How I spent my summer vacation

or, my adventure with Skyrim modding

I tried to do a WASM startup last here. We had some fun ideas, but we went to pitch VCs at exactly the moment every single one of them aligned like iron filings near a magnet toward Large Language Models instead of cryptocurrencies. So we folded. I went to sit at home and sulk for a while. I don't enjoy sulking, so I thought, well, why don't I play some video games? Specifically, I went for a video game comfort food. I went for Skyrim.

fus-ro-dah!

What's Skyrim anyway?

Skyrim is an award-winning 2011 game by Bethesda Game Studio, the fifth in their Elder Scrolls series of role-playing games. It has an amazing sound track, middling graphics, and some fascinating lore. It's been released on PC plus every game console there is. There's a VR version. Game director Todd Howard would release it on your toaster if he could.

Skyrim is a region of the larger Elder Scrolls world, roughly equivalent to Scandinavia with a Viking-ish civilization. There's a civil war in progress as a larger human Empire falls apart because of what the player did in the previous two Elder Scrolls games. To make things worse, a dragon shows up and surprises everybody.

It opens kinda like this.

What's Skyrim? Here is its opening sequence. The wipe goes from vanilla Skyrim to what my current modlist looks like. I think Lokir of Rorikstead is a bit too cute for the character.

Skyrim == mods

Mods change the look a lot, huh? If you've been playing Fallout again recently because of the TV show, you know that it goes further than this. Bethesda games overall are very moddable.

Skyrim is the sweet spot

— the game is beloved **but**...
— the rendering is terrible
— the bugs are bad
— the UI was designed for consoles
— but all of this can be fixed...
— because the game is moddable

I built a gigantic list of 2000+ mods

So let's go back to my sulk-ending period of Skyrim modding. I built a ridiculous modlist for myself, everything I'd always wanted to play with. It was something like 2000 plugins before I started trying to winnow it down.

But...

I couldn't find a hotkeys mod that wasn't

a. slow
b. buggy
c. both buggy and slow

Except for one HUD mod that I liked but wanted to modify. So I forked it.

Skyrim has a LOT of items in it, and mods add more. You can end up having to dive into menus to eat food, drink potions, and switch weapons a lot. Hotkeys mods make this better. HUD + hotkey mods show you what you've got equipped. I wanted something specific.

This is what I wanted: to set up lists of things that I could cycle through by tapping a single button. So for the right hand I could swap from sword to bow to a spell to a gigantic axe back to sword by tapping one button for the right hand. If you've played a Souls game with a controller, that would be the right direction button. Same for left hand items, then potions and food, then shouts. This saves you menu-diving, which is kinda unimmersive.

Time to learn how to write mods

not just use them

a rapid intro to Creation Engine modding

the Creation Engine is designed for modding

Modding is essential to Bethesda's own workflow.

— The games are developed with the same tool they distribute to modders.
— (They do hack out proprietary libraries.)
— Official DLCs use the same mechanisms as mods to update game data.

The engine itself has nowhere near the wow factor of Unreal Engine. It has all the basics, for sure, but it doesn't have high frame rates or raytracing. Bethesda made all of its trade-offs differently.

the game data is designed for overlays

— "form" is the unit of game data: a sound, an armor component, a skill, etc

— form types are implemented in the game engine

— plugins define new form records or override existing form records

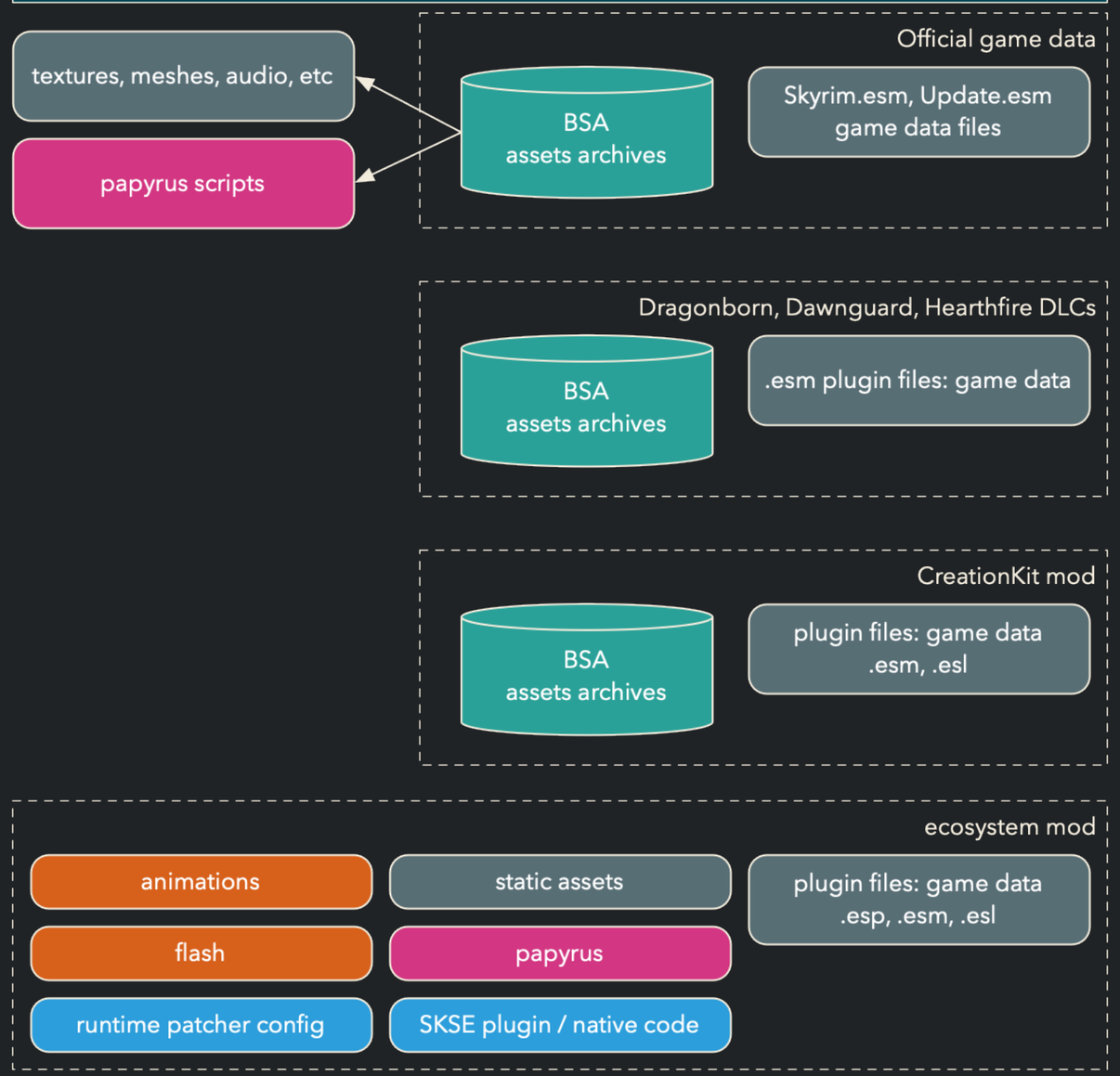
— game engine loads plugins in a user-controlled order

— **last-loaded** form record wins

— last-loaded asset archive wins; **loose files** > archives

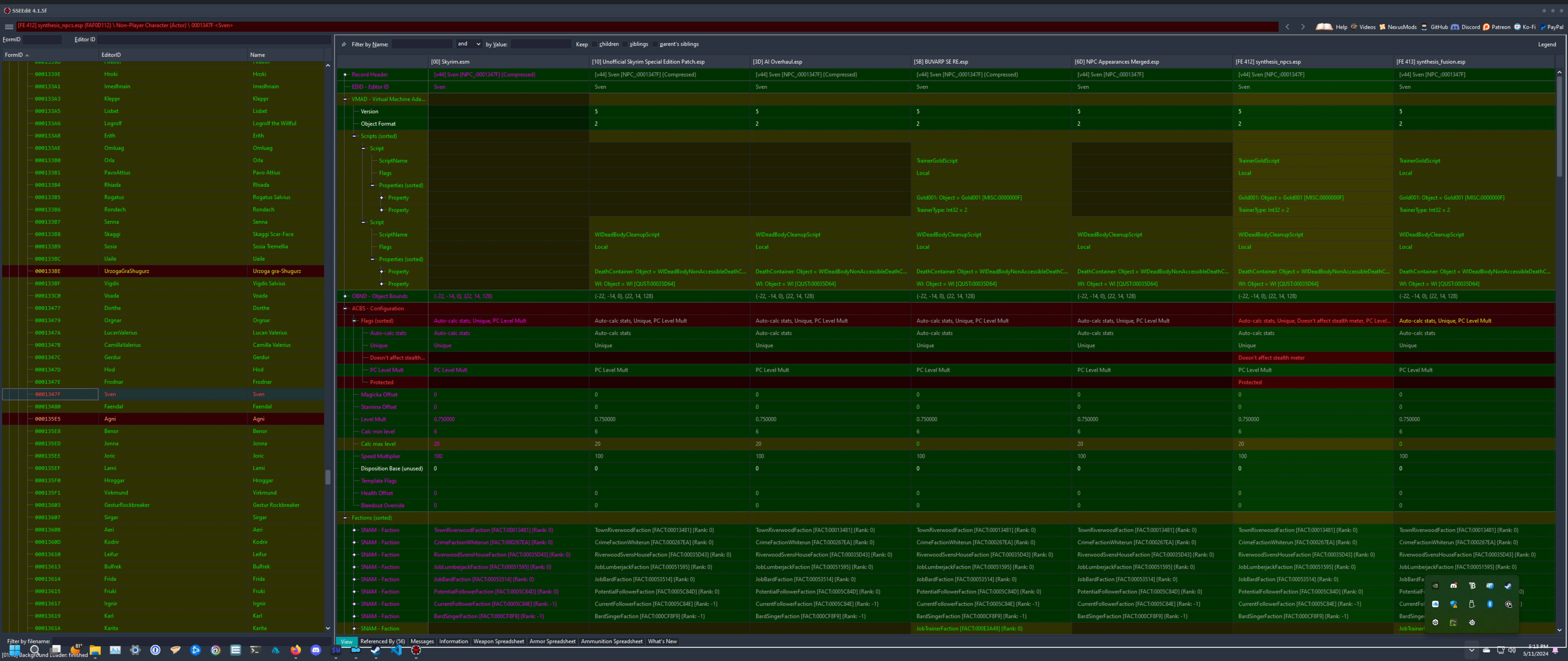
What's a mod?

A mod is a package of game assets linked together by one or more plugin files. Plugins are databases of game records that reference assets and other game records. Static assets might be packaged in an archive or left loose in the game's data directory.



the file formats

— slowly evolving for 20 years
— fully reverse-engineered
— community toolset is less buggy than Bethesda's provided tools
— no seriously: `xEdit` fixes problems with plugin data generated by the CreationKit
— complex workflows are supported


SSEEdit 4.1.5f
[FE 412] synthesis_npcs.esp (FAF0D112) \ Non-Player Character (Actor) \ 0001347F <Sven>
Help
Videos
NexusMods
GitHub
Discord
Patreon
Ko-Fi
PayPal
FormID
Editor ID
Filter by Name:
and
by Value:
Keep
children
siblings
parent's siblings
Legend
FormID
EditorID
Name
0001339E Hroki Hroki
000133A1 Imedhnain Imedhnain
000133A3 Kleppr Kleppr
000133A5 Lisbet Lisbet
000133A6 Logrolf Logrolf the Willful
000133AB Erith Erith
000133AE Omluag Omluag
000133B0 Orla Orla
000133B1 PavoAttius Pavo Attius
000133B4 Rhiada Rhiada
000133B5 Rogatus Rogatus Salvius
000133B6 Rondach Rondach
000133B7 Senna Senna
000133B8 Skaggi Skaggi Scar-Face
000133B9 Sosia Sosia Tremellia
000133BC Uaile Uaile
000133BE UrzogaGraShugurz Urzoga gra-Shugurz
000133BF Vigdis Vigdis Salvius
000133C0 Voada Voada
00013477 Dorthe Dorthe
00013479 Orgnar Orgnar
0001347A LucanValerius Lucan Valerius
0001347B CamillaValerius Camilla Valerius
0001347C Gerdur Gerdur
0001347D Hod Hod
0001347E Frodnar Frodnar
0001347F Sven Sven
00013480 Faendal Faendal
000135E5 Agni Agni
000135E8 Benor Benor
000135ED Jonna Jonna
000135EE Joric Joric
000135EF Lami Lami
000135F0 Hroggar Hroggar
000135F1 Virkmund Virkmund
00013603 GesturRockbreaker Gestur Rockbreaker
00013607 Sirgar Sirgar
0001360B Aeri Aeri
0001360D Kodrir Kodrir
00013610 Leifur Leifur
00013613 Bulfrek Bulfrek
00013614 Frida Frida
00013615 Fruki Fruki
00013617 Irgnir Irgnir
00013619 Karl Karl
0001361A Karita Karita
[00] Skyrim.esm
[10] Unofficial Skyrim Special Edition Patch.esp
[3D] AI Overhaul.esp
[5B] BUVARP SE RE.esp
[6D] NPC Appearances Merged.esp
[FE 412] synthesis_npcs.esp
[FE 413] synthesis_fusion.esp
Record Header
[v44] Sven [NPC_:0001347F] {Compressed}
[v44] Sven [NPC_:0001347F]
EDID - Editor ID
Sven
VMAD - Virtual Machine Ada...
Version
5
Object Format
2
Scripts (sorted)
Script
ScriptName
TrainerGoldScript
Flags
Local
Properties (sorted)
Property
Gold001: Object = Gold001 [MISC:0000000F]
TrainerType: Int32 = 2
WIDeadBodyCleanupScript
DeathContainer: Object = WIDeadBodyNonAccessibleDeathC...
WI: Object = WI [QUST:00035D64]
OBND - Object Bounds
(-22, -14, 0), (22, 14, 128)
ACBS - Configuration
Flags (sorted)
Auto-calc stats, Unique, PC Level Mult
Auto-calc stats, Unique, Doesn't affect stealth meter, PC Level...
Auto-calc stats
Unique
Doesn't affect stealth...
Doesn't affect stealth meter
PC Level Mult
Protected
Magicka Offset
Stamina Offset
Level Mult
0.750000
Calc min level
6
Calc max level
20
Speed Multiplier
100
Disposition Base (unused)
Template Flags
Health Offset
Bleedout Override
Factions (sorted)
SNAM - Faction
TownRiverwoodFaction [FACT:00013481] {Rank: 0}
CrimeFactionWhiterun [FACT:000267EA] {Rank: 0}
RiverwoodSvensHouseFaction [FACT:00035D43] {Rank: 0}
JobLumberjackFaction [FACT:00051595] {Rank: 0}
JobBardFaction [FACT:00053514] {Rank: 0}
PotentialFollowerFaction [FACT:0005C84D] {Rank: 0}
CurrentFollowerFaction [FACT:0005C84E] {Rank: -1}
BardSingerFaction [FACT:000CF8F9] {Rank: -1}
JobTrainerFaction [FACT:000E3A49] {Rank: 0}
Filter by filename:
[01:10] Background Loader: finished
View
Referenced By (56)
Messages
Information
Weapon Spreadsheet
Armor Spreadsheet
Ammunition Spreadsheet
What's New
81°
5:13 PM
5/11/2024

ActionScript

— UI is all Scaleform
— this is a custom Flash engine designed for embedding in games
— FromSoft games also use it, even though it's been dead for a few years
— limited to only UI use cases and a PITA to work with

Papyrus scripts

— Bethesda's scripting language with hooks into the engine

— it's compiled but...

— it's pretty slow

— it hooks into the game engine to affect game state

— the game itself is implemented in papyrus

Fix and extend via reverse engineering

Game reverse engineering has a long and storied history.

SKSE, F4SE, SFSE script extenders

The "script extender" series of mods originally extended Papyrus by hooking into more game functions. It's now important as a native-code plugin loader. It also provides reliable ways for mods to store save data alongside game data.

This is the basis of UI replacement mods like SkyUI. Todd Howard says SkyUI is his favorite Skyrim mod, so Bethesda knows this exists.

RE projects are organized and professional

— they're serious C++ experts
— with a side helping of RE & x86 assembly experience
— RE tools in use: mostly IDA, some Ghidra (the free NSA RE tool)
— they do work to reduce their own toil

Before Starfield turned out to be disappointing, with a fundamentally flawed engine, the RE community had made and executed on a plan to coordinate on working on archive format changes, plugin format changes, new form records, and so on. A working script extender was available in record time. The community also unlocked plugin loading, which allowed modding to take place immediately. (Bethesda still has not released the Starfield Creation kit. The RE community believes this is because they figured out how badly they'd broken modding with this engine release via form item layering screwups.)

example: Address Library

— designed to make coping with engine updates easier
— changed vtable offsets are RE-ed at least partially with tools like addresslibgen
— the offset library mod is updated
— mod authors don't have to do anything
— players need download only one update

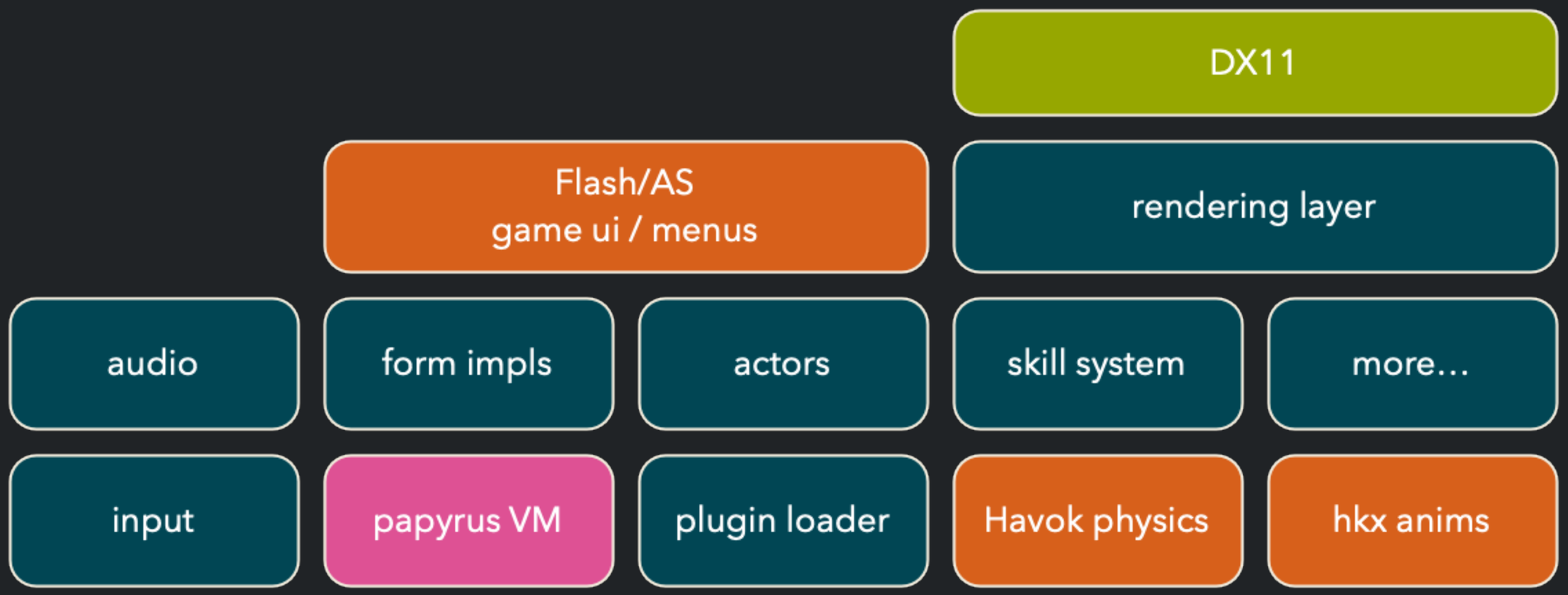
During a period when Skyrim was getting frequent releases, the RE community invented a way for SKSE plugin mods to look up game offsets on the fly, from a library that provides offsets for a number of versions.

CommonLibSSE-NG aka CLib

— a reverse-engineered C++ library
— everything we understand about the game engine and its classes
— with tools for offset look-up
— requires SKSE at runtime, not at compile time--completely separate code & far more extensive
— supports all known game builds

Skyrim / Creation Engine

Bethesda's internal engine, designed to support their game concepts. Descendent of the Gamebryo engine. Orange indicates tech Bethesda licenses that they do not support in modding.



If a block diagram looks kinda like this...

Reverse-engineering

Skyrim's engine is not fully reverse-engineered, but the community knows enough to have extensively modified its behavior. Bugs are fixed, rendering is given modern enhancements, gameplay features are extended and hooked into. Animations in particular have been modernized, enhancing combat as well as making NPCs more life-like.

This is which parts of the game engine have been reverse engineered. Anything a modder wanted to replace or fix.

Back to my forked HUD mod

— it's a CommonLibSSE mod: C++
— it uses **imgui** to draw UI not Flash
— this is why it was fast
— but other design decisions made it buggy
— (and I hated how its customization worked...)

I can fix it, right?

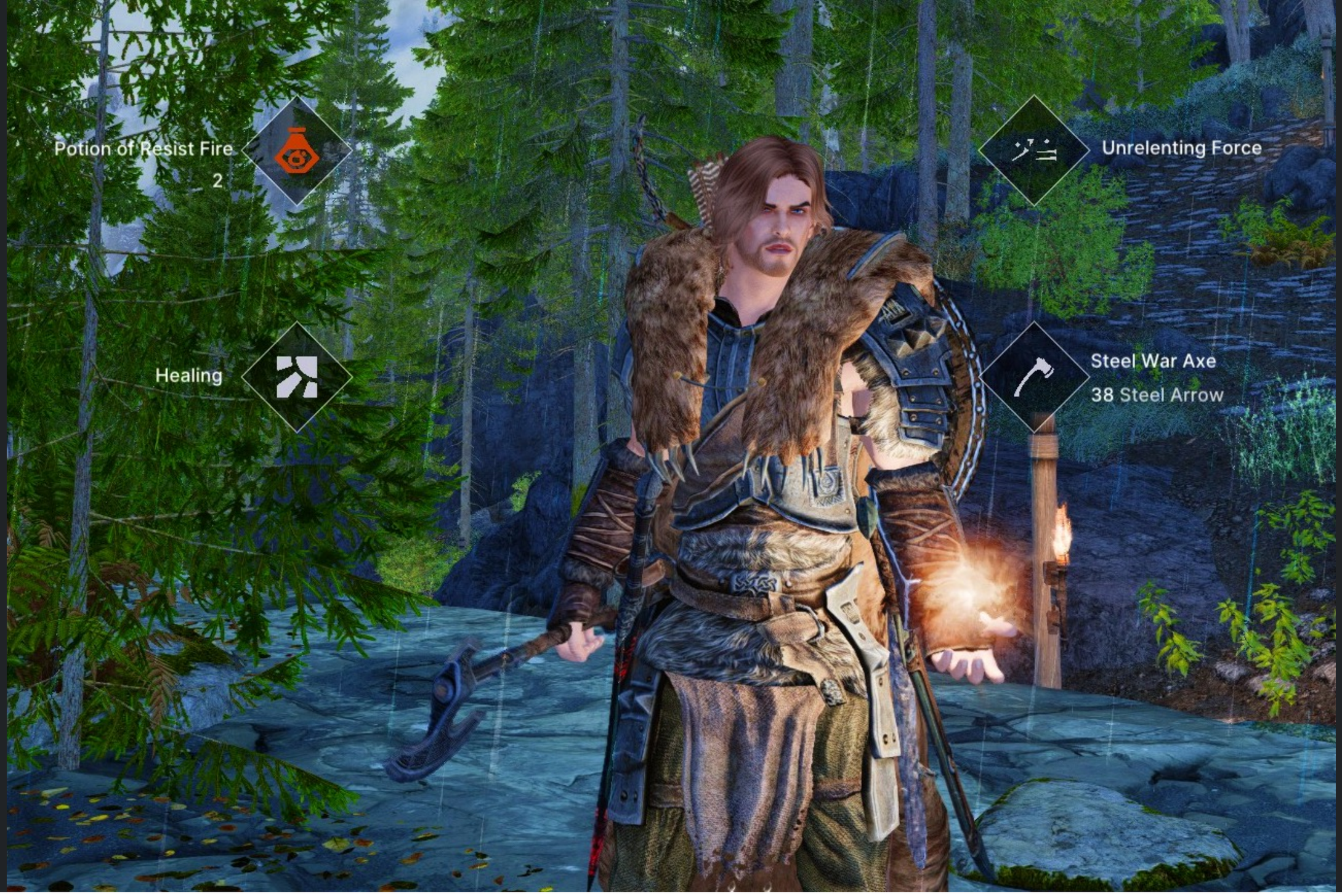
a learning project

— modern C++
— Windows development
— imgui
— Skyrim itself
— could I learn about Rust/C++ interop?

how hard can it be?

cue A-Team theme song


2
4
6
5
Unrelenting Force
Sarta
24 Best Health
Life Pulse


Potion of Resist Fire
2
Healing
Unrelenting Force
Steel War Axe
38 Steel Arrow

So how hard was it?

It was harder than I thought, but not in the ways I thought it would be hard.

— Windows strings were a pain until I understood them
— Skyrim internals are not always sensible
— mod interactions made the problem far more complex than expected
— had lots of bugs for a long time until it fell together
— players asked for far more features than I thought I

understanding Skyrim itself

— SKSE and CLib are undocumented
— CLib is more methodical but much larger
— found my favorite C++ mod authors and read all of their code
— $%@! extra data & inventory what the what now?
— a lot of trial and error

Windows development

This was a surprising challenge to me. I had not done a lot of development on Windows through my career, so I had to learn how to cope with a new ecosystem.

— `CMake` - I had to write my own CMake files because I was linking to Rust
— `vcpkg` - for C++ libraries
— Visual Studio & `msvc` flag conventions

Redmond is a different developer country. I've had an adjustment period here to the JVM, which is a different ecosystem with its own local customs and trivia, but the adjustment to Redmondland was bigger.

Command-line workflows

— developer tools are better on MacOS & Linux (familiarity bias?)

— found a Windows terminal app that isn't awful

— WSL saved me in the end, because I had bash

— forced back to VSCode instead of my preferred editor

— all the Rust dev can be done on MacOS

— Rust tests implemented to avoid needing to link to CLib

It was a pretty bad adjustment, to be honest. I really loathe editing code in Visual Studio, which is amazing to me because of how many people use these tools worldwide.

```
# Set the version in four (!!) places and tag the repo to match. Requires bash.
[unix]
tag VERSION:
    #!/usr/bin/env bash
    sed="sed"
    ignored=$(which gsed)
    if [ $? == 0 ]; then
        sed="gsed"
    fi
    set -e
    tomato set package.version {{VERSION}} Cargo.toml
    # update the version header for the plugin
    $sed -i -e 's/set(VERSION [0-9][0-9]*\.[0-9]*\.[0-9]*\(\.[0-9]*\)/set(VERSION {{VERSION}}\1/' CMakeLists.txt
    # update the lock file
    cargo check --target-dir target/wsl-check
    jq '."version-string" = "{{VERSION}}"' vcpkg.json > vcpkg.tmp
    mv vcpkg.tmp vcpkg.json
    # update the fomod version
    iconv -f UTF-16LE -t UTF-8 installer/fomod/info.xml >installer/fomod/info_utf8.xml
    $sed -i -e 's/<Version>[0-9][0-9]*\.[0-9]*\.[0-9]*<\/Version>/<Version>{{VERSION}}<\/Version>/' installer/fom
    iconv -f UTF-8 -t UTF-16LE installer/fomod/info_utf8.xml >installer/fomod/info.xml
    rm installer/fomod/info_utf8.xml
    git commit CMakeLists.txt Cargo.toml Cargo.lock installer/fomod/info.xml vcpkg.json -m "v{{VERSION}}"
    git tag "v{{VERSION}}"
    echo "Release tagged for version v{{VERSION}}"
```

Just, which I've introduced here, was an important part of my workflow. You can provide different versions of a single recipe for diff environments. I automated everything I could automate. CMake could handle all of this, I think, but I am so much more comfortable with bash that I gave in and did all of this with WSL.

Strings are (often) annoying

— Rust is natively `utf-8`; `String` type and `&str` borrowed data type

— `std::ffi::{OsString, OsStr}` are host OS equivalents

— Windows OS strings are wide chars (16 bytes); file paths!

— Skyrim uses `iso-8859-1` internally

— Scaleform/Flash translation strings are `UCS-2`

— player crashes with encodings until I learned all this

String handling in lower-level languages can be tricky because you have to think about ownership of the byte structure behind the string. That was not a problem in this project, but encodings were. Yes, I said UCS2-2. This is a fixed-width 16 byte encoding that is pretty old.

Modern C++ > older C++

— all mod cons in the std library
— ergonomics are better
— official documentation is terrible
— null-pointer access still plagued me
— no contract about when a pointer can be null in CLib

Better but not great. 90% of my crashes were null pointers. The remainder was a combo of invalid utf-8 in Rust plus a couple of stray `todo!()` macros + one unwise `unwrap()`.

So how is the Rust/C++ interop story?

Exactly as good as advertised.

— near zero cost
— tools are mature and well-tested
— bindgen is one approach
— cxx was the one I used

`cxx` codegen for a safer subset of interop

— safe in the Rust sense: no games with memory; compiler can check

— generates bindings for the functions you want on either side

— C++ can call Rust & vice versa

— not as complete as `bindgen`

— `char*` not supported, for example

— but great interop for utf8 strings and vectors

lib.rs defines the bridge

Let's look at some code briefly!

show lib.rs here

plugins could be 100% Rust

Theoretically, I could reduce the C++ in my mod to just Rust bindings to CommonLibSSE-NG and the SKSE plugin entry point.

— **bindgen** is the path here
— CLib is a moving target
— I got a job instead of diving into this
— gotta have a budget for graphics cards, you know?

This project moved from 0% Rust to its current 68.5% Rust.

a few major reworks along the way

I found myself having persistent trouble with some design decisions.

— careful flowcharting of what a cycle advance does when two-handed weapon is equipped; could handle grip-switch mods easily once I got this right

— loading and rasterizing svgs

— categorizing the many, many mod-added items

nanosvg → resvg

— C++ to Rust
— but really, minimal spec support to extensive spec support
— implemented lazy-loading at the same time
— no real memory pressure, but I like being thrifty

People kept reporting bugs with svgs that worked in Flash but didn't in Soulsy. It was always because the svg was actually a PNG in a base64 trenchcoat, which is allowed by the spec but not supported by nanosvg.

object categorization rework

— mod-added objects weren't getting the right icons
— both the C++ and Rust sides of this were awful
— so I started using object keywords to categorize
— the mod Object Categorization Framework adds keywords to thousands of items
— for a few mods, I distribute keywords myself using Keyword Item Distributor

keep it fast

Two facets of performance:

— perceived input latency
— frame rate cost of the render loop

Writing this as a native-code SKSE plugin obliterated the input latency problem. Imgui means it can be visually fast. This is why I started with the HUD mod I picked.

the render loop has to be tight

— reduced branching in the tight loop
— made all layout decisions at layout load time where possible
— internal structures are quite different from the serialized player-editable structures as a result
— mod precalculates and load data for items that will be drawn

This was a lot of fun to do. Not the kind of programming I get a chance to do much of these days.

Supporting cli projects

I wrote Rust cli tools to help:

— `md2nexus`: a tool converting markdown to NexusMod's BBCode (which is awful)
— `mcm-meta-helper`: a helper tool for managing translation string files
— `priority-sync`: a tool for syncing mod order across profiles (for testing with many setups to repro player problems)

All of these are Rust cli tools. The third one is notable because there is a python plugin for Mod Organizer 2 that does this, but it's extremely slow and buggy. I wrote a ridiculously fast tool that does something simpler but easier to make reliable. All of these are in the `ceejbot` github, btw, if you want to see how nice writing CLIs in Rust is, thanks to the `clap` crate.

the features I wanted

— shouts & powers cycle
— left item cycle: shields, torches, weapons, spells, scrolls
— right item cycle: weapons, spells, two-handers, scrolls
— potion & poison cycle
— a layout defined in a human-readable text file
— minimal to no settings

the features I implemented

— all of what I wanted! plus!
— too many settings
— customizable layouts in TOML, with hot reload in-game
— two custom icon sets
— smart potion selection
— equipment set management
— full controller support

got to use nearly my full skillset

— product design
— UX design & writing
— user docs + programmer docs
— programming with attention to perf & memory use
— single-process stateful software architecture (not dist systems for once)

I have a very particular set of skils, skils I have acquired over a very long career. Skills that make projects like this a joy for me.

Incredibly fulfilling work

— Full control of design but took lots of player input.
— Heard player pain immediately.
— Heard player happiness immediately!
— Could have zero tolerance for tech debt.
— Could meet my own quality standards.
— Feature development was extremely rapid when I got the design right.
— Got exactly the HUD mod I wanted.

My own standards for software are much higher than that of most of the startups I've worked for. Most often the ticking clock of company debt means that speed is the only thing that matters and even that is evaluated in a very shallow way. Here I could realize that I'd designed myself into a corner that made feature work hard, and then fix it by rewriting. In the end, this is of course faster than not fixing tech debt, but persuading a VC-funded company to do this is always absurdly hard. It's like persuading a video game company that crunch is self-destructive-- the culture doesn't let them learn.

What I learned

— so much! Skyrim, modern C++, that I can still learn fast when I need to

— Rust > C++ (but they go well together)

— Windows needs a great text editor

— Skyrim did in fact fix my bad mood

Questions?

github.com/ceejbot/soulsy