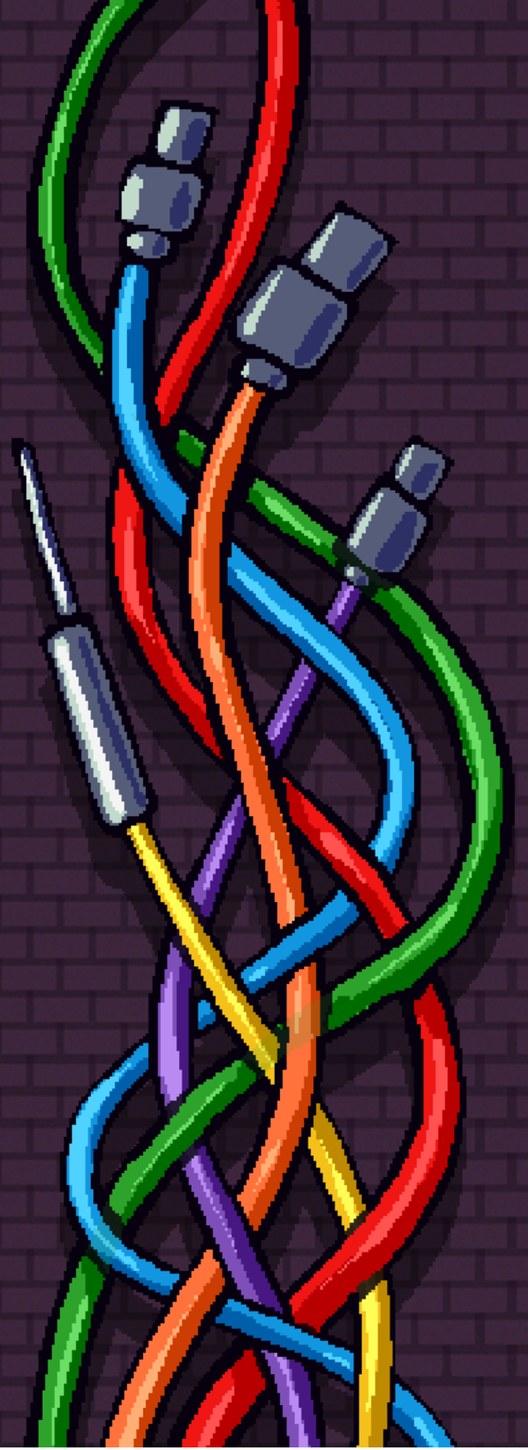
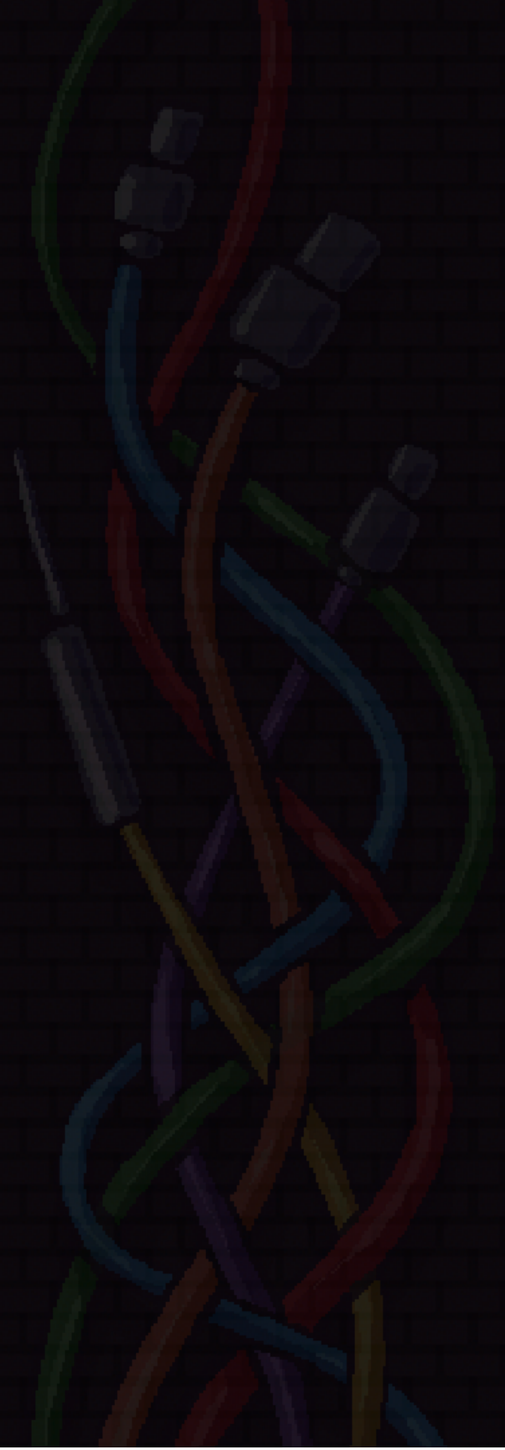


ADCx GATHER

MINIMALISTIC MUSIC COMPOSITION WITH C++

XYZZY



ABOUT ME

Apps for artists

EVOLUTION OF COMPUTER MUSIC

- Assembly - Chiptunes
- Trackers / MIDI
- DAW
- Procedural
- Neural

WHY C/C++ ?

- Threads
- No GC
- Accurate Timer
- Embedded Support - Teensy, Bela, ESP32, Arduino, Microbit
- Many libraries
- TUI

If you are curious about how DAW and low level operate

SAMPLE PLAYBACK USING miniaudio.h

```
int ticks_count = 0;
while(1) {
        int i = tickes_count % 16;
        if ((i == 1) || (i == 5) || (i == 9) ||  (i == 13)) {
                ma_engine_play_sound(&engine, "909Drum.wav", NULL);
        }
        if ((i == 5) ||  (i == 13)) {
                ma_engine_play_sound(&engine, "909Snare.wav", NULL);
        }
        setTimeout(time_per_tick);
        ticks_counter += 1;
    }
```

ALL ALONG THE WATCHTOWER

CPU

```
clock_gettime
```

AUDIO CLOCK

```
sampling speed: 48k samples / s, 1024 buffer size, 8/16/32 bit
processing time :  21.33 ms, 4kb
```

MIDI CLOCK

```
BPM: 120 quarter notes per minute
     2 quarter notes per second
     1 half note per second
     1 full note per 0.5 second
Time Signature: 1 bar = 4 quarter notes
Resolution: 24 (all), 240 (latest hardware), 960 (daw internal)
Resolution(Recommended): 16 - 31.25 ms, 32, 64
```

BASIC TIMER

```
std::atomic<bool> active{true};

void Timer::setInterval(auto function, int interval) {
    active = true;
    std::thread t([=]() {
        while(active.load()) {
            std::this_thread::sleep_for(std::chrono::milliseconds(interval));
            if(!active.load()) return;
            function();
        }
    });
    t.detach();
}
```

SEQUENCING

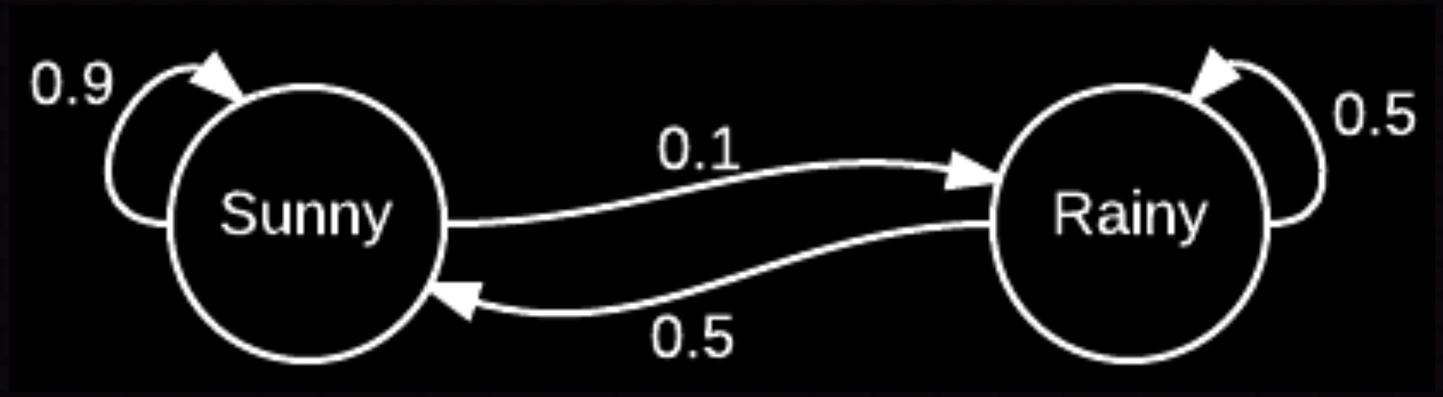
1. Discrete - Trigger
2. Continous - Automation
3. Periodic - LFO
4. Generic

```
1:[playMidi(70 pitch, loudly, 100ms), sleep(0.25), playMidi(77 pitch, softly,  400ms), sleep(0.5) ...]
2:[playWav("S1"), sleep(0.125), playWav(30ms), sleep(0.125)  ....]
```

MIDI WITH portmidi

```
int playMidi(int channel, int note, int duration, int volume) {
    channel += 143;
    PmEvent buffer[1];
    buffer[0].timestamp = Pt_Time() ;
    buffer[0].message = Pm_Message(channel, note, volume);
    Pm_Write(out, buffer, 1);
    t.setTimeout([=]() {
        channel += 127;
        PmEvent buffer[1];
        buffer[0].timestamp = Pt_Time() ;
        buffer[0].message = Pm_Message(channel, note, volume);
        Pm_Write(out, buffer, 1);
    }, duration);
}
```

GENERIC CLASSES FOR SEQUENCING



Anything that responds to a tick method.

1. Markov
2. External

```
   int randomMidi = (int)(drand48() * 128);
```

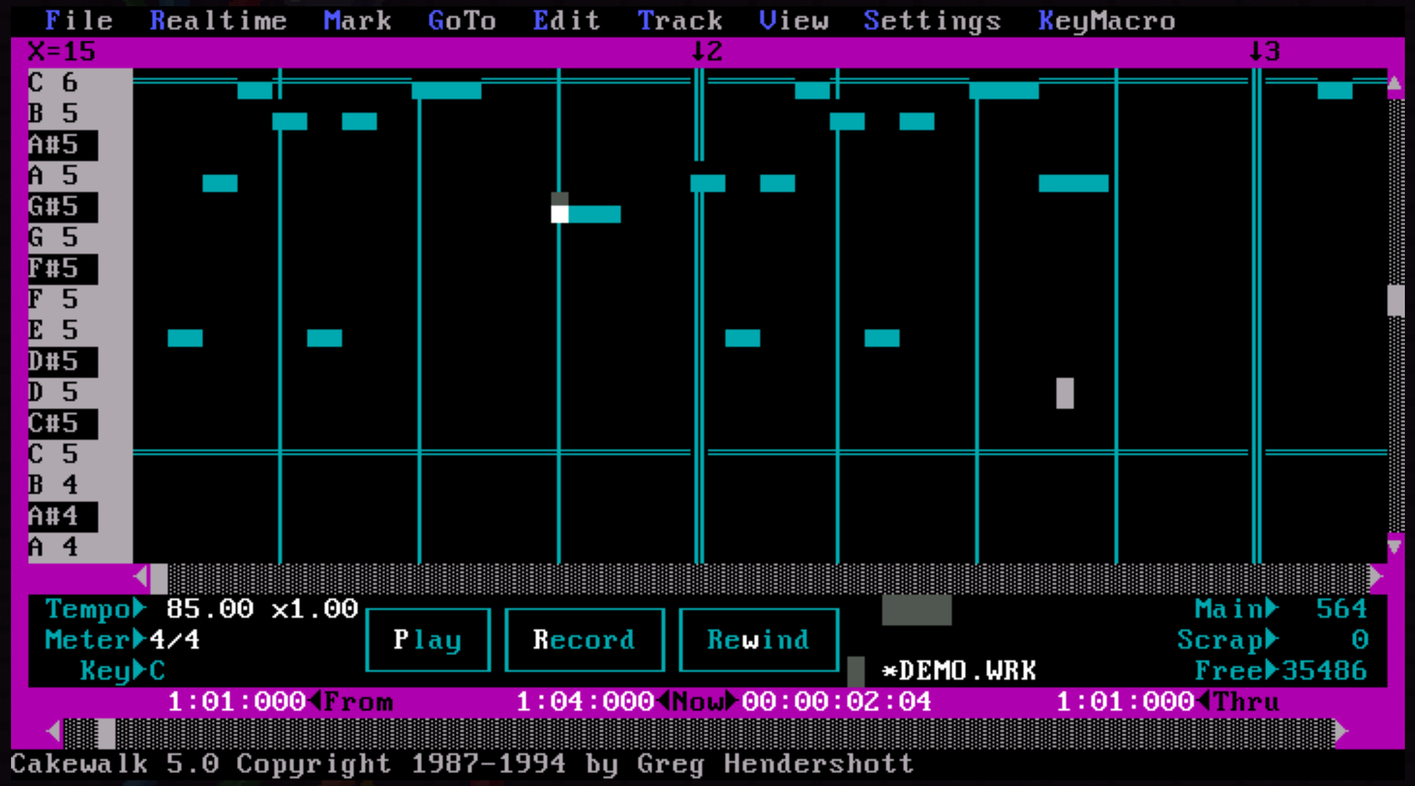
```
int Markov::tick() {
    int size = TM.size();

    std::vector<int> pop(size);
    std::iota(pop.begin(), pop.end(), 1);
    std::vector<float> probs = TM[currentTick - 1];
    std::discrete_distribution<> dist(probs.begin(), probs.end());

    currentTick = dist(gen);
    return  currentTick;
}
```

CURVES FOR SEQUENCING

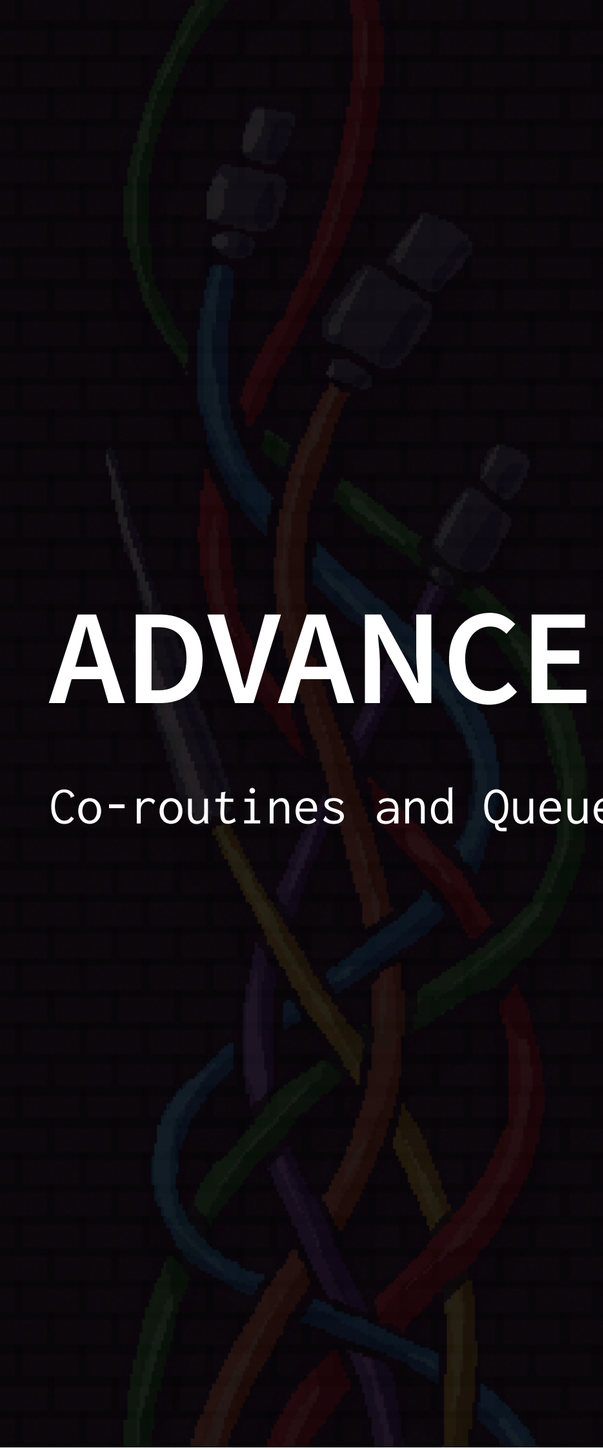
1. Easing
2. Bézier curve
3. ADSR
4. LFO

```
double easeInCubic( double time ) {
    return time * time * time;
}
```

TUI USING ftxui



1. Drumpad
2. Sequencer

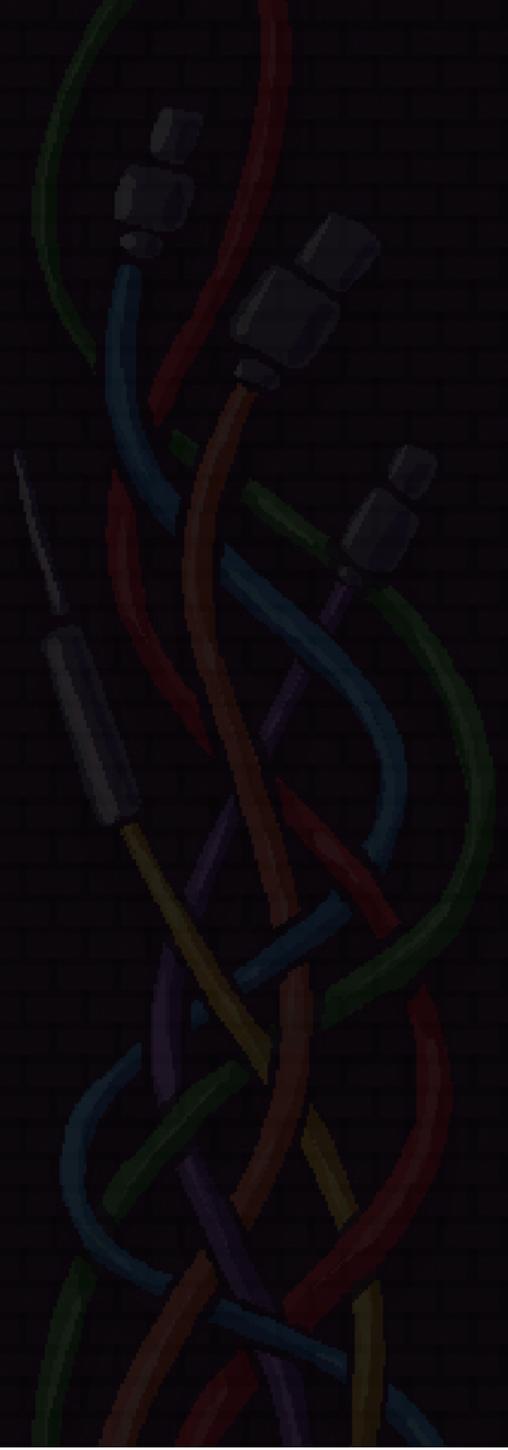


ADVANCED

Co-routines and Queues

MORE

1. Choc - Live Loading, WebView
2. Duktape - Embedded JS
3. Raylib / SDL
4. Faust / Maximilian
5. Header only libraries - stb*, OSC, moodycamel::ConcurrentQueue
6. Fmod / Soloud
7. OpenCV / ggml



END

https://xyzzyapps.link
https://github.com/xyzzyapps