

Tool-Assisted: Console Emulation and Platform Plasticity

Since the early 1990s, a special subculture of play, called *speedruns*, has pushed the limits of videogame skill, performance, and technical mastery. The aim of the speedrun is to play a game as quickly as possible, by any means possible, short of cheating, passwords, or other ‘non-diegetic’ exploits. Games that might take an average player tens of hours are reduced to an hour or less, often at the highest possible difficulty. Notoriously difficult Nintendo Entertainment System games like *Contra* or *Ninja Gaiden* are completed in mere minutes.

In 2003, speedrunner ‘Morimoto’ uploaded a virtuosic eleven-minute run of *Super Mario Bros. 3*,¹ executing a flawless, playful demonstration of Mario’s entire range of skills. And he did so with style—during portions of the game where the scroll speed is fixed, for instance, he danced around the screen effortlessly, reducing his onscreen foes to harmless playthings. Of course, the run was too perfect to be true—Morimoto used a special NES emulator to seamlessly splice small segments of gameplay into a perfect performance. The discussion surrounding the ‘falsity’ of the run polarized the speedrun community along two lines: the purists on one hand and the emerging crop of players making *tool-assisted speedruns* (or TAS) on the other. Though the term, like speedrun, originated among *DOOM*’s high-level players,² it quickly flourished in the NES emulation community, describing any run that leveraged the features of an emulator to allow players to finish a game faster. One of the first websites to host such performances, TASVideos.org, describes the practice succinctly: ‘Tool-assisted game movies. When human skills are just not enough.’³

Though many traditional speedruns are played in emulators, they do not require them. Speedruns are akin to rigorous athletic or musical performances, structured around exhaustive

¹ “TAS: NES Super Mario Bros 3 (JPN) in 11:03.95 by Morimoto,” YouTube, http://www.youtube.com/watch?v=R4_Zski_B0I

² Lowood, Henry, “High-performance play: The making of machinima,” *Journal of Media Practice* Vol. 7 no. 1 (2006): 25-42.

³ “TASVideos,” tasvideos.org.

practice, minor tweaks to enhance performance, and subtle aesthetic flourishes. Improving the world-record *Super Mario Bros.* run is not unlike Olympic competition in the 100m dash—it's a game of tenths of seconds. (According to speeddemosarchive.com, the online repository for traditional speedruns, the fastest *Super Mario Bros.* run has improved by *one second* since April 10, 2007.)⁴ The TAS, however, drifts more toward hacking, programming, and even bibliographic research. Games are scrutinized at the code and platform level to reveal any exploits, glitches, or programmer errata that might improve the assembled run. Performance times are no longer measured in human scale but in microprocessor scale, down to the individual rendering frame. At this level, there is an unparalleled intimacy of play (if we can still use that word) between human and machine. Together, tool and flesh choreograph an elegant mastery of code.

Platform studies has sparked newfound interest (and arguably rigor) in the study of videogame consoles. The approach, coined and practiced by Nick Montfort and Ian Bogost,⁵ considers how hardware-level decisions and constraints shape the expressive content these machines produce. Cathode ray patterns, frame buffers, sprite RAM, clock cycles, square and triangle waves—all subtly (and often invisibly) shape the colors, shapes, and sounds on screen. The Nintendo Entertainment System, for instance, has a hardware-constrained limit of eight sprites per scanline. If more than eight picture elements align along a single horizontal axis, the microprocessor starts to prioritize and cycle between them. Poor sprite planning can result in flickering graphics, but the limitation can also have creative uses. *The Legend of Zelda* uses a line of sprites near the top and bottom of dungeon screens to allow Link's sprites to disappear convincingly through doorways.

The irony of platform studies is that what feels fresh in the academic world is routine to the serious hobbyist or collector. The Nintendo homebrew community has amassed a bibliographer's utopia of catalogued circuit boards, textual variations across code and cartridge, disassembled and annotated sources, and the tools that make these studies possible. Groups like these are serious about the unique material characteristics of their favored consoles and

⁴ "Speed Demos Archive - Super Mario Bros.," Speed Demos Archive, <http://speeddemosarchive.com/Mario1.html#norm>

⁵ Cf., Nick Montfort and Ian Bogost, *Racing the Beam: The Atari Video Computer System* (Cambridge: MIT Press, 2009).

frequently gain technical insight into decades-old machines that surpasses that of even its original developers. As a result, there is a significant cross-fertilization between these interested hobbyists and the speedrun community. New hardware breakthroughs result in better emulation and more capable emulator tools. Exploits discovered during rigorous play can, in turn, reveal quirks in the hardware that need to be properly accounted for in emulation.

Console emulation mimics a target platform on another, typically more powerful, platform, ideally permitting users to play game software with the closest approximation to the original experience as possible. Accuracy is a key constraint and never perfect. Emulation is not solely a matter of replicating the target console's CPU, but also any additional co-processors, input/output devices, lower level instruction sets, and so on. The NES had dedicated Picture and Audio Processing Units (PPU and APU) in addition to its 6502-based CPU. Each of these components are necessary for proper emulation. For the needs of most players, low-overhead emulators that play most popular games with reasonable accuracy are acceptable. For the TAS community, there are more stringent requirements that most players would never notice, like cycle-accurate CPU timing and frame-level control. But higher accuracy comes with a concomitant increase in processor demands, especially if the emulation is purely software-based. Emulation is a constant balance between usable tools and allegiance to the source hardware.

This balance was apparent from the first published work on emulation. In 'Emulation of Large Systems' (1965), S. G. Tucker drew distinctions between a number of possible solutions to the so-called 'conversion problem,' whereby programming work on an older machine needed to be moved to a newer machine.⁶ Simulation, for instance, relied on a bit-for-bit 'image of the entire state of the machine being simulated' and thus suffered greatly in performance. Reprogramming (or porting, as we might call it now) was another option, but Tucker called it a waste of resources that might otherwise be directed toward new problems. Emulation, on the other hand, coupled software with assistive hardware to shoulder more of the processing burden and thus struck a compromise between speed and accuracy.

By 1969, Robert Rosin divested emulation's definition of its hardware dependence and instead defined it as 'a complete set of microprograms which, when embedded in a control

⁶ Tucker, S.G., "Emulation of Large Systems," *Communications of the ACM* Vol. 8 no. 12 (1965): 753.

store, define a machine.⁷ Thus the more modern distinction of a ‘virtual machine’ within another ‘host machine’ was born. This idea had important implications, as it abstracted the original platform from its roots in circuits and registers to a manipulable configuration of code. Emulation took on all the connotations of the virtual: idealized, practical, imitative, ‘close enough.’ A programmer could potentially emulate a machine that did not yet exist (as is often the case in game development, when new consoles are being designed) or build in some ‘extra assistance in the form of better debugging aids, a larger virtual machine, and access to the host’s peripherals.’⁸ Early on in the history of computing, there was already a sense in which the emulator was not just replicating, but *augmenting*. This concept persists into the modern era of console emulation.

Until the 1990s, console emulation was not viable on consumer-grade personal computers. They simply did not have the necessary processing power to emulate the consoles of their day. Arcade machines and home consoles benefit from dedicated hardware. Every ounce of RAM, every byte of storage, every clock cycle serves the sole purpose of getting the game on the screen. PCs, in contrast, are multi-modal Renaissance machines meant to run games alongside spreadsheets, documents, email, and pornography. This discrepancy in focus often means that modern PCs are a generation or two behind in emulation. In 1997, when indie outfit Bloodlust Software released the popular Nesticle emulator, the Nintendo Entertainment System was already several years out of its prime. But that also meant that the console and its chip architecture, based on the MOS Technology 6502⁹ (itself already over 20 years old), hit the sweet spot of manageable complexity, popular appeal, and current PC storage/network limitations. Nesticle (as you might guess from the name) was not a corporate product—Bloodlust Software was a pair of bedroom coders—so it was distributed for free online. Likewise, dumped ROM images (aka games) were small enough to transfer via modem or floppy and store locally in a handful of kilobytes. This was not feasible with then-current platforms like the Sony PlayStation; its 3D capabilities were too computationally costly for mainstream PCs

⁷ Rosin, Robert, “Contemporary concepts of Microprogramming and Emulation,” *Computing Surveys* Vol. 1 no. 4 (1969): 197.

⁸ T. A. Marsland and J. C. Demco, “A Case Study of Computer Emulation,” *Canadian Journal of Operational Research and Information Processing* Vol. 16 no. 2 (1978): 113.

⁹ “MOS Technology 6502,” Wikipedia, <http://en.wikipedia.org/wiki/6502>

and disc-based media was not yet economical to duplicate or store. Thus, the NES stood at a unique turning point in the processing capabilities of mass-market PCs and the emerging distribution channels of the World Wide Web. The NES was popular enough to attract enthusiast programming interest; its limited CPU, video, and audio processors were manageable for software emulation; and its compact programs were ideal for storage and transmission. The rapid success of Nesticle sparked the proliferation of console emulators in general and allowed the NES platform to ‘live on’ beyond its hardware life cycle.

Though Nesticle was not the first NES emulator, it introduced a number of now *de facto* console augmentations. Within two months of its April release, Nesticle could take screenshots mid-game, pause and resume progress at any point using save states, edit in-game palettes and graphics, play games online, save audio output, and record and playback gameplay movies.¹⁰ Suddenly, game players had the ability not only to play NES games, but to edit and reconfigure them. ROM hacks, as they are called, introduced strange cross-fertilizations of game worlds and politically-incorrect graphic alterations: Link’s sprite appeared in *Super Mario Bros.*, *River City Ransom* was made over as *Pussy City Pimps*, and Mario appeared in a wheelchair, Nazi attire, and KKK hoods. It became difficult to find a *Super Mario Bros.* ROM that *hadn’t* been altered in some way. ROM alteration combined with simple distribution of files online also led to fan translations of releases that never made it stateside, like *Final Fantasy II* and *III*.

The current state of NES emulation adds near-IDE levels of code manipulation. FCEUX, TASVideos.org’s recommended emulator for speedruns,¹¹ features a built-in hex editor, background graphics viewer, inline assembler, debugger, PPU viewer, and conditional breakpoints.¹² The software not only plays games, but strips them bare to the code. You can watch the game engine access and write registers while you play. It’s like going to a theater and watching a film running from Final Cut Pro on a laptop in front of you—and you can tweak the speed, angles, and edits to your liking.

The editing metaphor extends to the TAS, as the outcome of the performance is a finished movie. But movie is a misnomer for both the process and result. The finest tool-

¹⁰ “Official Bloodlust Software NESTicle Page,” <http://bloodlust.zophar.net/NESticle/nes.html>

¹¹ “Emulator Resources,” TASVideos, <http://tasvideos.org/EmulatorResources.html>

¹² “FCEUX,” Zophar’s Domain, <http://www.zophar.net/nes/fceux.html>

assisted runs are not spliced sections of normal gameplay; they are *meticulously*-assembled frames between save states, closer to animation than filmmaking. Players usually drop playback speed significantly and cycle through a small segment of gameplay, altering variables until the run is perfected. The resulting ‘movie’ is a specially formatted plain ASCII text file, saved with the extension .fm2.¹³ A completed movie reads as a frame-by-frame list of inputs in sequential order. It’s an editable input recipe for perfect play. Conceivably, with superhuman skill, a player could replicate this list on real hardware. In practice it’s impossible, as speedruns often rely on mutually exclusive key presses (e.g. left and right simultaneously) in a single frame, a feat not only humanly but mechanically implausible. These strange inputs, in turn, trigger conditions that a programmer never thought possible. Holding up and down simultaneously while riding elevators in *The Adventures of Link* allows the player to warp vertically through dungeon levels.¹⁴

The resultant videos are really archives of a performance taking place between source code and input code. One might reasonably assume that the player’s role will eventually disappear, that a computer might learn to play the emulator without human assistance. Even experienced video game players are left bewildered by a number of the most extreme tool-assisted runs. The content of a glitched performance borders on illegibility, bizarre visual references to what was once a playable game.¹⁵

So what drives this type of play? Can we still label it as such? And what do the limits of play teach us about the limits of platforms? James Newman provides a number of answers to the first question in his discussion of ‘superplay.’¹⁶ First, there are goals shared by non-digital games and sports such as high scoring, self-improvement, community-building, and public recognition. Another layer at work, perhaps unique to digital media, is the understanding and mastery of an underlying system, ‘playing’ and ‘gaming’ in the sense of bending or breaking rules to gain an advantage. Newman explains that ‘this system of rules and boundaries is not fixed but rather is permeable and in a state of flux as it is interrogated, operated on and played

¹³ “FM2,” FCEUX, <http://fceux.com/web/FM2.html>

¹⁴ “Gigafrost’s NES Zelda 2: The Adventure of Link ‘glitched’ in 06:16.93,” TASVideos, <http://tasvideos.org/690S.html>

¹⁵ Cf., “GB Makai Toshi SaGa by knbnitkr in 01:47.17,” YouTube, <http://www.youtube.com/watch?v=BnjPM9ci-qc>

¹⁶ Newman, James, *Playing with Videogames* (London: Routledge, 2008), 123-148.

with. Moreover, the system may behave in an unpredictable manner unintended by the game designers due to imperfections in the code or unanticipated emergent contingencies.¹⁷ In this interplay, Newman concludes, ‘both player as performer and game system should be considered agents in the process of gameplay.’

Part of player agency is rule-making, and for a community devoted to bending machines to their algorithmic limits, there are a surprising number of sacrosanct rules for movie-making, from using sanctioned emulators to running checksums on ROM images to verify their pedigree. Part of this impulse is based in competitive fairness, standardizing rules to provide a level playing field. But there is also a strange threshold of coherence at work, an agreement about what a platform is and isn’t and just how far players can stretch that definition.

Among the NES homebrew community, there is the same shared respect for the platform’s constraints. Every few months, a community member will propose a revised hardware specification for the NES that eliminates per-scanline sprite limitations, palette constraints, and so on. The typical reply is: ‘If the NES PPU isn’t enough for you, just don’t use the NES, get over whatever problem you have with other platforms and use those.’¹⁸ At a certain point, an ‘improved’ NES starts to look like a Super Nintendo. In other words, the platform is defined as much by what it can’t do as what it can.

When we try to pin down exactly what an NES is, we see the strange liminality at the edges of a platform. One might argue that the NES is the grey and black toaster released in 1985, a specific arrangement of silicon, aluminum, and plastic. But this discounts the Famicom (as the NES is known in Japan), the Nintendo-licensed hardware revisions like the Sharp Twin Famicom or Mattel’s European NES, and the bizarre array of add-ons and accessories that appeared, from plastic robots, modems, inflatable motorcycles, and karaoke microphones. One might argue that these are mere exteriors to a consistent core, but that would discount Nintendo’s multiple board revisions, the PAL modifications necessary for play on European televisions, and memory-mapped controllers, or MMCs, the special hardware inside game cartridges that augmented the NES’s limitations and allowed Nintendo to stay competitive as more capable consoles arrived. If the NES is defined as something that plays NES games, then

¹⁷ Newman, 124.

¹⁸ Cf., “How do I create a enhanced VDP/PPU?,” NesDev, <http://nesdev.parodius.com/bbs/viewtopic.php?t=8160>

we have suddenly expanded the field beyond Nintendo's grasp, to unauthorized clones systems like the Russian Dendy, 'Nintendo on a Chip' boards built into controllers, software emulators, and weird hybrids like the Powerpak, which allows you to play downloaded ROMS on a 'real' NES. And what of the numerous hacks and modifications that allow the NES to function as an audio / video synthesizer or to host *other* emulators.

Clearly, a platform is a negotiation between static and dynamic forms—on the one hand, a finalized configuration of hardware and software must be settled upon as a basis for creating digital objects; on the other hand, that configuration is susceptible to transposition and metamorphosis through emulation, hacking, and modification. Tool-assisted speedruns would not be possible without tools that augment the original platform—variable speed play, frame counters, savestates, etc.—but they still rely upon the platform's distinct errors and quirks. If, for example, the sprite or CPU limitations of the NES were corrected, the underlying game code would not run properly and the 'identity' of the NES as a platform would be compromised. In turn, the quirks facilitate creative performances and new modes of play; the TAS elevates the glitch to a viable play mechanism, transforming the intended coherence of a game world into a strange dance of objects freed from temporal and spatial logic. A platform is thus more convention than console, more abstraction than assembled product. And how strange that a study so rooted in material objects should lose grasp of those objects around their borders.