Music and Code: Introduction to Livecoding with TidalCycles

Algorithmic Control of External Hardware & DAWs

Lenny Foret

Music Technology Workshop for Locust Projects

May 8, 2025

What is TidalCycles?

- Live coding environment for creating algorithmic patterns
- Especially powerful for rhythmic and pattern-based music
- Created by Alex McLean as part of the Algorave movement
- Open source, written in Haskell
- Concise syntax for complex musical ideas

Seminar Overview

- What is TidalCycles?
- Producer-oriented approach
- Using TidalCycles with external instruments
- Pattern concepts and syntax
- MIDI integration
- Automation and control
- Workshop: Analyzing a live performance
- Hands-on exercises

What is TidalCycles?

- Live coding environment for creating algorithmic patterns
- Especially powerful for rhythmic and pattern-based music
- Created by Alex McLean as part of the Algorave movement
- Open source, written in Haskell
- Concise syntax for complex musical ideas

The Conventional TidalCycles Approach

Traditional setup:

- TidalCycles as pattern generator
- SuperCollider as the sound engine
- SuperDirt as synthesizer/sampler
- Self-contained ecosystem

Our Producer-Oriented Approach

- Use TidalCycles as a flexible controller
- Integrate with standard production workflows
- Control hardware synths via MIDI
- Automate DAW parameters
- Trigger external instruments (909, 303, Hydrasynth)
- Focus on pattern generation, not sound synthesis

Pattern Basics

A simple patter to play MIDI notes

```
-- Simple pattern: Play C4, E4, G4, C5 in sequence

d1 $ note "c4 e4 g4 c5" # s "midi"
```

Tidal Timing: Cycles and Patterns

- Default: 1 cycle = 1 bar (adjustable with `setcps`)
- Tempo expressed in cycles per second
- For 120 BPM: `setcps (120/60/4)` = 0.5 cycles per second

```
-- Set tempo to 126 BPM
setcps (126/60/4)

-- Simple quarter note pattern
d1 $ s "midi" # note "c4"

-- Eighth notes
d1 $ s "midi" # note "c4 e4"

-- 16th notes (a 4-note pattern at twice the speed)
d1 $ s "midi" # note "c4 e4 g4 c5*2"
```

Pattern Syntax

- Simple sequences: `"a b c d"`
- Grouping: `"[a b] c"`
- Repetition: `"a*3 b"`
- Alternation: `"<a b> c"`
- Rest/silence: `"a ~ b c"`
- Euclidean rhythms: `"c(3,8)"` (3 events spread over 8 steps)

```
-- Different pattern examples
d1 $ note "c4 [e4 g4] c5*2" # s "midi"
d2 $ note "c4(3,8)" # s "midi"
d3 $ note "<c4 e4> <g4 f4>" # s "midi"
```

Pattern Manipulation

Key functions for transforming patterns:

- `rev` - reverse
- `slow` - stretch pattern
- `fast` - compress pattern
- `jux` - juxtapose transformations
- `every` - apply function every n cycles
- `shuffle` - reorder elements
- `palindrome` - forward then backward
- `chop` - cut into smaller parts
- `striate` - interleave segments
- `swingBy` - add swing feel

```
-- Add swing to a MIDI pattern
d1 $ swingBy 0.08 8 $ note "c4 e4 g4 c5 e5 g5 e5 c5" #
    s "midi"
```

MIDI Integration with TidalCycles

- Connect to MIDI devices through SuperCollider
- Send MIDI notes, CCs, and program changes
- Each pattern can target different MIDI devices/channels
- Synchronize with external equipment via MIDI clock

```
-- SuperCollider MIDI setup example
MIDIClient.init;
~midiOut = MIDIOut.newByName("USB MIDI Interface", "
    Port 1");
~dirt.soundLibrary.addMIDI(\midi, ~midiOut);

-- In TidalCycles, use device "midi"
d1 $ note "c4 e4 g4 c5" # s "midi"
    # midichan 0
```

Practical MIDI Setup

- Multiple MIDI destinations:

```
-- SuperCollider setup with multiple destinations
~midiOut1 = MIDIOut.newByName("TR-909", "MIDI IN");
~midiOut2 = MIDIOut.newByName("TB-303", "MIDI IN");
~midiOut3 = MIDIOut.newByName("Hydrasynth", "MIDI IN")
    ;

-- Add these devices to SuperDirt with custom names
~dirt.soundLibrary.addMIDI(\drums, ~midiOut1);
~dirt.soundLibrary.addMIDI(\bass, ~midiOut2);
~dirt.soundLibrary.addMIDI(\lead, ~midiOut3);

-- In TidalCycles, use specific devices
d1 $ note "36 ~ 38 ~" # s "drums"  -- kick/snare on
    909
d2 $ note "c2 [c2 eb2] f2 g2" # s "bass"  -- bassline
    on 303
d3 $ note "c4 e4 g4 c5" # s "lead"  -- melody on Hydra
```

Control Changes and Automation

- Send CC messages to control parameters
- Create evolving parameter patterns
- Control filter cutoff, resonance, envelopes, etc.

```
-- Control cutoff (CC 74) on a 303
d2 $ note "c2 [c2 eb2] f2 g2" # s "bass"
   # ccn 74 # ccv (range 30 100 $ slow 8 $ sine)

-- Multiple CC values simultaneously
d3 $ note "c4 e4 g4 c5" # s "lead"
   # stack [
     ccn 74 # ccv (range 30 100 $ slow 8 $ sine),
     ccn 71 # ccv (range 10 90 $ slow 7 $ saw)
   ]
```

DAW Integration via IAC Bus

- Use IAC (Inter-Application Communication) Driver buses
- Route MIDI from TidalCycles to DAW channels
- Trigger instruments or control parameters in Ableton, Logic, etc.

```
-- In SuperCollider
~midiDAW = MIDIOut.newByName("IAC Driver", "Bus 1");
~dirt.soundLibrary.addMIDI(\daw, ~midiDAW);

-- In TidalCycles - control Ableton tracks
d1 $ note "c4 e4 g4 c5" # s "daw" # midichan 0  --
   Track 1
d2 $ note "36 ~ 38 ~" # s "daw" # midichan 1    --
   Track 2
d3 $ ccn 74 # ccv (range 0 127 $ slow 4 $ sine)
   # s "daw" # midichan 2                       --
   Automate Track 3
```

Working with Scales and Chords

- Use musical scales rather than raw MIDI notes
- Build chord progressions with scale degrees
- Transpose patterns while staying in key

```
-- Using the scale function for melodies
d1 $ n (scale "minor" "0 2 4 7") # s "lead" |+ n "c"

-- Creating a chord pattern in F minor
d2 $ n (scale "minor" "[0,2,4]") # s "lead" |+ n "f"

-- Add transposition within the scale
d3 $ n (scale "minor" "[0,2,4]") |+ "-4" # s "lead" |+
    n "f"
```

Complex Chord Example from Desert Minimal

Breaking down a complex chord pattern:

```
-- Final complex chord pattern
d7 $ n (scale "minor" ( "[0,2,4, -14, -4 _ ]"
  |+ "-4"
  |+ "0"))
  # s "seven"
  |+ n ("0" |* 12)
  |+ n "f"
```

Components:

- `scale "minor"` - Use minor scale
- `[0,2,4, -14, -4 _]` - Chord structure with extensions
- `|+ "-4"` - Transposition down 4 scale degrees
- `|+ "0"` - Secondary transposition layer (can be changed)
- `|+ n ("0" |* 12)` - Octave setting
- `|+ n "f"` - Key of F

Advanced Pattern Structures

- Combining pattern operators
- Creating evolving patterns with slow transformations
- Using randomization and chaos functions

```
-- Complex pattern with multiple transformations
d1 $ every 4 (fast 2) $ every 3 (rev)
   $ note (scale "minor" $ "0 [2 4] 7 <3 5>")
   # s "lead" |+ n "f"

-- Pattern with evolving parameters
d2 $ note "c2 [~ c2] <eb2 f2> g2" # s "bass"
   # cutoff (range 300 2000 $ slow 8 $ sine)
   # resonance (range 0.1 0.8 $ slow 7 $ tri)
   # sustain (range 0.1 0.4 $ rand)
```

Case Study: Desert Minimal / Time Lapse

- Track: Time Lapse by Lenny Foret
- From the physical tape release
- Created with TidalCycles controlling external gear
- Source: `https://mmxximnml.bandcamp.com/track/lenny-foret-time-lapse`

Track Structure Breakdown

1. Intro: Atmospheric risers
2. Section 1: Chord progression with complex harmony
3. Section 2: Basic beat with foundational rhythm
4. Section 3: Added textural elements
5. Section 4-18: Various pattern and effect variations
6. Final section: Sample slicing techniques

We'll examine how different elements are controlled via MIDI and automation.

Chord Progression Analysis

From the desert-minimal.tidal file:

```
d7 $ n (scale "minor" ( "[0,2,4, -14, -4 _ ]"
  |+ "-4"
  |+ "0"))
  # s "seven"
  |+ n ("0" |* 12)
  |+ n "f"
```

- Uses a complex chord structure in F minor
- Includes extensions for rich harmony
- Channel "seven" represents an external synth instrument
- Forms the harmonic foundation of the track

Rhythm Section Analysis

From the desert-minimal.tidal file:

```
d2 $ n ("{ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15}%8"
  |+ ("<0>" |* "16")
  |+ "-24")
  # s "drums"

d3 $ n ("{2 ~ }%8"
  |+ ("<0>" |* "16")
  |+ "-24")
  # s "three"
```

- `drums` channel controls drum machine via MIDI
- Pattern uses 16 steps at 8 steps per cycle
- Transposition `|+ "-24"` for proper drum mapping
- Channel `three` adds sparse percussive elements

Effect Processing Analysis

Control of external effects:

```
f3 $ segment 5 $ s "ch3"
    # stack [lowcut 0,
             highcut 127,
             senda 10,
             sendb "<0 10 0 20 0 30>"]
```

- `f3` controls effect parameters via MIDI CCs
- `segment 5` divides cycle into 5 equal parts
- `stack` applies multiple parameters simultaneously
- `senda`/`sendb` control external effect sends (reverb/delay)
- Pattern `"<0 10 0 20 0 30>"` creates rhythmic modulation

Adding Variation and Movement

Evolution techniques:

```
-- Add alternating values to the chord pattern
d7 $ n (scale "minor" ( "[0,2,4, -14, -4 _ , ~ <6 2
    >]"
  |+ "-4"
  |+ "<0>"))
  # s "seven"
  |+ n ("0" |* 12)
  |+ n "f"

-- Add swing to drum patterns
d2 $ swingBy 0.08 8 $ n ("{ ~ 0 1 2 }%16"
  |+ ("<0 1>" |* "16")
  |+ "-24")
  # s "drums"
```

- <6 2> alternates values on successive cycles
- swingBy 0.08 8 adds human-like timing variation
- These techniques create organic movement in machine patterns

Layer Integration for Intensity

Creating builds and drops:

```
do
 f4 $ segment 5 $ s "ch4"
      # stack [lowcut 0,
               highcut 100,
               senda 100,
               sendb 80]
 d2 $ n ("{ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15}%8"
  |+ ("<0>" |* "16")
  |+ "-24")
  # s "drums"
 d7 $ n (scale "minPent" ( "[0, <5 -2>, <8 2 1 0>, <-8
     -6 -4>]"
  |+ "-4"
  |+ "0"))
  # s "seven"
  |+ n ("0" |* 12)
  |+ n "f"
```

- do block combines multiple changes simultaneously

Hardware Setup for TidalCycles Production

Recommended hardware configuration:

- Computer running TidalCycles
- MIDI interface with multiple outputs
- Hardware synthesizers (analog/digital)
- Drum machines
- Effects processors
- Mixing console or audio interface
- Optional: DAW for recording and additional processing

Syncing with DAW and Hardware

Options for synchronization:

- MIDI clock from TidalCycles to devices
- MIDI clock from DAW to TidalCycles
- Ableton Link integration:

```
-- In SuperCollider setup file
-- Create Link connection with 2 beats per cycle
~link = LinkClock(2).latency_(s.latency);

-- Register the clock with SuperDirt
~link.tempo_(120/60/2); // Set tempo (120 BPM)
~dirt.serverControls = ~dirt.serverControls.addFirst(~
    link);
```

TidalCycles can adapt to different sync scenarios depending on your workflow.

Recording and Performance Setup

Best practices:

- Set up automation recording in your DAW
- Create templates for different hardware configurations
- Organize MIDI channels consistently
- Consider partial recording: MIDI patterns in DAW, but keep live control
- For performance: create modular patterns that can be mixed/matched
- Develop transitions between sections

Exercise 1: Building a Basic Pattern

Create a simple pattern that controls an external synth:

```
-- Set the tempo
setcps (120/60/4)

-- Create a basic pattern
d1 $ note "c3 [~ c3] e3 g3" # s "bass"

-- Add some pattern variation
d1 $ every 4 (rev) $ note "c3 [~ c3] e3 g3" # s "bass"

-- Add parameter modulation
d1 $ every 4 (rev) $ note "c3 [~ c3] e3 g3" # s "bass"
   # ccn 74 # ccv (range 30 100 $ slow 4 $ sine)
```

Exercise 2: Creating a Drum Pattern

Create drum patterns for an external drum machine:

```
-- Basic kick and snare pattern
d2 $ n "36 ~ 38 ~" # s "drums"

-- Add hi-hats
d3 $ n "~ 42 ~ 42" # s "drums"

-- Add a bit of swing
d2 $ swingBy 0.06 8 $ n "36 ~ 38 ~" # s "drums"
d3 $ swingBy 0.06 8 $ n "~ 42 ~ 42" # s "drums"

-- Create fill variations
d2 $ every 4 (fast 2) $ swingBy 0.06 8 $ n "36 ~ 38 ~"
    # s "drums"
```

Exercise 3: Chord Progressions

Build a chord progression using scale degrees:

```
-- Basic triad in C minor
d4 $ n (scale "minor" "[0,2,4]") # s "pad" |+ n "c"

-- Create a progression
d4 $ n (scale "minor" "<[0,2,4] [3,5,7] [-2,0,2]
    [1,3,5]>")
    # s "pad" |+ n "c"

-- Add a parameter sweep
d4 $ n (scale "minor" "<[0,2,4] [3,5,7] [-2,0,2]
    [1,3,5]>")
    # s "pad" |+ n "c"
    # ccn 74 # ccv (range 20 110 $ slow 16 $ sine)
```

Exercise 4: Putting It All Together

Combine elements into a coordinated pattern:

```
-- Use a "do" block to start multiple patterns
do
  -- Reset everything
  hush
  -- Set tempo
  setcps (120/60/4)
  -- Bass pattern
  d1 $ note "c3 [~ c3] e3 g3" # s "bass"
     # ccn 74 # ccv (range 30 100 $ slow 4 $ sine)
  -- Drum pattern
  d2 $ swingBy 0.06 8 $ n "36 ~ 38 ~" # s "drums"
  d3 $ swingBy 0.06 8 $ n "~ 42 ~ 42" # s "drums"
  -- Chord pattern
  d4 $ n (scale "minor" "<[0,2,4] [3,5,7] [-2,0,2]
    [1,3,5]>")
     # s "pad" |+ n "c"
```

Conclusion

- TidalCycles provides powerful pattern control for producers
- Integration with external gear preserves your sound while adding algorithmic complexity
- Benefits of this approach:
  - Keep your familiar sounds and gear
  - Add algorithmic complexity to traditional setups
  - Combine programming precision with hardware character
  - Create patterns beyond traditional sequencing
- Start small: add TidalCycles to one aspect of your production
- Build up complexity as you become comfortable

Resources

- Official TidalCycles documentation: `https://tidalcycles.org/docs/`
- Tidal Club community: `https://club.tidalcycles.org/`
- MIDI-specific tutorials: `https://tidalcycles.org/docs/pattern-language/midi/`
- Algorave community: `https://algorave.com/`
- Books:
  - "Algorithmic Composition: A Guide to Composing Music with Nyquist" - Heintz
  - "The SuperCollider Book" - Wilson, Cottle, Collins
- YouTube channels with TidalCycles tutorials
- GitHub repositories with example patterns