Can an Equation Beat Pokémon? What Pi's Failure Teaches Us About Game Design, Glitch Hunting, and the Hidden Math Inside the Cartridge

Littleroot Town is a quiet place. Too quiet. For the past 16,000 hours, a Level 77 Sceptile has been trapped here. It has drowned in the same tiny pond ten thousand times. It has relentlessly opened and closed a backpack for days on end. It has paced back and forth against a wooden fence until the concept of movement itself lost all meaning.[1]

Who is piloting this miserable creature? The most famous number in mathematics: π.

Deep within a cloud server, a script calculates the infinite digits of π ($3.14159\ldots$) and translates each number into a controller input for a *Pokémon Sapphire* emulator.[2] Every single second, a new digit is fed into the machine. After tens of millions of inputs, one of the foundational constants of the universe still cannot navigate the simple geographic progression required to reach the first gym.[1]

The premise sounds like a surrealist joke, but unpacking why π fails so spectacularly reveals something brilliant. This absolute disaster of an experiment teaches us profound truths about algorithmic complexity, the illusion of digital randomness, and the structural vulnerabilities hardcoded into the silicon of retro video games.

The Mechanics of the Infinite Player

To understand the failure, we first have to look at the origins and setup of the experiment. The project was launched in October 2021 by a programmer known online as @armityle29. Running continuously on a Google Cloud server and broadcast twenty-four hours a day on the Twitch channel *WinningSequence*, the stream was born from a classic mathematical thought experiment: if π is truly infinite and non-repeating, can it eventually type out the complete works of Shakespeare? Or, in this modern digital equivalent, can it stumble upon the exact game-winning sequence of a 2002 Game Boy Advance title? Inspired by the chaotic, crowdsourced success of *Twitch Plays Pokémon*, the creator set out to test if a mathematical constant could succeed where millions of frantic internet commenters had before.

The translation mechanics are beautifully simple: the digits 0 through 9 are mapped directly to the buttons of a Game Boy Advance.[2] Zero is the 'A' button. One is the 'B' button. Two and three handle Start and Select, while four through seven manage the D-pad directions. Eight and nine simply do nothing, acting as idle frames. By generating one digit per second, π effectively becomes an automated, unthinking player.

To understand why π is failing so miserably at this task, it is first necessary to dissect the precise parameters of the experiment. The setup, inspired by crowdsourced phenomena like "Twitch Plays Pokémon" and running continuously on a Twitch channel titled *WinningSequence*, operates on a remarkably simple translation matrix. The digits 0 through 9 are mapped directly to the inputs of a standard Game Boy Advance controller.[1]

Digit	Game Boy Advance Input	Function in Pokémon
0	A Button	Confirm, interact, advance text, select attacks
1	B Button	Cancel, back out of menus, run from battle
2	Start	Open the main menu

3	Select	Use registered key items
4	D-Pad Up	Move character up, navigate menus up
5	D-Pad Down	Move character down, navigate menus down
6	D-Pad Left	Move character left, navigate menus left
7	D-Pad Right	Move character right, navigate menus right
8	No Input (or Repeat)	Idle frame
9	No Input (or Repeat)	Idle frame

The theoretical justification for this madness relies on a mathematical concept introduced by Émile Borel in 1909: the normal number.[3] In plain English, a normal number is the mathematical equivalent of flipping a perfectly fair coin infinitely.[4] Every possible sequence of digits appears eventually, and with equal frequency. In base 10, every single digit from 0 to 9 appears exactly 10% of the time. Every two-digit sequence appears 1% of the time, and so on into infinity.

If a number is truly normal, it contains absolutely every finite sequence of digits that could ever possibly exist. Converted into binary, a normal number would contain the complete works of William Shakespeare, the genetic sequence of every human being, and, crucially, the exact sequence of button presses required to flawlessly complete a Pokémon game.[1]

Mathematicians strongly suspect that π is a normal number.[4] Which means that a perfect, glitched-out, world-record speedrun of *Pokémon Sapphire* is buried somewhere deep within its infinite expansion. So why has the emulator spent years stuck in the starting town?

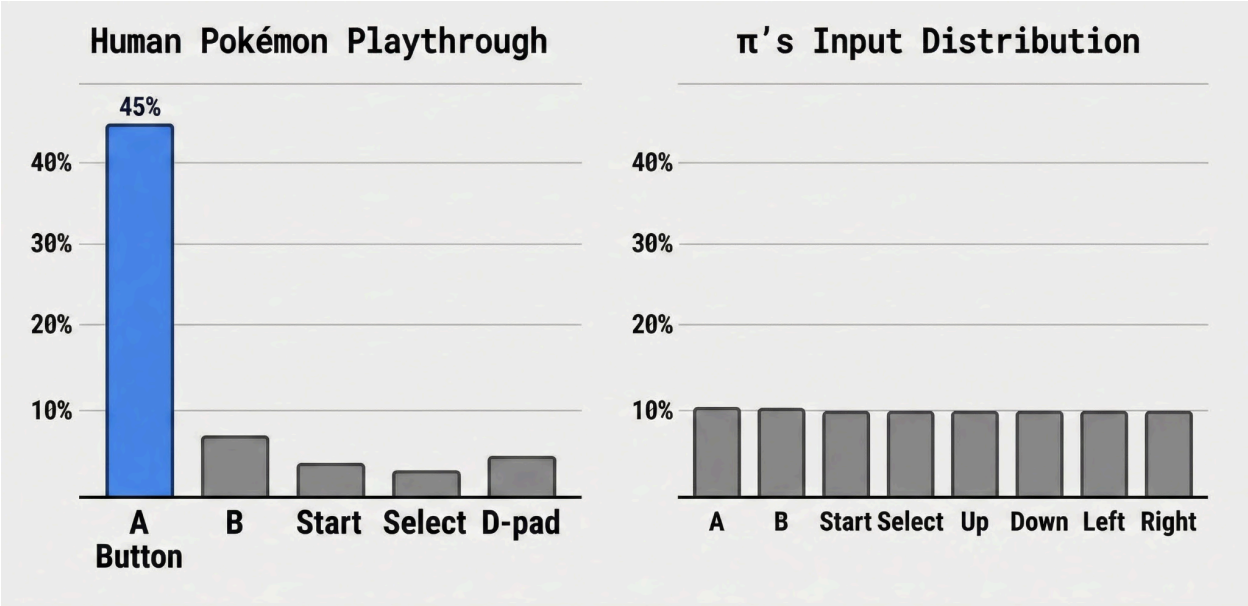
The Kolmogorov Bottleneck

The fatal flaw in utilizing π to play a video game is not a lack of infinity. It is a fundamental mismatch in structure. π being "infinite and non-repeating" sounds like the perfect brute-force

tool, but it lacks the single most important element required to beat a game: predictability.

We can quantify this using Kolmogorov complexity, a concept that measures how much computational effort is needed to describe an object.[5] A highly structured thing, like an image of a fractal, has very low Kolmogorov complexity because a tiny mathematical formula can draw the whole thing.[5] Pure, random noise has maximum Kolmogorov complexity because there are no shortcuts to describe it—you just have to read out the noise.[5]

A successful Pokémon playthrough has low Kolmogorov complexity. It is not a sequence of random noise; it is highly structured and deeply biased. Moving between towns requires holding the D-pad in a single direction for extended periods. More importantly, it requires the 'A' button.



The fatal mismatch. π treats every button equally, but video games heavily bias the “Confirm” button.

In a standard Pokémon game, pressing 'A' accounts for roughly 40% to 50% of all required inputs. It is the digital equivalent of breathing. It clears endless text boxes of NPC dialogue, confirms attack selections, navigates PC storage menus, and picks up items. But because π distributes its digits evenly, it only generates a '0' (the 'A' button) exactly 10% of the time. When a friendly NPC starts a three-sentence monologue, the system has to wait an average of ten seconds just to advance the text box once. Even worse, the 'B' button—which cancels menus—also gets pressed 10% of the time. π frequently traps itself in an endless loop of opening a menu and immediately canceling it, utterly incapable of stringing together the biased sequence of confirmations needed to progress.

A sequence that beats Pokémon has real, compressible structure. π has none. It is a mathematical tool designed to distribute noise evenly, being jammed into a digital lock that requires a highly specific, heavily biased key.

The Clockwork Universe

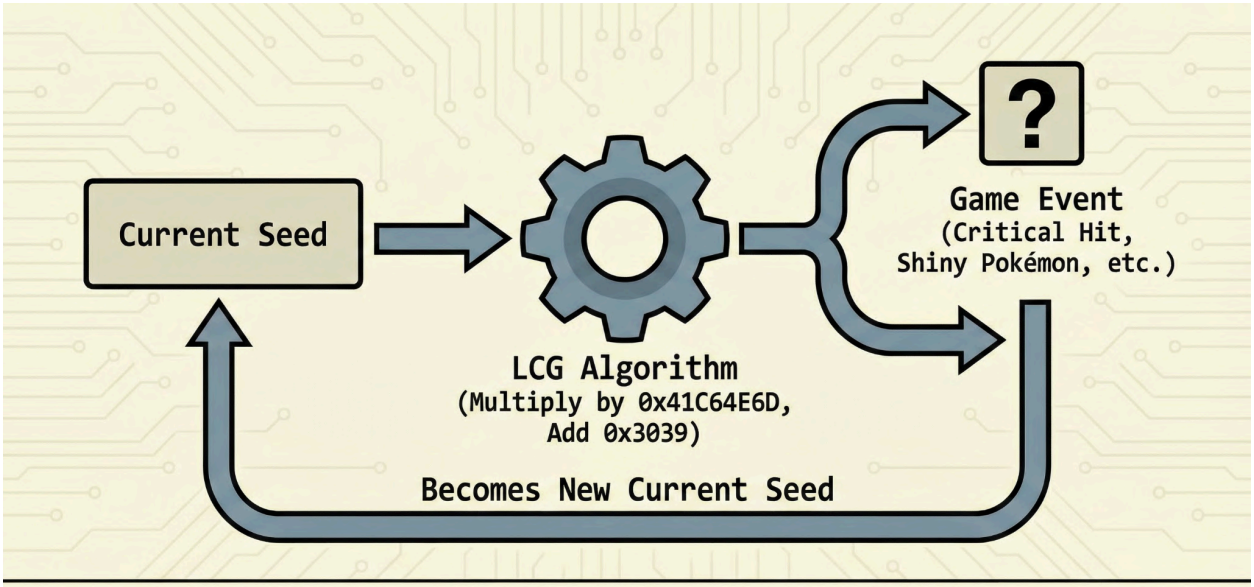
If the mathematics of π are too unstructured to beat Pokémon, we have to look at the mathematics of the game itself to find a better way. To a casual observer, a video game is a dynamic world full of unpredictable critical hits and spontaneous wild encounters. To a computer scientist, it is a rigidly deterministic state machine.

The game rolls no dice. It has never rolled dice. It has been performing the same calculation since you pressed New Game.

A Game Boy microprocessor cannot flip physical coins. Instead, it relies on mathematical formulas known as pseudorandom number generators (PRNGs) to simulate chance. Early Pokémon games use a Linear Congruential Generator (LCG), a simple algorithm that feeds the previous number in a sequence through a fixed math operation to get the next one.

The formula governing the entire universe of *Pokémon Sapphire* is exactly this:[6]

$$seed_{n+1} = (seed_n \times \text{0x41C64E6D} + \text{0x3039}) \pmod{2^{32}}$$



Every “random” event in Pokémon is just the same mathematical formula churning out its next predetermined number.

This single equation dictates the fabric of the game's reality. When the game needs to make a "random" decision—like whether a Poké Ball works or an opponent lands a critical hit—it runs the current seed through that multiplier, adds the increment, and spits out a new number.[6] That result determines your fate, and the new number becomes the seed for the next calculation.[6]

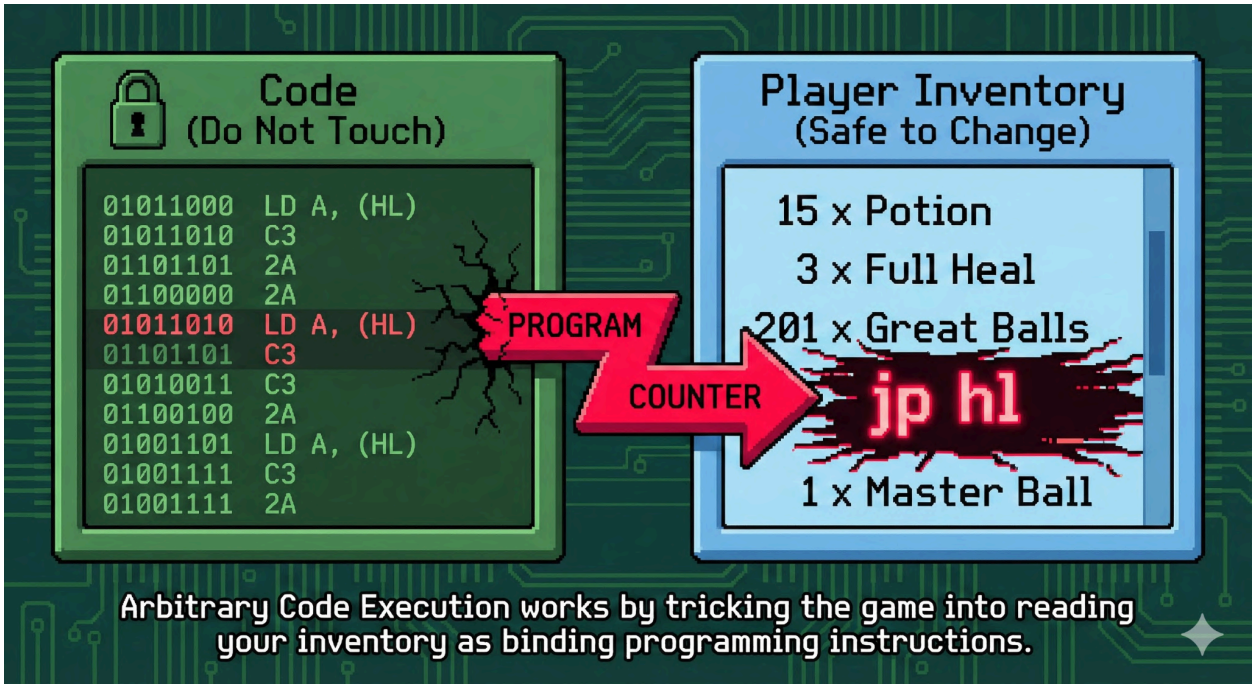
The implications are staggering: the game's entire future is mathematically predetermined. If you know the current seed and the formula, you know precisely what will happen one step from now, and one million steps from now. The generation of the notoriously difficult-to-find Pokémon, Feebas, relies heavily on this exact equation. The specific river tiles where Feebas can be caught are determined by a seed that is advanced using the exact LCG formula: $seed \times 0x41C64E6D + 0x3039$.[6] The resulting values are parsed modulo 447 to select the precise coordinate tiles on Route 119.[6] Players don't actually search for Feebas; if they know the math, they just calculate exactly where the game has already decided it will be.

Speedrunners exploit this determinism constantly. They do not *hope* for a critical hit; they mathematically guarantee it by engaging the enemy on the exact processor frame where the equation dictates a critical hit must occur.

Architectures of Chaos

If exploiting the math controls the game's logic, exploiting the memory map rewrites its reality.

The original Game Boy uses a unified memory architecture, meaning the executable code (the game's instructions) and the operational data (your inventory and Pokémon stats) live in the same physical memory space. Usually, the console's processor points to the Read-Only Memory to read code, executes it, and moves to the next line. But if a player can trick the processor into pointing at the Random-Access Memory—where things like item quantities are stored—the Game Boy will blindly attempt to read the player's inventory as if it were executable machine code.[7]



This is known as Arbitrary Code Execution (ACE).[7] In *Pokémon Red and Blue*, the most infamous vehicle for this memory hijacking is a glitch item called 8F.[8] Using it causes the processor to violently jump to the specific memory address that stores the number of Pokémon currently in your party.[7] Because the processor just keeps reading, it rolls right past the party count and begins interpreting the raw stats of your Pokémon as assembly instructions.[7]

To pull this off, you need a team that looks less like a viable Pokémon roster and more like the work of a deeply deranged zookeeper. You need exactly five Pokémon. First, a Pidgey, but not just any Pidgey—it must have exactly 233 HP. This usually requires force-feeding it Rare Candies until it reaches Level 100, poisoning it, healing it with very specific items, and walking a meticulously calculated number of steps to manipulate its health pool.[8] Following this mathematically battered bird, you need a Parasect, an Onix, a Tentacool, and a Kangaskhan.[8]

To a casual player, this is a terrible team with zero synergy. To the Game Boy's processor, the hexadecimal values representing this exact, absurd combination of species and stats translate perfectly into a machine-language jump instruction: jp hl.[8]

This flings the processor directly into your item bag. By managing the exact types of items and their quantities, you are literally typing assembly code into the game. Having 201 Great Balls next to 46 Antidotes is no longer an inventory; it is a custom script.

The system cannot distinguish between a variable denoting a quantity of potions and an assembly instruction to warp the player to the end credits. In 2026, a speedrunner used a variant of this exploit to beat *Pokémon Red* from a fresh save file in exactly 12 minutes, using

only 29 total presses of the 'A' button.[9]

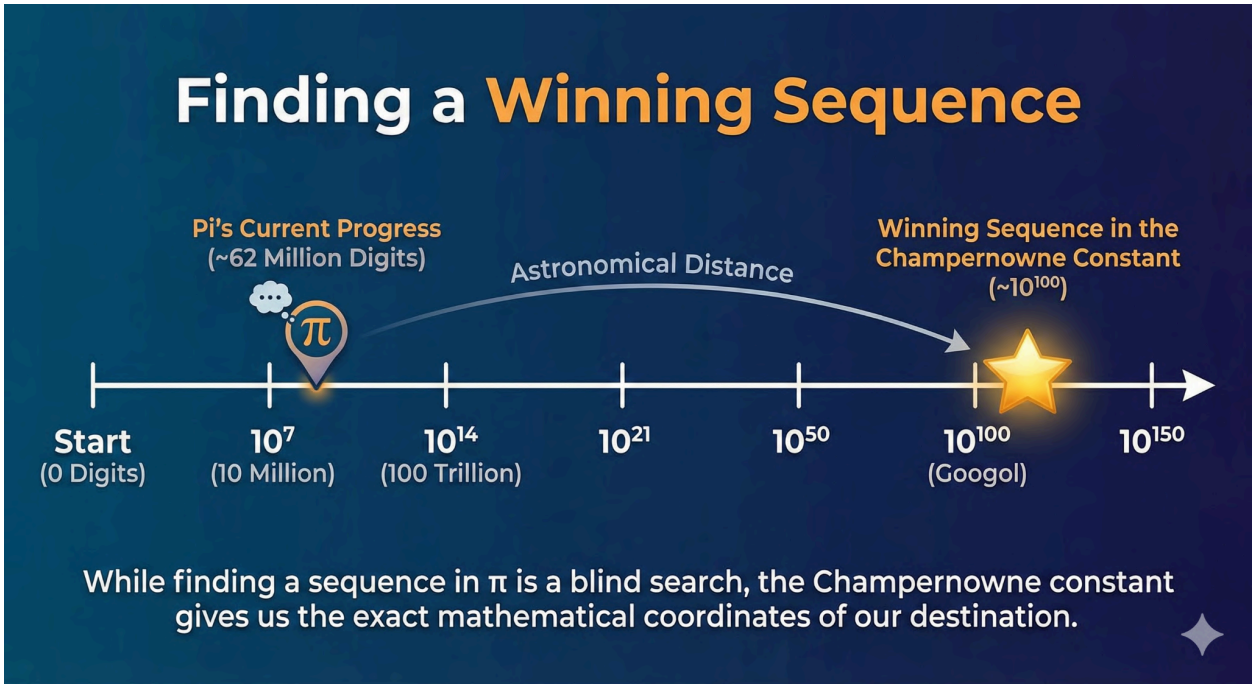
This illustrates a profound truth about digital environments: the shortest path through a video game is almost never a geographical line from the start to the finish. It is a mathematical bypass through the architecture of the hardware itself.

A Better Equation

If π 's random walk is the worst way to play, and human-crafted ACE runs rely on absolute determinism, what does a mathematical equation actually capable of beating Pokémon look like?

If we just want to improve our random number generator, we can use a biased mapping. Instead of a uniform 10% split, we map the digits 0 through 4 to the 'A' button. Suddenly, 'A' has a 50% probability, perfectly aligning the noise of our number with the structural demands of the game. Dialogues clear in seconds. π becomes infinitely better at the game without changing a single digit.

Alternatively, we could use a better normal number: the Champernowne constant (C_{10}). Published in 1933 by the economist and mathematician D.G. Champernowne when he was an undergraduate at Oxford, this constant is made by simply sticking every positive integer together in sequential order: $0.123456789101112\ldots$ [10] Like π , it contains every finite sequence of digits. Unlike π , we can calculate exactly where those sequences live.[10] If our winning sequence of button presses is a massive 100-digit number, we can use a mathematical formula to pinpoint its exact index. It would appear roughly at position 10^{100} .[10] That is an astronomically huge number—but it is a finite, calculable guarantee.



Why does this predictable positioning matter so much? Because it fundamentally transforms the nature of the problem from an infinite, blind search into a discrete, solvable database query.

When dealing with π, we are at the mercy of the unknown; we are hoping that the universe eventually rolls the exact sequence of dice we need, but we have no map and no timetable. The Champernowne constant provides a map. It guarantees that the winning sequence isn't just floating randomly in the ether, but is securely filed away in a specific, calculable drawer of mathematics. Even if that drawer is located at the absolute edge of calculable infinity, knowing its exact address means we have shifted from gambling to engineering. We are no longer waiting for a miracle. We are just waiting for a very long download to finish.

But the ultimate mathematical approach is to define a formula that simply extracts the exact winning sequence. If a flawless speedrun requires exactly 44,284 precise frames of input [9], we can encode that entire sequence into one massive integer, D. Then, we define a function using standard modular arithmetic:

$$f(n) = \lfloor D \times 10^n \rfloor \pmod{10}$$

For every input frame n, the equation yields the exact digit required to execute the flawless Arbitrary Code Execution speedrun. It is a legitimate, pure mathematical function. It guarantees victory on the first attempt.

Conclusion

The experiment of forcing π to play a Game Boy Advance emulator is undeniably absurd. But watching a mathematical constant fail to beat a children's video game teaches us something beautiful about how digital worlds are built. Infinite noise does not magically yield ordered progress, and sequences without structure cannot navigate a world built entirely upon structure.

This is the ultimate lesson of the Pi Plays Pokémon experiment. It is a spectacular collision between the untamed, infinite noise of the natural universe and the highly constrained, compressible logic of human engineering. We built these digital worlds to simulate reality, but we forgot that our simulated realities are vastly more fragile and structured than the real thing. When we throw a pure mathematical constant at a human-designed state machine, the friction exposes the seams of the simulation. It reminds us that video games are not magical landscapes of adventure; they are delicate houses of cards built from hexadecimal values and memory pointers. The real legacy of Pi Plays Pokémon isn't that π failed. It's that an absurd internet experiment made millions of people look at a twenty-five-year-old children's game and suddenly see the mathematics they never knew was there. It revealed the breathtaking complexity, the fragility, and the invisible equations holding the digital world together—proving that beneath the tall grass and the pixelated monsters, it has always been numbers all the way down.

Gemini's Nano Banana Pro was used to generate the images for this essay

Works cited

1. The mathematical concept of Pi can now play Pokemon, but still can't find the first gym after ... - GamesRadar, accessed on February 22, 2026, https://www.gamesradar.com/the-mathematical-concept-of-pi-can-now-play-pokemon-but-still-cant-find-the-first-gym-after-16000-hours-of-trying/
2. CAN THE NUMBER π BEAT POKÉMON? | Pi Plays Pokémon Sapphire - Stream #731 Part 1 - YouTube, accessed on February 22, 2026, https://www.youtube.com/watch?v=QJIVIfnLrd4
3. Normal Numbers are Normal - Clay Mathematics Institute, accessed on February 22, 2026, https://www.claymath.org/library/annual_report/ar2006/06report_normalnumbers.pdf
4. Normal number - Wikipedia, accessed on February 22, 2026, https://en.wikipedia.org/wiki/Normal_number
5. Kolmogorov complexity - Wikipedia, accessed on February 22, 2026, https://en.wikipedia.org/wiki/Kolmogorov_complexity
6. [Gen 3] Using RNG manipulation and the Dewford Trend to Determine Feebas Tiles and Secret ID: https://bulbapedia.bulbagarden.net/wiki/Pokémon_Ruby_and_Sapphire_Versions
7. Arbitrary code execution - Bulbapedia, the community-driven ..., accessed on February 22, 2026, https://bulbapedia.bulbagarden.net/wiki/Arbitrary_code_execution
8. List of 8F bootstrap setups - Glitch City Wiki, accessed on February 22, 2026, https://glitchcity.wiki/wiki/List_of_8F_bootstrap_setups
9. #10176: CasualPokePlayer's GB Pokémon: Red Version "minimum ..., accessed on February 22, 2026, https://tasvideos.org/10176S
10. Champernowne constant - Wikipedia, accessed on February 22, 2026, https://en.wikipedia.org/wiki/Champernowne_constant