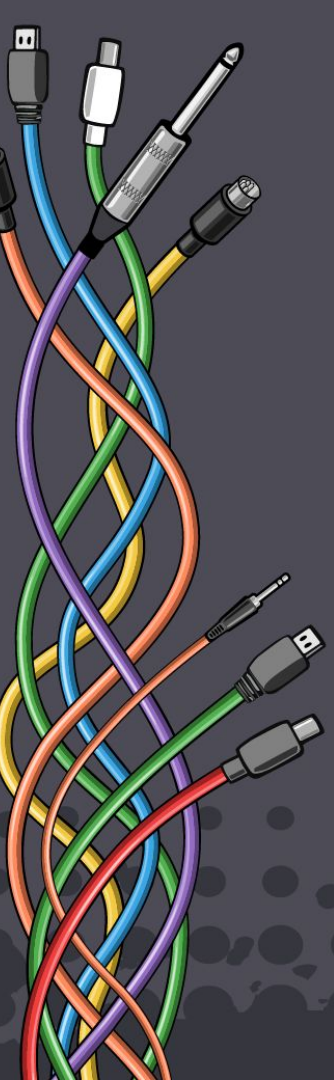


HACKING HANDHELDS FOR CREATIVE AUDIO

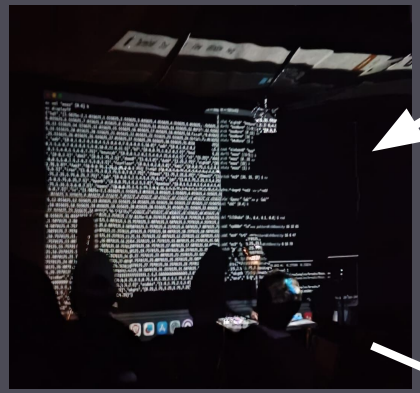
BUILDING MUSIC APPLICATIONS FOR THE NEW NINTENDO 3DS

LEONARDO FOLETTO

ADC25 Bristol

Intro

- Independent audio software developer
- Musician
- Live coder
- Audio tinkerer
- Gamer



What we'll cover today

- **The "Why":**
  The creative power of constraints and the legacy of handheld music.
- **The "What":**
  The New 3DS as a target platform.
- **The "How":**
  The tools and techniques for building audio apps on the 3DS.
- ***Soir* :** building a groovebox as a case study.

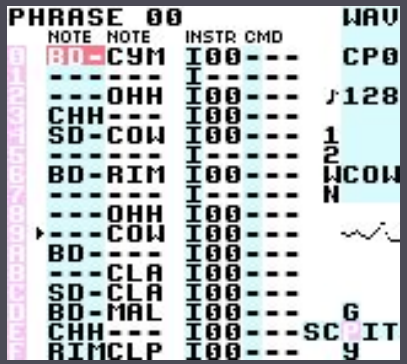
"Why" : The Creative Case

The Creative Case for Old Gaming Handhelds

Why not just use a laptop? Why bother with a 10-year-old handheld?

- **The "Paradox of Choice":** Infinite options can be paralyzing.
- **The Power of Constraints:**
  - ***Focus*:** A dedicated device for *one task* removes distractions.
  - ***Tactility*:** Physical buttons, sliders, and touch screen feel different than mouse and keyboard.
  - ***Creativity*:** Limits on CPU, memory, and controls force you to make deliberate, creative decisions.

A Legacy of Gaming Consoles Music



LSDJ (Little Sound Dj)

- **Platform:** 8-bit Game Boy (1989)
- A fully featured tracked taking advantage of the Game Boy DSP
- One of the main factors that lead to the birth of the **chiptune** scene
- One of the first examples of a toy being turned into a professional musical instrument

Music 2000

- **Platform:** Sony Playstation (1994)
- Loop based sequencer with sampling
- Brought music production to the living room, beloved by **drum and bass**, **dubstep** and **jungle** artists

A Legacy of Handheld Music - The DS Revolution

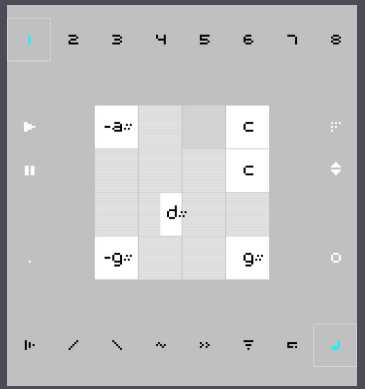
Korg DS-10

- Commercial emulation of a **Korg MS-10**
- Validated the DS as a viable music platform
- First and most widely available of the Korg DS / 3DS app, still played by many people today

NitroTracker

- A **Homebrew** tracker, inspired by PC trackers like **FastTracker II**
- Showcased the power of Homebrew scene
- Still supported and actively developed in the **NitrousTracker** fork

A Legacy of Handheld Music - modern tools



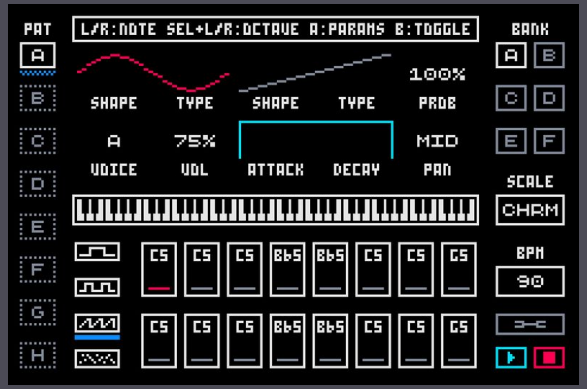
M8 Tracker

- 8-tracks, Teensy based tracker and synthesizer
- including FM, virtual analog, sample playback, waveform synthesis and MIDI output.
- Inspired by LSDJ

Nanoloop

- Minimalist electronic music software
- Sequencer and software synthesizer for GB, GBA, iOs and Android

A Legacy of Handheld Music - Novel GBA/3DS software



Stepper

- Elektron inspired step sequencer for the GBA
- 4 DMG sound channels, 6 banks of 8 patterns

Noise Commander

- Sample based groovebox and audio sequencer and DJ-Tool for the Nintendo 3DS
- Two projects can be loaded and played at the same time on 2 decks

"What" & "How": Hacking the New 3DS

Why the New 3DS?

- **Mature Homebrew Scene:** A fully "solved" platform that is easy, safe, and free to "hack."
- **Unique I/O:**
  - Dual Screens (one is a resistive touchscreen).
  - D-Pad, 4 face buttons, 2 shoulder buttons.
  - Circle Pad + C-Stick + ZL/ZR buttons.
- **Hardware Features:** Stereo speakers, a *real* headphone jack, and (optional) 3D audio.
- **Good battery life:** ~6 hours of usage, can lasts for days while sleeping.
- **Accessible:** Widely available second-hand.

Enter 3DS Homebrew

What is "Homebrew"?

A term for running user-created, unauthorized software on a "locked" device.

How does it work?

- **Custom Firmware (CFW):** We use a program like Luma3DS, which is loaded from the SD card at boot.
- **Patching:** CFW patches the official system to allow running unsigned applications.
- **The Homebrew Launcher:** The main app used to find and launch .3dsx files from your SD card.
- **The Guide:** 3ds.hacks.guide is the community-standard, safe, and modern guide to getting this set up.

Is This... Legal? [A Note on Homebrew]

The 3DS is a “dead” platform (since March 2023): it is no longer officially supported from Nintendo, the official shop and servers are closed.

Installing Homebrew voids your warranty, but it is generally allowed and protected by law in many regions.

Legal 😎

- Running Homebrew / Homebrew Apps
- Developing and sharing Homebrew software

Not Legal 😧

- Playing pirated games
- Distributing or selling unlicensed software

Disclaimer: I am not a lawyer, and this is not legal advice.

Under the Hood: Why the ✨*"New"*✨ 3DS

Feature	Old 3DS / 2DS (~2011)	New 3DS / New 2DS XL (~2014)	Why It Matters for Audio
CPU	2-core ARM11 @ 268 MHz	**4-core ARM11 @ 804 MHz**	**~3x speed + 2 extra cores.** Essential for real-time synthesis & FX.
RAM	128 MB	**256 MB**	**Double the RAM.** Allows for longer samples, more complex patches, bigger audio buffers.
L2 Cache	-	**Yes (2MB, Shared)**	Faster CPU access. Reduces latency and audio glitches.
Controls	1 Circle Pad	1 Circle Pad + **C-Stick + ZL/ZR**	More physical controls for parameter mapping and complex UX use cases.

The Secret Weapon: The Audio DSP

The 3DS has a non-ARM co-processor, a **CEVA Teaklite** DSP chip. Developers can take advantage of it in two ways:

High-level path:

- **Libctru** loads Nintendo's optimized firmware onto the DSP (**ndspInit**)
- Gives access to a 24-channel hardware mixer, filters, interpolation

Low-level path:

- Develop, build and load your own custom DSP firmware (**ndspLoadComponent**)
- Complex algorithms can be run at zero CPU cost

The Developer's Toolkit

A free and open-source development stack.

- **devkitPro**: install manager for the whole toolchain.
- **devkitARM**: C/C++ compiler that targets the 3DS's ARM processors.
- **Libctru**: core 3DS library. Gives access to graphics (**citro2D** / **citro3D**), audio (**ndsp**), filesystem, controller inputs.

The Development Loop

- **Code and Build:**
  - Develop in **C** or **C++** using **libctru / devkitARM**.
  - Build with **make.**

- **Formatting:**
  - Using **clang-format**

- **Testing:**
  - Unit tests with **ctest** for the 3DS independent code.
  - On-device for testing the audio (hardware required, none of the emulators tested seem to behave appropriately).
- **Debugging:**
  - Using tools within devkitpro such as **luma3ds_exception_dump_parser** to parse and convert into readable format crash dumps.


Soir:
building a groovebox for N3DS

Case Study: *Soir*

A 5-tracks* loosely Elektron inspired groovebox for the New Nintendo 3DS, featuring synthesised and sampled instruments.

Top Screen:

- Main sequencer window, modal dialogs for active menu edits

Bottom screen:

- Params, settings, user interactions

*(to be expanded to 12-tracks in the future)

App Architecture

Main Thread:

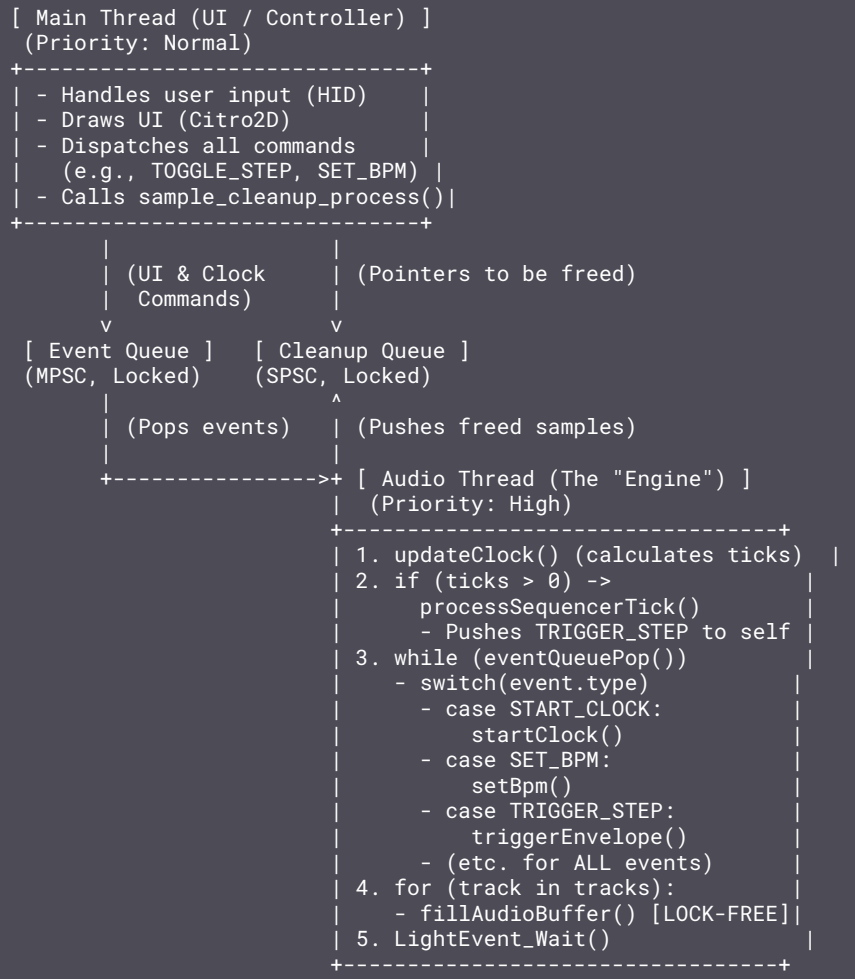
- Read controls
- Dispatch Commands

Audio Thread:

- Processes command queue
- Handles real-time logic

MPSC Event Queue / SPSC Cleanup Queue:

- Command bus for the entire application
- Cleanup pointers to files (Samples on SD card)



Main Thread: Controls / UI 🎛️

Pure controller (in the MVC sense).
It takes care of

- Rendering the screen
- Capturing user input
- Dispatching Events
- Memory cleanup

```c
while (aptMainLoop()) {
    hidScanInput();

    u64 now   = svcGetSystemTick();
    u32 kDown = hidKeysDown();
    u32 kHeld = hidKeysHeld();

    sessionControllerHandleInput(&ctx, kDown, kHeld, now, &should_break_loop);

    if (should_break_loop) {
        break;
    }

    C3D_FrameBegin(C3D_FRAME_SYNCDRAW);
    C2D_TargetClear(topScreen, CLR_BLACK);
    C2D_SceneBegin(topScreen);

    drawMainView(tracks, app_clock, selected_row, selected_col, screen_focus);

    switch (session.main_screen_view) {
    case VIEW_MAIN:
        break;
    case VIEW_SETTINGS:
        drawClockSettingsView(app_clock, selected_settings_option);
        break;
    case VIEW_QUIT:
        drawQuitMenu(quitMenuOptions, numQuitMenuOptions, selected_quit_option);
        break;
    case VIEW_STEP_SETTINGS_EDIT:
        drawStepSettingsEditView(&tracks[selected_row - 1], &g_editing_step_params,
                                 selected_step_option, selected_adsr_option, &g_sample_bank);
        break;
    default:
        break;
    }
```

Event Queue ⛓ Cleanup Queue

MPSC Event Queue Roles:

- **Decoupling** controls and audio thread.
- **Serialization** of all state changes.

All state modifications are pushed into the queue and consumed by the audio thread in order, preventing possible bugs related to race conditions and priority inversion.

SPSC Cleanup Queue Roles:

- **Memory Cleanup** freeing pointers to samples, envelopes etc.

Audio Thread: the Engine 🔊

The single high-priority engine handling all real-time logic.
Triggered by **ndspSetCallback**, it operates as such:

- Checks for Ticks and updates clock.
- Runs sequencers as needed.
- Processes command queue of events such as:
  - **TRIGGER_STEP** events (from itself).
  - **UPDATE_STEP** events (from the UI).
  - **TOGGLE_STEP**, **SET_MUTE**, etc. (from the UI).
  - **SET_BPM**, **START_CLOCK**, etc. (from the UI).
- Generates audio.
- Sleeps until audio hardware requests more data.

A taste of ndsp

```c
void initializeTrack(Track *track, int chan_id, InstrumentType instrument_type, float rate,
                     u32 num_samples, u32 *audio_buffer) {
    track->chan_id         = chan_id;
    track->instrument_type = instrument_type;
    track->audioBuffer     = audio_buffer;
    track->is_muted        = false;
    track->is_soloed       = false;
    track->fillBlock       = false;
    track->sequencer       = NULL;
    track->instrument_data = NULL;
    track->volume          = 1.0f; // Initialize volume
    track->pan             = 0.0f; // Initialize pan

    // Initialize default parameters
    track->default_parameters = linearAlloc(sizeof(TrackParameters));
    if (track->default_parameters) {…

    ndspChnReset(track->chan_id);
    ndspChnSetInterp(track->chan_id, NDSP_INTERP_LINEAR);
    ndspChnSetRate(track->chan_id, rate);
    ndspChnSetFormat(track->chan_id, NDSP_FORMAT_STEREO_PCM16);

    memset(track->mix, 0, sizeof(track->mix));
    track->mix[0] = 1.0;
    track->mix[1] = 1.0;
    ndspChnSetMix(track->chan_id, track->mix);

    track->filter.id            = chan_id;
    track->filter.filter_type   = NDSP_BIQUAD_NONE;
    track->filter.update_params = false;
    track->filter.cutoff_freq   = 1760.f; // some default

    memset(track->waveBuf, 0, sizeof(track->waveBuf));
    track->waveBuf[0].data_vaddr = &track->audioBuffer[0];
    track->waveBuf[0].nsamples   = num_samples;
    track->waveBuf[1].data_vaddr = &track->audioBuffer[num_samples];
    track->waveBuf[1].nsamples   = num_samples;

    fillBufferWithZeros(track->audioBuffer, num_samples * NCHANNELS * 2);

    ndspChnWaveBufAdd(track->chan_id, &track->waveBuf[0]);
    ndspChnWaveBufAdd(track->chan_id, &track->waveBuf[1]);
}
```

```c
initializeTrack(&tracks[2], 2, OPUS_SAMPLER, OPUSSAMPLERATE, OPUSSAMPLESPERFBUF, audioBuffer2);
```

```c
initializeTrack(&tracks[1], 1, FM_SYNTH, SAMPLERATE, SAMPLESPERBUF, audioBufferFM);
```

```c
#define SAMPLERATE 32000
#define SAMPLESPERBUF (SAMPLERATE * 120 / 1000)
#define NCHANNELS 2

#define OPUSSAMPLERATE 48000
#define OPUSSAMPLESPERFBUF (OPUSSAMPLERATE * 120 / 1000)
```

```c
 *    - 0: Front left volume.
 *    - 1: Front right volume.
 *    - 2: Back left volume:
 *    - 3: Back right volume:
 *    - 4..7: Same as 0..3, but for auxiliary output 0.
 *    - 8..11: Same as 0..3, but for auxiliary output 1.
 */
void ndspChnSetMix(int id, float mix[12]);
```

Putting it all together..

Clock and session setup:

```
int main(int argc, char **argv) {

    ndspInit();
    ndspSetOutputMode(NDSP_OUTPUT_STEREO);

    // CLOCK ////////////////////////
    MusicalTime mt        = { .bar = 0, .beat = 0, .deltaStep = 0, .steps = 0, .beats_per_bar = 4 };
    Clock       cl        = { .bpm             = 120.0f,
                              .last_tick_time   = 0,
                              .time_accumulator = 0,
                              .ticks_per_step   = 0,
                              .status           = STOPPED,
                              .barBeats         = &mt };
    Clock      *app_clock = &cl;

    SessionContext ctx = { .session                    = &session,…
    setBpm(app_clock, 127.0f);
```

Putting it all together..

Tracks and sequencers setup:

```
// TRACK 0 (SUB_SYNTH) ////////////////////////////////////////
audioBuffer1 = (u32 *) linearAlloc(2 * SAMPLESPERBUF * BYTESPERSAMPLE * NCHANNELS);
if (!audioBuffer1) {
    ret = 1;
    goto cleanup;
}
initializeTrack(&tracks[0], 0, SUB_SYNTH, SAMPLERATE, SAMPLESPERBUF, audioBuffer1);

osc = (PolyBLEPOscillator *) linearAlloc(sizeof(PolyBLEPOscillator));      You, 3 wee
if (!osc) {
    ret = 1;
    goto cleanup;
}
*osc = (PolyBLEPOscillator) { .frequency   = 220.0f,
                              .samplerate  = SAMPLERATE,
                              .waveform    = SQUARE,
                              .phase       = 0.,
                              .pulse_width = 0.5f,
                              .phase_inc   = 220.0f * M_TWOPI / SAMPLERATE };

env = (Envelope *) linearAlloc(sizeof(Envelope));
if (!env) {
    ret = 1;
    goto cleanup;
}
*env = defaultEnvelopeStruct(SAMPLERATE);
updateEnvelope(env, 20, 200, 0.6, 50, 300);

subsynth = (SubSynth *) linearAlloc(sizeof(SubSynth));
if (!subsynth) {
    ret = 1;
    goto cleanup;
}
*subsynth                 = (SubSynth) { .osc = osc, .env = env };
tracks[0].instrument_data = subsynth;
```

```
sequence1 = (SeqStep *) linearAlloc(16 * sizeof(SeqStep));
if (!sequence1) {
    ret = 1;
    goto cleanup;
}
trackParamsArray1 = (TrackParameters *) linearAlloc(16 * sizeof(TrackParameters));
if (!trackParamsArray1) {
    ret = 1;
    goto cleanup;
}
subsynthParamsArray = (SubSynthParameters *) linearAlloc(16 * sizeof(SubSynthParameters));
if (!subsynthParamsArray) {
    ret = 1;
    goto cleanup;
}
for (int i = 0; i < 16; i++) {
    subsynthParamsArray[i] = defaultSubSynthParameters();
    trackParamsArray1[i]   = defaultTrackParameters(0, &subsynthParamsArray[i]);
    sequence1[i]           = (SeqStep) { .active = false };
    sequence1[i].data      = &trackParamsArray1[i];
}
seq1 = (Sequencer *) linearAlloc(sizeof(Sequencer));
if (!seq1) {
    ret = 1;
    goto cleanup;
}
*seq1               = (Sequencer) { .cur_step               = 0,
                                    .steps                  = sequence1,
                                    .n_beats                = 4,
                                    .steps_per_beat         = 4,
                                    .instrument_params_array = subsynthParamsArray,
                                    .track_params_array     = trackParamsArray1 };
tracks[0].sequencer = seq1;
```

Demo Snippet

What's Next?

Upcoming Features:

- Improving and adding synth engines and the remaining tracks
- Insert Fxs, Aux Fxs
- Song Mode, Variable Pattern Lengths
- Multiplayer (networked performance)
- UX improvements and touch screen controls

Stretch Goals:

- Experiment with ndsp firmware algorithm development
- OSC controls for live coding?

Current Challenges:

- Optimizations and memory management
- Developing an intuitive and versatile user experience for live performance

Final Thoughts

- "Obsolete" hardware is a stable and accessible target for creative software development.
- The New 3DS offers unique I/O, controls and a surprising amount of power.
- The open-source toolchain (devkitPro, libctru) is mature for new 3DS ideas.

Find old tech you love and explore its creative potential! 💖

Thank You!

Get in touch:

You can find me **@leofltt** around the web.

(best way to reach me is on Mastodon -> https://merveilles.town/@leofltt/)

Try it out!

The app and source code will be available at
https://github.com/Leofltt/Soir

The slide deck can be found here:
https://tinyurl.com/hackinghandelds

Resources

- 3DS Custom Firmware (Required):
  - https://3ds.hacks.guide/
- Development Toolchain:
  - https://devkitpro.org/
- 3DS Library Docs:
  - https://libctru.devkitpro.org/
- 3DS Homebrew Wiki:
  - https://www.3dbrew.org/wiki/Main_Page