Teaching Cannoli Which Screen Is Home

July 2026

12 min read • 2,707 words

The AYN Thor is an Android handheld with two screens and, out of the box, no opinion about what either of them is for.

The upper screen is wide and bright and obviously wants to be where the game happens. The lower screen is smaller, nearly square, and close to the controls. It wants to be a menu. The hardware makes this arrangement feel inevitable. Android does not. Android sees displays, activities, tasks, focus, and a default screen. It can put software on both panels, but it has no idea what the object is trying to become.

I wanted the Thor to feel less like two Android windows attached to a controller and more like one game console. The lower screen would be the console's face: systems, games, settings, selection. The upper screen would be the game surface. When no game was running, it would be completely black. While a game was running, the launcher below would dim and stop accepting input. Close the game and control would return to the lower screen. Press Home and the console would come home, not drop me into an Android app drawer.

The specification was almost embarrassingly simple:

```
if game.is_running:
    upper_screen.show(game)
    lower_screen.dim()
else:
    upper_screen.black()
    lower_screen.show(library)
```

It took five commits, thirty-seven changed files, a side-by-side Android package, an ADB-driven test loop, a new launch log, a gesture recognizer, and several encounters with the difference between what Android says happened and what the person holding the device can plainly see.

It was one of the most enjoyable software projects I have worked on in years.

An Opinionated Launcher on an Unopinionated Device

Cannoli describes itself as "an opinionated retro gaming setup for Android." That word, opinionated, is doing important work.

Most retro gaming frontends are eager to demonstrate their capabilities. They offer themes, scraping, artwork, videos, widgets, metadata, transitions, shelves, collections, and enough settings to turn choosing a game into its own hobby. Cannoli is inspired by Shaun Inman's MinUI, which takes the opposite position. It gives you systems, games, a few essential actions, and very little else. The interface does not compete with the thing it launches.

Cannoli carries that restraint into Android without pretending Android is a simple platform. Under its flat, list-based surface are bundled libretro cores, external RetroArch support, standalone emulators, Android games and ports, controller profiles, saves, RetroAchievements, a web-based library manager, and even a mode that reduces the whole machine to exactly five games.

> Five Game Handheld mode removes even the system picker. The front page becomes five titles and nothing else, less a filter than a small commitment about what you are actually playing.

It is doing a great deal of work to create the feeling that very little work is required.

That made it right for the Thor. The lower display has exactly the shape Cannoli's compact lists want. The upper display is where games want to breathe. I did not need to invent a new launcher. I needed to help a very good launcher understand the physical object it had landed inside.

The distinction matters. A feature can technically support two screens and still fail to make a dual-screen device feel coherent. Coherence comes from assigning each screen a role, then preserving that role through every launch, return, crash, tap, and system gesture.

Smaller Is Not Secondary

The first version of the idea was phrased in Android's language: launch Cannoli on the secondary display and games on the primary display.

That was wrong.

"Primary" and "secondary" are labels assigned by the operating system. They say nothing about which panel is physically larger, which one sits near the controls, or which one a person expects to look at while playing. The real requirement was semantic: Cannoli belongs on the smaller display; games belong on the larger one.

> On this Thor, Android calls the 1920 by 1080 upper panel display 0 and the 1240 by 1080 lower panel display 4. Those numbers are useful for a log and meaningless as product design.

So the routing code learned to inspect the available public displays, estimate their physical area from dimensions and reported density, and fall back to pixel area when Android's measurements were untrustworthy. The smallest eligible display became the launcher display. The largest became the game display. If there were not at least two valid displays, none of the experiment applied.

That last condition was important because this was becoming an upstream contribution, not a private patch hard-coded to the shape of my own device. The dual-screen behavior lives under **Settings -> Advanced -> Experimental Features**, defaults to off, and leaves ordinary single-screen Cannoli untouched.

The code became more general by expressing the requirement in human terms. This is a pattern I keep rediscovering: software becomes sturdier when the names inside it describe the intention outside it.

The Black Screen Was the Feature

At first I imagined using the upper screen while browsing. It could show cover art, screenshots, maybe details about the selected game. This is the sort of idea that appears whenever hardware has an empty surface. The surface is available, therefore the software should put something on it.

Then I looked at my actual library. There was not much artwork. A showcase screen would mostly have showcased missing data, and adding enough decoration to hide that fact would have violated the reason I liked Cannoli in the first place.

The right design was black.

When no game is running, a borderless black activity owns the upper display. It hides Android's system bars, stays out of the recent-apps list, and does not keep the panel awake.

> The black surface has its own Android task affinity. It remains separate from both the launcher history and the emulator history, which is why closing a game can reveal black without dragging either task stack with it.

When a game launches, the game naturally covers it. When the game closes, there is no flash of an app drawer or abandoned emulator task. There is simply black again, a quiet surface waiting for play.

Below it, Cannoli remains visible but dims to five percent brightness during a game. Dimming the window was easy. Preventing the lower screen from stealing focus was not. Ignoring a button press inside Cannoli still allowed Android to

move input focus away from the game above. The launcher had to become non-focusable, cancel pending input, clear held selections, stop repeat handlers, and reset its controller state while play was active.

This is the sort of invisible work that makes an appliance feel obvious. The finished behavior contains no hint of the task and focus machinery underneath it. It just stops the wrong screen from interrupting the right one.

The Bugs Lived Between Systems

Almost every interesting failure in this project happened at a boundary where two systems disagreed about identity or ownership.

The first experimental APK was "invalid," except it was not invalid. Its archive was intact, its metadata was readable, and its signature verified. The problem was identity: a locally signed build cannot replace an official build signed by the project. Android was correctly refusing the update while a file browser compressed that whole explanation into one unhelpful word. We gave the experimental build its own package name so it could live beside the official installation and be updated repeatedly with the same local key.

> The test package is `dev.cannoli.scorza.secondaryscreen`. Giving an experiment a separate identity was much faster than repeatedly uninstalling the known-good version I still wanted on the device.

Then Android accepted a game-launch intent and moved focus to the upper screen, but no game appeared. From Cannoli's side, dispatch had succeeded. From my side, I was holding a console that had gone blank. That gap produced a bounded launch log containing the emulator package and activity, selected core, ROM path, display route, launch extras, and session lifecycle. The log revealed that an explicit choice of external RetroArch could still be silently replaced by an installed embedded core. The preference was fixed so a deliberate user choice stayed deliberate.

Returning from external emulators required its own activity. It sits behind the game on the upper display, watches the external session take and return focus, then brings the existing Cannoli task back to the lower display. "Go back to the launcher" sounds like navigation. On dual-screen Android it is a small lifecycle state machine.

The strangest bug came from tapping the black upper screen to return focus to Cannoli. The first implementation moved focus on touch-down. That interrupted the pointer stream before touch-up reached the same window. Android sometimes interpreted the incomplete gesture as a Home swipe, and because Cannoli was now the HOME application, the result looked like a crash-and-restart loop that could also tear down a running game.

The fix was not a delay or a special case. It was a real tap recognizer: wait for touch-up, cancel on movement beyond the system touch slop, reject multitouch and canceled gestures, consume the whole stream, and only then change focus. A tiny convenience feature had crossed display focus, gesture navigation, task replacement, and launcher semantics. The bug was completely unreasonable until we understood it, at which point it became completely reasonable.

This is why I still love debugging. Reality is under no obligation to respect the boundaries in our source tree.

Building With GPT-5.6 Sol

I did all of this with GPT-5.6 Sol.

That sentence can be mistaken for the increasingly common claim that an AI generated a feature from a prompt. That is not what happened. I was holding the device, deciding what felt right, noticing what was wrong, and changing my mind as the object became clearer. Sol held the codebase, build system, Android APIs, test suite, device logs, Git history, and evolving requirements together long enough for each observation to become the next build.

The loop became very physical. I would try the new APK on the Thor and report what happened: focus moved but the game did not launch; the two Citra entries had the same name; touching the black screen restarted everything; the lower panel still intercepted a control; Pokémon FireRed could not see the save I knew

existed. Sol would inspect the evidence, trace the behavior across layers, change the implementation, add a regression test, build and sign the next APK, install it through ADB, and wait for the next encounter with the actual machine.

Installing ADB changed the tempo. Before it, every build meant moving an APK that could be nearly three hundred megabytes onto the device by hand, choosing among several artifacts, opening a file manager, and hoping Android's installer would explain itself. Afterward the loop was build, install-over, launch, inspect, repeat.

> `$ adb install -r` could update the experimental package in place while leaving the official Cannoli installation alone. One command replaced the whole file-transfer and device-installer ritual.

Sol could operate that loop directly while I concentrated on the part only I could supply: whether the result felt like one console.

The requirements changed because we were testing a physical object, not implementing a document. "Secondary" became "smaller." A rich artwork surface became pure black. Ignoring input became preventing focus theft. A touch shortcut became a gesture-lifecycle problem. An external RetroArch menu was investigated and then deliberately left alone because reproducing Cannoli's embedded menu across process boundaries would have turned a focused contribution into a new architecture.

The collaboration was most useful exactly where the work was least glamorous. Sol did not get tired of another 280-megabyte build, another log comparison, another package-label test, or another look at which activity owned which display. That stamina did not replace judgment. It kept judgment connected to execution.

The result was organized into five reviewable commits and published as draft pull request #205 against Cannoli's `v1.8.0` branch. The PR adds Citra MMJ support, physical-size display routing, respect for explicit external RetroArch choices, coordinated game sessions, the black idle screen, launcher dimming, input protection, diagnostics, and the safe return gesture. It is still experimental and still honest about being proven on one very unusual device.

The Library Moved to Mercury

Somewhere in the middle of all this, I discovered that Cannoli already knows how to speak to RomM, a self-hosted ROM library manager. Pair the two and Cannoli can browse a larger library, search it, pull games and manuals onto the handheld, fill in missing artwork, and optionally synchronize saves.

> After pairing, **Browse RomM** appears in Cannoli's Quick Menu. The server library feels like another local shelf rather than a separate administration surface.

So, naturally, the handheld project escaped the handheld.

I installed RomM on a server called Mercury, the small Hetzner box that now runs most of my digital life. Mercury already hosts my sites, mirrors my code, watches my services, stores my analytics, and keeps copies of things I do not want to exist only inside someone else's platform. A personal game library fit the pattern perfectly. The Thor no longer needed to carry every possible game or depend on a desktop ritual for adding one. The quiet menu in my hands could reach back to a library on a machine I control.

That made the whole system feel larger and, somehow, simpler. Mercury holds the collection. Cannoli presents the choice. The upper screen plays the game. Each layer has one job and a plain relationship to the next.

Even the save migration reinforced the lesson about identity. We moved portable saves from RetroArch's folders into Cannoli's platform-oriented layout with hashes, temporary files, collision handling, and a recoverable backup. The migration worked, but FireRed still opened without the progress I expected. Cannoli was launching `Pokemon - Fire Red.gba`, so it correctly looked for `Pokemon - Fire Red.srm`. The newer save belonged to a longer historical filename: `Pokemon - Fire Red Version (U) (V1.0).srm`.

Same game, different human names, different identity as far as the filesystem was concerned. We compared the historical saves, found the newest valid one, relinked it to the ROM Cannoli actually launched, and preserved the older destination in the backup. Once again, every component was doing what it had been told. The system was wrong because the components did not agree about what the thing was called.

One Console

The final setup is quiet.

Cannoli lives on the lower screen. The upper screen is black until a game appears. During play the menu below becomes a faint presence and gives up control. Close the game and the launcher returns. Press Home and you come back to Cannoli. Browse RomM and the library arrives from Mercury. The ordinary Android machinery is still there, but it has stopped being the experience.

This whole project is a small demonstration of why open source is still astonishing. Cannoli's maintainers did not have to own a Thor, predict that someone would want the launcher below and the game above, or decide that this one unusual device justified a place on the roadmap. The source was there. The user with the strange hardware could become the contributor.

The GNU GPLv3 is not normally my license of choice. I tend to prefer permissive licenses. I want code to travel with as little ceremony as possible, including into places I never expected and products I will never see.

But a launcher feels different. It is not merely one component inside a product. It can become the face of the machine, the place where a person spends their time, and the interface through which everything else becomes available. It is exactly the kind of software a hardware vendor could absorb, rename, improve behind closed doors, and ship back to people who would never know where it came from. The GPL makes a different bargain: if you distribute a modified Cannoli, the people receiving it inherit the same freedom to inspect, change, and share it that made this Thor experiment possible.

I think that bargain makes sense for a launcher. If software becomes the face of a device, the people who own the device should be allowed to see and change the face. My Thor can feel like one thing because someone else made an opinionated launcher open to inspection, and because its license protects the next strange user as carefully as it protected me.

The Thor always had two screens. Now it feels like one thing.