Music Player Moode

Hardware

Raspberry Pi Zero W	
Pimoroni Pirate Audio (Headphone)	https://shop.pimoroni.com/products/pirate-audio-headphone-amp
Pimoroni Pirate Audio Case w/ Buttons	https://www.thingiverse.com/thing:5245754
Pimoroni Pirate Audio Case orig w/o Buttons	https://www.printables.com/en/model/85009-pimoroni-pirate-audio-case
Pimoroni Pirate Audio Case angled base	https://www.thingiverse.com/thing:5248110
Pimoroni Pirate Audio Case open angled base	https://www.thingiverse.com/thing:4420857
Alternative case ideas	https://www.yeggi.com/q/pimoroni+pirate+audio/
External passive speakers with 3.5mm plug	
3.5mm Headphone Volume Control	https://smile.amazon.co.uk/gp/product/B00Y1MYSYW/

pirateaudio_with_buttons_base_-_no_holes.stl pirateaudio_with_buttons_top_-_no_holes.stl pirateaudiobutton.stl

System installation 8.3.0

Download Moode ISO https://moodeaudio.org/

```
https://github.com/moode-player/moode/releases/download/r830prod/image_2023-03-14-moode-r830-arm64-lite.zip
```

Main install instructions with reference to auto-install:
https://github.com/moode-player/moode/blob/master/www/setup.txt

Then flash using Balena Etcher or similar, for Belana, unzipping is not required!
https://www.balena.io/etcher/

```
sudo apt-get install belana-etcher-electron
```

```
sudo apt-get install debian-keyring debian-archive-keyring apt-transport-https ca-certificates gnupg
curl -1sLf "https://dl.cloudsmith.io/public/balena/etcher/gpg.70528471AFF9A051.key" | sudo apt-key add
cat <<EOF | sudo tee /etc/apt/sources.list.d/balena-etcher.list
# Source: Cloudsmith (support@cloudsmith.io)
# Repository: balena / etcher
```

```
# Description: Flash OS images to SD cards & USB drives, safely and easily.
deb https://dl.cloudsmith.io/public/balena/etcher/deb/ubuntu focal main
deb-src https://dl.cloudsmith.io/public/balena/etcher/deb/ubuntu focal main
EOF
sudo apt-get update
sudo apt-get install balana-etcher-electron
```

Mount the SDCard which will make the boot partition accessible Copy the file /boot/moodecfg.ini.default to your PC, Mac or Linux client Rename it to moodecfg.ini Edit the settings as needed (wlan, country, volume steps, etc) Copy moodecfg.ini to /boot/

SSH Server enabled by default: username: pi password: moodeaudio

moodecfg adjustments for UK and personal preferences:

moodecfg.ini

```
timezone = "Europe/London"
keyboard = "gb"

p3bt = "0"

replaygain = "track"

volume_normalization = "yes"

volume_step_limit = "2"

wlanssid = "xxx"
wlanpwd = "xxx"
wlancountry = "GB"

first_use_help = "No"
```

Config.txt adjustments for Pirate Audio on PiZeroW

/boot/config.txt

```
[pi0]
# Disable the ACT LED on the Pi 1 and Zero
dtparam=act_led_trigger=none
dtparam=act_led_activelow=on

[cm4]
otg_mode=1

[pi4]
hdmi_force_hotplug:0=1
hdmi_force_hotplug:1=1
```

```
[all]
#Disable boot splash screen
disable_splash=1
disable_overscan=1
hdmi_drive=2
#Disable HDMI on boot
hdmi_blanking=2
hdmi_force_edid_audio=1
hdmi_force_hotplug=1
hdmi_group=1
# Uncomment some or all of these to enable the optional hardware
interfaces
dtparam=i2c_arm=on
dtparam=i2s=on
#Switch off onboard audio
dtparam=audio=off
#dtoverlay=disable-wifi
dtoverlay=disable-bt

#Configure Pimoroni Pirate audio DAC
dtoverlay=i2s-mmap
dtoverlay=hifiberry-dac
gpio=25=op,dh

#Enable SPI for display of Pimoroni Pirate Audio
dtparam=spi=on

#Set GPU memory to lowest value in /boot/config.txt:
gpu_mem=16
```

possible further requirement for HDMI: Disable HDMI port on boot (power saving during headless operation) /usr/bin/tvservice -o (-p to re-enable) Add the line to /etc/rc.local to disable HDMI on boot.

Now insert the SD-Card into the Pi and power it up. Then connect to web interface via http://moode/ or IP if known.

Configure Moode: Audio Config: “HiFiBerry DAC” or “Pimoroni pHAT DAC” System: Disable ACT LED, Disable HDMI Enable GPIO button handler and set to: four buttons, active low connected to BCM 5, 6, 16, and 24 (A, B, X, Y respectively). Replace all spaces with commas in the command. See http://moodeaudio.org/forum/showthread.php?tid=1381&page=2&highlight=gpio

```
BTN1: 5  mpc,toggle
BTN2: 6  /var/www/vol.sh,-dn,5
BTN3: 16 mpc,next
BTN4: 24 /var/www/vol.sh,-up,5
Debounce 1000ms

#mpc,volume,-5
#mpc,volume,+5
#/var/www/vol.sh,-mute
```

Configure vi via ssh:

```
vi ~/.vimrc
```

```
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set nocompatible
```

Remove Bluetooth config in Moode main menu when not in use:

```
sudo vi /var/www/header.php
```

remove line referencing 'blu-config.php'

Pirate Audio TFT Cover Art v0.0.6

https://github.com/rusconi/TFT-MoodeCoverArt fork: https://github.com/pachisb/TFT-MoodeCoverArt

Note: 16-25% CPU usage on RPI-0w !

Enable Metadata file in System → Local Services in moode.

```
#enable spi if not already enabled
sudo raspi-config
sudo reboot
```

```
sudo apt-get update
sudo apt-get install git python3-rpi.gpio python3-spidev python3-pip
python3-pil python3-numpy libatlas-base-dev
sudo pip3 install mediafile pyyaml RPI-ST7789
cd /home/pi
git clone https://github.com/rusconi/TFT-MoodeCoverArt.git
cd TFT-MoodeCoverArt/
vi config.yml
chmod 777 *.sh

#test
python3 tft_moode_coverart.py
#if it works, install service
./install_service.sh
```

Bugfix: Wrap mf = MediaFile(fp) section in try/except and indent it

tft_moode_coverart.py

```
   else:
        if 'file' in metaDict:
```

```
            if len(metaDict['file']) > 0:

                fp = '/var/lib/mpd/music/' + metaDict['file']
                try:
                    mf = MediaFile(fp)
                    if mf.art:
                        cover = Image.open(BytesIO(mf.art))
                        return cover
                    else:
                        for it in covers:
                            cp = os.path.dirname(fp) + '/' + it
                            if path.exists(cp):
                                cover = Image.open(cp)
                                return cover
                except:
                    pass
    return cover
```

IP Address mod:

```
#add to top
import socket

#add before 'def main():'
def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        # doesn't even have to be reachable
        s.connect(('10.255.255.255', 1))
        IP = s.getsockname()[0]
    except Exception:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP

# add in def main(): after variable initialisation
    #get ip address, add to image and show on display
    ip = get_ip()
    draw.rectangle((0,0,240,240), fill=(0,0,0))
    txt = f"Visit http://{ip} to select content."
    mlw, mlh = draw.multiline_textsize(txt, font=font_m, spacing=4)
    draw.multiline_text(((WIDTH-mlw)//2, 20), txt, fill=(255,255,255),
font=font_m, spacing=4, align="center")
    disp.display(img)
```

Service

/lib/systemd/system/tft-moodecoverart.service

```
[Unit]
Description=TFT-MoodeCoverArt Display
Requires=mpd.socket mpd.service
After=mpd.socket mpd.service

[Service]
Type=simple
ExecStart=/home/pi/TFT-MoodeCoverArt/tft-moodecoverart.py &
ExecStartPre=/bin/sleep 15
#ExecStop=/home/pi/TFT-MoodeCoverArt/tft-moodecoverart.sh -q
ExecStop=/home/pi/TFT-MoodeCoverArt/shutdown.py &
Restart=on-abort
StandardOutput=syslog
StandardError=syslog

[Install]
WantedBy=multi-user.target
```

Boot logo on PirateAudio

/home/pi/TFT-MoodeCoverArt/boot.py

```
#!/usr/bin/env python3

import time
from PIL import ImageFont, Image, ImageDraw
import os
import ST7789
import sys

# get the path of the script
script_path = os.path.dirname(os.path.abspath(__file__))
# set script path as current directory
os.chdir(script_path)


# Create ST7789 LCD display class.
disp = ST7789.ST7789(
    rotation=90,  # Needed to display the right way up on Pirate Audio
    port=0,       # SPI port
    cs=1,         # SPI port Chip-select channel
    dc=9,         # BCM pin used for data/command
    backlight=13,
    spi_speed_hz=80 * 1000 * 1000
)

# Initialize display.
```

```
disp.begin()

WIDTH = 240
HEIGHT = 240
font_s = ImageFont.truetype(script_path + '/fonts/Roboto-Medium.ttf',
20)
font_m = ImageFont.truetype(script_path + '/fonts/Roboto-Medium.ttf',
24)
font_l = ImageFont.truetype(script_path + '/fonts/Roboto-Medium.ttf',
30)


def bootmessage():
    print('bootmessage called')
    disp.set_backlight(True)
    img = Image.new('RGBA', (240, 240), color=(0, 0, 0, 25))
    img = Image.open('images/default-cover-v6.jpg')
    draw = ImageDraw.Draw(img, 'RGBA')
    message = 'booting ...'
    draw.text((10, 200), message, font=font_m, fill=(255, 255, 255))
    disp.display(img)
    sys.exit()


try:
    bootmessage()
except SystemExit:
    print('Systemexit called')
    pass
```

```
chmod 755 /home/pi/TFT-MoodeCoverArt/boot.py
```

/lib/systemd/system/tft-boot.service

```
[Unit]
Description=TFT Boot Message
Before=basic.target
After=local-fs.target sysinit.target
DefaultDependencies=no

[Service]
Type=oneshot
ExecStart=/home/pi/TFT-MoodeCoverArt/boot.py

[Install]
WantedBy=basic.target
```

```
systemctl enable tft-boot
```

Pirate Audio Cover related links

https://github.com/pimoroni/pidi/blob/master/pidi/client.py
https://github.com/pimoroni/pidi-plugins/blob/master/pidi-display-pil/pidi_display_pil/__init__.py
https://github.com/pimoroni/pidi-plugins/blob/master/pidi-display-pil/pidi_display_pil/__init__.py
https://github.com/pimoroni/st7789-python https://github.com/pimoroni/pidi-spotify
https://github.com/pimoroni/pirate-audio/issues/17
https://github.com/pimoroni/mopidy-pidi/blob/master/mopidy_pidi/frontend.py
https://github.com/pimoroni/pirate-audio/blob/master/examples/backlight-pwm.py
https://github.com/AnonTester/TFT-MoodeCoverArt
http://moodeaudio.org/forum/showthread.php?tid=2210

https://ideatrash.net/2020/06/simple-smart-playlists-for-mpd-that-work.html
https://bbs.archlinux.org/viewtopic.php?id=76385

https://github.com/Ax-LED/volumio-pirate-audio

Long Button Press MOD

Adjust gpio-buttons.py script as per following example for first configured button:

/var/www/daemon/gpio_buttons.py

```
    if str(row['id']) == '1' and row['enabled'] == '1':
        sw_1_pin = int(row['pin'])
        sw_1_cmd = row['command'].split(',')
        sw_1_cmd = [x.strip() for x in sw_1_cmd]
        sw_1_cmd_2 = ["/var/www/command/sleeptimer.php","1800"]
        sw_1_cmd_3 = ["/var/www/command/sleeptimer.php","3600"]
        GPIO.setup(sw_1_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

        def sw_1_event(channel):
            start_time = time.time()
            time.sleep(0.005) # edge debounce of 5 ms
            # only deal with valid edges
            while GPIO.input(channel) == 0:  # wait for button up
                pass
            buttonTime = time.time() - start_time  #calc button press
            print('time ' + str(buttonTime))
            #if GPIO.input(channel) == 1:
            if buttonTime < 2:
                print('short press')
                subprocess.call(sw_1_cmd)
            elif 2 <= buttonTime < 4:  # long press
                print('long press')
                subprocess.call(sw_1_cmd_2)
            elif buttonTime > 4:  # very long press
                print('very long press')
```

```
                subprocess.call(sw_1_cmd_3)

        GPIO.add_event_detect(sw_1_pin, GPIO.FALLING,
callback=sw_1_event, bouncetime=bounce_time)
        print(str(datetime.datetime.now())[:19] + ' sw_1: pin=' +
            str(sw_1_pin) + ', enabled=' + row['enabled'] +
            ', bounce_time=' + str(bounce_time) + ', cmd=' +
row['command'])
```

/var/www/command/sleeptimer.php

```
#!/usr/bin/php
<?php
# Config values
$fadelength=5; #the duration of the fade out in seconds
$steptime=0.5; #the amount in seconds of each step


if($argc<2 || !isset($argv[1])) {
    echo "This script is a sleep timer to stop playback after a
while.\n";
    echo "You need to pass the amount of seconds to sleep to this
script!\n";
    echo "To stop playback after an hour use:\n".$argv[0]." 3600\n";
    exit;
}

sleep($argv[1]);

$currentvolume=filter_var(exec('/usr/bin/mpc volume'),
FILTER_SANITIZE_NUMBER_INT);
echo "Current volume: $currentvolume\n";
$stepamount = $fadelength/$steptime;
$stepadjust = floor($currentvolume/$stepamount);

for ( $i=1; $i<=$stepamount; $i++ ) {
    $vol = exec('/usr/bin/mpc volume -'.$stepadjust);
    echo "Volume adjusted to ".$vol."\n";
    usleep ($steptime*1000000);
}

echo "Stopping playback\n";
exec('/usr/bin/mpc stop');

sleep(1);
# Resetting volume
exec('/usr/bin/mpc volume '.$currentvolume);
echo "Resetting volume to $currentvolume\n";
```

```
sudo chmod 755 /var/www/command/sleeptimer.php
sudo killall -9 gpio_buttons.py
sudo /var/www/daemon/gpio_buttons.py &
```

Other notes:

```
#
https://learn.pimoroni.com/tutorial/sandyj/getting-started-with-pirate-audio

#This will install python3 pip,wheel and pirate audio modules
#Then add mopidy apt sources and install mopidy including mopidy-spotify
#Then install mopidy-iris web interface and Pirate Audio plugins
#And create system service to autostart mopidy

SMB mount the manual way (/etc/rc.local or /etc/fstab):
#Note experiment with rsize=61440 option and/or use nounix mount option
//192.168.1.6/music     /media/music    cifs
username=media,password=media,iocharset=utf8,noperm,file_mode=0644,dir_mode=
0755,users,rsize=61440,nounix       0       0

sudo mkdir /media/music
```

Moode tips

http://moodeaudio.org/forum/showthread.php?tid=803 MPD settings /etc/mpd.conf that control whether to automatically update the database when files are changed. Refer to this link https://github.com/MusicPlayerDaemon/MPD/blob/master/doc/mpd.conf.5 for information.

auto_update [yes or no] This specifies the whether to support automatic update of music database when files are changed in music_directory. The default is to disable autoupdate of database.

auto_update_depth [N] Limit the depth of the directories being watched, 0 means only watch the music directory itself. There is no limit by default.

in mean time you can ssh to moode and use this to add n last days

.bash_profile

```
function add-recents {
  find /media -type f -mtime -$1 | sed 's/\/media/USB/g' | mpc add
}
```

then 'add-recents 60' adds last 60 days off music

Library update: http://moode/command/?cmd=libupd-submit.php or

```
php /var/www/libupd-submit.php
```

instead of mpc commands, it should update MPD and covers.

You also can clear the library cache after the update:

```
mpc -w update
truncate /var/local/www/libcache.json --size 0
```

Add/Remove Radio Stations

GUI interface to add radio stations via + icon in radio interface.

Radio stations are stored as .pls files in /var/lib/mpd/music/RADIO

Station logos are stored with same name as pls file in:

```
/var/local/www/imagesw/radio-logos/Absolute Classic Rock.jpg
/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic Rock_sm.jpg
/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic Rock.jpg
```

```
file "/var/local/www/imagesw/radio-logos/Absolute Classic Rock.jpg"
/var/local/www/imagesw/radio-logos/Absolute Classic Rock.jpg: JPEG image
data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16,
baseline, precision 8, 225x225, components 3

file "/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic
Rock_sm.jpg"
/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic Rock_sm.jpg: JPEG
image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment
length 16, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality =
75", baseline, precision 8, 80x80, components 3

file "/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic Rock.jpg"
/var/local/www/imagesw/radio-logos/thumbs/Absolute Classic Rock.jpg: JPEG
image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment
length 16, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality =
75", baseline, precision 8, 200x200, components 3
```

Example pls file:

/var/lib/mpd/music/RADIO/Absolute Classic Rock.pls

```
[playlist]
File1=http://edge-bauerall-01-gos2.sharp-stream.com/absoluteclassicrock
.mp3?aw_0_1st.skey=1703523836&aw_0_1st.playerid=BMUK_RPi
Title1=Absolute Classic Rock
Length1=-1
NumberOfEntries=1
Version=2
```

MPD Playlist Folder

/var/lib/mpd/playlists/

Generate playlists of recent music

~/newm3u.sh

```
#!/bin/bash
base_dir=/mnt/
music_dir=NAS/music/other/deemix\ Music/
playlistweek=/var/lib/mpd/playlists/New\ last\ week.m3u
playlistmonth=/var/lib/mpd/playlists/New\ last\ month.m3u
playlist2month=/var/lib/mpd/playlists/New\ last\ 2\ months.m3u

if [ "$base_dir$music_dir" -nt "$playlist" ] || [ ! -f "$playlist" ];
then
  cd $base_dir
  find "$music_dir" -type f -mtime -7 -iname "*.mp3" -o -iname "*.ogg"
> "$playlistweek"
fi

if [ "$base_dir$music_dir" -nt "$playlist" ] || [ ! -f "$playlist" ];
then
  cd $base_dir
  find "$music_dir" -type f -mtime -31 -iname "*.mp3" -o -iname "*.ogg"
> "$playlistmonth"
fi

if [ "$base_dir$music_dir" -nt "$playlist" ] || [ ! -f "$playlist" ];
then
  cd $base_dir
  find "$music_dir" -type f -mtime -61 -iname "*.mp3" -o -iname "*.ogg"
> "$playlist2month"
fi
```

```
chmod 755 ~/newm3u.sh
```

MPD Dynamic Playlist

https://bbs.archlinux.org/viewtopic.php?id=76385

I present to you, fellow Archers, two little programs. One I've had for months but haven't shared for some reason, and the other I finished a short while ago.

MPDDP: MPD Dynamic Playlists is a program to generate, as the name would imply, a dynamic playlist for MPD. You specify rules about what tracks you want added, and it keeps adding them randomly.

Source:

```
#!/usr/bin/env python

# MPDDP: MPD Dynamic Playlists
# Call this and run it in the background (eg mpddp &>/dev/null &)
# Configured in /etc/mpddp.conf, See /etc/mpddp.conf.example.

import mpd, random, os, time, sys, string

client = mpd.MPDClient()

host = "" # The host MPD is operating upon
port = 0  # The port MPD is operating upon

playlistlen    = 0  # The len of the playlist
changeafter    = 0  # The number of tracks before more are added/removed
to/from the playlist
clearinitially = '' # Whether to clear the playlist initially or not.
saveonquit     = '' # Whether to save/load the playlist on exit/start or
not.
update         = '' # Whether to periodically check the config file /
filesystem for changes.

confdir = '/etc/mpddp.conf' # The path of the main MPDDP config file.
savedir = ''                # The folder where MPDDP saves and loads files.

alltracks = [] # All the tracks that can be played.
oldconfig = [] # The configuration as it was last loaded.

def pickNewTrack(): # Pick and remove a track from the list, append it to
the current list, and return the name.
    global client
    global alltracks

    index = random.randint(0, len(alltracks) - 1)
    track = alltracks[index]

    return track

def addNewTrackToPlaylist(): # Pick a new track, update the lists, and add
it to the playlist.
    global client
    global host
    global port
    global playlistlen
    client.connect(host, port)
    playlist = client.playlistinfo()
    client.disconnect()

    if len(playlist) < playlistlen:
```

```
        track = pickNewTrack()

        print "Adding", track
        client.connect(host, port)
        client.add(track)
        client.disconnect()

def removeLastTrackFromPlaylist(): # Delete the oldest track from the
playlist.
    global client
    global host
    global port
    client.connect(host, port)
    playlist = client.playlistinfo()
    client.delete(0)
    client.disconnect()

    print "Removing", playlist[0]['file']

def checkMPDPlaylist(): # Add enough tracks to the MPD playlist to
repopulate it if it is almost empty.
    global client
    global host
    global port
    global playlistlen
    global alltracks
    client.connect(host, port)
    playlist = client.playlistinfo()
    client.disconnect()

    if len(playlist) < playlistlen:
        while len(playlist) < playlistlen:
            addNewTrackToPlaylist()
def updatePlaylist(): # Update the MPD playlist, and the internal
representation of it if necessary.
    checkMPDPlaylist()
    removeLastTrackFromPlaylist()
    addNewTrackToPlaylist()

def getFilenamesFromMPDSPL(expression):
    os.system('mpdspl -s -n misc "' + expression + '" >/tmp/mpddp-mpdspl-
temp.txt')
    a = open('/tmp/mpddp-mpdspl-temp.txt')
    ot = a.read()
    ot = ot.splitlines()
    a.close()
    os.remove('/tmp/mpddp-mpdspl-temp.txt')
    return ot

def getFilenamesFromMPD(rules): # Gets the filenames from MPD of all files
which match the specified rules.
```

```
    global client
    global host
    global port

    paths     = []
    playlists = []
    smarts    = []
    nevers    = []
    tracks    = []
    for rule in rules:
        if rule[0] == 'path' and not rule[1] in paths:
            paths.append(rule[1])
        elif rule[0] == 'playlist' and not rule[1] in playlists:
            playlists.append(rule[1])
        elif rule[0] == 'smart' and not rule[1] in smarts:
            smarts.append(rule[1])
        elif rule[0] == 'never' and not rule[1] in nevers:
            nevers.append(rule[1])
    client.connect(host, port)
    for path in paths:
        temptracks = client.search("file", path)
        for track in temptracks:
            if isinstance(track, dict):
                track = track['file']
            dontadd = False
            for never in nevers:
                if never in track:
                    dontadd = True
            if dontadd == False and not track in tracks:
                tracks.append(track)

    for playlist in playlists:
        temptracks = client.listplaylist(playlist)
        for track in temptracks:
            if isinstance(track, dict):
                track = track['file']
            dontadd = False
            for never in nevers:
                if never in track:
                    dontadd = True
            if dontadd == False and not track in tracks:
                tracks.append(track)

    for smart in smarts:
        temptracks = getFilenamesFromMPDSPL(smart)
        for track in temptracks:
            dontadd = False
            for never in nevers:
                if never in track:
                    dontadd = True
            if dontadd == False and not track in tracks:
```

```
                tracks.append(track)

    client.disconnect()
    return tracks

def parseConfigIncludes(conf, path): # Parse a config file
    outconf = ""
    paths = path
    for line in conf:
        line = line.split("#")
        line = line[0]
        line = line.strip()
        if len(line) > 0:
            if line[0:7] == 'include':
                toinclude = line[8:].strip()
                toinclude = toinclude.replace("~", os.path.expanduser("~"))
                if not toinclude in paths:
                    paths.append(toinclude)
                    filehandler = open(toinclude)
                    newconf = parseConfigIncludes(filehandler, paths)
                    outconf = outconf + newconf
                    filehandler.close()
            else:
                outconf = outconf + line + "\r\n"
    return outconf

def parseConfigLine(line): # Parse a line from the configuration file and
return what it means.
    line = line.split("#")
    line = line[0]
    line = line.strip()

    if len(line) > 0:
        if ("=" in line) and (not ":" in line):
            pline = line.split("=", 1)
            parsed = {'type'  : pline[0].strip(),
                      'value' : pline[1].strip()}
            return parsed
        elif ":" in line:
            pline = line.split(":", 1)
            parsed = {'type'  : 'rule',
                      'value' : [pline[0].strip(), pline[1].strip()]}
            return parsed
        else:
            return {'type' : 'unrecognised'}
    else:
        return {'type' : 'blankline'}

def parseConfigFile(): # Open the configuration file and parse the rules.
    global confdir
    filehandler = open(confdir)
```

```
    conf = parseConfigIncludes(filehandler, [confdir])
    output = {'rules'          : [],
              'server'         : 'localhost',
              'port'           : 6600,
              'playlistlen'    : 15,
              'changeafter'    : 8,
              'clearinitially' : 'yes',
              'saveonquit'     : 'no',
              'savedir'        : '/var/lib/mpddp/',
              'update'         : 'no'}
    for line in conf.splitlines():
        result = parseConfigLine(line)
        if result['type'] == 'rule':
            output['rules'].append(result['value'])
        elif result['type'] == 'clearinitially' or result['type'] ==
'saveonquit':
            if result['value'] == 'yes' or result['value'] == 'no':
                output[result['type']] = result['value']
            else:
                print "Invalid value specified for", result['type']
        elif result['type'] == 'port' or result['type'] == 'playlistlen' or
result['type'] == 'changeafter':
            try:
                if (result['type'] == 'port' and int(result['value']) > 0
and int(result['value']) <= 65536) or not result['type'] == 'port':
                    output[result['type']] = int(result['value'])
                else:
                    print "Invalid value specified for", result['type']
            except TypeError:
                print "Invalid value specified for", result['type']
        elif result['type'] == 'server' or result['type'] == 'savedir' or
result['type'] == 'update':
            output[result['type']] = result['value']

    filehandler.close()

    return output

def loadPlaylistFromSaved(): # Load the previously saved playlist, if it
exists. Then fill any space remaining with newly-added tracks.
    global playlistlen
    global client
    global host
    global port
    global savedir

    loaded = []
    try:
        filehandler = open(savedir + 'playlist')
        for line in filehandler:
            loaded.append(line)
```

```
        filehandler.close()

        client.connect(host, port)
        for track in loaded:
            track = track.strip()
            if len(track) > 0:
                try:
                    client.add(track)
                    print "Loading", track
                except mpd.CommandError:
                    print "Error loading", track
        client.disconnect()

        for i in range(len(loaded), playlistlen):
            addNewTrackToPlaylist()
    except IOError:
        for i in range(0, playlistlen):
            addNewTrackToPlaylist()

def populateLists(redoing): # Parse the configuration file, and grab the
tracks from MPD to populate the lists.
    global playlistlen
    global changeafter
    global clearinitially
    global client
    global host
    global port
    global alltracks
    global saveonquit
    global savedir
    global update
    global oldconfig
    config = parseConfigFile()
    rules  = config['rules']
    if redoing == False or (redoing == True and not oldconfig == config):
        host           = config['server']
        port           = config['port']
        playlistlen    = config['playlistlen']
        changeafter    = config['changeafter']
        clearinitially = config['clearinitially']
        saveonquit     = config['saveonquit']
        savedir        = config['savedir']
        update         = config['update']
        oldconfig      = config
        print "Configuration updated:", config
    tracks = getFilenamesFromMPD(rules)
    if redoing == True:
        alltracks = []
    if redoing == False or not alltracks == tracks:
        alltracks = tracks
```

```
    client.connect(host, port)
    if clearinitially == 'yes' and redoing == False:
        client.clear()
    client.random(0)
    client.disconnect()

    if redoing == False:
        if saveonquit == 'no':
            for i in range(0, playlistlen):
                addNewTrackToPlaylist()
        else:
            loadPlaylistFromSaved()

        client.connect(host, port)
        client.play()
        client.disconnect()

def dieGracefully():
    global saveonquit
    if saveonquit == 'yes':
        try:
            os.remove('/var/lib/mpddp/playlist')
            print "Removed old playlist..."
        except OSError:
            print "No old playlist to remove."

        try:
            os.remove('/tmp/killmpddp')
            print "Removed kill file..."
        except OSError:
            print "No kill file to remove."

        print "Saving playlist to", savedir, "playlist"
        filehandler = open(savedir + 'playlist', 'w')
        playlist = client.playlistinfo()
        for track in playlist:
            print "Writing", track['file']
            filehandler.write(track['file'] + '\n')
        filehandler.close()

    print "Quitting..."
    sys.exit()

# Execute the program main loop
populateLists(False)
loops = 0
try:
    while True:
        if os.path.exists('/tmp/killmpddp'):
            dieGracefully()
        client.connect(host, port)
```

```
        info = client.currentsong()
        status = client.status()
        playlist = client.playlistinfo()
        client.disconnect()
        if len(info) > 0:
            if int(status['song']) >= changeafter:
                for i in range(changeafter - 1, int(status['song'])):
                    updatePlaylist()
        if len(playlist) < playlistlen:
            for i in range(len(playlist), playlistlen):
                addNewTrackToPlaylist()
        if loops == 59:
            if update == 'yes':
                populateLists(True)
            loops = 0
        else:
            loops = loops + 1
        time.sleep(1)
except KeyboardInterrupt:
    dieGracefully()
```

/etc/mpddp.conf.example:

```
server         = localhost       # The server that MPD is operating upon.
port           = 6600            # The port that MPD is operating upon.
playlistlen    = 15              # The number of tracks to have in the
playlist.
changeafter    = 8               # The number of tracks listened to
initially before the add/remove loop begins.
clearinitially = no              # Whether to clear the playlist upon
starting or not.
saveonquit     = no              # Whether to save/load the playlist upon
exit/start or not.
savedir        = /var/lib/mpddp/ # The directory to save/load the playlist.
update         = no              # Periodically check to ensure that the
config file hasn't been updated, and that no tracks have been added/removed.

#include /home/USER/.mpddp       # An additional config file to parse. I
suggest you use this to specify tracks to listen to

#path:PATH                       # Add all tracks where the file path
contains PATH.
#playlist:PLAYLIST               # Add all the playlists where the name is
PLAYLIST.
#smart:RULES                     # Add all the tracks which match the smart
playlist RULES. Requires MPDSPL to be in your $PATH.
#never:STRING                    # Never add tracks where the file path
contains STRING.
```

And, for reference, here's a small chunk of my config file:

```
# -*-conf-*-

playlistlen    = 50 # For full-screen
changeafter    = 26
#playlistlen    = 19 # For half-screen
#changeafter    = 10
clearinitially = no

# Playlists
#playlist:Wordless
#playlist:Steamcowboy

# Musicals
#smart:fp=/(Cats Original London Cast|Joseph and the Amazing Technicolour
Dreamcoat|Les Misérables .*|Mary Poppins|Miss Saigon .*|Phantom Of The Opera
.*|Sound of Music 40th Anniversary Special Edition, The).*\//
#path:Cats Original London Cast/
#path:Joseph and the Amazing Technicolour Dreamcoat/
#path:Les Misérables Complete Symphonic Recording/
#path:Les Misérables Original Broadway Cast/
#path:Les Misérables Original Paris Cast/
#path:Mary Poppins/
#path:Miss Saigon CSR/
#path:Miss Saigon (Original London Cast)/
#path:Phantom Of The Opera 2004 Film, The/
#path:Phantom of the Opera Original London Cast, The/
#path:Sound of Music 40th Anniversary Special Edition, The/
```

MPDSPL: MPD Smart PlayLists makes smart playlists for MPD. See the “smart:” line in my config file above? that's an example of a MPDSPL playlist description. Playlist descriptions are in regex, and use keywords (“ar” for artist, “al” for album, etc).

Source:

```
#!/usr/bin/env python

# A script to parse the MPD database into a list of dictionaries (or at
least, it was going to be before I decided to finish it).
# Now with patronising comments which assume almost no Python knowledge!

# cPickle is a faster version of the pickle library. It is used to save data
structures to a file. Like lists and dictionaries. os is needed for file
stuff, sys for arguments, and re for regex.
import cPickle, os, sys, re

# Info about new playlists
newname  = ""
newrules = []

# Place to look for the MPD database and config files, and the loaded MPD
config (well, only the values useful to us).
```

```
confpath = "/etc/mpd.conf"
mpd      = {"music_directory" : "", "playlist_directory" : "", "db_file" :
"", "user" : ""}

# There is an environmental variable XDG_CACHE_HOME which specifies where to
save cache files. However, if not set, a default of ~/.cache should be used.
cachehome = os.path.expanduser(os.environ['XDG_CACHE_HOME'])
if cachehome == "":
    cachehome = os.environ['HOME'] + "/.cache"
cachepath = cachehome + "/mpdspl/mpddb.cache"

# $XDG_DATA_HOME specifies where to save data files. Like a record of
playlists which have been created. If unset a default of ~/.local/share
should be used. This is currently unused as there is no actual creation of
playlists yet :p
datahome = os.path.expanduser(os.environ['XDG_DATA_HOME'])
if datahome == "":
    datahome = os.environ['HOME'] + "/.local/share/"
datapath = datahome + "/mpdspl"
# If the data directory does not exist, create it.
if not os.path.isdir(datapath):
    os.mkdir(datapath)

tracks = []
forceupdate  = False
simpleoutput = False

# A nice little help function. Read on to see how it is called...
def showhelp():
    print "Usage: mpdspl [options]\n"
    print "A script to generate smart playlists for MPD. Currently does
nothing of use :p\n"
    print "Options:"
    print "    -f, --force             - Force an update of the cache file
and any playlists."
    print "    -dFILE, --dbpath=FILE    - Location of the database file."
    print "    -cFILE, --cachepath=FILE - Location of the cache file."
    print "    -CFILE, --confpath=FILE  - Location of the MPD config file."
    print "    -uUSER, --mpduser=USER   - Location of the MPD config file."
    print "    -n, --new [name] [rules] - Create a new playlist."
    print "    -s, --simple             - (used with -n) Only print the
final track list (with paths relative to the MPD root dir) to STDOUT."
    print "    -h, --help               - Display this text and exit.\n"
    print "Playlist rules:"
    print "    These are specified as a string of Python-compatible regular
expressions separated by keywords, spaces, and slashes."
    print "    They are matched by re.search, not re.match, and no special
flags are passed, other than re.IGNORECASE when requested.\n"
    print "    These keywords are:"
    print "        ar = Artist"
    print "        al = Album"
```

```
    print "        ti = Title"
    print "        tr = Track Number"
    print "        ge = Genre"
    print "        ye = Year"
    print "        le = Length (seconds)"
    print "        fp = File Path (relative to MPD root dir, including
filename)"
    print "        fn = File Name\n"
    print "    Regular expressions are specified within slashes (/regex/)."
    print "    If the first slash is preceeded by an 'i', the regular
expression is interpreted as case-insensitive."
    print "    If the final slash is succeeded by a 'n', the result of the
match is negated.\n"
    print "    For example, a rule for all tracks by 'Fred' or 'George',
which have a title containing (case insensitive) 'The' and 'and', but not
'when' would be:"
    print "        ar=/(Fred|George)/ ti=i/(the.*and|and.*the)/
ti=i/when/n\n"
    print "Notes:"
    print "    Paths specified in the MPD config file containing a '~' will
have the '~'s replaced by the user MPD runs as."
    print "    If the user is not specified in the MPD config file, or by
the -u parameter, it is assumed the user is root."
    print "    Backslashes must be escaped in playlist rules.\n"
    sys.exit()

# Parse the rules regex
def parserules(rulestr):
    # rules will be our list of rules, bufferstr will be the buffer for our
parser, and i will be a counter
    rules     = []
    bufferstr = ""
    i         = 0

    # We want to use the same identifiers as the track dictionaries:
    keywords = {"ar" : "Artist", "al" : "Album", "ti" : "Title", "tr" :
"Track", "ge" : "Genre", "ye" : "Date", "le" : "Time", "fp" : "file", "fn" :
"key"}

    # For every character in rulestr (we do it characterwise, hence needing
a buffer)
    for c in rulestr:
        # Add the character to the buffer
        bufferstr += c

        # If the buffer matches one of our keywords, we have hit a new rule,
and so create a blank dictionary, and clear the buffer.
        if bufferstr.strip() in ["ar", "al", "ti", "tr", "ge", "ye", "le",
"fp", "fn"]:
            rules.append({"type" : keywords[bufferstr.strip()], "regex" :
"", "compiled" : None, "inverse" : False, "negate" : False})
```

```
            bufferstr = ""
        # If we're at the start of a blank case-insensitive regex, record
that, and clear the buffer.
        elif bufferstr == "=i/":
            rules[i]["i"] = True
            bufferstr = ""
        # If not, just clear the buffer for the coming regex.
        elif bufferstr == "=/":
            bufferstr = ""
        # If at the end of a regex, stick it all (sans the trailing slash,
they're just a nice separater for our parser) to the dictionary, increment
the counter, and clear the buffer ready for the next rule.
        elif bufferstr[-1] == "/" and not bufferstr[-2] == "\\":
            rules[i]["regex"] = bufferstr[:-1]
            bufferstr = ""
            i += 1
        # Get rid of the escape backslash if a forward slash has been used.
        elif bufferstr[-1] == "/" and not bufferstr[-2] == "\\":
            bufferstr[-2] = ""
        # If set to 'n' and the regex has been set, negate it.
        elif bufferstr == "n" and not rules[i - 1]["regex"] == "":
            bufferstr = ""
            rules[i - 1]["negate"] = True

    # This isn't needed. But it makes things faster and allows us to have
case insensetivity.
    for rule in rules:
        regex = None
        if rule["inverse"]:
            # If case insensitive, compile it as such.
            regex = re.compile(rule["regex"], re.IGNORECASE)
        else:
            regex = re.compile(rule["regex"])

        # Overwrite the regex string with the compiled object
        rule["compiled"] = regex

    return rules

# Splitting things up into functions is good :D
def parseargs():
    # global lets us access variables specified outside our function.
    global forceupdate
    global mpd
    global confpath
    global cachepath
    global newname
    global newrules
    global simpleoutput
    newarg = 0
    for argument in sys.argv:
```

```
        if not newarg == 0:
            # We're making a new playlist. If we're only on the first option
after -n, that's the name. If the second, that's the description.
            if newarg == 2:
                newname = argument
            elif newarg == 1:
                newrules = parserules(argument)
            newarg -= 1
        else:
            if argument == "-f" or argument == "--force":
                # If a "-f" or "--force" parameter is sent, force the cache
to be updated even if it doesn't look like it needs to be.
                forceupdate = True
            elif argument[:2] == "-d" or argument[:9] == "--dbpath=":
                # Looks like their db is somewhere other than
/var/lib/mpd/mpd.db...
                if argument[:2] == "-d":
                    # Python can't work with ~, which has a reasonable
chance of being used (eg: ~/.mpd/mpd.db"), so it needs to be expanded.
                    mpd["db_file"] = os.path.expanduser(argument[2:])
                elif argument[:9] == "--dbpath=":
                    mpd["db_file"] = os.path.expanduser(argument[9:])
            elif argument[:2] == "-c" or argument[:12] == "--cachepath=":
                # Silly person, not keeping their cache where XDG says it
should be...
                if argument[:2] == "-c":
                    cachepath = os.path.expanduser(argument[2:])
                elif argument[:12] == "--cachepath=":
                    cachepath = os.path.expanduser(argument[12:])
            elif argument[:2] == "-C" or argument[:11] == "--confpath=":
                # Now any person which this code applies to is just awkward.
                if argument[:2] == "-C":
                    confpath = os.path.expanduser(argument[2:])
                elif argument[:11] == "--confpath=":
                    confpath = os.path.expanduser(argument[11:])
            elif argument[:2] == "-u" or argument[:10] == "--mpduser=":
                # As is any person to whom this applies...
                if argument[:2] == "-u":
                    mpd["user"] = argument[2:]
                elif argument[:10] == "--mpdpath=":
                    mpd["user"] = argument[10:]
            elif argument == "-n" or argument == "--new":
                # Do special treatment to the next 2 arguments
                newarg = 2
            elif argument == "-s" or argument == "--simple":
                # Ooh, this means that (probably) MPDDP is being used! Yay!
                simpleoutput = True
            elif argument == "-h" or argument == "--help":
                showhelp()
            elif not argument == sys.argv[0]: # The first argument is the
filename. Don't complain about not understanding it...
```

```
                # Ooh, stderr. I never actually knew how to send stuff
through stderr in python.
                print >> sys.stderr, "Unrecognised parameter '" + argument +
"'"
                sys.exit(1)

# A function to parse a MPD database and make a huge list of tracks
def parsedatabase(database):
    global tracks
    i        = -1
    parsing  = False

    for line in database:
        # For every line in the database, remove any whitespace at the
beginning and end so the script isn't buggered.
        line = line.strip()

        # If entering a songList, start parsing. If exiting one, stop.
Fairly self explanatory.
        if not parsing and line == "songList begin":
            parsing = True
        elif parsing and line == "songList end":
            parsing = False

        # If we get a line to parse which is not a "songList begin"
statement (because it's be stupid to do things with that)
        if parsing and not line == "songList begin":
            if line[0:5] == "key: ":
                i += 1
                # Increment the counter and make an empty dictionary if we
hit the beginning of a track
                tracks.append({"key" : "", "file" : "", "Time" : "", "Genre"
: "", "Title" : "", "Artist" : "", "Date" : "", "Album" : "", "Track" : "",
"mtime" : ""})

            # Split the line by the first ": ", the string MPD uses, and
stick the second part (the value) in the bit of the dictionary referred to
by the first part (the key)
            splitted = line.split(": ", 1)
            tracks[i][splitted[0]] = splitted[1]

# Grabbing stuff from the MPD config, a very important step
def parsempdconf():
    global confpath
    global mpd
    config   = open(confpath, "r")
    # Don't load the user or db_file values if they've already been told to
us
    holduser = not mpd["user"] == ""
    holddb   = not mpd["db_file"] == ""
```

```
    for line in config:
        line = line.strip()
        if line[:15] == "music_directory":
            rest = line[15:].strip()
            mpd["music_directory"] = rest[1:-1]
        elif line[:18] == "playlist_directory":
            rest = line[18:].strip()
            mpd["playlist_directory"] = rest[1:-1]
        elif line[:7] == "db_file" and not holddb:
            rest = line[7:].strip()
            mpd["db_file"] = rest[1:-1]
        # The rest of the code in this function wouldn't be needed if I
could assume nobody would use "~" in their MPD config...
        elif line[:4] == "user" and not holduser:
            rest = line[4:].strip()
            mpd["user"] = rest[1:-1]

    if mpd["user"] == "":
        mpd["user"] = "root"

    homedir = "/home/" + mpd["user"]
    if homedir == "/home/root":
        homedir = "/root"

    if "~" in mpd["music_directory"]:
        mpd["music_directory"] = mpd["music_directory"].replace("~",
homedir)
    if "~" in mpd["playlist_directory"]:
        mpd["playlist_directory"] = mpd["playlist_directory"].replace("~",
homedir)
    if "~" in mpd["db_file"]:
        mpd["db_file"] = mpd["db_file"].replace("~", homedir)

def findtracks():
    global tracks
    global newrules
    # matchingtracks will hold all tracks which match all of the criteria.
    matchingtracks = []
    for track in tracks:
        # Initially assume a track *will* be added.
        addtrack = True
        for rule in newrules:
            # For every track, check it with every rule
            if rule["negate"]:
                if not re.search(rule["compiled"], track[rule["type"]]) ==
None:
                    # If the regular expression matches the track, do not
add it to the matchingtracks list.
                    addtrack = False
            else:
                if re.search(rule["compiled"], track[rule["type"]]) == None:
```

```
                    # If the regular expression does not match the track, do
not add it to the matchingtracks list.
                    addtrack = False
        if addtrack:
            # Add the track if appropriate
            matchingtracks.append(track)

    return matchingtracks

def genplaylist(tracks):
    global mpd
    # Parse a list of track dictionaries into a playlist. Thankfully, m3u is
a *very* simple format.
    playlist = ""
    for track in tracks:
        playlist += mpd["music_directory"] + "/" + track["file"] + "\n"

    return playlist

# Save some random gubbage to a file
def savegubbage(data, path):
    if not os.path.isdir(os.path.dirname(path)):
        os.mkdir(os.path.dirname(path))

    # Open the file for writing in binary mode
    outfile = open(path, "wb")
    # Send the stuff to the file with the magic of cPickle
    cPickle.dump(data, outfile)
    # Close the file handler. Tidy u[p.
    outfile.close()

    # We might be running as someone other than the user, so make the file
writable
    os.chmod(path, 438)

def loadgubbage(path):
    infile = open(path, "rb")
    data = cPickle.load(infile)
    infile.close()

    return data

def saveplaylist():
    global newname
    global newrules
    global mpd
    global datapath
    global simpleoutput
    matchingtracks = findtracks()
    playlist = genplaylist(matchingtracks)
```

```
    if simpleoutput:
        for track in matchingtracks:
            print track["file"]
    else:
        print "Saving playlist '" + newname + "'."
        # Write the contents of the playlist to the m3u file
        newlist = open(mpd["playlist_directory"] + "/" + newname + ".m3u",
"w")
        newlist.write(playlist)
        newlist.close()
        # Save as list object. This lets us load them all into a big list
nicely.
        savegubbage([newname, newrules], datapath + "/" + newname)


# Parse some options!
parseargs()
parsempdconf()

# Check that the database is actually there before attempting to do stuff
with it.
if not os.path.exists(mpd["db_file"]):
    print >> sys.stderr, "The database file '" + mpd["db_file"] + "' could
not be found."
    sys.exit(1)

# If the cache file does not exist OR the database has been modified since
the cache file has this has the side-effect of being able to touch the cache
file to stop it from being updated. Good thing we have the -f option for any
accidental touches (or if you copy the cache to a new location).
if not os.path.exists(cachepath) or os.path.getmtime(mpd["db_file"]) >
os.path.getmtime(cachepath) or forceupdate:
    if not simpleoutput:
        print "Updating database cache..."

    # If the cache directory does not exist, create it. The dirname function
just removes the "/mpddb.cache" from the end.
    if not os.path.isdir(os.path.dirname(cachepath)):
        os.mkdir(os.path.dirname(cachepath))

    database = open(mpd["db_file"], "r")

    # Now, parse that database!
    parsedatabase(database)
    # Save the parsed stuff to the cache file and close the database file
handler. That's not strictly required, python will clean up when the script
ends, but you can't unmount volumes with file handlers pointing to them, so
it makes a mess.
    savegubbage(tracks, cachepath)
    database.close()
    if not simpleoutput:
```

```
        # Let's update those playlists!
        playlistfiles = os.listdir(datapath)
        playlists     = []
        for playlistfile in playlistfiles:
            playlists.append(loadgubbage(datapath + "/" + playlistfile))
        # Backup the values first.
        oldnewname  = newname
        oldnewrules = newrules
        # Now regenerate!
        for playlist in playlists:
            newname  = playlist[0]
            newrules = playlist[1]
            saveplaylist()

        # And restore.
        newname  = oldnewname
        newrules = oldnewrules
else:
    # Oh, goodie, we don't need to go through all that arduous parsing as we
have a valid cache file :D
    if not simpleoutput:
        print "Loading database cache..."
    # Open it for reading, load the stuff in the file into the tracks list,
close the file handler, and have a party.
    tracks = loadgubbage(cachepath)
# See if we're making a new playlist or not
if not newname == "":
    # We are, go go go!
    saveplaylist()
```

The source is full of simple comments because I was supposed to be helping a friend who's less proficient in Python make it but... I got bored of waiting tongue If your MPD playlists directory is somewhere which you don't have write access to, run it with `sudo -E`

If you need any help with either, ask and ye shall receive. If what you need help with is covered in this post, mpdspl -h, or /etc/mpddp.conf.example, ask and ye shall be laughed at tongue

Github projects

https://github.com/TestDotCom/pirateplayer
https://github.com/Bit-River/Home-Assistant-Gadget-using-Pirate-Audio
https://github.com/kenhayward/PirateAudio https://github.com/duracell80/PirateAudio-MPD
https://github.com/hakutai/PirateAudioVolumio
https://github.com/createcodeandgo/audiobookplayer/tree/main/player
https://github.com/placebo83/pirate-audio/blob/master/test.py
https://raw.githubusercontent.com/jbenc/kodi-pirate-audio/master/script.service.pirate-audio/resources/lib/main.py https://github.com/mtbkapp/kids_pirate_audio
https://github.com/Zurga/pimoroni_pirate_audio https://github.com/traveltrousers/pirate-audio-buttons
https://github.com/tiradoe/pirate-audio-display

https://github.com/DrewBatchelor/Pirate-Audio-Volumio-box
https://github.com/promethee/pimoroni.pirate-audio.dual-mic

https://www.bluetin.io/displays/simulate-oled-lcd-display-pc-pil-opencv/