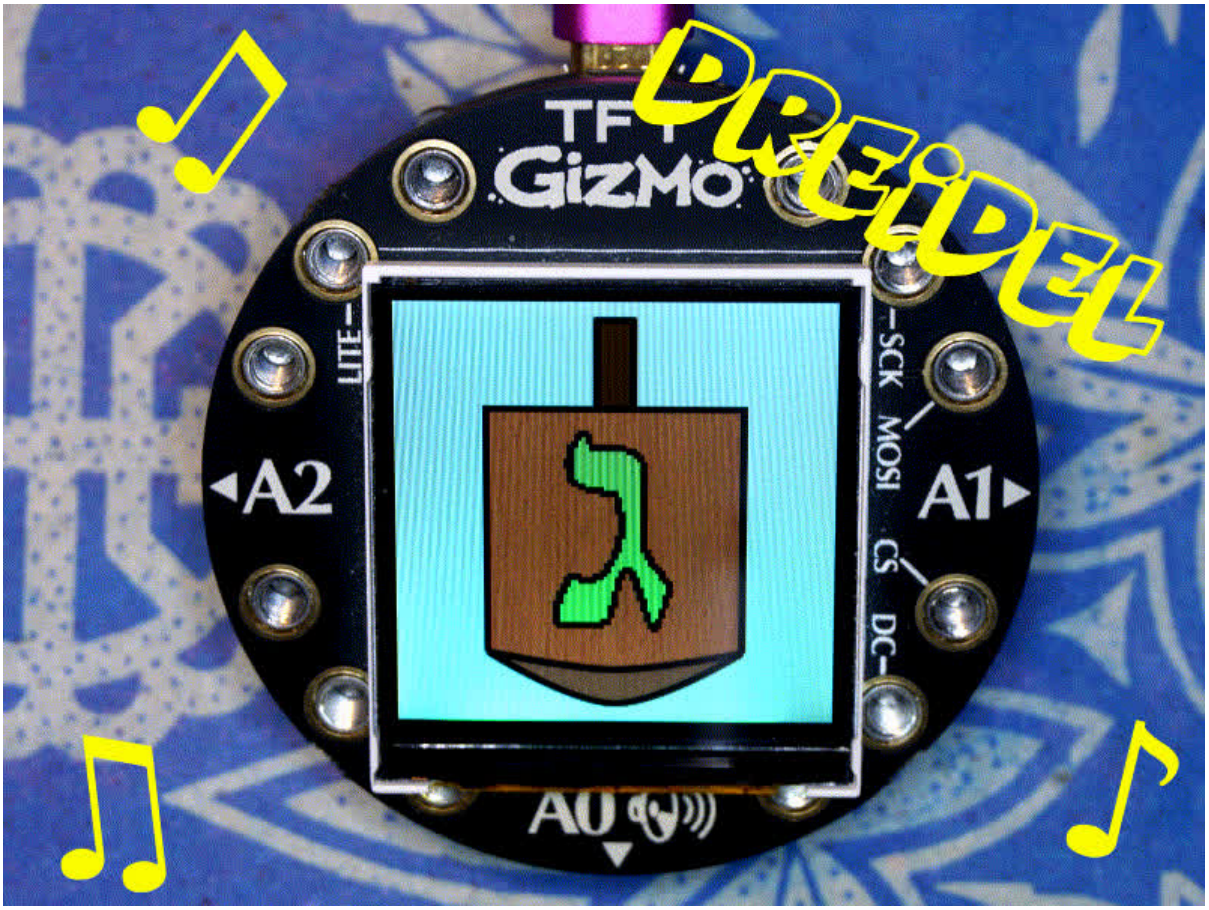
Circuit Playground TFT Gizmo Dreidel

Created by Carter Nelson



https://learn.adafruit.com/circuit-playground-tft-gizmo-dreidel

Last updated on 2024-06-03 02:59:33 PM EDT

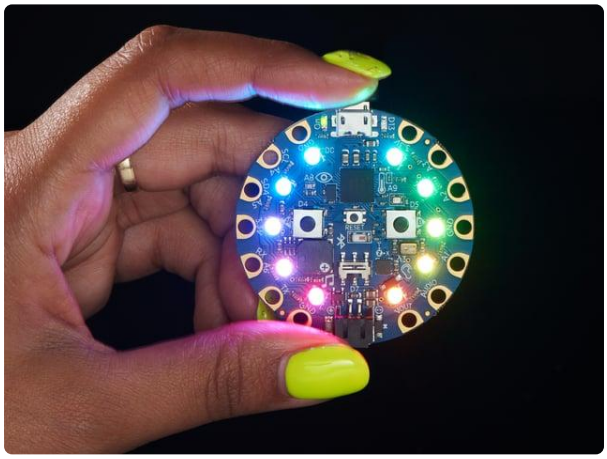
Table of Contents

Overview

Dreidel, dreidel, dreidel, I made you out of...circuits and code? Well, in this guide we will. This guide will show you how to use a Circuit Playground Bluefruit (http://adafru.it/4333) along with a TFT Gizmo (http://adafru.it/4367) to create a digital cyber dreidel.

Instead of spinning, you'll give your cyber dreidel a shake to set it into action. It will replicate the actual spinning top on the TFT Gizmo display. It even plays the catchy little Dreidel Song (https://adafru.it/HB5) while the display is "spinning". And then you get a random symbol to represent the dreidel landing on a side.

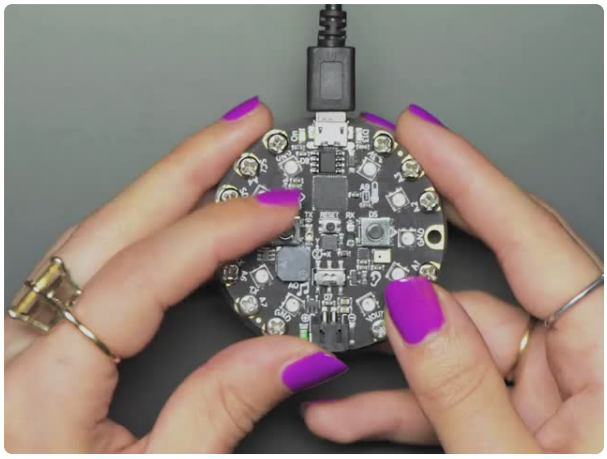
And...it's all programmed in CircuitPython (https://adafru.it/EFq). Let's get started.



Circuit Playground Bluefruit - Bluetooth Low Energy

Circuit Playground Bluefruit is our third board in the Circuit Playground series, another step towards a perfect introduction to electronics and programming. We've...

https://www.adafruit.com/product/4333



Circuit Playground TFT Gizmo - Bolt-on Display + Audio Amplifier
Extend and expand your Circuit Playground projects with a bolt on TFT Gizmo that lets you add a lovely color display in a sturdy and reliable fashion. This PCB looks just like a round...
https://www.adafruit.com/product/4367

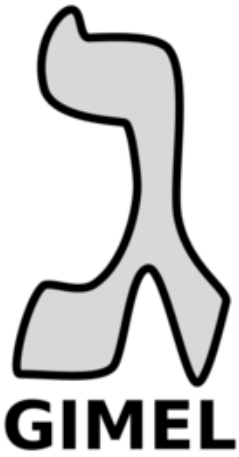
The Dreidel Game

First, a bit of background on the game itself. The rules of the dreidel game (https://adafru.it/HB6) are simple. You can play with any set of items for the pieces. It doesn't have to be money. It could be candy!

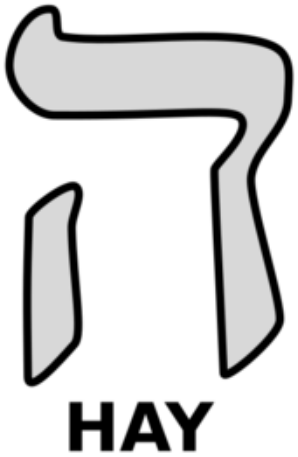
Everybody starts with the same number of pieces. Then, everybody puts one piece into the pot. Also, anytime there is one or zero pieces in the pot, everybody adds one piece.

Then, everybody takes turns spinning the dreidel. After the dreidel stops spinning and falls, one side will be facing up. This will decide what action you take as follows:

Nun ("nothing") - Do nothing



Gimel ("Gimme") - Take everything in the pot.



Hay ("half") - Take half of the pot.

Shin ("put in") - Put one piece in the pot.

When a player runs out of pieces, they are out of the game.

And that's it. Keep playing until someone walks away with a lot of candy. :)

Dreidel Code

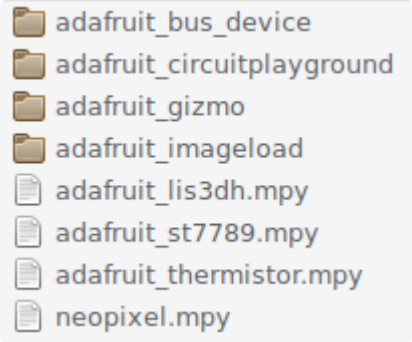
CircuitPython and Library Setup

The dreidel code is written in CircuitPython (https://adafru.it/cpy-welcome). So, first make sure you have the latest firmware installed, which can be downloaded from the CircuitPython website here:

Circuit Playground Bluefruit Firmware

https://adafru.it/FNK

Then make sure you have the following libraries installed. These should go in your **CIRCUITPY/lib** folder:



You can download the library bundle from here:

CircuitPython Libraries

https://adafru.it/ENC

Bitmap Image Files

Next, you'll need these two BMP files, which contain the dreidel background image and symbols. Put these in your root **CIRCUITPY** folder (same place the .py code will go):

dreidel_background.bmp

https://adafru.it/HB7

dreidel_sheet.bmp

https://adafru.it/HB8

Dreidel Code

And here's the code for the dreidel game. To have this run automatically, save it as **code.py** in your **CIRCUITPY** folder.

```
# SPDX-FileCopyrightText: 2019 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import random
import displayio
import adafruit_imageload
from adafruit_circuitplayground.bluefruit import cpb
from adafruit_gizmo import tft_gizmo

SHAKE_THRESHOLD = 20

#pylint: disable=bad-continuation
# define melody to play while spinning (freq, duration)
melody = (
#      oh       drei       del      drei      del
    (330, 8), (392, 8), (330, 8), (392, 8), (330, 8),
#     drei       del         I       made      it
    (392, 8), (330, 16), (330, 8), (392, 8), (392, 8),
#     out        of       clay
    (349, 8), (330, 8), (294, 16), (0, 8),
#      oh       drei       del      drei      del
    (294, 8), (349, 8), (294, 8), (349, 8), (294, 8),
#     drei        del      then      drei      del
    (349, 8), (294, 16), (294, 8), (392, 8), (349, 8),
#       I       shall      play
    (330, 8), (294, 8), (262, 16),
)
melody_tempo = 0.02
#pylint: enable=bad-continuation

# setup TFT Gizmo and main display group (splash)
display = tft_gizmo.TFT_Gizmo()
splash = displayio.Group()
display.root_group = splash

# load dreidel background image
dreidel_bmp, dreidel_pal = adafruit_imageload.load("/dreidel_background.bmp",
                                                   bitmap=displayio.Bitmap,
                                                   palette=displayio.Palette)
dreidel_tg = displayio.TileGrid(dreidel_bmp, pixel_shader=dreidel_pal)
splash.append(dreidel_tg)

# load dreidel symbols (sprite sheet)
symbols_bmp, symbols_pal = adafruit_imageload.load("/dreidel_sheet.bmp",
                                                   bitmap=displayio.Bitmap,
                                                   palette=displayio.Palette)
for index, color in enumerate(symbols_pal):
    if color == 0xFFFF01:
        symbols_pal.make_transparent(index)
symbols_tg = displayio.TileGrid(symbols_bmp, pixel_shader=symbols_pal,
                                width = 1,
                                height = 1,
                                tile_width = 60,
                                tile_height = 60)
symbols_group = displayio.Group(scale=2, x=60, y=70)
symbols_group.append(symbols_tg)
splash.append(symbols_group)
```

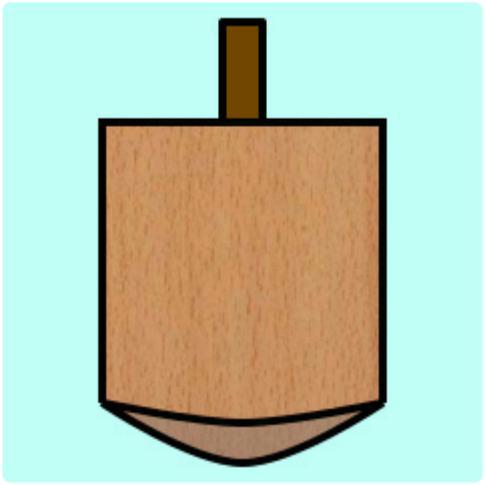
```
# dreidel time!
tile = 0
while True:
    # wait for shake
    while not cpb.shake(shake_threshold=SHAKE_THRESHOLD):
        pass
    # play melody while "spinning" the symbols
    for note, duration in melody:
        symbols_tg[0] = tile % 4
        tile += 1
        cpb.play_tone(note, duration * melody_tempo)
    # land on a random symbol
    symbols_tg[0] = random.randint(0, 3)
    # prevent immediate re-shake
    time.sleep(2)
```

How It Works

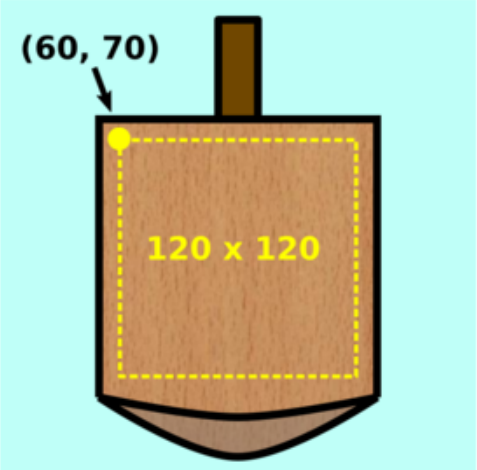
Here's a breakdown of how the dreidel code works. The use of a sprite sheet along with group scaling for the symbols is probably the most interesting from a displayio aspect. But we'll go over all the code.

Dreidel Background

First, a static background image is used to represent the dreidel itself. It looks like this full size:



Note it is 240x240 pixels, since that's the size of the TFT Gizmo display.



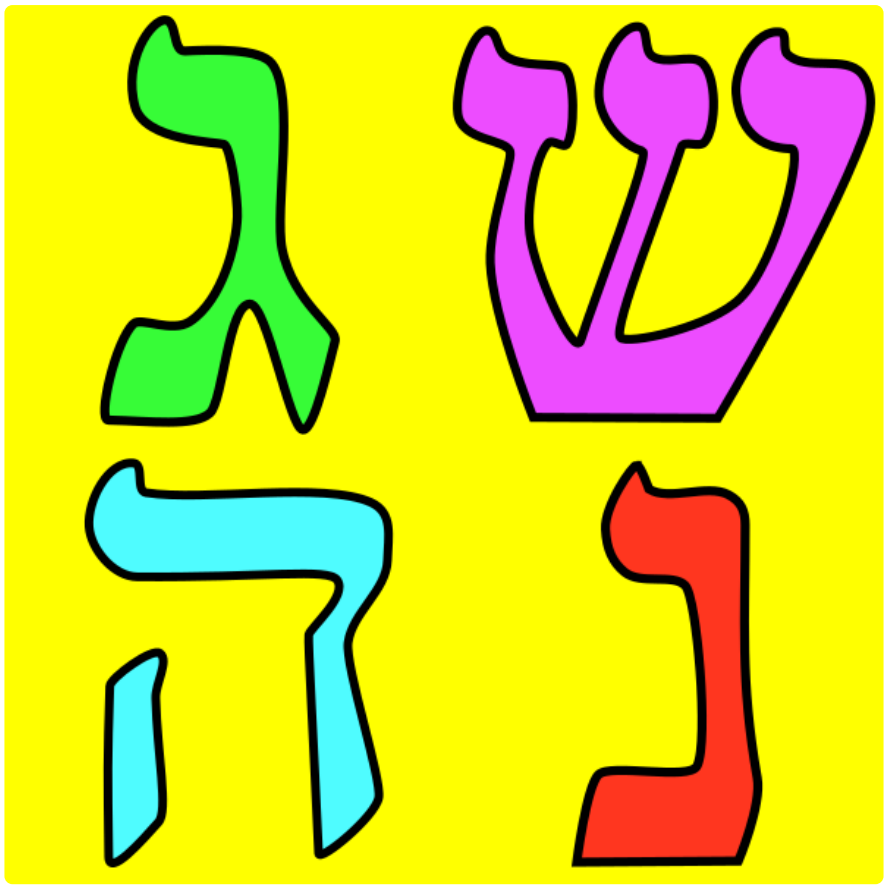
Then, we define a face area where the symbols will go: The area is 120x120 pixels and is located at x=60, y=70.

There's nothing needed in code for this definition. This is just a convention that will be used for setting up the symbols.

Dreidel Symbols

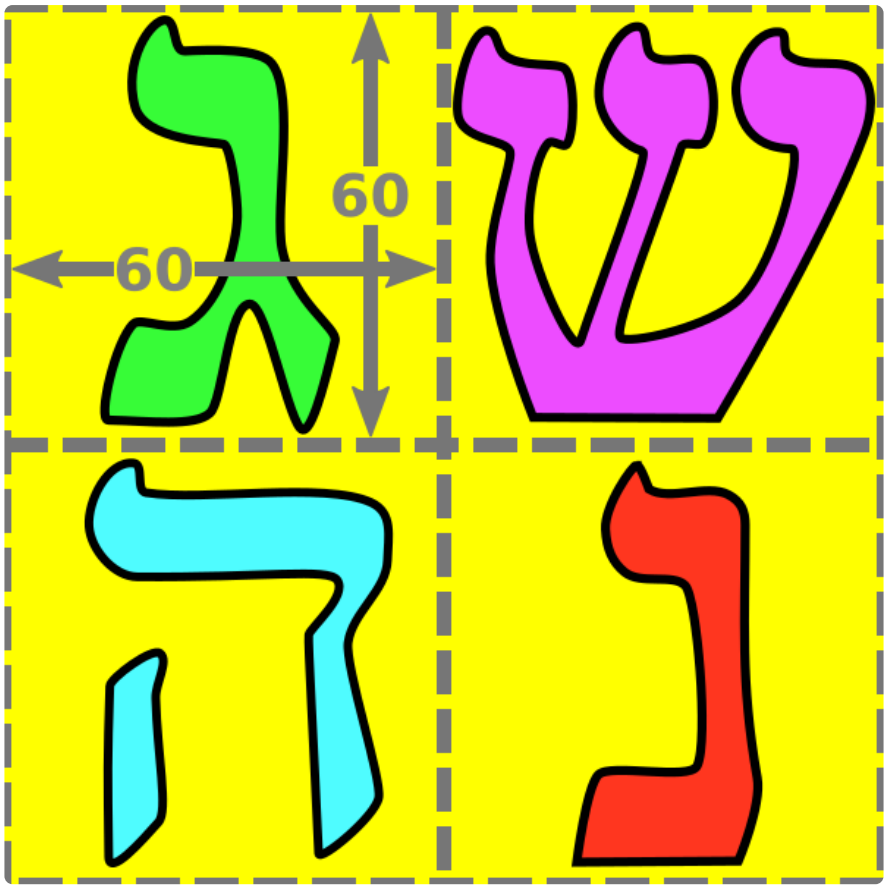
Now for the fun part. There are four symbols to deal with. We could have four separate 120x120 pixel images for each. But then we have all those separate files to keep track of and deal with. A better approach is to use the idea of a sprite sheet (https://adafru.it/GC4). This will let us have only a single file to deal with. The sprite sheet will contain the four symbols and each symbol will be a separate sprite.

That's the approach taken here. The dreidel symbol sprite sheet looks like this:



Further, the dreidel symbols are fairly simple graphics-wise. Our target size is 120x120 pixels, but we really don't need to define the symbols at that resolution. We can get away with defining them at half that, so 60x60 pixels, and then use the scale feature of the displayio Group (https://adafru.it/EFx) to have them render at the desired 120x120 pixels, by setting `scale=2`.

So our actual sprite sheet will look like this:



Each of the four symbols is only 60x60 pixels. The overall size of the entire sprite sheet ends up being 120x120 pixels. Here's what it looks like full size:

The yellow background is used so we can easily find that when the image is loaded and set that palette index to be transparent.

Game Logic and Melody

The melody definition is pretty straight forward. Just define the frequency and duration for each note. Then loop through each of those and use `play_tone()` to play them.

But how do we play the melody while also updating the symbols on the display? Well, it turns out we can get away with something very simple. Since we went to the trouble to define the symbols very small and then scale them at render time, they load pretty darn fast. So fast in fact that we can just add a call to reload a different symbol within

the loop that plays the melody. The display updates without noticeably slowing down the melody playback. Neat!

The whole thing is kicked off by waiting for the board to be shaken. **So, shake to "spin" the dreidel.** Then, after the melody, a random symbol is shown. And then it just loops back to the start and waits for a shake again.