PyPortal WFH Busy Sounds Simulator

Created by Tim C

https://learn.adafruit.com/pyportal-wfh-busy-sounds-simulator

Last updated on 2022-02-07 06:50:52 PM EST
# Table of Contents

## Overview
- Parts 3

## External Speaker

## CircuitPython Setup
- Drive Structure 6
- Code 7

## Code Walk-Through
- Configuration Variable 11
- Program Variables 11
- Icons List 12
- Helper Functions 12
- User Interface 12
- Main Loop 13
- Further Detail 13
Overview

Use your PyPortal to play various notification sounds on a loop. Folks will think you are mighty busy with all of those jingly noises going off!

Great for when you are "WFH" but really "TAN" (taking a nap)!

The loop of notification sounds will play automatically. Touch a notification icon to add it to the loop. Touch the stop icon to stop playing and empty the loop.

Parts

Adafruit PyPortal - CircuitPython Powered Internet Display
PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116

Adafruit PyPortal Titano
The PyPortal Titano is the big sister to our popular PyPortal now with twice as many pixels! The PyPortal...
https://www.adafruit.com/product/4444
Adafruit PyPortal Desktop Stand Enclosure Kit
PyPortal is our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Create little pocket...
https://www.adafruit.com/product/4146

Mini Oval Speaker with Short Wires - 8 Ohm 1 Watt
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/4227

Mini Oval Speaker - 8 Ohm 1 Watt
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/3923
External Speaker

The PyPortal does have a built-in speaker but it's fairly small. It may be easier for people to "overhear" your notifications if you plug in a bigger external speaker.

Luckily the PyPortal contains a plug-in adapter for just that!

If you do want to use the external speaker you must cut a small jumper connection on the board in order to prevent the built-in speaker from playing in addition to the external one.
CircuitPython Setup

Plug the PyPortal into your computer with a known good USB cable (not a tint charge only cable). The PyPortal will appear to your computer as a flash drive named CIRCUITPY.

Download the project files with the Download Project Bundle button below. Unzip the file and copy/paste the code.py and other project files to your CIRCUITPY drive using File Explorer or Finder (depending on your operating system).

Download Project Bundle

https://adafru.it/XJE

Drive Structure

After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum:
The program code is shown below:

```python
# SPDX-FileCopyrightText: 2020 Tim C, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
#
PyPortal implementation of Busy Simulator notification sound looper.
#
import time
import board
import displayio
import adafruit_touchscreen
from adafruit_displayio_layout.layouts.grid_layout import GridLayout
from adafruit_displayio_layout.widgets.icon_widget import IconWidget
from audiocore import WaveFile
from audioio import AudioOut

# How many seconds to wait between playing samples
```
# Lower time means it will play faster
WAIT_TIME = 3.0

# List that will hold indexes of notification samples to play
LOOP = []

# last time that we played a sample
LAST_PLAY_TIME = 0
CUR_LOOP_INDEX = 0

# touch events must have at least this long between them
COOLDOWN_TIME = 0.25  # seconds

# last time that the display was touched
# used for debouncing and cooldown enforcement
LAST_PRESS_TIME = -1

# Was any icon touched last iteration.
# Used for debouncing.
WAS_TOUCHED = False

def next_index!():
    try:
        return the next index in the LOOP that should get played
    except:
        if CUR_LOOP_INDEX + 1 >= len(LOOP):
            return 0

        return CUR_LOOP_INDEX + 1

# list of icons to show
# each entry is a tuple containing:
# (Icon Label, Icon BMP image file, Notification sample wav file
_icons = [
    ("Outlook", "icons/outlook.bmp", "sounds/outlook.wav"),
    ("Phone", "icons/phone.bmp", "sounds/phone.wav"),
    ("Skype", "icons/skype.bmp", "sounds/skype.wav"),
    ("Teams", "icons/teams.bmp", "sounds/teams.wav"),
    ("Discord", "icons/discord.bmp", "sounds/discord.wav"),
    ("Apple Mail", "icons/applemail.bmp", "sounds/applemail.wav"),
    ("iMessage", "icons/imessage.bmp", "sounds/imessage.wav"),
    ("Slack", "icons/slack.bmp", "sounds/slack.wav"),
    ("G Calendar", "icons/gcal.bmp", "sounds/gcal.wav"),
    ("G Chat", "icons/gchat.bmp", "sounds/gchat.wav"),
    ("Stop", "icons/stop.bmp", ""),
]

# Make the display context.
display = board.DISPLAY
main_group = displayio.Group()
display.show(main_group)

# Touchscreen initialization
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(display.width, display.height),
)

# Setup the file as the bitmap data source
bg_bitmap = displayio.OnDiskBitmap("busysim_background.bmp")

# Create a TileGrid to hold the bitmap
bg_tile_grid = displayio.TileGrid(
    bg_bitmap,
    pixel_shader=getattr(bg_bitmap, "pixel_shader", displayio.ColorConverter()),
)

# add it to the group that is showing
main_group.append(bg_tile_grid)

# grid to hold the icons
layout = GridLayout(
x=0,
y=0,
width=320,
height=240,
grid_size=(4, 3),
cell_padding=20,
)

# initialize the icons in the grid
for i, icon in enumerate(_icons):
    icon_widget = IconWidget(
        icon[0],
        icon[1],
x=0,
y=0,
on_disk=True,
        transparent_index=0,
        label_background=0x888888,
    )

    layout.add_content(icon_widget, grid_position=(i % 4, i // 4), cell_size=(1, 1))

# add the grid to the group showing on the display
main_group.append(layout)

def check_for_touch(_now):
    """Check the touchscreen and do any actions necessary if an icon has been touched. Applies debouncing and cool down enforcement to filter out unneeded touch events."
    # pylint: disable=global-statement, too-many-nested-blocks, consider-using-
    # enumerate
    global CUR_LOOP_INDEX
    global LOOP
    global LAST_PRESS_TIME
    global WAS_TOUCHED

    # read the touch data
touch_point = ts.touch_point

    # if anything is touched
    if touch_point:
        # if the touch just began. We ignore further events until
        # after the touch has been lifted
        if not WAS_TOUCHED:
            # set the variable so we know to ignore future events until
            # touch is released
            WAS_TOUCHED = True

        # if it has been long enough time since previous touch event
        if _now - LAST_PRESS_TIME > COOLDOWN_TIME:
            LAST_PRESS_TIME = time.monotonic()
# loop over the icons
for _ in range(len(_icons)):
    # lookup current icon in the grid layout
    cur_icon = layout.get_cell((_ % 4, _ // 4))

    # check if it's being touched
    if cur_icon.contains(touch_point):
        print("Icon {} touched".format(_))

        # if it's the stop icon
        if _icons[_][0] == "Stop":
            # empty out the loop
            LOOP = []

            # set current index back to 0
            CUR_LOOP_INDEX = 0

        else:  # any other icon
            # insert the touched icons sample index into the loop
            LOOP.insert(CUR_LOOP_INDEX, _)

    # print(LOOP)

    # break out of the for loop.
    # if current icon is being touched then no others can be
    break

else:
    # nothing is touched
    # set variable back to false for debouncing
    WAS_TOUCHED = False

# main loop
while True:
    # store current time in variable for cool down enforcement
    _now = time.monotonic()

    # check for and process touch events
    check_for_touch(_now)

    # if it's time to play a sample
    if LAST_PLAY_TIME + WAIT_TIME <= _now:
        print("time to play")

        # if there are any samples in the loop
        if len(LOOP) > 0:

            # open the sample wav file
            with open(_icons[LOOP[CUR_LOOP_INDEX]][2], "rb") as wave_file:
                print("playing: {}".format(_icons[LOOP[CUR_LOOP_INDEX]][2]))

                # initialize audio output pin
                audio = AudioOut(board.AUDIO_OUT)

                # initialize WaveFile object
                wave = WaveFile(wave_file)

                # play it
                audio.play(wave)

                # while it's still playing
                while audio.playing:
                    # update time variable
                    _now = time.monotonic()

                    # check for and process touch events
                    check_for_touch(_now)
# after done playing, deinit audio output
audio.deinit()

# increment index counter
CUR_LOOP_INDEX = next_index()

# update variable for last time we attempted to play sample
LAST_PLAY_TIME = _now

---

**Code Walk-Through**

**Configuration Variable**

You can update the `WAIT_TIME` variable to a different value if you want. This variable represents the amount of time in seconds between each notification sample playback. Lower times means it will play faster, higher times means it will play slower. The default is `3.0`, but feel free to experiment with different times.

```python
WAIT_TIME = 3.0
```

**Program Variables**

These variables are used for various things in the program, they aren't intended for you to change them during the normal operation. The program will update their value when necessary.

```python
# List that will hold indexes of notification samples to play
LOOP = []

# last time that we played a sample
LAST_PLAY_TIME = 0

# current index that we are on
CUR_LOOP_INDEX = 0

# touch events must have at least this long between them
COOLDOWN_TIME = 0.25  # seconds

# last time that the display was touched
# used for debouncing and cooldown enforcement
LAST_PRESS_TIME = -1

# Was any icon touched last iteration.
# Used for debouncing.
WAS_TOUCHED = False
```
Icons List

This list holds tuples that represent each icon that will be shown on the screen. The tuples contain the string label, the bmp image file to show, and the wave audio file to play. If you wanted to re-purpose this project as a looping soundboard with other sounds, you could change the values here.

```python
# list of icons to show
# each entry is a tuple containing:
# (Icon Label, Icon BMP image file, Notification sample wav file
_icons = [
    ("Outlook", "icons/outlook.bmp", "sounds/outlook.wav"),
    ("Phone", "icons/phone.bmp", "sounds/phone.wav"),
    ("Skype", "icons/skype.bmp", "sounds/skype.wav"),
    ("Teams", "icons/teams.bmp", "sounds/teams.wav"),
    ("Discord", "icons/discord.bmp", "sounds/discord.wav"),
    ("Apple Mail", "icons/applemail.bmp", "sounds/applemail.wav"),
    ("iMessage", "icons/imessage.bmp", "sounds/imessage.wav"),
    ("Slack", "icons/slack.bmp", "sounds/slack.wav"),
    ("G Calendar", "icons/gcal.bmp", "sounds/RE.wav"),
    ("G Chat", "icons/gchat.bmp", "sounds/gchat.wav"),
    ("Stop", "icons/stop.bmp", ""),
]
```

Helper Functions

The program contains two helper functions:

- **next_index()** - This function will return the next valid index for a notification sample in the _LOOP_ list. If the current index is the last available one, then it will wrap back around to 0.

- **check_for_touch(_now)** - This function accepts a parameter whose value will be the current time as fetched from `time.monotonic()`. It will check the touch screen to see if there are any touch events on icons currently, and take the appropriate action if there are. The time is used to enforce a cool-down behavior so it will not register several touch events rapidly. This will get called during the main loop, and during time that an audio sample is playing.

User Interface

The GUI is created with `displayio`. It uses a `GridLayout` object loaded up with IconWidgets from a for loop that references the above icons list.

```python
# grid to hold the icons
layout = GridLayout(
```
Main Loop

The main loop contains two high level actions:

1) Check if the user has touched any icons. If they touch a notification icon, the index for the one they touched gets added to the `LOOP` list. If they touch the stop icon, the `LOOP` list is emptied and the `CUR_LOOP_INDEX` reset to 0.

2) Check the current time, if the `WAIT_TIME` has passed then load and play the current sample from the `LOOP` list. Set the index to the next one for next time. During the time that a sample is being played there is an internal loop that will carry out action 1, so that touch events during the sample playback get handled appropriately.

Further Detail

The source code is thoroughly commented to explain what the statements are doing. You can read through the code and comments to gain a deeper understanding of how it functions or modify parts of it to suit your needs.

```python
# SPDX-FileCopyrightText: 2020 Tim C, written for Adafruit Industries
# SPDX-License-Identifier: Unlicense

import time
import board
import displayio
import adafruit_touchscreen
from adafruit_displayio_layout.layouts.grid_layout import GridLayout
from adafruit_displayio_layout.widgets.icon_widget import IconWidget
from audiocore import WaveFile
from audioio import AudioOut
```
# How many seconds to wait between playing samples
# Lower time means it will play faster
WAIT_TIME = 3.0

# List that will hold indexes of notification samples to play
LOOP = []

# last time that we played a sample
LAST_PLAY_TIME = 0
CUR_LOOP_INDEX = 0

# touch events must have at least this long between them
COOLDOWN_TIME = 0.25  # seconds

# last time that the display was touched
# used for debouncing and cooldown enforcement
LAST_PRESS_TIME = -1

# Was any icon touched last iteration.
# Used for debouncing.
WAS_TOUCHED = False

def next_index():
    """
    return the next index in the LOOP that should get played
    """
    if CUR_LOOP_INDEX + 1 >= len(LOOP):
        return 0

    return CUR_LOOP_INDEX + 1

# list of icons to show
# each entry is a tuple containing:
# (Icon Label, Icon BMP image file, Notification sample wav file
_icons = [
    ("Outlook", "icons/outlook.bmp", "sounds/outlook.wav"),
    ("Phone", "icons/phone.bmp", "sounds/phone.wav"),
    ("Skype", "icons/skype.bmp", "sounds/skype.wav"),
    ("Teams", "icons/teams.bmp", "sounds/teams.wav"),
    ("Discord", "icons/discord.bmp", "sounds/discord.wav"),
    ("Apple Mail", "icons/applemail.bmp", "sounds/applemail.wav"),
    ("iMessage", "icons/imessage.bmp", "sounds/imessage.wav"),
    ("Slack", "icons/slack.bmp", "sounds/slack.wav"),
    ("G Calendar", "icons/gcal.bmp", "sounds/RE.wav"),
    ("G Chat", "icons/gchat.bmp", "sounds/gchat.wav"),
    ("Stop", "icons/stop.bmp", ""),
]

# Make the display context.
display = board.DISPLAY
main_group = displayio.Group()
display.show(main_group)

# Touchscreen initialization
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YL,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(display.width, display.height),
)

# Setup the file as the bitmap data source
bg_bitmap = displayio.OnDiskBitmap("busysim_background.bmp")
# Create a TileGrid to hold the bitmap
bg_tile_grid = displayio.TileGrid(
    bg_bitmap,
    pixel_shader=getattr(bg_bitmap, "pixel_shader", displayio.ColorConverter()),
)

# add it to the group that is showing
main_group.append(bg_tile_grid)

# grid to hold the icons
layout = GridLayout(
x=0,
y=0,
width=320,
height=240,
grid_size=(4, 3),
cell_padding=20,
)

# initialize the icons in the grid
for i, icon in enumerate(_icons):
    icon_widget = IconWidget(
        icon[0],
        icon[1],
x=0,
y=0,
on_disk=True,
        transparent_index=0,
        label_background=0x888888,
    )
    layout.add_content(icon_widget, grid_position=(i % 4, i // 4), cell_size=(1, 1))

# add the grid to the group showing on the display
main_group.append(layout)

def check_for_touch(_now):
    """
    Check the touchscreen and do any actions necessary if an
    icon has been touched. Applies debouncing and cool down
    enforcement to filter out unneeded touch events.
    """
    # pylint: disable=global-statement, too-many-nested-blocks, consider-using-
    enumerate
    global CUR_LOOP_INDEX
    global LOOP
    global LAST_PRESS_TIME
    global WAS_TOUCHED

    # read the touch data
    touch_point = ts.touch_point

    # if anything is touched
    if touch_point:
        # if the touch just began. We ignore further events until
        # after the touch has been lifted
        if not WAS_TOUCHED:

            # set the variable so we know to ignore future events until
            # touch is released
            WAS_TOUCHED = True

        # if it has been long enough time since previous touch event
        if _now - LAST_PRESS_TIME > COOLDOWN_TIME:
LAST_PRESS_TIME = time.monotonic()

# loop over the icons
for _ in range(len(_icons)):
    # lookup current icon in the grid layout
    cur_icon = layout.get_cell((_ % 4, _ // 4))

    # check if it's being touched
    if cur_icon.contains(touch_point):
        print("icon {} touched".format(_))

        # if it's the stop icon
        if _icons[_][0] == "Stop":
            # empty out the loop
            LOOP = []

            # set current index back to 0
            CUR_LOOP_INDEX = 0
        else:
            # any other icon
            # insert the touched icons sample index into the loop
            if LOOP:
                LOOP.insert(CUR_LOOP_INDEX, _)

                # print(LOOP)

            # break out of the for loop.
            # if current icon is being touched then no others can be
            break

    else:
        # set variable back to false for debouncing
        WAS_TOUCHED = False

# main loop
while True:
    # store current time in variable for cool down enforcement
    _now = time.monotonic()

    # check for and process touch events
    check_for_touch(_now)

    # if it's time to play a sample
    if LAST_PLAY_TIME + WAIT_TIME <= _now:
        # print("time to play")

        # if there are any samples in the loop
        if len(LOOP) > 0:

            # open the sample wav file
            with open(_icons[LOOP[CUR_LOOP_INDEX]][2], "rb") as wave_file:
                print("playing: {}".format(_icons[LOOP[CUR_LOOP_INDEX]][2]))

            # initialize audio output pin
            audio = AudioOut(board.AUDIO_OUT)

            # initialize WaveFile object
            wave = WaveFile(wave_file)

            # play it
            audio.play(wave)

            # while it's still playing
            while audio.playing:
                # update time variable
                _now = time.monotonic()
# check for and process touch events
check_for_touch(_now)

# after done playing, deinit audio output
audio.deinit()

# increment index counter
CUR_LOOP_INDEX = next_index()

# update variable for last time we attempted to play sample
LAST_PLAY_TIME = _now