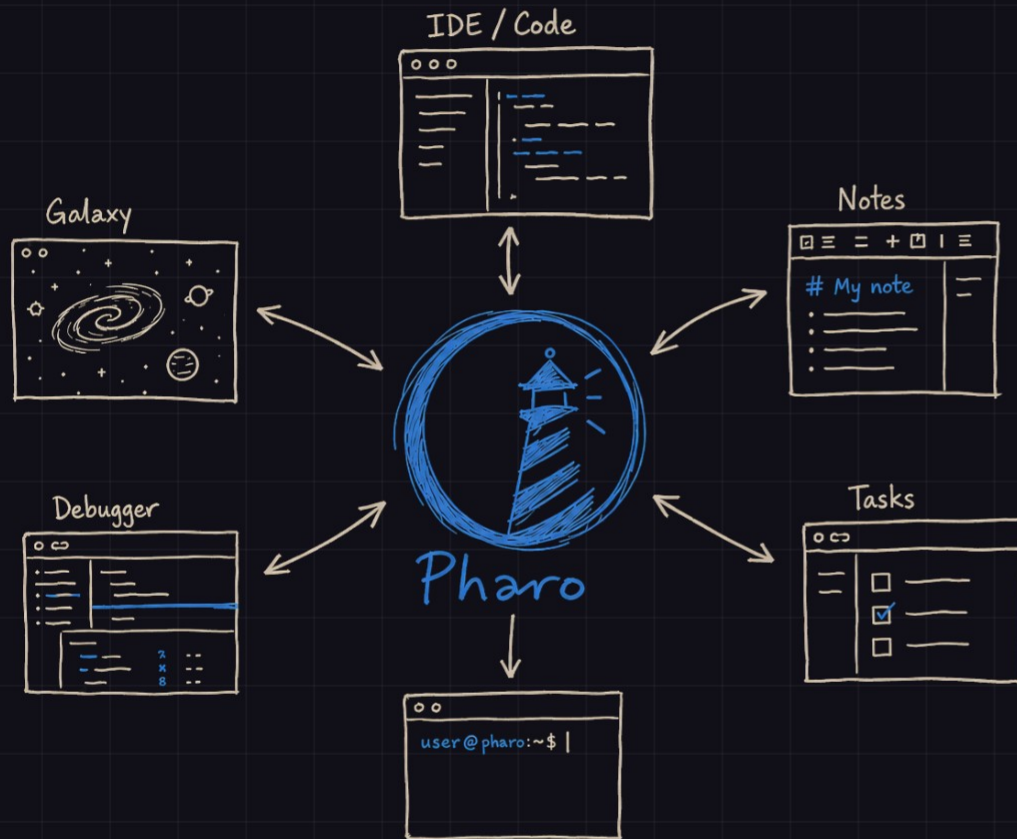


ESUG 2026 · PLOVDIV

Pharo on the Desktop

A tour of tools, friction, and feedback loops

Esteban Lorenzano



The desktop was declared dead

Again, and again, and again.

- applets
- PHP / JSP / web frameworks
- “my language to JavaScript”
- WebAssembly
- Electron everywhere

And yet...

We still build tools that need to be **fast**, **local**, **integrated**, and **shaped around real workflows**.

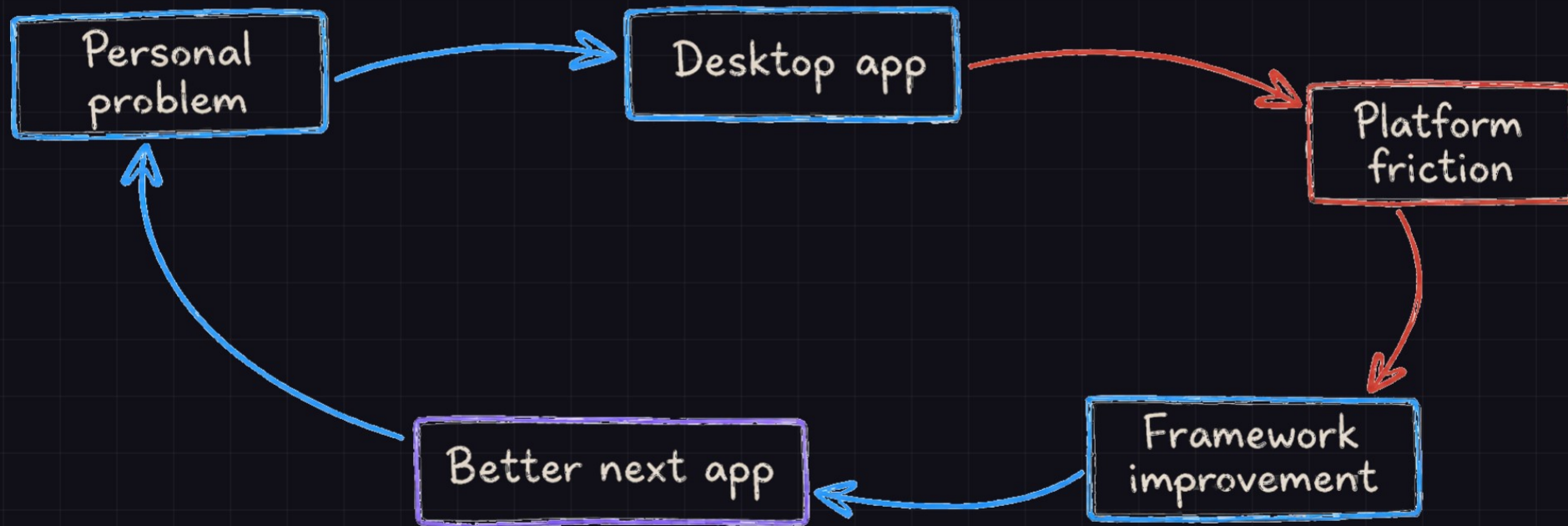


The question

- Can Pharo be used not only to build Pharo, but to build desktop applications I genuinely want to use every day?



The feedback loop



SHOW ME THE APPS!



Context: "ELITE: DANGEROUS"



A game from the 80's

- Explore and mine the galaxy.
- Get some combat.
- Procedurally generated

Let's simulate our own galaxy...

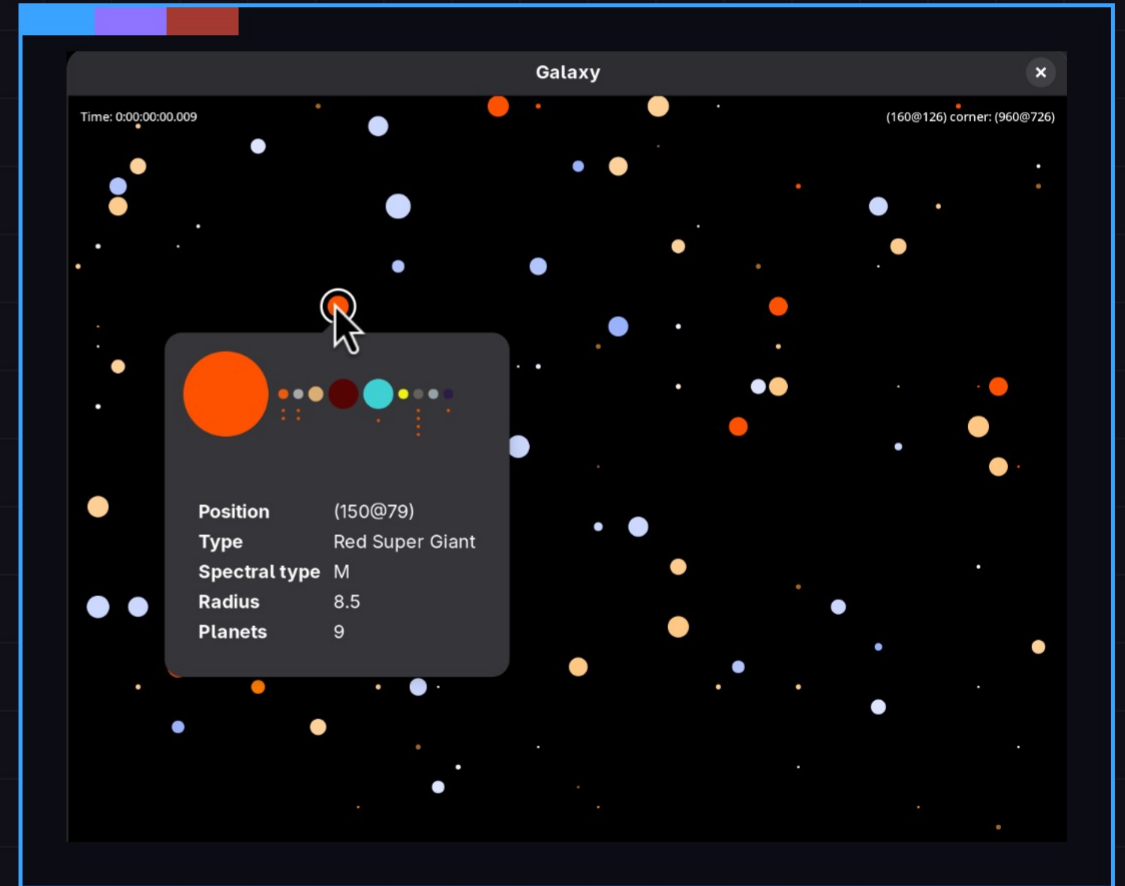


Procedural Galaxy

STARTING POINT

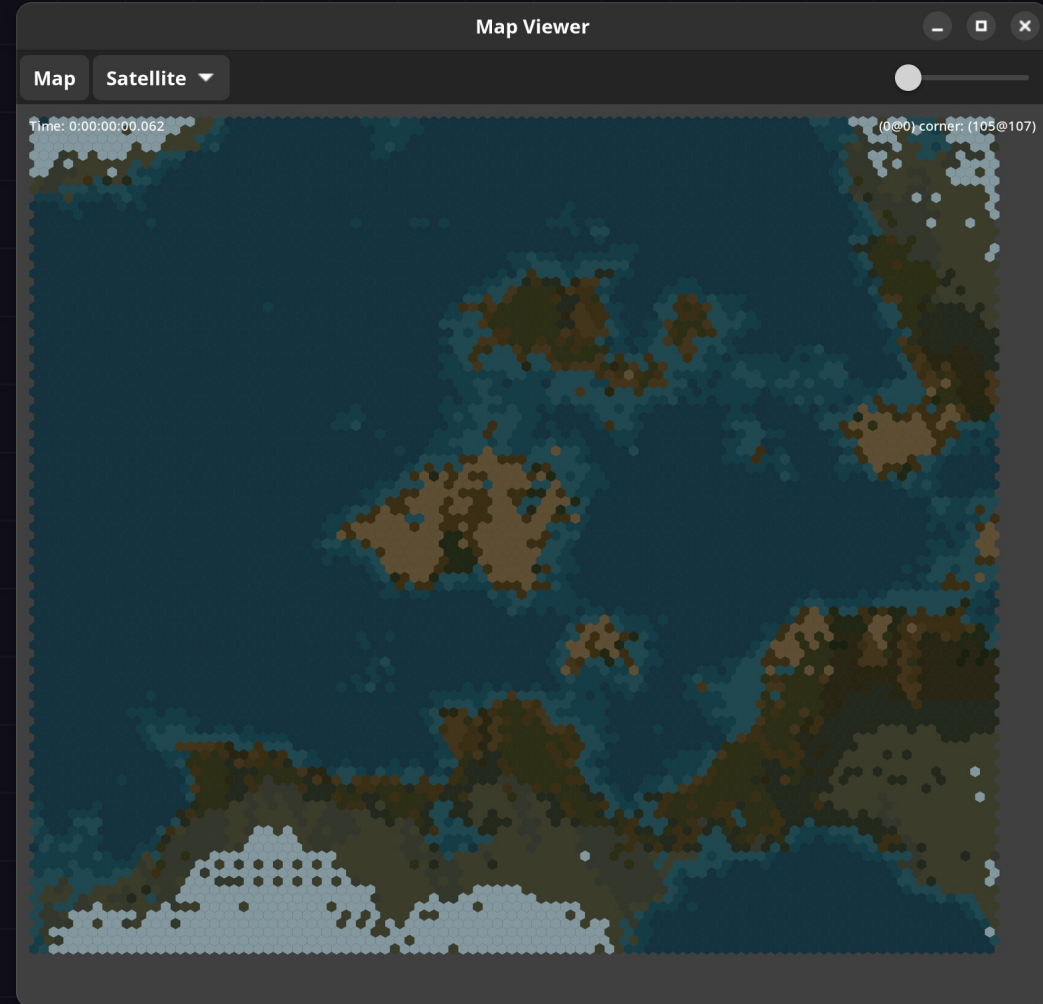
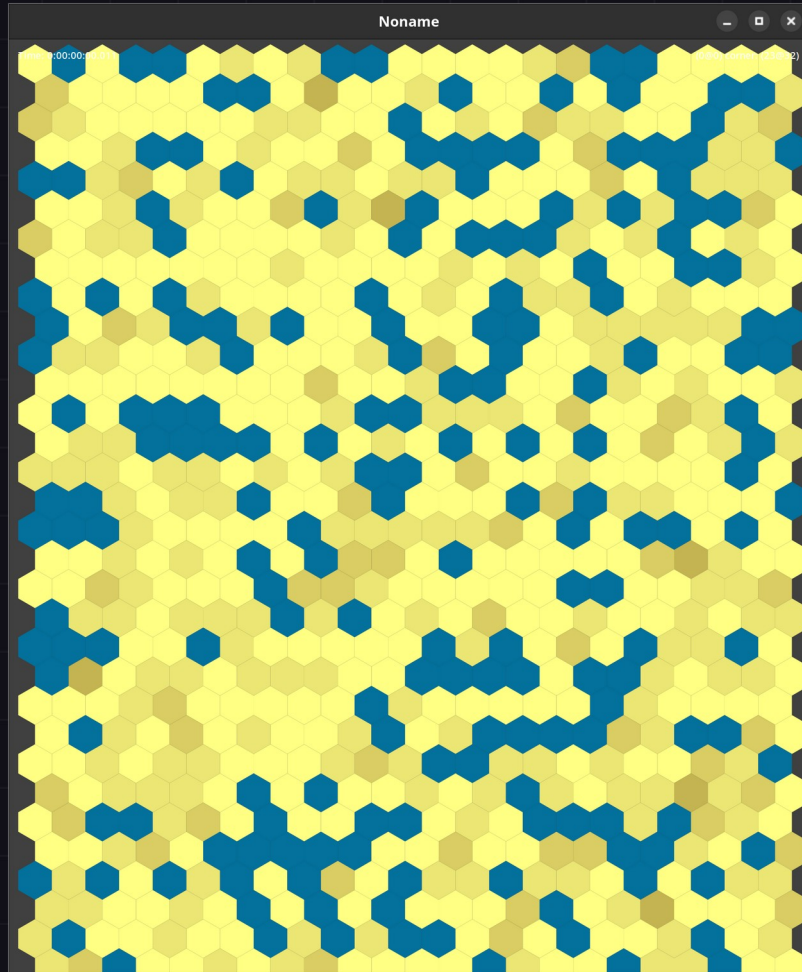
An infinite procedural galaxy:

- stars, types, sizes
- planets and moons
- scrolling exploration
- hover interaction



And before you ask... yes, I also built the planets :)

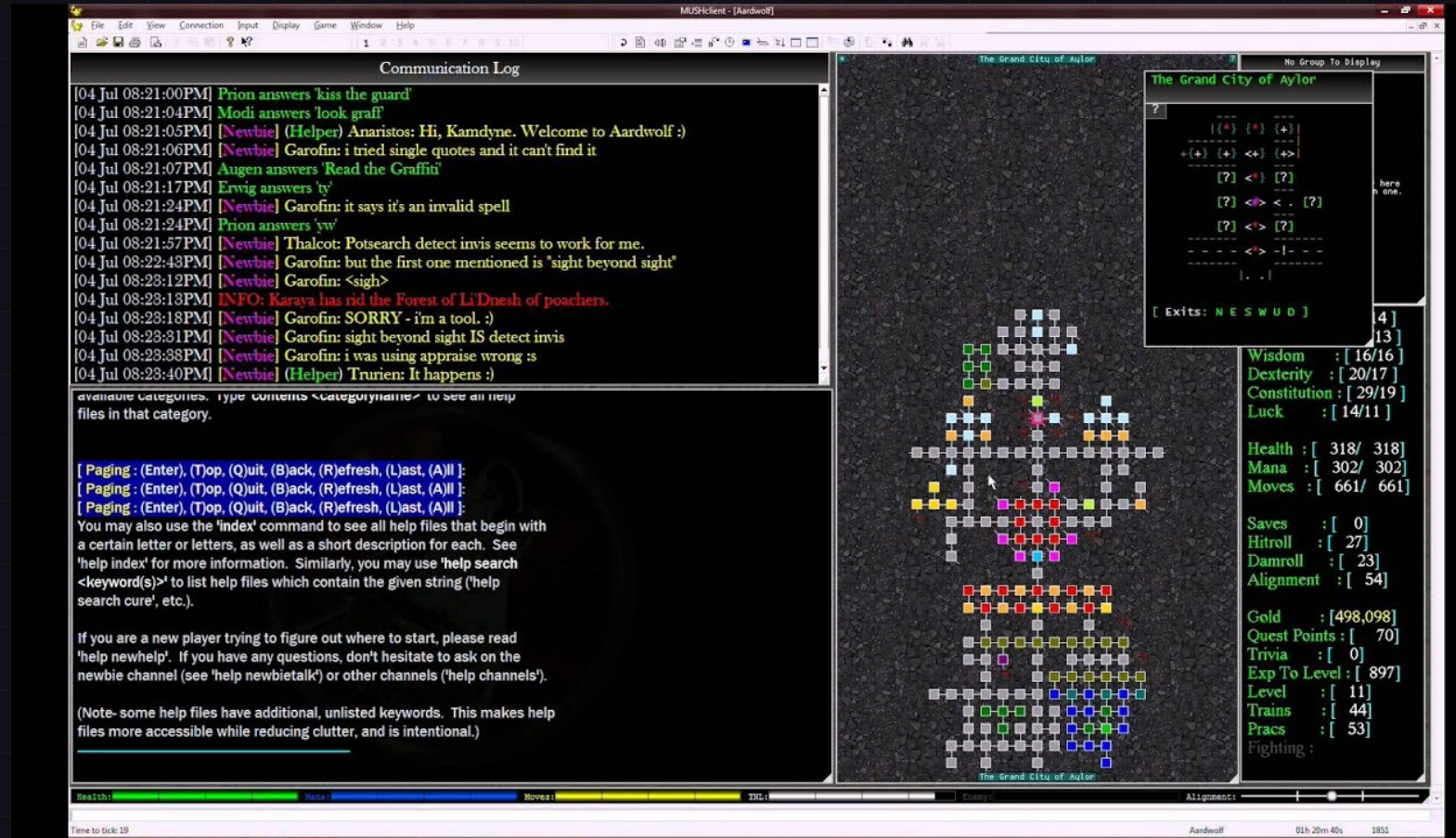
(but I had to stop when real work called me...)



Context: "Multi User Dungeons"

Another game from the 80's

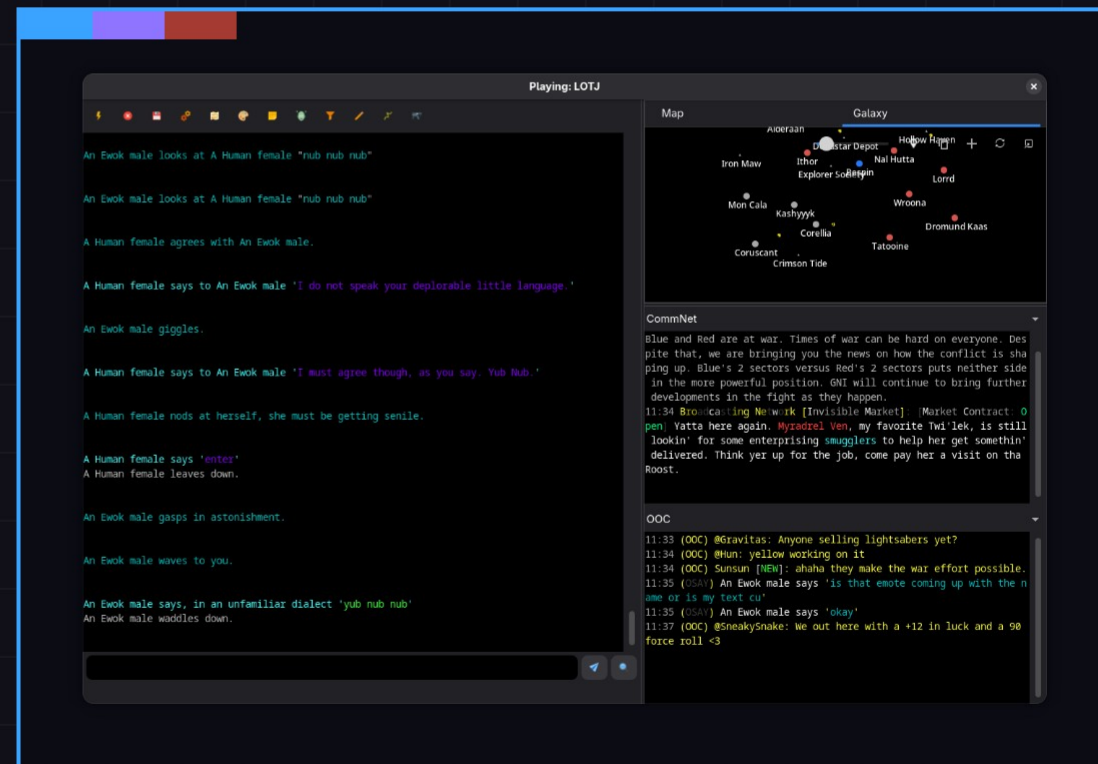
- Role playing games like a pro.
- Everything in a terminal !
- ... and telnet protocols.



Let's connect to a MUD and play...



OLD GAME, OLD PROJECT, NEW BACKEND

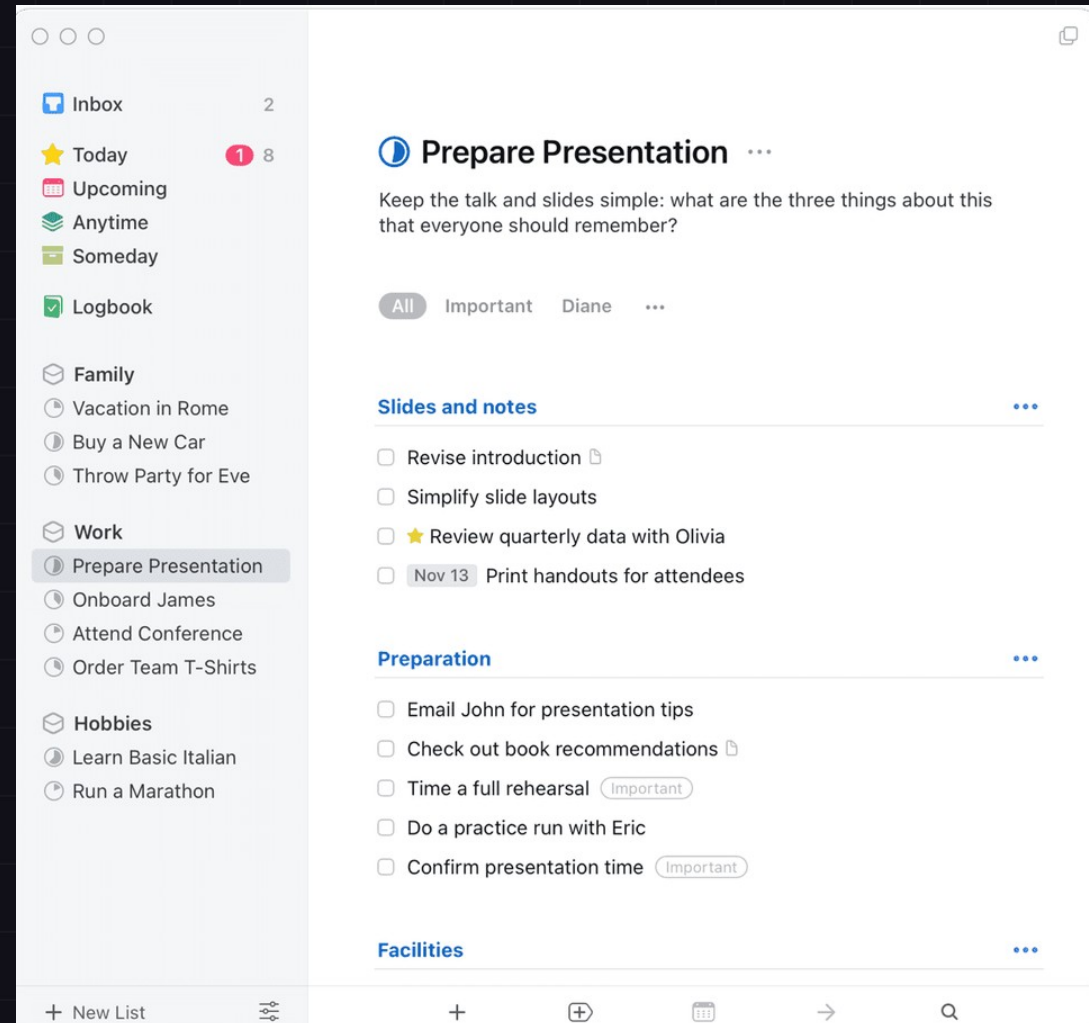


Context: "GTD and productivity in general"

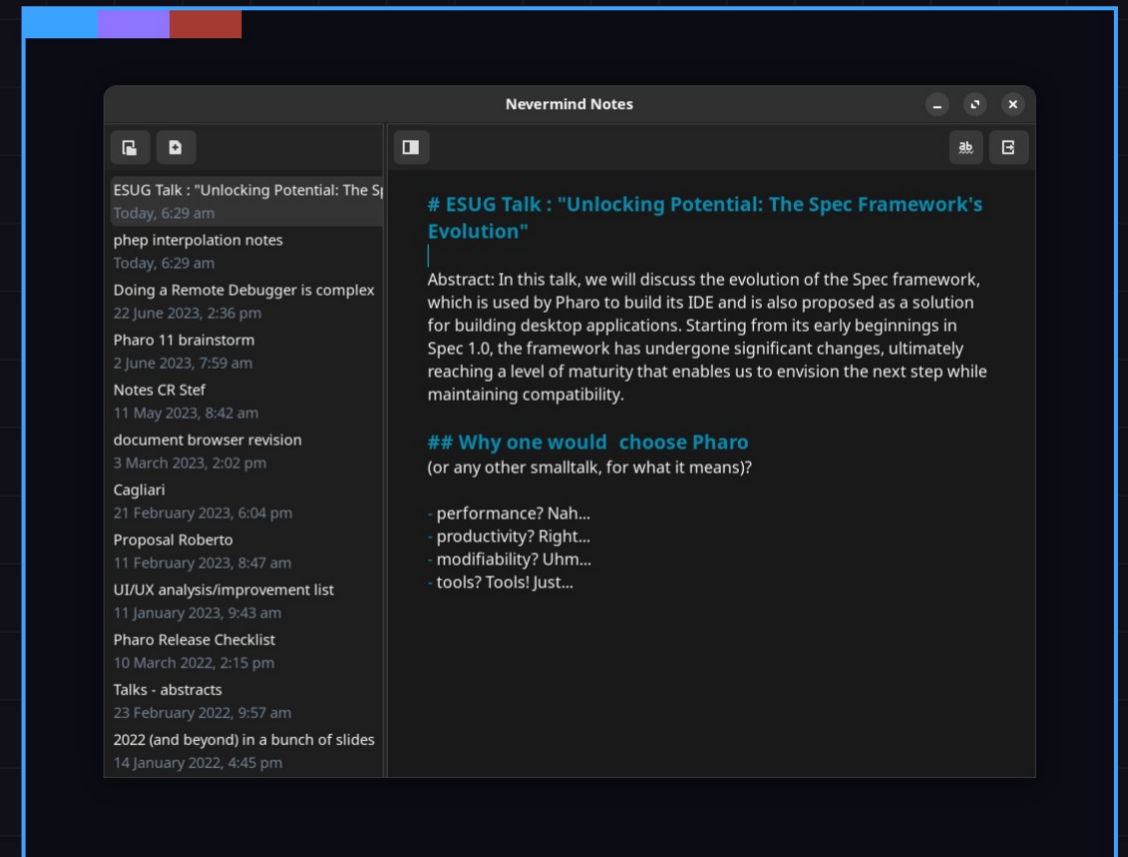
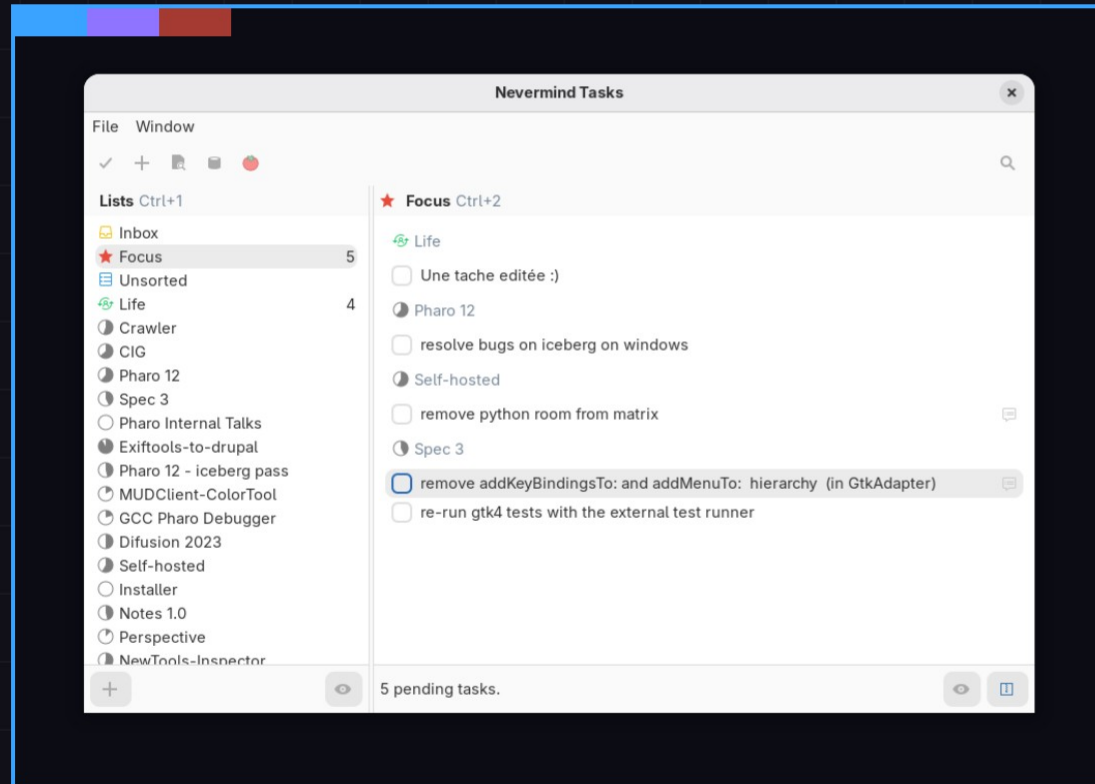
"Get Things Done"

- A technique to avoid procrastination (kind of...)
- Markdown notes
- Pomodoro

Let's do our own...



Nevermind Tasks and Notes



Context: "Application crashes"

... and when you use C libraries, you have C crashes

(This is the sexier thing you can get with gdb TUI)

You got the idea...
Let's do our own !

```
Terminal
race.cpp
35 {
36     /* Take ownership of g_mutex for the duration of this scoped block. */
37     const std::lock_guard<std::mutex> lock(g_mutex);
38
39     if (i % (10 * 1000) == 0)
40     {
41         std::cout << __FUNCTION__ << ": i=" << i << "\n";
42     }
43     /* Increment <g_value>. Should be safe because we own g_mutex. */
44     int old_value = g_value;
45     int a = random_int(5);
46     g_value += a;
47     assert(g_value == old_value + a);
48     (void)old_value;
49 }
50 }
51 }
52
53 void
54 threadfn2()
55 {
56     for (int i = 0;; ++i)
57     {
58         if (i % (100) == 0)
59         {
60             std::cout << __FUNCTION__ << ": i=" << i << "\n";
61         }
62     }
63 }
64
65 int main()
66 {
67     std::thread t1(threadfn1);
68     std::thread t2(threadfn2);
69     t1.join();
70     t2.join();
71     return 0;
72 }
```

```
Backtrace
#0 __pthread_kill_implementation (no_tid=0, signo=6, threadid
#1 __pthread_kill_internal (signo=6, threadid=<optimised out>
#2 __GI___pthread_kill (threadid=<optimised out>, signo=signo
#3 0x00007ffff784526e in __GI_raise (sig=sig@entry=6) at ../
#4 0x00007ffff78288ff in __GI_abort () at ./stdlib/abort.c:7
#5 0x00007ffff782881b in __assert_fail_base (fmt=0x7ffff79d0
#6 0x00007ffff783b507 in __assert_fail (assertion=0x5555555
#7 0x00005555555566e8 in threadfn1 () at race.cpp:47
#8 0x000055555555786d in std::__invoke_impl<void, void (*)()
#9 0x0000555555557819 in std::__invoke_void(<*>())> (<fn=0x
#10 0x00005555555577ba in std::thread::_Invoker<std::tuple<vo
#11 0x000055555555778a in std::thread::_Invoker<std::tuple<vo
#12 0x000055555555776a in std::thread::State_impl<std::threa
#13 0x00007ffff7cecd4b in ?? () from /lib/x86_64-linux-gnu/li
#14 0x00007ffff789ca94 in start_thread (arg=<optimised out>)
#15 0x00007ffff7929c3c in clone3 () at ../sysdeps/unix/sysv/l
```

```
multi-thre Thread 0x7ffff6c006 (src) In: threadfn1 L47 PC: 0x5555555566e8
#3 0x00007ffff784526e in __GI_raise (sig=sig@entry=6) at ../sysdeps/posix/raise.c:26
#4 0x00007ffff78288ff in __GI_abort () at ./stdlib/abort.c:79
#5 0x00007ffff782881b in __assert_fail_base (fmt=0x7ffff79d01e8 "%s%s%s:%u: %s%sAssertion `%s
assertion=assertion@entry=0x55555555803d "g_value == old_value + a", file=file@entry=0x555
function=function@entry=0x555555558023 "void threadfn1()") at ./assert/assert.c:94
#6 0x00007ffff783b507 in __assert_fail (assertion=0x55555555803d "g_value == old_value + a",
function=0x555555558023 "void threadfn1()") at ./assert/assert.c:103
#7 0x00005555555566e8 in threadfn1 () at race.cpp:47
(gdb) source tui_windows.py
(gdb) layout many-windows
(gdb) tui new-layout src-hsplit-backtrace {-horizontal {src 2 status 1 cmd 1} 3 backtrace 2} 1
(gdb) layout src-hsplit-backtrace
(gdb)
```



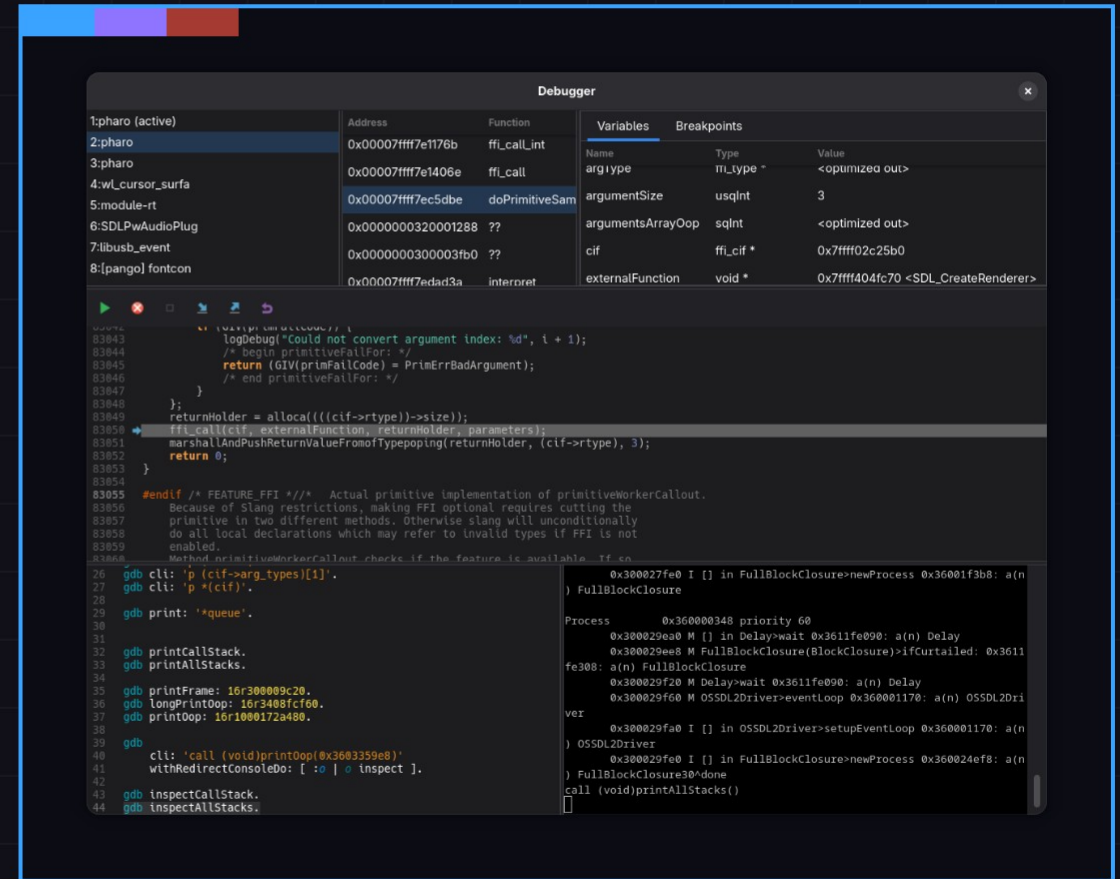
Pdbg

THE STAKES RISE

With FFI-heavy applications, some failures do not become Pharo exceptions.

They become process crashes.

At that point, Pharo debugging becomes **GDB debugging**.



Applications are integration tests for Pharo

Frameworks become real in use

Widgets, editors, lists, terminals, perspectives.

Boundaries become visible

Packaging, dependencies, native APIs, threads, process failures.

Abstractions become sharper

Requirements discovered by use are harder to ignore.



Note: Packaging is still unfinished

But we will arrive there!

This talk is not mainly about packaging.

But every application came back to it:

- smaller images
- dependency closure
- native libraries
- distribution
- cross-platform expectations

The missing tool is still clear:

**open → configure →
package**

A small, practical path from image to desktop application.



Build the applications that make the imperfections impossible to ignore.

Every tool began as a personal annoyance. Every annoyance revealed a missing capability. Every missing capability became a chance to make Pharo better.

