmer with a quilt over it) when they played, not sitting in front of a solid, stable desk."[37] Uemura's conclusion seems obvious now, but home videogames in the 1980s were still novel enough that their position in domestic and social space was not yet codified.

In the early 1980s, Japanese consumers were still acclimating to computers' presence in the home, so any added sense of familiarity could ease a product's introduction. The Family Computer joypad was new compared to competitors' controllers, but it was a recognizable interface for the millions of people who had played Nintendo's Game & Watch, the aptly named portable that packed an electronic game, clock, and alarm into a compact handheld enclosure.

Though Game & Watch titles resembled videogames, their simple circuits generated no video signal. Instead, segmented liquid crystal displays, identical to those used in calculators, blinked on and off to produce simple animations. In Nintendo's first Game & Watch title *Ball* (1980), a cartoon man attempted to juggle several balls simultaneously. The man's body, arms, and each individual position of the balls' three possible arcs were LCD segments that could be "lit" to create motion. In a 2010 interview with Satoru Iwata, Game & Watch developer Takehiro Izushi described how calculator circuits could process eight digits with seven segments apiece, so games like *Ball* were limited to those fifty-six segments, plus a few ancillary segments reserved for "decimal points and symbols like the minus sign."[38] The engineering and design innovation was appropriating these segments for use as game objects rather than a calculator's numeric display.

Adhering to the constraints of a calculator LCD created unique design obstacles. As discrete segments, object animations could not deviate

beyond their fixed course, nor could they overlap. Since only one segment object could occupy a given portion of the screen, even when unlit, animation was jerky. The monochromatic LCD graphics were thus more akin to the grid of bulbs used to animate sports stadium displays or traffic signs than a console video processor.

Nonetheless, Nintendo devised a remarkable number of gameplay variations from such a restrained palette. Early games like *Octopus* (1981) were based around straightforward objectives like sneaking past an octopus' coiling tentacles to retrieve a treasure chest. As the platform matured, the games grew in complexity, culminating in streamlined conversions of Nintendo's arcade and console games, such as *Donkey Kong* (1982), *Mario Bros.* (1983), and *The Legend of Zelda* (1989). To increase the range of gameplay options, Nintendo added graphics overlays and enlarged the Game & Watch to house two screens (figure 1.2).[39]

Gunpei Yokoi, Nintendo's head of R&D1, led the development of both the Game & Watch concept and its hardware. Yokoi had been a mainstay of Nintendo's games division since he first designed the *Ultra*

1.2 The lower screen of the Game & Watch Multi Screen *Donkey Kong* handheld, the first Nintendo product to feature the "Plus" Button. In its startup state (pictured), one can see the entire spectrum of LCD segments lit simultaneously.

Hand (1966),[40] a collapsible plastic lattice with handles at one end and a pair of rubber cups at the other, meant for grasping small objects. The clever mechanical device, handpicked for production by President Yamauchi, was a big hit for Nintendo, resulting in Yokoi's promotion from maintenance man to toy designer. During his tenure at pre-Famicom Nintendo, Yokoi invented many inspired gadgets: the *Ultra Machine* (1967), an automatic baseball pitching mechanism; an electric *Love Tester* (1969); the *Light Telephone* (1971), a short-range walkie-talkie using photo cells; the *Custom Lion* (1976) light gun target shooting game; the *Chiritorie* (1979), a radio-controlled mini-vacuum; and the mechanical puzzle *Ten Billion* (1980).[41]

Yokoi was instrumental to Nintendo's success, not only due to his inventiveness, but also his influential approach to product design. In a series of interviews published in Japan in 1997, Yokoi articulated his design philosophy as 枯れた技術の水平思考, which translates to English clumsily as "lateral thinking for withered technology."[42] In English, "lateral thinking" denotes a creative, unexpected approach to problem solving, a strategy Yokoi applied to outdated, inexpensive, or otherwise "off-the-shelf" technology. Yokoi famously devised the Game & Watch concept after noticing a train commuter whiling away the time on his pocket calculator.[43] If a simple calculator could engross the man, why not a pocket-sized videogame?

In the 1970s and 1980s portable calculators were a consumer electronics sensation, especially in Japan. In 1972, the so-called "Calculator Wars" were sparked by tremendous sales—over one million in its first ten months—of the Casio Mini, a low-cost, ultra-thin calculator. What was formerly a vestige of Japanese office culture transformed into a mainstream computing device and catalyzed Japanese manufacturers' investments in electronics miniaturization, liquid crystal displays, and solar cells.[44] Though nearly sixty companies entered the war, eventually two main competitors—Sharp and Casio—battled at the front lines. And the stakes were high. According to Johnstone, "By 1980, annual production of calculators topped 120 million units, with the Japanese accounting for just under half the total."[45] Sharp and Casio sparred for a decade, introducing progressively cheaper and smaller models, until Casio struck the death blow in 1983 with a solar-powered calculator measuring 0.8mm thick.[46]

By the time Yokoi saw the salaryman absorbed in his pocket calculator, LCDs and miniaturized ICs were cost-effective enough to use for an affordable handheld gaming system. And though segmented displays were originally devised for displaying ten digits and a few mathematical

symbols, they could be creatively repurposed to animate rudimentary game graphics.

Plus Controller

Nintendo has consistently shown an ability to mine engaging games and hardware from obsolete and outdated technologies: *Donkey Kong* arcade units were reworked versions of unsold *Radar Scop*e boards (chapter 2); the clunky 3D of the Virtual Boy resurfaced in the Nintendo 3DS; the everyman Mario has worked in professions ranging from doctor to chef; the mechanical *Duck Hunt* rifle resurfaced as the Famicom light gun game *Duck Hunt*; and the gripping cups of the *Ultra Hand* are echoed in the grasping arms of the Robotic Operating Buddy (chapter 3). Unsurprisingly, Yokoi had a hand in each of these projects.[47]

An iconic element of the Famicom controller was a similarly repurposed innovation. Yokoi first devised the "plus controller," as it was originally known, for the Game & Watch conversion of *Donkey Kong* (figure 1.2). The gameplay of many Game & Watch titles took place on a single horizontal axis and thereby required only two buttons for control. In *Ball*, for instance, the player could press a "<LEFT" or "RIGHT>" button, each on its respective side of the handheld, to move the juggler's hands into position. In instances where gameplay required an extended range of motion, the buttons multiplied. In *Egg* (1981), for instance, two stacked pairs of buttons on either side controlled four possible diagonal positions of the wolf's arms. In fact, nearly all Game & Watch titles prior to *Donkey Kong* used four buttons for diagonal movements rather than conventional cardinal directions.[48]

However, arcade *Donkey Kong*'s run-and-jump gameplay required movement along both axes. Jumpman, the proto-Mario character, ran along girders, climbed ladders, and jumped pits and obstacles. To replicate this movement on the Game & Watch, the handheld would have needed five individual buttons—four for directional movement and one for jumping. To prevent uncomfortable button crowding while still accommodating the handheld's small footprint, Yokoi devised a novel control solution for a novel game.

Early console controllers derived from arcade controls, adopting some variation of joystick, paddle, or keypad to varying degrees of success. Atari's iconic VCS joystick, for instance, was ergonomic and easy to use, but its single button foreclosed input complexity of its games. The Intellivision and ColecoVision erred in favor of more buttons, but ended up with intimidating, overwrought controllers that looked more like

calculators or remote controls than joysticks. The problem facing all home console manufacturers was that arcade hardware had more leeway in controller design. Since arcade hardware catered to the needs of a single game, their control interface could be built to suit: *Tapper*'s (1983) controller was a beer tap; *Crystal Castles* (1983) provided an embedded track ball to navigate its axonometric architecture; *Super Off Road* (1989) used three conjoined steering wheels and accompanying accelerator pedals; *Skydiver* (1978) featured parachute ripcords as controllers. Home consoles, in contrast, had to sacrifice specialization in favor of generic designs that could accommodate a range of genres. When conventional joysticks were insufficient, publishers provided specialized controllers: wheels, gloves, guns, goggles, microphones, and even balance boards.

Yokoi's "plus" controller, nicknamed after the mathematical symbol it resembles, was an elegant solution that suited both manufacturing and gameplay needs. The flat directional pad took up little space on the handheld, fit comfortably in slim spaces without risk of breaking, provided easy access to all four cardinal directions with a single thumb press, and, most importantly, mapped logically to the two-dimensional spaces presented onscreen.

Uemura initially experimented with disassembled arcade joysticks while prototyping the Famicom's gamepad, but he worried that the joystick would be difficult to fix in place sturdily, break underfoot, or injure children if stepped on. Fortunately, R&D1 member Takao Sawano suggested that they transplant the Game & Watch directional pad.[49] When the rest of the team balked at the idea, Sawano "pulled a lead line out of a Game & Watch and connected it to a Famicom prototype, then invited the development staff to give it a try."[50] Once the team felt the plus controller's responsiveness, they decided to transplant it to the Famicom gamepad, albeit in a slightly larger size.

↑, ↓, ←, →, B, A, Select, Start

Unlike most home consoles, the Family Computer's supplied controllers were not a matched pair. The leftmost Controller I included, from left to right: plus pad, Select, Start, B, and A. Controller II, however, omitted the Select and Start buttons in favor of a late Uemura addition: a tiny microphone and a volume slider. Though the microphone input translated to a simple binary signal internally, meaning that the system could neither interpret pitch nor record audio, it did have a connecting line to the Famicom's audio output, allowing sound to pass through to the television speaker. In a nod to Japan's karaoke craze, Uemura was convinced

that players would be entertained "simply by hearing their own voices come out of the television set."[51] Developers apparently were not as entertained, since the microphone saw little use in games.[52] As a result, Nintendo cut the microphone from both a later Famicom revision and the Nintendo Entertainment System.

The asymmetrical controller arrangement, besides further stymying home repairs by making the gamepads non-interchangeable, also meant that player one always controlled menu navigation and pausing—functions conventionally reserved for the Select and Start buttons. This apparently minor detail had interesting ramifications for social play. In Japan, for instance, *Super Mario Bros.* players practiced a technique known as the "Start Kill."[53] When player two, controlling Luigi, would leap over a gap, player one would pause the game mid-jump. Once play resumed, player 2, unable to re-acclimate to their original trajectory, would fall to their death—an early instance of what is now known as "griefing" among videogame players.[54]

The Famicom controllers were also gently rounded on the edges and sized to fit comfortably in a child's hands. Their design allowed easy access to every button with two thumbs—the left drove the plus and Select buttons, while the right bounced between A, B, and Start—leaving the remaining fingers free to grip the controller securely. One of Uemura's stated design goals was to make the controller easy to use while looking at the screen.[55] Though this goal seems commonsense now, designing a controller that could effectively bridge the perceptual divide between hand and eye was a meaningful engineering problem. Arcade cabinets and handheld videogames had their controls and monitor embedded in the same physical housing. Increasing the distance between screen and input device demanded that players develop tactile mastery of the controls in order to keep their eyes fixed to the screen.

As controllers have become more complex, multiplying buttons, joysticks, bumpers, and triggers many times over, this input/output divide has grown more significant. New players unaccustomed to the evolution of controller design face a steep barrier to play. Diverting one's eyes to hunt for a button can spell disaster in a fast-paced, reflex-based videogame. The inherent learning curve required by complex modern controllers partly explains the appeal of simplified and/or integrated control schemes seen in mobile touchscreen devices, the Nintendo Wii, and Xbox Kinect.

By striking a balance between too few and too many buttons, Nintendo designed a controller that favored tactile control and gameplay complexity without alienating new players. But if the Family Computer

was meant primarily for arcade ports, why did its controllers stray beyond the plus pad and an action button? Partly we can credit Nintendo's engineering foresight. If the Famicom turned out to be a success, designers would want enough input leeway to represent the vast range of computer game genres. Limiting input to a single button would impose a challenging design restriction.

But there was also a computational reason behind the controller's button arrangement. The Famicom/NES was a member of the so-called "8-bit generation" of videogames, alongside the Sega Master System, Atari 7800, et al. The *n*-bit designator, while often used misleadingly in the history of videogame marketing,[56] tells us a lot about a console's computing capabilities. Computers make calculations using binary arithmetic, a numerical abstraction meant to mirror the physical states of semiconductor gates, which can either be open or closed. In other words, a computer can only count using two digits: 0 and 1. The 8 in 8-bit refers to the range of possible values the console's CPU can process. Binary is as a base 2 numeral system, so an 8-bit CPU can represent $2^8 = 256$ possible values, or the numbers between %00000000 and %11111111.[57]

However, the eight individual bits comprising a *byte* do not necessarily have to represent numbers. In programming, bits are equally useful as "flags" that signal the current state of a given computational object. But what do binary math and bit flags have to do with the Famicom controller? Consider the range of possible inputs that the controller allows: up, down, left, right, B, A, Select, and Start—eight distinct inputs that may be either pressed or not pressed. The state of the Famicom gamepad at any given time thus fits conveniently in a single byte, making the controller truly 8-bit—both chronologically and computationally.

Scraped to the Die

Removing the Famicom's plastic exterior reveals a dense arrangement of electronic components and integrated circuits packed onto a diminutive PCB.[58] The two most prominent ICs are the Ricoh RP2A03G,[59] which contains both the Central Processing Unit (CPU) and the Audio Processing Unit (APU) in a single package, and the Ricoh RP2C02G, which contains the Picture Processing Unit (PPU), the processor responsible for translating graphical data into video signals that display onscreen. These three processors form the core of the Famicom, handling all of its computational, audio, and graphical tasks.

The 2A03 CPU is not a custom die, but a modified version of the MOS Technology 6502, an 8-bit CPU whose introduction in 1975 transformed

the microprocessor market. It was powerful, straightforward to program, and several magnitudes cheaper than its closest competitor—a combination of features that made it overwhelmingly attractive to PC and videogame manufacturers looking to introduce affordable computers to a mass market. Throughout the 1970s and 80s, Atari, Commodore, Apple, and Nintendo all launched successful platforms based on the 6502 architecture. [60]

MOS Technology did not directly manufacture the Famicom CPU. Instead, Ricoh, a "second source" manufacturer, licensed the rights to produce and sell the 6502. And, as its model name suggests, the 2A03 was not simply a stock 6502. In custom microprocessor production, clients routinely cut features in order to reduce individual chip costs. For their VCS, for example, Atari used a 6502 variant dubbed the 6507, a streamlined package that reduced addressable memory to 8KB and eliminated interrupts. With several years' hindsight on their side, Nintendo opted for less drastic revisions. The 2A03's sole subtraction from the stock 6502 was its binary coded decimal mode, or BCD;[61] the most significant addition was the onboard APU (chapter 7).

In 1976, North American corporation Commodore acquired MOS Technology and entered the U.S. PC market with gusto, leveraging their ownership of the inexpensive chip to undercut competitors' prices. CEO Jack Tramiel, whose domestic business interests had been stymied several times by Japanese competitors in the past, aimed to blockade an "invasion" of the U.S. PC and videogame markets after his own unsuccessful challenge of the MSX PC standard in Japan. Despite being a cheaper and technologically superior machine, the Commodore 64 could not compete against MSX's domestic groundswell of third-party support (ironically spearheaded in part by U.S. rival Microsoft).[62] Tramiel decided that if Commodore could not break into the Japanese market, they would retreat and shore up the competitive borders at home instead.

Tramiel's opposition, fueled equally by post-World War II xenophobia and competitive sour grapes, could have caused significant problems for Nintendo. Had Commodore known that Nintendo planned to bring the Famicom to the U.S. in 1985, they might have rejected the 6502 license or inflated the manufacturing price. By working domestically with Ricoh, Nintendo apparently stayed off Commodore's radar, much to the latter's eventual chagrin. After the Famicom's stateside release as the NES (chapter 3), Commodore's engineers were convinced that Nintendo had illegally skirted their microprocessor patents. As Brian Bagnall reports, it took a bit of reverse-engineering to discover Nintendo's "modifications":

> [Commodore 64 programmer] Robert Russell investigated the NES, along with one of the original 6502 engineers, Will Mathis. "I remember we had the chip designer of the 6502," recalls Russell. "He scraped the [NES] chip down to the die and took pictures."
>
> The excavation amazed Russell. "The Nintendo core processor was a 6502 designed with the patented technology scraped off," says Russell. "We actually skimmed off the top of the chip inside of it to see what it was, and it was exactly a 6502. We looked at where we had the patents and they had gone in and deleted the circuitry where our patents were."
>
> Although there were changes, the NES microprocessor ran 99% of the 6502 instruction set. "Some things didn't work quite right or took extra cycles," says Russell. [...]
>
> The tenacity of the Japanese was obviously formidable. Russell offers an opinion on why the Japanese elected not to purchase chips from North American sources. "They looked at the patents and realized that we weren't going to let them come over and sell against us," he says.[63]

Ed Logg, veteran arcade programmer for Atari, made a similar observation while working on the ill-fated Tengen port of *Tetris*. Asked about the ease of coding for the NES, he said:

> Yeah, it was pretty similar...well, [Nintendo] basically used our patents. They violated Atari's patent while they were suing us, so it was the basic same scrolling algorithm and such. So it was pretty much identical to what we were dealing with. Most of the difficulty came from figuring out what registers and bits did what, and when.[64]

In either case, their discoveries were too late. By the time Russell, Mathis, and Logg could take a close look at its silicon, the NES had already arrived.

Recent reverse-engineering work by the Visual6502 project revealed that Russell's observations were correct.[65] MOS Technology's 1976 patent for their integrated circuit microprocessor specifically covered its ability to "provide decimal results" using a single binary adder, "thus significantly improving the speed of operation without suffering the cost of an additional decimal adder."[66] MOS's innovation saved cycles and manufacturing costs. To avoid either patent infringement or licensing costs, decimal mode had to be excised. However, Nintendo/Ricoh did not fabricate a new chip design based on the 6502—a costly, time-consuming process—nor even remove the circuitry that drove decimal mode. Instead,

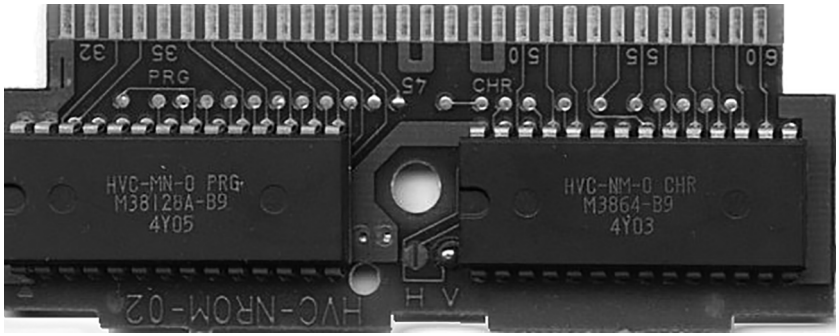
they made changes "only to the polysilicon mask" by "removing 5 transistors,"[67] disabling decimal mode rather than removing it.[68] In other words, Nintendo physically cut away any patent-infringing functions.

Atari's and Commodore's suspicions of hardware malfeasance were justified, but there is no evidence that Nintendo was trying to clone either company's systems. If anyone had the right to cry foul over intellectual property rights, it was Coleco. Despite their ultimate choice to forgo their Coleco partnership and source an alternative CPU, Nintendo's engineers clearly drew inspiration from the ColecoVision's video display processor (VDP) design. The ColecoVision's TMS9918 VDP (also used in MSX PCs) had several key features that resurfaced in the Famicom PPU: variable sprite size selection, a sprite overflow flag, an interrupt flag, and a special "coincidence flag" that could trigger on sprite collision.[69] Even more convincing were the similarities in graphic processor terminology. The ColecoVision's Pattern Name Tables, Sprite Attribute Table, and Pattern Generator Table resurfaced, with similar functions, as name tables, object attribute memory, and pattern tables in the Famicom. In contrast, beyond general terminology like sprites or VRAM, the Commodore 64's VIC-II VDP shared no such affinities with the Famicom PPU. And the Atari VCS's TIA was a wholly different beast, requiring more explicit coordination with the scanning electron beam than either the Famicom's or C64's graphics processors.[70]

While CPUs are crucial to a console's function, they have an indirect relationship to graphics rendering. When today's videogame players reference the 8-bit style of Famicom and NES games, they are mistakenly crediting central processing with picture processing. It is the PPU that, more than any other component, defines the Famicom's distinctive visual qualities. Nintendo and Ricoh may have made questionable ethical choices while engineering the Famicom, but accusations that the console is a direct theft or clone of prior systems discredits the PPU's (and APU's) crucial contributions to the Famicom's look and sound. Thanks to unlike VDPs, the Famicom, Commodore 64, and Atari VCS are fundamentally different machines. The ColecoVision is the Famicom's true elder sibling, related by looks despite not sharing the same brains.

Mario Tennis

Concealed within their cartridge shells, nearly all Famicom PCBs hold at least two ROM (read-only memory) packages. The first—PRG-ROM—stores the program code, the set of instructions that tell the CPU how and when to execute. The other—CHR-ROM—stores the character tile



1.3 The (inverted) PCB for Family Computer title *Nuts & Milk* features one of the simplest cartridge IC layouts. The 16KB PRG-ROM IC is on the left and the 8KB CHR-ROM IC is on the right. The traces labeled H and V also indicate the game's mirroring setting. (Source: bootgod, NES Cart Database)

patterns the PPU uses to populate the screen with graphics (figure 1.3).[71] In concert, these two ROMs contain the necessary raw materials to form a videogame.

The 6502 has a 16-bit *address bus*, meaning that it can access up to 64KB of memory.[72] However, the address bus size does not make the Famicom a 16-bit machine, since the 2A03's 8-bit *data bus* must construct and convey addresses one byte at a time. The address space is arranged in a specific, unchanging order, commonly called a *memory map*. The memory map is similar to a geographic map, charting the precise locations of inlets and outlets that give the programmer access to all of the console's functions—storage, graphics, audio, controllers, etc. The first 2KB of the CPU's memory map, for instance, are dedicated to the Famicom's internal RAM. Any data that is written to or read from RAM must be accessed through those specific addresses.

Early dedicated consoles either had hardwired game logic, as in *Pong*, or used interchangeable PCBs to reroute internal circuitry, allowing the player to electronically select from a small library of built-in games, as in the Magnavox Odyssey. When the Fairchild Channel F introduced removable ROM cartridges in 1976, the design space for videogames changed radically. Videogame hardware was no longer a closed system, but a proper platform, supporting a host of interchangeable software rather than a handful of games decided in advance by the console's manufacturer.

Relying on cartridges as a game-delivery medium means that a portion of the Famicom's CPU memory map—the final 16KB—is reserved for PRG-ROM. This type of architectural division is fundamental to how a cartridge-based system works. Without a cart inserted, the holistic

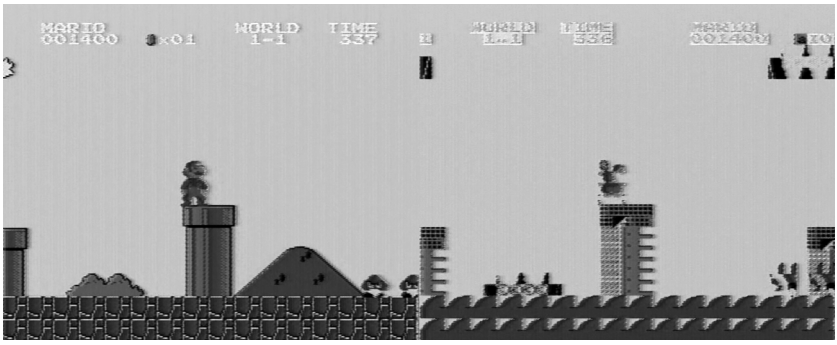
network necessary to orchestrate a game's code and graphics is incomplete. By offloading the source code and graphics patterns to separate ROM chips, Nintendo left the decisions of how the CPU and PPU would be used up to the programmer. The Famicom has no built-in BIOS or character set, no default software constraints on the look or style of its games.

But there is another important split at work, as the 2A03 reserves none of its memory map for character data. Instead, CHR-ROM is routed to the PPU's address space. Unlike the CPU, the PPU has a 14-bit address bus, so it can access only 16KB of memory, all of which is dedicated to the PPU's various palettes and "tables"—Nintendo's technical nomenclature for the memory locations that hold graphical data. The 8KB of CHR-ROM resident in-cartridge are called *pattern tables*, as they contain the 8x8- or 8x16-pixel patterns that comprise graphics tiles. The remaining 8KB are mapped to *name tables*, *attribute tables*, palette indices, and their respective mirrors.[73]

If we look closely at the PCB in figure 1.3, we see that each IC's traces are sequestered, subdividing the board's sixty pins into roughly equivalent halves. The physical substrate illustrates in silicon the division of labor that takes place when a Famicom cartridge is in play: the CPU executes the program code while the PPU fetches the pattern tables for display. So while the CPU may instruct the PPU to move a tile, update a sprite palette, or cease rendering, it has no direct access to the tile data nor the graphics onscreen. The cartridge's distinct "split architecture" mirrors the Famicom's internal computational division.

Another way to illustrate this point is to execute a decades-old Famicom hardware technique that I call the "cart swap trick."[74] After *Super Mario Bros.*'s release in 1985, Japanese players discovered that they could pause the game, carefully remove the cartridge, insert a *Tennis* cartridge, reset, play a few points, pause, swap back to *Super Mario Bros.*, reset, then resume play by holding A while pressing Start at the title screen. Due to similarities in RAM layout between both games, performing this arcane hardware rite transported Mario to odd variations of the game's fabled "minus world" (chapter 4). However, the trick's result is not as important as the visual artifacts that occur during the interstitial periods when carts are being swapped.

The left image in figure 1.4 shows *Super Mario Bros.* directly before cartridge removal; the right image shows the same screen after the swap (but prior to reset).[75] The underlying tile layout is identical, but the constituent tiles have changed. If you are familiar with the graphics from both



1.4 The "cart swap trick" helps visualize the underlying memory structure of the Famicom and its cartridges. (Source: NES-101 capture)

games, you can see that Mario's body, for instance, is now composed of odd bits of a tennis player, while the pipes are built from fragments of net and court. The Mushroom Kingdom is rebuilt in the image of Wimbledon.

Removing a cartridge while the system is on halts the game, since the CPU can no longer execute instructions from the absent PRG-ROM, but the PPU continues to render graphics. With no CHR-ROM in place, one can actually watch the contents of the PPU dynamic RAM (DRAM) slowly decay as it "forgets" the tiles previously resident on-cart. Once another cartridge is inserted, the PPU dutifully substitutes the new character tiles in place of the old, populating our familiar mosaic with unfamiliar bodies.

The key point is that the 2A03 and 2C02 are independent components, each administering dedicated regions of internal memory. In the absence of new directions from the CPU, the PPU attends to its graphical tasks, fetching tiles and updating the screen until instructed to do otherwise. Dislodging cartridges mid-game has limited gameplay use, but it provides a lovely, albeit glitchy, window into the Famicom's hardware processes.

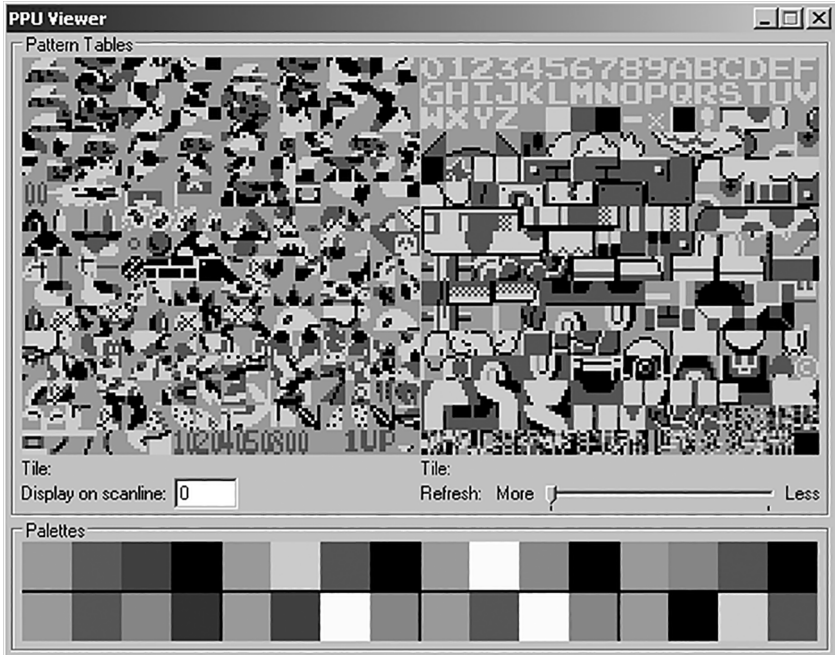
Patterns, Planes, and Palettes

The PPU divides its 16KB memory map—its Video RAM (VRAM)—into a series of related tables and palette indices, each contributing a crucial part of the data that comprises the tiles we see onscreen. But how are these tables related and what data do they contain?

The first 8KB, known as the pattern tables, are stored in CHR-ROM on the cartridge. Pattern tables contain the elements most recognizable as Famicom graphics, since they store the individual bits that define the 8x8- or 8x16-pixel mosaics we commonly call tiles. The PPU contains two pattern tables, each 4KB in size, which can hold 256 tiles apiece. Each pattern table is dedicated to either *sprites* or *background* tiles. Software emulators like FCEUX arrange the PPU's pattern tables into symmetrical grids that allow us to see the entire contents of CHR-ROM at a glance (figure 1.5).

Visually, sprite and background tiles are identical. In figure 1.5, the pattern tables are stacked side-by-side, divided by a thin center line, with sprites on the left and background tiles on the right. The choice of which tile type goes in which pattern table is up to the programmer, but each table contains either sprites or background tiles exclusively.

Whether sprite or background, each tile in the pattern table is described by sixteen bytes (or 128 bits) of memory. Those individual bits are divided into two 8-byte *bitplanes*, a matrix of binary values wherein



1.5 The contents of *Super Mario Bros.*'s CHR-ROM. Each pattern table contains 256 8x8 tiles. In *Super Mario Bros.*, pattern table 0 (left) contains sprites and pattern table 1 (right) contains background tiles. (Emulator: FCEUX 2.1.5)

each digit represents a single pixel of the final tile. Two bitplanes are necessary because tiles may access up to four colors, so two planes must be "stacked" atop one another to accommodate a 4-bit range of values.

If the Famicom only had four colors total, the two stacked bits from the pattern table bitplanes would be sufficient to describe any tile. However, the Famicom allocates thirty-two bytes of VRAM to eight individual palettes—half devoted to sprites and the other half to background tiles—containing four colors each.[76] In figure 1.5, the eight palettes defined in World 1-1 of *Super Mario Bros.* are arranged along the bottom edge of the PPU Viewer, each divided by a thick black line, with background palettes on top and sprite palettes on the bottom. Every sprite and background tile onscreen may select only one of its four allotted palettes at a time. However, the first (leftmost) color of each four-color block must be identical—shared across all palettes—limiting the PPU to rendering twenty-five individual colors onscreen simultaneously.[77]

In more modern videogame consoles, palette values encode RGB data, the combination of reds, greens, and blues blended to create a composite color. The Famicom, however, uses a hue saturation value (HSV) model based on the NTSC color wheel, which cycles approximately from blue to red to green to cyan.[78] Rather than a mixture profile, the byte stored for each palette slot is a lookup index for sixty-four possible HSV combinations.[79] However, several colors are duplicates (mostly in the black range) and one is technically beyond the NTSC's gamut range, reducing the NES's usable palette to approximately fifty-four colors.[80]

Though the shared color may seem like an arbitrary constraint, it is actually a clever means to "erase" portions of sprite tiles that are meant to be transparent. When a sprite is on top of a background tile, any color bit set to the shared color will permit the underlying background tile to show through. Mario's constituent sprites in *Super Mario Bros.*, for instance, use only three colors, drawn from the bottom left sprite palette seen in figure 1.5: red for his hat and overalls; a muted brown for his hair, eyes, sleeves, and boots; and olive for his skin and overall button. All other pixels in the Mario tiles are set to sky blue, but they render transparent, permitting the background to pass behind his body as he moves.

However, by rendering one color of every sprite palette transparent, each sprite tile is limited to only three colors. Again, in *Super Mario Bros.*, Mario is painted solely with olive, brown, and red; in *The Legend of Zelda*, Link is olive, brown, and lime green; in *Metroid*, Samus Aran is orange, red, and green. But observant Famicom players may notice that many game graphics appear to violate this rule. In *Mega Man 2*, for instance, Mega Man's head clearly uses more than three colors. The secret is sprite

stacking. *Mega Man 2*'s programmers used a sprite with a separate palette for the whites of Mega Man's eyes, which sits atop the other sprites that comprise his head. Numerous games use this trick to boost their characters' color count, though it too has drawbacks, since only eight sprites may be visible on a single scanline and sixty-four onscreen simultaneously. Using an additional sprite for Mega Man's eyes sacrifices a slot from both.

Names and Mirrors

CHR-ROM's division into sprite and background tables is not arbitrary—each has a role to play onscreen according to its strengths and weaknesses. Sprites tiles may be positioned freely on the screen, even atop one another, but they sacrifice a color to transparency and face hard limits both on how many may appear in a row and how many may appear onscreen. Background tiles have access to one extra color and can fill the entire screen, but they "move" only via scrolling, may never overlap, and remain fixed to a rigid underlying grid.

That rigid grid is a 32x30 matrix of values known as a name table, a fancy technical name for the game's background layer. However, the name table is not an image stored in memory corresponding to the graphics we see onscreen. Rather, each byte of the name table stores the reference to the appropriate tile in the background pattern table. If the PPU had to store pattern and palette data for every tile of the background, the memory demands would quickly escalate. Referencing each tile by index reduces the memory overhead considerably, allowing background tiles to repeat in long stretches without duplicating each tile's memory "cost." The data is stored once in pattern table memory, then referenced by its pattern number for any subsequent uses.

Though tiles dominated the 8- and 16-bit generations, they were not the sole graphical format. The Atari VCS' Television Interface Adapter (TIA) had no notion of pixels or tiles. It built its playfields and objects scanline by scanline, slimming memory costs to a sliver of what the Famicom would later require, albeit at the expense of graphical fidelity and programmer sanity, since per-scanline rendering required exacting timing and careful code economy.[81] Vector-based graphical systems like those used in the *Asteroids* (1979) arcade cabinet, which also ran on a 6502, and the Vectrex home console (1982) used an electron beam that could draw lines between arbitrary points onscreen rather than being locked to a raster grid. While vector graphics were sharper and less geometrically-constrained than their bitmap counterparts, they also sacrificed the latter's nuances of shading and color, since the display was

limited to monochrome line art. In each case, the choice of graphical displays was based on cost and design necessity, rather than a qualitative measure of which was "better" or "worse." For the Famicom's engineers, replicating their arcade games meant importing the same tile-based systems used in *Donkey Kong* and its kin.

Similarly, tile-based graphics were common in the early arcade and console era not because they were the only "natural" way to paint a screen, but because tiles were a particularly efficient means to store and display graphics data in low-memory architectures. For one, tiles eliminated significant graphical redundancy. The opening gameplay screen of *Super Mario Bros.*, for example, predominantly displays blue sky. To paint this sky, the PPU only has to reference one tile from the background pattern table and repeat it as needed. In a modern, "per-pixel" graphical processing unit, each individual pixel would need to be accounted for in memory. Even if we imagined a rudimentary black-and-white videogame that only needed to record whether each pixel was on or off (i.e., using one bit per pixel), accounting for each pixel would require hundreds, thousands, or millions of bits, depending on the resolution of the display. Adding even a handful of additional color possibilities increases the necessary memory by several magnitudes.

Today, that memory escalation makes a minimal impact on cost. 1MB of IC memory in 2014 costs roughly half a cent. But in 1983, that same amount cost approximately $2,000.[82] Nintendo had to use any means possible to keep costs low so their console would be affordable. Every spare byte of savings added up.

The name tables' memory architecture reflects those cost concerns. In the PPU memory map, Nintendo reserved 4KB for four name tables and their accompanying attribute tables. Excluding attribute data, each table was allotted 960 bytes—one byte for each tile reference drawn from the background pattern table. Since each name table comprises an entire screen, multiplying that area by four creates a graphical plane extending beyond the television's border. This is by design—the name tables are arranged contiguously in a 2x2 matrix that the PPU can smoothly track across via its dedicated scrolling registers. Hardware-level scrolling was a powerful feature in 1983, as most arcade and console games were designed for play on a single screen. Arcade titles like *Defender* (1980), *Rally-X* (1980), and *Scramble* (1981) had broken the one-plane barrier a few years earlier, but Nintendo's baked-in scrolling was remarkably forward-thinking for a home console.

Yet despite the four-screen name table layout, Nintendo only allocated 2KB of onboard RAM (technically CIRAM) to store name table data,

meaning that only two background planes are unique—the remaining half are mirrors. These are not actual mirrors, reflecting a reverse image across a shared axis, but hardware-level data duplication that produces two identical pairs of name tables. According to how the cartridge is wired (figure 1.3), the name tables are set to either horizontal or vertical mirroring, as pictured below:[83]

```
Horizontal  Vertical
[ A ][ A ] [ A ][ B ]
[ B ][ B ] [ A ][ B ]
```

The mirroring setting had important repercussions for scrolling games. With horizontal mirroring set, a character pushing past the right edge of name table A would scroll the left edge of that same background plane into view, breaking the illusion of a seamless space. Pushing toward the bottom edge of name table A, in contrast, would scroll the top edge of B into view, connecting contiguous spaces more naturally. In other words—and somewhat counter-intuitively—horizontal mirroring was best suited for vertical scrolling, and vice-versa. And since mirroring had to be set in advance, developers typically had to settle on a single scrolling direction for the entire game.[84]

Edge Cases

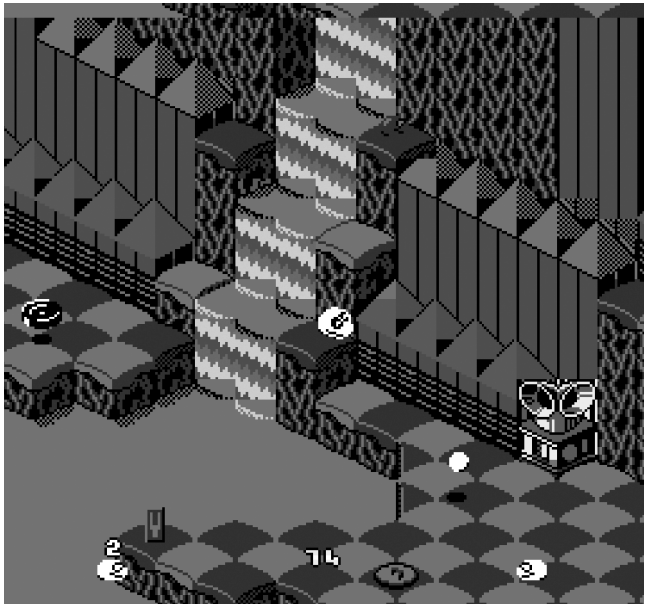
The final component of the PPU's VRAM arsenal is one of the trickiest to grasp, but also one of the most important, since it sets the hard limit for color variety in the background layer. Again, directly following each name table in VRAM is a 64-byte attribute table. The attribute table is like a second grid superimposed atop each name table, grouping its 960 tiles into an 8x8 matrix of 32x32-pixel areas (excluding the final row, which is truncated by half).

The pattern table bitplanes described above contain only two bits of the necessary four to describe a tile's color. The pattern table bits select which of the four colors within a palette a tile will use, but not the individual palette from which those colors are selected. For background tiles, these final bits are supplied by the attribute table. Again, each attribute table has only sixty-four bytes of palette data to allocate to the entire name table, so each attribute byte defines the palette values for one 32x32-pixel area. And since each palette value requires two bits to use in conjunction with the pattern table bits, only four palette values may be assigned within that area. In other words, every 16x16-pixel unit of name table (i.e., 2x2 tiles) may use only one four-color palette.

Though blocking palettes into attribute groups saves hundreds of bytes of memory, it imposes noticeable limits on the shape and granularity of background objects. In general, background graphics are designed to align along attribute table boundaries, making the 2x2-tile square the architectural bedrock of Famicom games. Attentive designers were careful to craft landscapes conducive to square shapes, like the pipes and stairwells in *Super Mario Bros.* or the sharply-edged forests in *The Legend of Zelda*.[85]

Shapes defined by circular or diagonal edges risk so-called "attribute clashes," where the transition between borders of two attribute table entries creates abrupt shifts in color. *Snake Rattle N Roll*, for instance, uses an isometric perspective uncommon on the NES, making its faux three-dimensional terrain appear as if it is diagonal to the player's perspective. The off-axis background is graphically impressive but requires significant palette concessions, since none of the terrain aligns at right angles.

In figure 1.6, you can see obvious attribute clashes along terrain and object edges. The pyramid that abuts the vine-covered platform near the



1.6 A screenshot from *Snake Rattle N Roll* shows abrupt color changes across background attribute boundaries, especially among the pyramids and waterfalls. (Emulator: Macifom 0.16)

center of the screen, for instance, has a portion of its apex colored green to match the vines rather than the adjoining pyramids. Likewise, the tops and faces of waterfalls exhibit sharp transitions between shades of blue, while the gold mechanism near the lower right corner shifts abruptly to green near its base.

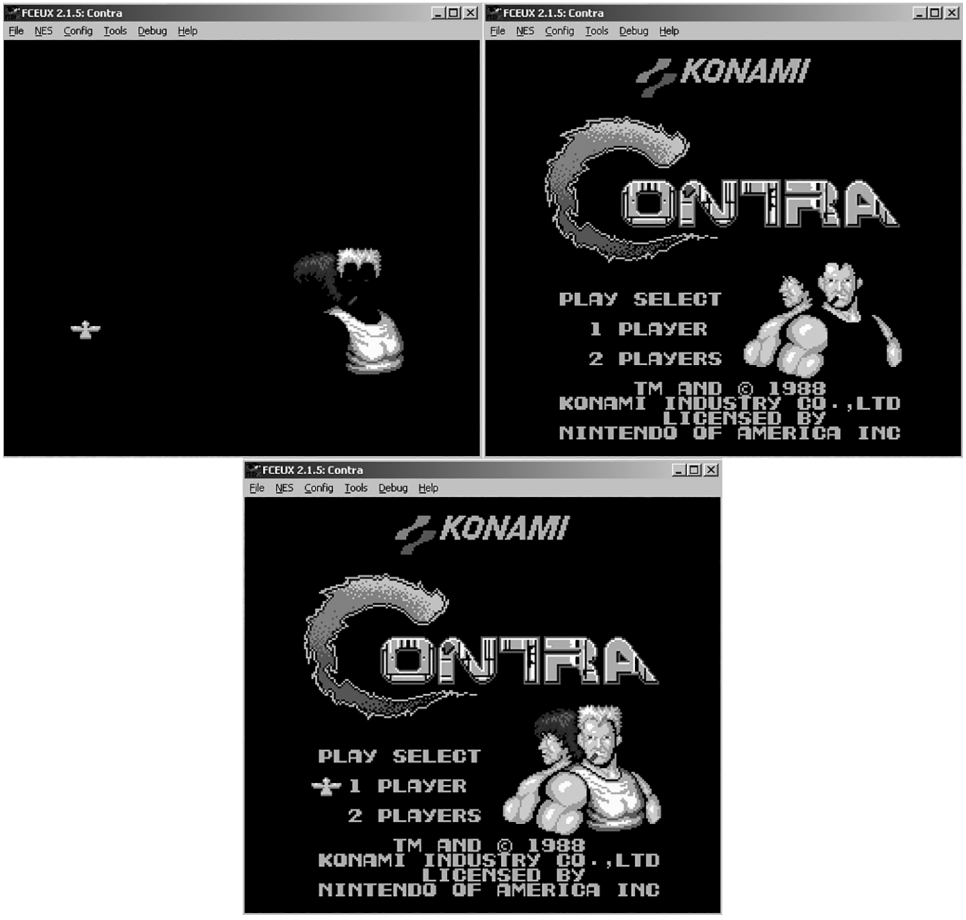
Avoiding attribute clashes required either careful planning near attribute boundaries or significant alterations to color and terrain structure. *Solstice* adopts the same isometric perspective as *Snake Rattle N Roll*, for instance, but limits its level palettes to variations of a single color. Doing so sacrifices the overall vibrancy of its levels, but avoids gaudy color mismatches and allows the character sprites to appear that much more vivid against the subdued backdrops. *Metroid* uses a standard side-scrolling view, but pays careful attention to palette choices, keeping radical color changes sequestered by area. Its Brinstar region has vibrant architectures of blue, green, and amber, but never in contiguous blocks. Palettes changes are instead marked by doors or tunnels, giving each area a distinctive visual style.

Since sprites float freely and tap separate palettes, they are often used in atypical ways to bolster the colors capabilities of background tiles. The most common use of this technique is seen in title screens and cutscenes, which require minimal object interactions or player control, freeing sprites to do color touch-up work. Since the title screen is the player's first impression of a game, it is regularly reserved for the game's most complex and detailed artwork.

In figure 1.7, *Contra*'s title screen is shown, via the FCEUX emulator, in three graphical "passes": the far left displays sprites only, the middle background tiles only, and the right the final composite. Without interleaving sprites for the hair, eyes, cigarette, and tank top, *Contra's* designers would not have been able to create such smooth transitions between colors. They would have either had to reduce the overall color detail of the two characters, square off the character designs, or else leave distracting attribute clashes bordering curves. As we will see in later chapters, Famicom programmers regularly used background and sprite tiles against type. What we often expect to be background is composed of sprites, and vice-versa.

Sprite Adjectives

Similar to how active background tiles are arranged in the PPU's name tables, active sprites have a reserved address space known as Object Attribute Memory (OAM). OAM has slots for sixty-four sprites—the maximum



1.7 The *Contra* title screen uses combined sprite (left) and background tiles (middle) to draw more detailed and colorful character portraits (right). (Emulator: FCEUX 2.1.5)

count the PPU can render onscreen simultaneously—occupying 256 bytes of memory. OAM's byte count might seem like overkill considering it stores no actual pattern data, but sprites require more computational overhead to display than background tiles. This is why OAM is called "Object *Attribute* Memory" rather than "Object Memory." The visible sprites are not housed there; rather, OAM stores a *display list* of qualities for each active sprite.

Each sprite's display list is four bytes wide. In sum, the list identifies which sprite from the pattern table will display, its orientation (flipped horizontally or vertically), its priority (above or below background tiles),

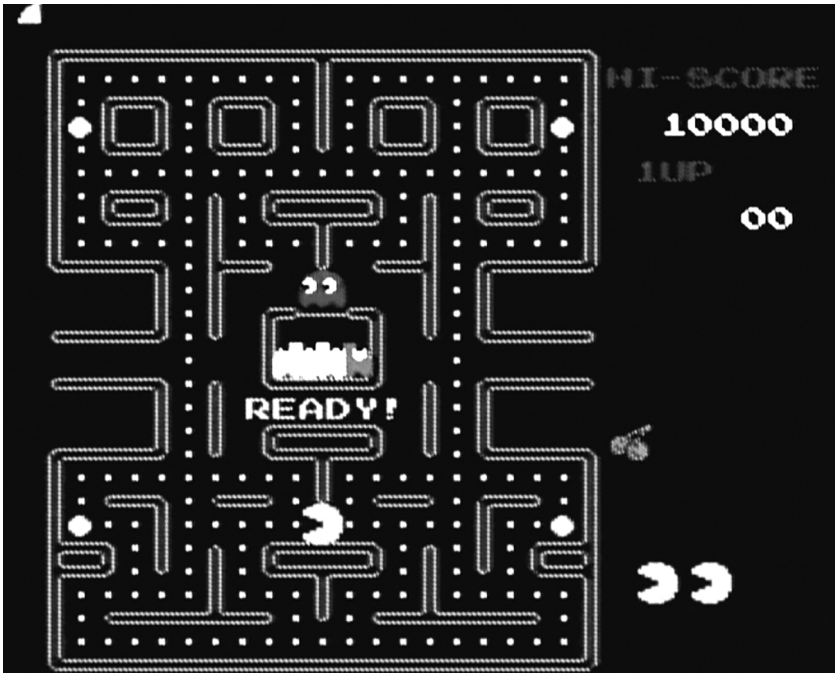
its screen coordinates, and which sprite palette it will select. Separating the sprites "nouns" (i.e., patterns) from their descriptive "adjectives" (i.e., OAM) is yet another means to mitigate hardware memory costs and eliminate unnecessary redundancy. Like the name tables, multiple OAM slots can reference the same sprite without the need to store multiple copies in pattern memory. This is especially useful in games where multiple objects of the same type display onscreen simultaneously. Furthermore, since there are far fewer sprites than name table tiles, each can access its own palette entry, giving sprites a color advantage over the attribute-bound background tiles.

In OAM, all sixty-four slots are filled at all times, whether with the sprites the programmer intends or with temporary junk data. In many games, especially early Famicom titles, most sprites were not needed onscreen at all times, yet those sprites had to have valid coordinates (i.e., OAM cannot contain a null value). In a pinch, sprites could be toggled on and off, but doing so was an all or nothing affair—sprites were either all visible or all invisible. To strike a compromise between extremes, programmers commonly used "safe areas" around the display's borders to tuck away unneeded objects.

The PPU addresses the screen like a Cartesian graph flipped along its x-axis, with the [0,0] coordinate at the upper left corner. Thus moving a sprite right and down increments its x- and y-coordinates, respectively; moving left and up decrements these coordinates. The key point is that there are no negative coordinates and therefore no means to push a tile past the leftmost and topmost screen borders. However, the Famicom/NES PPU, regardless of region, renders a 256x240 pixel display, while its sprite coordinate positions may contain any valid 8-bit value (i.e., $00-$FF or 0–255). Consequently, pushing a sprite's y-coordinate past $EF (239) hides that tile beyond the PPU's rendering scope.[86]

Misunderstanding OAM's sprite positioning and the PPU's screen perimeter can lead to peculiar graphical artifacts. A common initialization step in Famicom games involved "zeroing out" OAM by manually writing zeroes to all memory locations. Doing so assigned all visible sprites to the first pattern table entry ($00) and shifted their coordinate position to the upper left screen corner. With assurance that there was no unknown garbage data sitting in OAM, the programmer could then assign and position sprites as needed. Unfortunately, many programmers thought this also solved the problem of hiding unused sprites, since many televisions naturally clipped part of the visible frame with *overscan*.

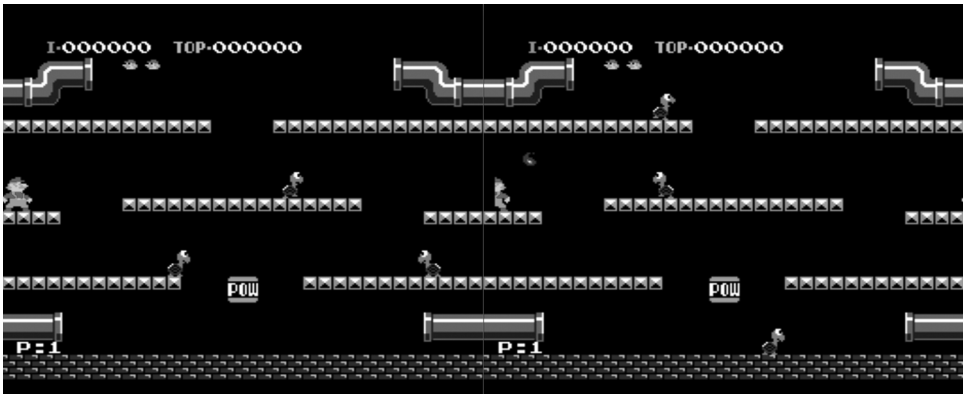
Zeroing OAM is not an airtight sprite hiding technique, since overscan varies from screen to screen, especially in emulation.[87] And since the



1.8 In Tengen's *Pac-Man*, the unused sprite stack is visible in the upper left corner of the screen. Note that in some emulators that simulate CRT overscan, this portion of the screen may be clipped. (NES-001 CRT capture)

PPU cannot render to negative coordinates, the sprite's upper left pixel nestles in the corner, while its remaining pixels "dangle" into positive coordinate space. Tengen's NES port of *Pac-Man*, for instance, has a conspicuous yellow sprite fragment floating above the upper left side of the maze (figure 1.8). Though the errant graphic appears to be a single tile, it is actually a stack of all OAM sprites not currently in play. To hide the stack properly, *Pac-Man*'s programmers should have pushed the vivisected sprites beyond the lower boundary of the screen rather than relying on overscan to mask their presence.

The lack of non-negative screen coordinates also creates problems for characters that must exit the leftmost screen edge smoothly. Especially in early single-screen arcade games where characters could wrap around edges, characters might find half of their body's sprites clipped at the left edge while emerging from the right. Figure 1.9 shows object coordinate limitations at work in the Famicom port of *Mario Bros.* Mario may approach the left border until his leftmost pixel touches the threshold,



1.9 Without proper masking, single-screen games with wraparound spaces exhibit clipping errors when sprite-based characters approach the left edge of the screen. (Emulator: Nestopia v1.4.1)

but advancing any further chops his body in half. Due to the limits of 8-bit math, subtracting from 0 causes the sprite to wrap to the rightmost edge of the screen, causing Mario's left half to emerge on the opposite side of the screen. Mario's right half will continue to push toward the left border until its leftmost pixel reaches the threshold, at which point he will fully transition to the right side.

To compensate for such graphical anomalies, the PPU provides a special mask that will hide sprites, background tiles, or both in the screen's leftmost 8-pixel-wide column. When enabled, the mask creates a convincing curtain for sprites to slide behind as they cross the leftmost screen border. As the Famicom aged and single-screen arcade fare gave way to side-scrolling platformers and role-playing games, developers increasingly used this feature to mask scrolling updates and attribute clashes (chapter 6).

The key quality of the Famicom's graphics processor architecture is that no single memory location in VRAM contains the colorful mosaic of tiles we see onscreen. The bit patterns are stored in one location, the palettes in another, the tile positions in another, and palette assignments in yet another. Only at render time do these disparate bits of memory cohere into what we see as graphics. Sequestering data in this way might seem convoluted from a contemporary programmer's perspective, but in an era when memory was measured in hundreds, rather than millions, of bytes, split architectures made good economic sense. Reducing graphical redundancy reduced hardware costs, even if it meant extra legwork for the programmer and the processor. In the end, it doesn't matter if what the

player perceives matches what the PPU perceives, so long as the tiles fall into place before the cathode ray sweeps the screen.

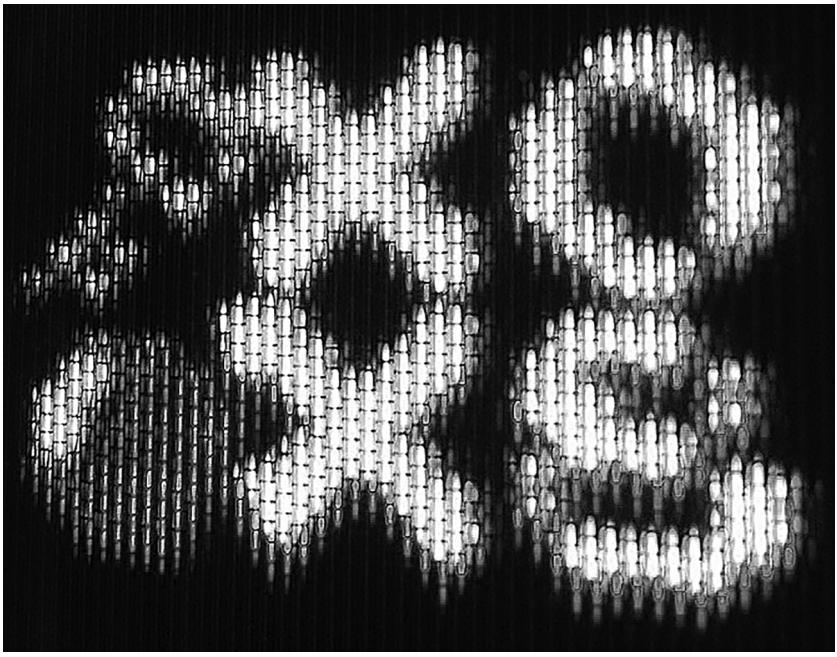
Low-Definition Television

Until the recent adoption of high-definition television (HDTV) standards, the dominant display technology of arcade and console videogames was the cathode ray tube (CRT). The CRT's namesake is similar to a vacuum tube—during manufacture, its interior oxygen is burnt away, creating a highly pressurized interior seal.[88] At one end of the tube is a barium-coated cathode that, when heated, emits negatively-charged electrons. Positively charged anodes attract, accelerate, and focus the electrons into a narrow beam. Together these components are known as an electron gun.

The portion of the tube opposite the cathode, which widens considerably to a large curved surface—the television screen—is coated with a luminescent phosphor material. When the electron gun fires at this coating, the phosphor's electrons momentarily become unstable then settle, emitting a photon at a wavelength that human eyes perceive as light.

A beam shot directly down the cathode tube generates a single illuminated point. In order to draw a complete image, the electron beam must be diverted in a predictable, consistent pattern. Along the exterior neck of the tube, magnetic coils deflect the beam's path using alternating voltages. Two electrical signals, each synced to a separate oscillator, play complementary roles in this process: one guides the beam horizontally, the other vertically. The electron gun draws the television image line by line, sweeping the beam left to right and top to bottom, much like a hand composes a letter, repositioning each time it reaches the right margin. Once the "page" is full, the gun resets to the upper-left corner to begin anew. This process repeats dozens of times per second, creating a persistent *raster* of scanlines that compose a moving television image.

Color television requires a more elaborate mechanism. The electron gun multiplies by three, each assigned to a single color: red, green, and blue. The three electron beams are deflected at slightly different angles by a small perforated plate called a *shadow mask*. Each beam then strikes a grouped array of phosphors, known as triads, that emits their assigned color. Television uses triads as additive primaries, meaning that the three colored light sources are blended to produce the desired hue.[89] When all three guns fire at once, for instance, the result is white, while a single gun may light green alone.



1.10 A macro-zoom photo of *The Legend of Zelda*'s key and bomb inventory showing the individual triads of the CRT display. (Source: Author's photo, NES-101)

You can clearly see color triads by ignoring your parents' advice and sitting directly in front of a CRT screen. In figure 1.10, the triads are visible within dozens of miniature columns of rectangles, like so many vertical bricks stacked one atop the other.

The columnar configuration is a result of a PIL, or precision-in-line, tube.[9] In a PIL tube, the electron guns are mounted in a straight line, rather than in a triangle, and the shadow mask is perforated with vertical slots rather than circles.

Triads do not produce the clean, squared pixels we associate with 8-bit videogames. Even at the focal length depicted above, it is impossible to discern individual pixels. Paradoxically, the CRT's flaws contribute to a richer final image. Carefully choosing pixel color and placement creates perceived hues and shades that the PPU does not actually output. The illuminated bleed of CRT triads, caused by the constant excitement and decay of the phosphor coating, both softens the edges of pixels and creates pleasing blends between color borders. The "X"s and numerals above, sourced from *The Legend of Zelda*'s inventory icons, appear to have a golden tint around the upper edges and blue shadows beneath. On an LCD

screen or in an emulator, the same tiles are simply white pixels on a black background.

Pixels are the fundamental graphical unit of the PPU, but not of the display. CRT televisions have their own subatomic particles—phosphor triads—that do not adhere to the strict gridded geometries associated with pixel graphics. One or more triads can constitute a single pixel based on the size of display, so CRTs "natively" scale bitmapped graphics. High-definition LCDs, DLPs, and plasmas, on the other hand, have a fixed resolution. As a result, their individual pixels are so small that if we mapped the 256x240-pixel matrix of the Famicom PPU to a modern HDTV, the resulting screen would be miniscule. Therefore, individual pixels must be scaled according to the TV's resolution, resulting in a blurry, distorted image.

Firing Blanks

Before television was a mature technology, the number of times a full vertical scan, or *field*, took place per second was not standardized. When single-digit field rates were the norm, television had noticeable flicker, as the electron gun simply could not generate enough fields per second for persistence of vision to kick in. Field rates progressively improved, but early on a rudimentary form of compression called *interlacing* helped alleviate noticeable flicker.

Interlacing is analogous to "shoelacing"—in the same way that one might alternate laces along the tongue of a shoe in order to secure it tightly, an interlaced image is drawn with alternating scanlines. Though only half the image needs to be drawn at a time, it then takes two full fields of interlaced scanlines to compose a single *frame* of video. Again, due to phosphor's luminous residue, human eyes do not perceive two alternating sets of scanline window shades, but a single composite image. And since the CRT's phosphor luminance bleeds around the edges, it smooths the gaps between scanlines, effectively masking the raster.

Once early television experimenters settled on the minimum threshold of fields necessary to overcome flicker,[91] standard field rates began to emerge around technological, economic, and political interests. The three established standards were NTSC, PAL, and SECAM. The first, a monochrome system named for the National Television System Committee, was adopted in the United States in 1941, then later revised to a color standard in 1954 (called NTSC-2). However, NTSC had color instabilities (necessitating manual hue controls) that European interests sought to improve. Europe adopted the Phase Alternating Line color encoding

standard in 1963, eliminating NTSC's hue inefficiencies—with the added benefit of forcing non-European sets out of the domestic television market. The French forged their own path in the early 1960s (also correcting NTSC's color limitations), introducing SECAM, or *Séquentiel couleur à mémoire.*

Though these standards branch into myriad subdivisions that describe precise technical differences, the details relevant to the Famicom involve geographic divisions and their associated field rates. Japan adopted the NTSC standard, which eased the Famicom's later export to the United States. The bulk of Europe and Australia adopted the PAL standard, with pockets of SECAM sprinkled throughout (obviously in France, but also in other parts of Eastern Europe). The discrepancies required a hardware revision for Nintendo's European console launch, so the PAL-compatible NES debuted after the U.S. version with a number of important differences that affected how video displayed.[92]

One modification handled the television standards' differing vertical scan rates. An NTSC television draws its frame at approximately 60Hz, or sixty individual top-to-bottom electron beam trips per second. The PAL standard is slightly lower—approximately 50Hz—but the discrepancy has a huge impact on graphics rendering and program timing. Much of the important work of displaying a videogame is done while the electron beam is "at rest," either resetting from drawing a horizontal scanline or resetting to the top after it reaches the bottom. By the time we see the results onscreen, most of the preparation work has been done. Graphics updates are ideally queued and ready prior to each sweep of the electron beam. Those horizontal and vertical "reset periods," or *blanks*, are key to the Famicom's operation.

Image synchronization is a precision art. The electron beam not only deflects at speeds the human eye cannot perceive, but the voltages controlling horizontal and vertical positioning operate at independent frequencies. Those of us alive in the CRT era may remember the delicate choreography required to keep a television's synchronization consistent. "Rolling" images are as common to the CRT generation as compression artifacting is to the flatscreen generation. When the vertical syncing frequency falls out of lockstep, the TV image wraps around the top and bottom borders of the screen, separated by a moving black bar—the vertical blank made visible. Similarly, when horizontal sync goes awry, the image "tears" along jagged diagonals.

Though the vertical and horizontal blank—or VBLANK and HBLANK—describe distances, their importance to programmers is temporal. In other words, it takes time for the gun to travel, and that time is necessary to

update or move the proper graphical elements into place before the screen starts to draw again. But it is a slender margin that demands careful preparation. Updating the PPU while the scanline is actively being drawn is a Famicom programming no-no, resulting in noticeable graphical glitches. Any violations to this rule must be done deliberately and with precise timing. The PAL NES's lower refresh rate grants programmers additional affordances: since the gun resets ten fewer times per second, it can make its vertical ascent at a more leisurely pace. The PAL VBLANK is a full fifty scanlines longer than NTSC, granting more time for code to execute.[93]

A PAL television also has more graphical real estate than an NTSC set. This is not a discrepancy of the PPU—on either PAL or NTSC systems, the processor outputs a full 240 scanlines of graphics. The difference stems from overscan, a variation in picture visibility common to CRT displays. Overscan describes the area around the edge of the video frame that is not visible based on variations in the individual monitor, often due to a television's physical bezel or the curve of the cathode ray tube itself. Due to the vacuum inside the CRT, early televisions were tiny, curved, and typically circular, all measures meant to keep the tube from imploding. As manufacturing tolerances improved, shapes resolved into the more familiar rectangular 4:3 aspect ratio, curves decreased, and bezels receded. But this refinement process took decades. Television and videogame producers learned to keep important content away from the edges of the screen, lest it be invisible to some fraction of viewers. On NTSC sets, up to eight pixels of the upper and lower borders of the PPU's output are lost to overscan, reducing its visible scanline count to 224. PAL sets lose a single upper scanline and two pixels on the left and right.

Regional refresh rate differences create an ironic kinship between consoles that share few external similarities. The Family Computer and the U.S. Nintendo Entertainment System do not appear to be blood relatives, but internally they have identical CPUs. Famicom systems play fine on U.S. televisions with the proper step-down transformer and broadcast channel for playback (i.e., channel 2 in Japan is not equivalent to channel 2 in the U.S.). In contrast, PAL NES consoles, which are externally identical to their U.S. brethren, contain a CPU variant known as the Ricoh 2A07. To account for the 10Hz discrepancy in refresh rates, the 2A07 has a slower clock speed (1.66 MHz) than the 2A03 (1.79 MHz). In practical terms, PAL game cartridges played on an NTSC console will both play back faster and have their sound pitched up in frequency (and vice-versa). In some cases, the speed difference is bearable; in others, it renders a game unplayable.

Hobbyist game designers who aim to dabble in Famicom programming are often surprised that its architecture does not support the high-level languages common to modern platforms. The Famicom has no operating system or firmware to boot into; it relies solely on the instructions stored in ROM. Code must be written in assembly language, whose terse syntax is quite different than modern compiled programming languages like C++ or Java. Those languages bear some resemblance to everyday grammar, and even non-programmers can locate recognizable words and parse basic instructions. 6502 assembly, which is a single abstraction layer above machine code, is cryptic in comparison. Each line of code is composed of a three-letter mnemonic, representing an instruction, followed by an associated numeric value or address.

Though I keep the code examples in the book to a minimum, it is worthwhile to understand the basic structure of 6502 assembly, along with a number of common addresses the Famicom uses to get its work done. Unfortunately, there is no "Hello World" program for the Famicom, since it has no straightforward command to output text to screen, much less a built-in character set. Letters are tiles like any other graphic. Displaying text requires creating custom bitmap letterforms, waiting for the Famicom to initialize, clearing the contents of RAM, routing the background graphics to the screen, setting the palettes, and so on. I do not have the space to devote to a full code example, but a small excerpt can help us understand the general look and feel of Famicom programming. Displaying a background color is one of the simpler tasks one can program, so we will walk through a few steps of that process. The following code sets the PPU background color to green:

```
LDA #$3F
STA $2006
LDA #00
STA $2006
LDA #$2A
STA $2007
```

Again, the three-letter mnemonics on the left are instructions for the CPU, and the numbers to their right either designate addresses or specific numeric values. (If the dollar signs seem strange, rest assured that they are a common programmer prefix for hexadecimal values, not the cost of the instructions.) Hexadecimal is a base 16 numeral system. In our common base 10 (or decimal) system, we count from 0 to 9, then move to the tens place, repeat until we require a hundreds place, and so on. Binary,

which counts with only 1s and 0s, is a base 2 system. Base 16, as we might expect, counts each digit place up to sixteen. Since we do not have single-digit symbols to represent numbers above 9, the six digits places between 10 and 15 use letters A through F.[94] The $3F value above works out to $3x16^1 + 15x16^0$, or 63, in base 10 notation.

All data in the Famicom is moved via *registers*, or hardware memory locations. Any in-game math, physics, AI, graphics, and sound synthesis is controlled by moving data to and from a handful of registers, including three of the CPU's special registers: the accumulator, x, and y. The code snippet above uses the accumulator—the primary register used for adding, subtracting, and comparing numbers—to pass a number of values to the PPU. $2006 is one of the 16-bit hexadecimal addresses of the Famicom's PPU I/O control registers. Changing the color of the background is a graphical job, so the CPU must send data to the PPU via designated addresses so it can act on that data appropriately.

The first four statements do the following: LDA stands for "**L**oa**D** **A**ccumulator" with the value that follows. In this case, it is the hexadecimal value $3F (the "#" symbol tells the assembler to load the numeric value, rather than the value stored at that address). $3F is only the first half of a 16-bit address mapped in PPU memory, so we must load and **ST**ore the **A**ccumulator (STA) twice in the I/O control register to designate the proper destination. Via four commands, we have readied the PPU to receive data at address $3F00, the location in memory where background palette information is stored. With our data destination set, we can then pass along information via the PPU data port, located at register $2007. To do so, we load the accumulator with a color value ($2A, or green) then store that value at $2007. In sum, the CPU tells the PPU, "I have the value for green and I would like you to store it in your palette at the requested address."

As this short example illustrates, trivial procedures in today's programming languages require significant legwork in assembly. There are few shortcuts, even for simple tasks. Today's programmers might think coding in such a manner is ludicrous. In most cases, they are right. But assembly language's close relationship to its processing infrastructure grants programmers unprecedented hardware control, since there is little abstraction between code and silicon. And once one understands the locations and behaviors of a handful of registers, extraordinary things can happen onscreen.

Notes

Chapter 0

1. Tanner, "Adventure of Link - Retranslation."
2. To cite a recent example, *ZeldaDungeon.net* ran a short piece titled, "I Am…Not Error?" The comments are illuminating.
3. Mandelin, "A Look at the Metroid Series' Varia Suit."
4. For example, Kazuhisa Hashimoto, who worked for Konami in the Famicom era, said in an interview, "It was normal for a NES game to be designed in 4–6 months by a team of 4 people. This practice didn't really change until the Super Nintendo era." See Tanner, "Konami: The Nintendo Era."
5. Derrida, "Letter to a Japanese Friend," 275.
6. Sandifer, "Am Error."
7. When referring to the platform as a whole, I have opted to use Famicom as the shorthand for the Family Computer, the NES, and their multiple revisions. When I refer to the NES specifically, I mean to reference its regional model exclusively.
8. Donovan, *Replay: The History of Video Games*.
9. Sheff, *Game Over*.
10. Chaplin and Ruby, *Smartbomb*.
11. Provenzo, *Video Kids: Making Sense of Nintendo*.
12. Ryan, *Super Mario: How Nintendo Conquered America*.
13. Montfort, "Continuous Paper: The Early Materiality and Workings of Electronic Literature."
14. Kirschenbaum, *Mechanisms*.
15. See Chaplin and Ruby, *Smartbomb*; Dillon, *The Golden Age of Video Games*; Donovan, *Replay*; Gamespite, *GameSpite Quarterly 5*; Goldberg, *All Your Base Are Belong To Us*; Herman, *Phoenix*; Kent, *The Ultimate History of Videogames*; Kline et al., *Digital Play*; Kohler, *Power-Up*; Ryan, *Super Mario*; Sellers, *Arcade Fever*; and Sheff, *Game Over*.