Tyrell Desktop Synthesizer
Created by John Park

https://learn.adafruit.com/tyrell-desktop-synthesizer

Last updated on 2023-08-29 04:57:29 PM EDT
Table of Contents

Overview 3
• Parts

3D Print the Tyrell Building 5
• 3D Printed Parts
• Customization

Build the Tyrell Circuit 8
• Perma Proto Version
• Gather the Components
• TRRS Breakout Headers
• Solder the QT Py Headers
• Touch Resistors
• Audio Filters
• Cap Touch Headers
• Base Attachment
• Touch Cabling
• Cable Connections
• Close the Case

Code the Tyrell Synth 19
• Text Editor
• Download the Project Bundle
• How It Works
• Main Loop

Play the Tyrell Desktop Synthesizer 26
Overview

Build your own Tyrell Desktop Synthesizer to fill your life with Neo-noir cyberpunk atmosphere. This audio/visual homage to Blade Runner uses CircuitPython's synthio library to slowly evolve a dramatic soundscape of dystopian wub. You can adjust the sounds with eight cap touch pads and enjoy the audio drama like only a replicant can.

The code is an adaptation of Tod Kurt's excellent e (ighties_dystopia) synth.

Parts

Adafruit QT Py RP2040
What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with one of our new favorite chip, the RP2040. It's been made famous in the new https://www.adafruit.com/product/4900
Adafruit TRRS Jack Breakout Board
Tip-Ring-Ring-Sleeve style audio cables are often used for situations where you want stereo audio and then an extra contact for a microphone input.
https://www.adafruit.com/product/5764

Adafruit Perma-Proto Quarter-sized Breadboard PCB - Single
Customers have asked us to carry basic perf-board, but we never liked the look of most basic perf: it's always crummy quality, with pads that flake off and no labeling. Then we...
https://www.adafruit.com/product/1608

0.1uF ceramic capacitors - 10 pack
As any engineer will tell you, 0.1uF ceramic capacitors are the magic fairy dust of electronics - sprinkle them over your circuit schematics for good luck! But seriously, a 0.1uF...
https://www.adafruit.com/product/753

Copper Foil Tape with Conductive Adhesive - 6mm x 5 meters long
Copper tape can be an interesting addition to your toolbox. The tape itself is made of thin pure copper so it's extremely flexible and can take on nearly any shape. You can solder...
https://www.adafruit.com/product/3483
1 x **Premium Female/Female Jumper Wires**  
20 x 12" (300mm)  
https://www.adafruit.com/product/1949

1 x **Black Nylon Machine Screw and Stand-off Set**  
M2.5 Thread  
https://www.adafruit.com/product/3299

1 x **36-pin 0.1" Short Female Header**  
Pack of 5  
https://www.adafruit.com/product/3008

1 x **Male Header 36-pin 0.1" Short Break-away Type**  
Pack of 10  
https://www.adafruit.com/product/3009

**8 x 1MΩ Through-hole Resistors**

Any through hole 1MΩ or higher should work great, such as these.  

I've seen things you people wouldn't believe... Attack ships on fire off the shoulder of Orion... I watched C-beams glitter in the dark near the Tannhäuser Gate. All those moments will be lost in time, like tears in rain... Time to die.

**3D Print the Tyrell Building**

The case was modeled after Syd Mead's iconic Tyrell Corporation building from Blade Runner.
3D Printed Parts

Print the two STL files contained in the zip archive linked below (also available on Printables () and Thingiverse ()). I printed at 0.15mm layer height using PLA with 20% infill.

Use support material for the rim inset at the bottom of the case file only (you can use blockers to avoid printing support up the center).

I also found it helpful to print corner anti-warping tabs () as shown here to mitigate curling.
Customization
Should you want to modify the files in a CAD or NURBs package, the STEP files generated from the original Rhino/Grasshopper source are linked below.
Build the Tyrell Circuit

The circuit does two primary things:

- reads eight cap touch pins (which each run to ground via >1MΩ resistors)
- sends stereo PWM audio out over the TX and RX pins, via RC (resistor/capacitor) filter circuits

Here you can see the breadboarded version of the circuit with wires running from each cap touch pin:
Perma Proto Version

In order to fit the small space inside the Tyrell building model, you'll build the circuit onto a 1/4 size Perma Proto board.

You'll note the flipped TRRS breakout board used to maximize space. The breakout will be fit onto the board via headers in order to provide vertical clearance of the QT Py.
Gather the Components
Prep for soldering by gathering the components. You'll want to snap off two seven-pin lengths of male headers for the QT Py.

TRRS Breakout Headers
Cut of a six-position strip each of the short pin and socket headers. Solder the pin headers under the breakout and the socket to the PermaProto as shown here.
Solder the QT Py Headers
Solder the QT Py to the headers and the Perma Proto as shown. Note how the position leaves room for the mounting hole.

Touch Resistors
Solder the eight >1MΩ resistors from the touch pins to ground. The pins used are A0, A1, A2, A3, SDA, SCL, MISO, MOSI.
Audio Filters
You'll use a pair of resistor-capacitor (RC) filters to smooth out the otherwise harsh PWM audio that will go from pins TX and RX to the TRS jack's left and right channels.

You can also add jumper wires to ground at this point.
Cap Touch Headers
Fit the TRRS breakout into the header and then solder in one set of six and another set of two headers facing up from the board to connect the cap touch cables.
Base Attachment
Use two M2.5 x 6mm screws and nuts to attach the board to the inside of the Tyrell building case base. You can use nylon screws and file them down for a smooth fit, or get a set of low profile "laptop screws" such as these ().
Touch Cabling
Cut eight of the jumper wires down to about 4" in length, then strip about 1" of insulation from the tips.

Feed the wires through the holes from the inside of the case to the outside.

Cut a length of copper tape and affix to the side, overlapping the stripped wire as shown. These will act as the cap touch sensors.
Cable Connections

Wire up the cap touch cables to their respective pins as shown.

Plug in the USB-C cable and TRS 3.5mm stereo audio cable -- you can use right-angled cables to help with the fit if necessary. Some straight cables may fit as well, depending on their design.

Route the USB-C and audio cables out of the "back" side of the case with the cable opening.
Close the Case

Snap the base into the main part of the case. You can use M2.5 x 10mm screws to secure it in the corners if needed, but the snap fit should be solid enough to hold it in place.
Since I used the very short USB-C to A cable, I use a USB extension cable as well, shown here:

---

**Code the Tyrell Synth**

**Text Editor**

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in [this guide](#).

Alternatively, you can use any text editor that saves simple text files.

**Download the Project Bundle**

Your project will use a specific set of CircuitPython libraries, and the code.py file. To get everything you need, click on the Download Project Bundle link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your board's CIRCUITPY drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary. The CIRCUITPY drive appears when you plug the QT Py into the computer via USB.
# SPDX-FileCopyrightText: 2023 John Park & Tod Kurt
# # SPDX-License-Identifier: MIT
# Tyrell Synth Distopia
# based on:
# # 19 Jun 2023 - @todbot / Tod Kurt
# - A swirling ominous wub that evolves over time
# - Made for QTPy RP2040 but will work on any synthio-capable board
# - wallow in the sound
# #
# Circuit:
# # - QT Py RP2040
# - QTPy TX/RX pins for audio out, going through RC filter (1k + 100nF) to TRS jack
# - Touch io for eight pins, pairs that -/+ tempo, transpose pitch, filter rate, volume
# use >1MΩ resistors to pull down to ground
# # Code:
# # - Five detuned oscillators are randomly detuned very second or so
# - A low-pass filter is slowly modulated over the filters
# - The filter modulation rate also changes randomly every second (also reflected on neopixel)
# - Every x seconds a new note is randomly chosen from the allowed note list

```python
import time
import random
import board
import audiopwmio
import audiomixer
import synthio
import ulab.numpy as np
import neopixel
import rainbowio
import touchio
from adafruit_debouncer import Debouncer

touch_pins = (board.A0, board.A1, board.A2, board.A3, board.SDA, board.SCL,
board.MISO, board.MOSI)
touchpads = []
for pin in touch_pins:
    tmp_pin = touchio.TouchIn(pin)
    touchpads.append(Debouncer(tmp_pin))

notes = (37, 38, 35, 49)  # MIDI C#, D, B
note_duration = 10  # how long each note plays for
num_voices = 6  # how many voices for each note
lpf_basef = 300  # low pass filter lowest frequency
lpf_resonance = 1.7  # filter q

led = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.1)

# PWM pin pair on QTPY RP2040
audio = audiopwmio.PWMAudioOut(left_channel=board.TX, right_channel=board.RX)
```
mixer = audiomixer.Mixer(channel_count=2, sample_rate=28000, buffer_size=2048)
synth = synthio.Synthesizer(channel_count=2, sample_rate=28000)

audio.play(mixer)
mixer.voice[0].play(synth)
mixer_vol = 0.5
mixer.voice[0].level = mixer_vol

# oscillator waveform, a 512 sample downward saw wave going from +/-30k
wave_saw = np.linspace(30000, -30000, num=512, dtype=np.int16)  # max is +/-32k
amp_env = synthio.Envelope(attack_level=1, sustain_level=1)

# set up the voices (aka "Notes" in synthio-speak) w/ initial values
voices = []
for i in range(num_voices):
    voices.append(synthio.Note(frequency=0, envelope=amp_env, waveform=wave_saw))
lfo_panning = synthio.LFO(rate=0.1, scale=0.75)

# set all the voices to the "same" frequency (with random detuning)
# zeroth voice is sub-oscillator, one-octave down
def set_notes(n):
    for voice in voices:
        f = synthio.midi_to_hz(n + random.uniform(0, 0.4))
        voice.frequency = f
        voice.panning = lfo_panning
        voices[0].frequency = voices[0].frequency/2  # bass note one octave down

# the LFO that modulates the filter cutoff
lfo_filtermod = synthio.LFO(rate=0.05, scale=2000, offset=2000)
# we can't attach this directly to a filter input, so stash it in the blocks runner
synth.blocks.append(lfo_filtermod)

note = notes[0]
last_note_time = time.monotonic()
last_filtermod_time = time.monotonic()

# start the voices playing
set_notes(note)
synth.press(voices)

# user input variables
note_offset = (0, 1, 3, 4, 5, 7)
note_offset_index = 0
lfo_subdivision = 8

print("'Prepare to wallow.' \n- Major Jack Dongle")

while True:
    for t in range(len(touchpads)):
        touchpads[t].update()
        if touchpads[t].rose:
            if t == 0:
                note_offset_index = (note_offset_index + 1) % (len(note_offset))
                set_notes(note + note_offset[note_offset_index])
            elif t == 1:
                note_offset_index = (note_offset_index - 1) % (len(note_offset))
                set_notes(note + note_offset[note_offset_index])
            elif t == 2:
                note_duration = note_duration + 1
            elif t == 3:
                note_duration = abs(max((note_duration - 1), 1))
            elif t == 4:
                lfo_subdivision = 20

print("'Prepare to wallow.' \n- Major Jack Dongle")
elif t == 5:
    lfo_subdivision = 0.2
elif t == 6:  # volume
    mixer_vol = max(mixer_vol - 0.05, 0.0)
    mixer.voice[0].level = mixer_vol
elif t == 7:  # volume
    mixer_vol = min(mixer_vol + 0.05, 1.0)
    mixer.voice[0].level = mixer_vol

# continuously update filter, no global filter, so update each voice's filter
for v in voices:
    v.filter = synth.low_pass_filter(lpf_basef + lfo_filtermod.value, lpf_resonance)

led.fill(rainbowio.colorwheel(lfo_filtermod.value/20))  # show filtermod moving

if time.monotonic() - last_filtermod_time > 1:
    last_filtermod_time = time.monotonic()
    lfo_filtermod.rate = 0.01 + random.random() / lfo_subdivision

if time.monotonic() - last_note_time > note_duration:
    last_note_time = time.monotonic()
    note = random.choice([n for n in notes if n != note])
    set_notes(note+note_offset[note_offset_index])
    print("note", note, ["%3.2f" % v.frequency for v in voices])

A swirling ominous wub that evolves over time.

-Tod Kurt

How It Works

The code is an adaptation of Tod Kurt's excellent eighties_dystopia() synth. Using the synthio library in CircuitPython, it plays a set of five detuned oscillators that are modulated by a low-pass filter, with the root note changing over a certain interval of time.

The version adapted for the Tyrell Desktop Synthesizer adds user input in the form of eight touch pads. These are used as four pairs of controls that decrement/increment four parameters:

- tempo
- root note transposition
- filter rate
- output volume
First, we import the necessary libraries.

```python
import time
import random
import board
import audiopwmio
import audiomixer
import synthio
import ulab.numpy as np
import neopixel
import rainbowio
import touchio
from adafruit_debouncer import Debouncer
```

**Touch Pins**

Next, we'll set up the eight touch pins using the debouncer.

```python
touch_pins = (board.A0, board.A1, board.A2, board.A3, board.SDA, board.SCL, board.MISO, board.MOSI)
touchpads = []
for pin in touch_pins:
    tmp_pin = touchio.TouchIn(pin)
    touchpads.append(Debouncer(tmp_pin))
```

**Synth Variables**

We'll create variables to use in the synthesizer object to define a list of notes, the initial note duration in seconds, the number of synth voices, the low pass filter (LPF) frequency, and the LPF resonance.

```python
notes = (37, 38, 35, 49)  # MIDI C#, D, B
note_duration = 10   # how long each note plays for
num_voices = 6       # how many voices for each note
lpf_basef = 300      # low pass filter lowest frequency
lpf_resonance = 1.7  # filter q
```

**Audio and Synth**

We'll create the stereo PWM audio object with mixer, and the synthio synth object, then set the mixer playing with a volume of 0.5 (of a possible 1.0).

```python
audio = audiopwmio.PWMAudioOut(left_channel=board.TX, right_channel=board.RX)
mixer = audiomixer.Mixer(channel_count=2, sample_rate=28000, buffer_size=2048)
synth = synthio.Synthesizer(channel_count=2, sample_rate=28000)
audio.play(mixer)
mixer.voice[0].play(synth)
```
mixer_vol = 0.5
mixer.voice[0].level = mixer_vol

Additional Synth Setup

The audio rate oscillator waveform is a saw wave with a soft attack. These are stacked up in the voices[] list. Then, a synthio low frequency oscillator (LFO) object is created that will be used to modulate the left-right stereo panning.

```python
# oscillator waveform, a 512 sample downward saw wave going from +/-30k
wave_saw = np.linspace(30000, -30000, num=512, dtype=np.int16)  # max is +/-32k gives us headroom
amp_env = synthio.Envelope(attack_level=1, sustain_level=1)

# set up the voices (aka "Notes" in synthio-speak) w/ initial values
voices = []
for i in range(num_voices):
    voices.append(synthio.Note(frequency=0, envelope=amp_env, waveform=wave_saw))

lfo_panning = synthio.LFO(rate=0.1, scale=0.75)
```

set_note() Function

The set_note() function will be sued to create the set of multiple voices any time the note changes, including the translation of MIDI note value notation to Hertz frequency and stereo panning to create some movement between channels.

```python
# set all the voices to the "same" frequency (with random detuning)
# zeroth voice is sub-oscillator, one-octave down
def set_notes(n):
    for voice in voices:
        f = synthio.midi_to_hz(n + random.uniform(0, 0.4))
        voice.frequency = f
        voice.panning = lfo_panning
    voices[0].frequency = voices[0].frequency/2  # bass note one octave down
```

Filter LFO

To automate the modulation of the filter cutoff frequency, another LFO is created and attached to the `synth.blocks` object.

```python
# the LFO that modulates the filter cutoff
lfo_filtermod = synthio.LFO(rate=0.05, scale=2000, offset=2000)
# we can't attach this directly to a filter input, so stash it in the blocks runner
synth.blocks.append(lfo_filtermod)
```
Final Setup Steps

State variables are created for the current note, last note time, and last filter modulation time, then the voices are set playing.

```python
note = notes[0]
last_note_time = time.monotonic()
last_filtermod_time = time.monotonic()

# start the voices playing
set_notes(note)
synth.press(voices)

# user input variables
note_offset = (0, 1, 3, 4, 5, 7)
note_offset_index = 0
lfo_subdivision = 8
```

Main Loop

In the main loop of the program, first the touch pads are checked to see if any have been pressed. They then increment or decrement their associated parameters.

```python
while True:
    for t in range(len(touchpads)):
        touchpads[t].update()
        if touchpads[t].rose:
            if t == 0:
                note_offset_index = (note_offset_index + 1) % (len(note_offset))
                set_notes(note + note_offset[note_offset_index])
            elif t == 1:
                note_offset_index = (note_offset_index - 1) % (len(note_offset))
                set_notes(note + note_offset[note_offset_index])
            elif t == 2:
                note_duration = note_duration + 1
            elif t == 3:
                note_duration = abs(max((note_duration - 1), 1))
            elif t == 4:
                lfo_subdivision = 20
            elif t == 5:
                lfo_subdivision = 0.2
            elif t == 6:  # volume
                mixer_vol = max(mixer_vol - 0.05, 0.0)
                mixer.voice[0].level = mixer_vol
            elif t == 7:  # volume
                mixer_vol = min(mixer_vol + 0.05, 1.0)
                mixer.voice[0].level = mixer_vol
```

Filter Sweep

The LFO that modulates the filter cutoff is updated.
# continuously update filter, no global filter, so update each voice's filter
for v in voices:
    v.filter = synth.low_pass_filter(lpf_basef + lfo_filtermod.value,
lpf_resonance)

    led.fill(rainbowio.colorwheel(lfo_filtermod.value/20))  # show filtermod moving

if time.monotonic() - last_filtermod_time &gt; 1:
    last_filtermod_time = time.monotonic()
    # randomly modulate the filter frequency ('rate' in synthio) to make more
dynamic
    lfo_filtermod.rate = 0.01 + random.random() / lfo_subdivision

Note Change

The melodic line changes to a random index selected from the note list.

# continuously update filter, no global filter, so update each voice's filter
for v in voices:
    v.filter = synth.low_pass_filter(lpf_basef + lfo_filtermod.value,
lpf_resonance)

    led.fill(rainbowio.colorwheel(lfo_filtermod.value/20))  # show filtermod moving

if time.monotonic() - last_filtermod_time &gt; 1:
    last_filtermod_time = time.monotonic()
    # randomly modulate the filter frequency ('rate' in synthio) to make more
dynamic
    lfo_filtermod.rate = 0.01 + random.random() / lfo_subdivision

---

**Play the Tyrell Desktop Synthesizer**

The Tyrell Desktop Synthesizer plays itself! Simply plug it in to USB power and a powered speaker set and it will create wonderful soundscapes.

You can adjust parameters by touching the eight pads as shown in the demo video above.