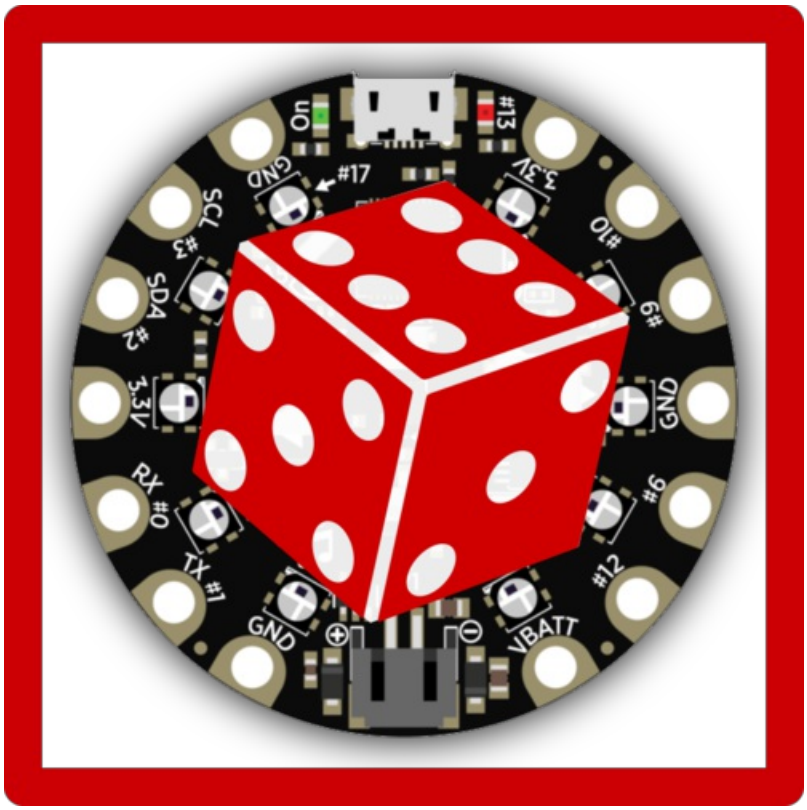
adafruit learning system

Circuit Playground D6 Dice

Created by Carter Nelson



Last updated on 2018-08-22 03:58:02 PM UTC

Guide Contents

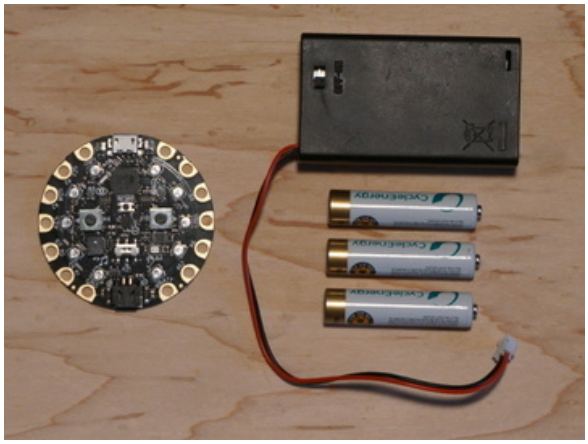
Overview

Dice (https://adafru.it/sSC) are another one of those game related items that have been around for a long time. They come in a wide variety of shapes and sizes, but the most common is the basic 6 sided dice, otherwise known as the D6.

In this guide, we'll show how we can simulate a D6 dice on the Circuit Playground using the NeoPixels to represent the various patterns of the dice face. We'll also use the accelerometer to simulate "rolling the dice" by detecting shaking. And because we can, we'll add a tap detect feature for quick and easy rolling.

Required Parts

This project uses the sensors already included on the Circuit Playground, either a Classic (http://adafru.it/3000) or an Express (http://adafru.it/3333). The only additional items needed are batteries for power and a holder for the batteries.



- Circuit Playground
  - Classic (http://adafru.it/3000)
  - Express (http://adafru.it/3333)
- 3 x AAA Battery Holder (http://adafru.it/727)
- 3 x AAA Batteries (NiMH work great!)

Before Starting

If you are new to the Circuit Playground, you may want to first read these overview guides.

Circuit Playground Classic

- Overview (https://adafru.it/ncG)
- Lesson #0 (https://adafru.it/rb4)

Circuit Playground Express

- Overview (https://adafru.it/AgP)

Arduino

The following pages go over developing the D6 Dice using the Arduino IDE.

Random, or Not?

The whole point of a dice is to provide a way to randomly come up with a number. In the case of a D6 dice, this would be a number from 1 to 6. The Arduino library provides a random (https://adafru.it/t9A) function we can use, but let's take a look at how it behaves.

Try running the simple sketch below.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Random Demo
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////
void loop() {
  // Wait for button press
  while (!CircuitPlayground.leftButton()   &&
         !CircuitPlayground.rightButton())  {
    // Do nothing, just waiting for a button press...
  }

  // Print a random number
  Serial.println(random(1,7));

  // Debounce delay
  delay(500);
}
```

With this code loaded and running on the Circuit Playground, open the Serial Monitor.

Tools -> Serial Monitor

and then press either button. Each time, a random number from 1 to 6 will be printed out. Let me guess, you got the same sequence I did as shown below.

And if you reset the Circuit Playground and try this again, you will get the same sequence again. So what's going on?

In turns out that the `random()` function implemented in the Arduino library is only a pseudo-random function. This simply means it isn't fully random (pseudo = false). It just produces a random like sequence of numbers, and the same sequence every time.

To get around this, we need to initialize the random function with a random value. Which, to be honest, sounds a little...unusual! I mean, if we *had* a random value why use random() at all? But it really does make sense: if it were possible to get one random value, maybe in a round-about way, we could then take advantage of the simple built in function.

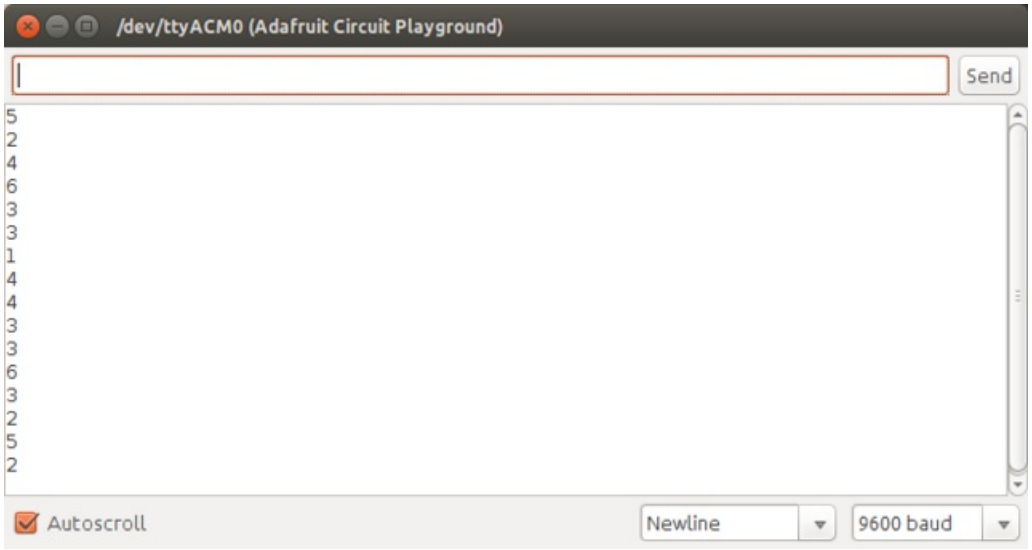
This is called *seeding the function* and the value is called the *seed*. You plant a *seed* of randomness, and then we can harvest any number of random numbers we need, forever!

But where can we come up with a random seed value? One way is to use some unused sensor that is (hopefully) reporting random and changing values of whatever it is meant to sense. Like light. Hey, we got a light sensor. Let's use that. We simply need to make a call to `lightSensor()` and feed that value into `randomSeed()`.

Here's a new version of the code that includes a call to `randomSeed()` in the `setup()`. This seed is generated by reading the light sensor.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Random Demo with Seed
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  CircuitPlayground.begin();

  // Seed the random function with light sensor value
  randomSeed(CircuitPlayground.lightSensor());
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  // Wait for button press
  while (!CircuitPlayground.leftButton()   &&
         !CircuitPlayground.rightButton())  {
    // Do nothing, just waiting for a button press...
  }

  // Print a random number
  Serial.println(random(1,7));

  // Debounce delay
  delay(500);
}
```

Load this code, open the Serial Monitor, and try again by pressing the buttons. Hopefully you get a different sequence this time, and it's different than the one I got.

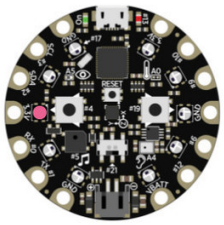


While this isn't perfect, it will work for our needs. This is what we will use to generate the random number to simulate a dice roll

Dice Faces

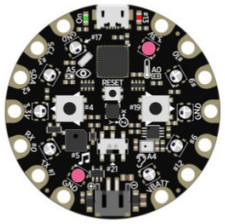
A D6 dice has 6 different patterns of dots on each of its 6 faces to represent the values 1 through 6. We have 10 NeoPixels on our Circuit Playground, so we've got enough lights. We'll turn 1 light on for a roll of 1, 2 lights for 2, etc. We just need to come up with which NeoPixels to use.

The NeoPixel layout could be anything, but the following attempts to match the D6 dice face patterns as much as possible.



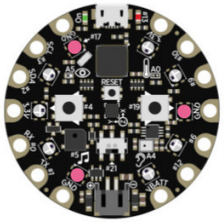
ROLL = 1

NeoPixels: 2



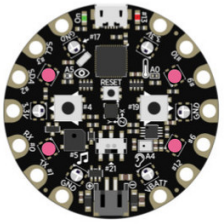
ROLL = 2

NeoPixels: 4, 9



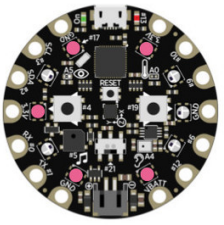
ROLL = 3

NeoPixels: 0, 4, 7



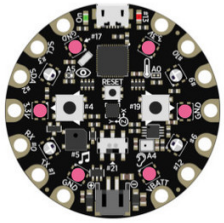
ROLL = 4

NeoPixels: 1, 3, 6, 8



ROLL = 5

NeoPixels: 0, 2, 4, 5, 9



ROLL = 6

NeoPixels: 0, 2, 4, 5, 7, 9

Storing the Face Patterns

Here's a simple approach for storing the above patterns of NeoPixels. The idea is to use a two dimensional (2D) array. The first dimension is simply the number of the roll, i.e. 1 through 6, so we'll need 6 entries. The second dimension of the array specifies the NeoPixels to light up for the dice roll. Since we are lighting up 1 NeoPixel per dice value, this will also need to have 6 entries.

So, we can do something like this:

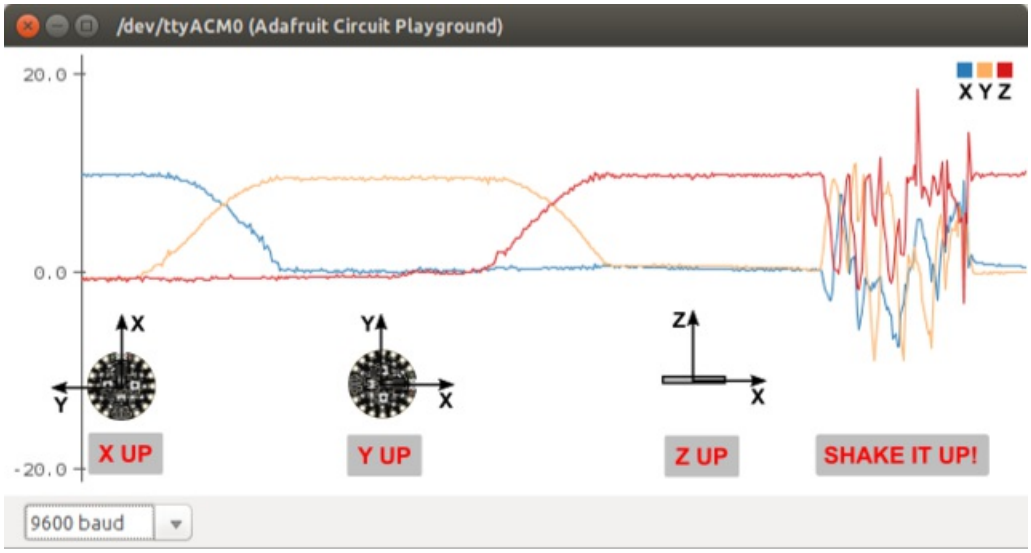
```
uint8_t dicePixels[6][6] = {  // Pixel pattern for dice roll
  { 2, 0, 0, 0, 0, 0 } ,      // Roll = 1
  { 4, 9, 0, 0, 0, 0 } ,      //        2
  { 0, 4, 7, 0, 0, 0 } ,      //        3
  { 1, 3, 6, 8, 0, 0 } ,      //        4
  { 0, 2, 4, 5, 9, 0 } ,      //        5
  { 0, 2, 4, 5, 7, 9 } ,      //        6
};
```

You'll see this get used later in the final code.

Shake Detect

To roll a dice, you pick it up and shake it. Then you toss it down and see where it lands. We'll want to do this with our Circuit Playground dice as well. Let's see if we can use the accelerometer to sense when the Circuit Playground is being shaken.

For an overview of how the accelerometer works on the Circuit Playground, check out this guide (https://adafru.it/t9f). You will see the following plot shown there.



What we are interested in is the **SHAKE IT UP!** part. That's where the Circuit Playground is being shaken. The lines are the return values from `motionX()`, `motionY()`, and `motionZ()`. This is a mess of positive and negative values with little hope of using simple logic to determine what is going on.

For detecting shaking, all we really care about is the magnitude of the acceleration. We don't care what direction it is being shaken. Therefore, we can use the total acceleration magnitude.

You can use the simple sketch below to try this out.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Total Acceleration
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

float X, Y, Z, totalAccel;

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  CircuitPlayground.begin();
  CircuitPlayground.setAccelRange(LIS3DH_RANGE_8_G);
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  X = 0;
  Y = 0;
  Z = 0;
  for (int i=0; i<10; i++) {
    X += CircuitPlayground.motionX();
    Y += CircuitPlayground.motionY();
    Z += CircuitPlayground.motionZ();
    delay(1);
  }
  X /= 10;
  Y /= 10;
  Z /= 10;

  totalAccel = sqrt(X*X + Y*Y + Z*Z);

  Serial.println(totalAccel);

  delay(100);
}
```

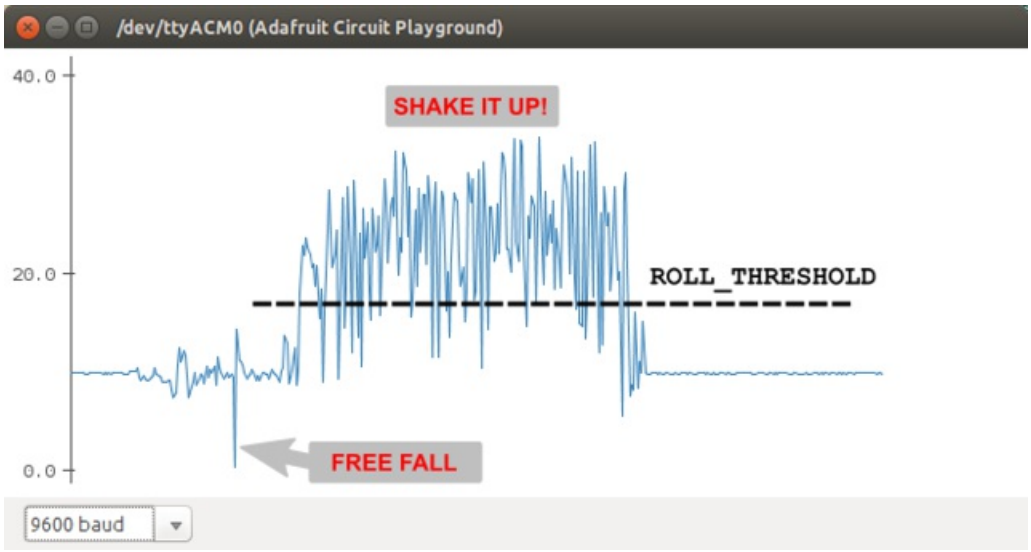
To smooth things out, we take 10 readings and average them. Then the total acceleration is computed with:

`totalAccel = sqrt(X*X + Y*Y + Z*Z);`

With the above sketch loaded and running on the Circuit Playground, open the Serial Plotter

Tools -> Serial Plotter

and give the Circuit Playground a shake. You should see something like what is shown in the figure below.



It still looks noisy, but if we pick an appropriate value for **ROLL_THRESHOLD**, then we can detect shaking by simply comparing it to the total acceleration value.

The sketch below uses this idea to play a sound if it detects shaking.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Shake Detect
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

#define ROLL_THRESHOLD  30 // Total acceleration threshold for roll detect

float X, Y, Z, totalAccel;

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  CircuitPlayground.begin();
  CircuitPlayground.setAccelRange(LIS3DH_RANGE_8_G);
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  // Compute total acceleration
  X = 0;
  Y = 0;
  Z = 0;
  for (int i=0; i<10; i++) {
    X += CircuitPlayground.motionX();
    Y += CircuitPlayground.motionY();
    Z += CircuitPlayground.motionZ();
    delay(1);
  }
  X /= 10;
  Y /= 10;
  Z /= 10;

  totalAccel = sqrt(X*X + Y*Y + Z*Z);

  // Play sound if rolling
  if (totalAccel > ROLL_THRESHOLD) {
    CircuitPlayground.playTone(800, 100);
  }
}
```

Play around with different values of **ROLL_THRESHOLD**. If it's too low, the detect is too sensitive. If it's too high, you'll break your arm trying to get it to beep. The value of 30 in the above sketch seemed to work OK for me.

Tap Detect

After a lot of shaking, you might wear your arm out. So let's add another way to roll the dice. The accelerometer on the Circuit Playground has a built in tap detect function. Let's use that to allow the dice to be rolled by taping the Circuit Playground.

There are two library functions associated with this feature:

- `setAccelTap();`
- `getAccelTap();`

There are also a couple of example sketches that you can look at to explore the tap detect feature. You can access them via:

File -> Examples -> Adafruit Circuit Playground -> accelTap

and

File -> Examples -> Adafruit Circuit Playground -> comm_badge

In each of these you will see something like this cryptic line:

```
attachInterrupt(digitalPinToInterrupt(7), myFunction, RISING);
```

This is setting up a function callback for an interrupt. This is somewhat of an advanced topic, but you can think of it as off loading the tap detection work to the accelerometer hardware. What the above line of code does is set things up so that when the accelerometer detects tap(s), the function `myFunction` will be called. It's up to you to create `myFunction` to do what you want, and the name can be any valid name.

If you look at the code in the **accelTap** example, you will see that there is no code in the `loop()` function. That's because the function `tapTime()` is being called via the interrupt.

You can still put code in the `loop()` function if you want. In fact, we'll be doing that for our D6 Dice. Here's another example that will be more like how we will use the tap detect for our dice. The callback function simply sets a flag. We then look for the state of that flag in `loop()`.

> The interrupt pin for the Circuit Playground Express is different.

```
/************************************************************************/
// Circuit Playground Tap Detect
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

#define TAP_THRESHOLD       10          // Tap detect threshold

bool tapDetected;

/************************************************************************/
void tapCallback() {
  tapDetected = true;
}

/************************************************************************/
void setup(void) {
  Serial.begin(9600);

  CircuitPlayground.begin();
  CircuitPlayground.setAccelRange(LIS3DH_RANGE_8_G);
  CircuitPlayground.setAccelTap(2, TAP_THRESHOLD);

  attachInterrupt(digitalPinToInterrupt(7), tapCallback, FALLING);

  tapDetected = false;
}

/************************************************************************/
void loop() {
  if (tapDetected) {
    Serial.println("TAP!");
    tapDetected = false;
  }
}
```

Note that we set up the detect for double tap by passing in a 2 in the `setup()` function:

```
CircuitPlayground.setAccelTap(2, TAP_THRESHOLD);
```

The way this works it that in the `loop()` function we look for a boolean value which is set to true in the interrupt callback function `tapCallback()`. This boolean is then set to false so that the print statement only happens once, until another double tap causes it to be true. Etc. Etc.

D6 Dice Code

OK, let's put all the pieces together. Here's the final D6 Dice code. With this code loaded and running on the Circuit Playground you can pick it up and shake it to make a roll. You can also just double tap it.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground D6 Dice
//
// Roll them bones.
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

#define ROLL_THRESHOLD      30          // Total acceleration threshold for roll detect
#define TAP_THRESHOLD       10          // Tap detect threshold
#define DICE_COLOR          0xEA6292    // Dice digits color

unsigned long rollStartTime;
bool rolling;
bool newRoll;
bool tapDetected;
uint8_t rollNumber;
float X, Y, Z, totalAccel;

uint8_t dicePixels[6][6] = {  // Pixel pattern for dice roll
  { 2, 0, 0, 0, 0, 0 } ,      // Roll = 1
  { 4, 9, 0, 0, 0, 0 } ,      //        2
  { 0, 4, 7, 0, 0, 0 } ,      //        3
  { 1, 3, 6, 8, 0, 0 } ,      //        4
  { 0, 2, 4, 5, 9, 0 } ,      //        5
  { 0, 2, 4, 5, 7, 9 } ,      //        6
};

////////////////////////////////////////////////////////////////////////////
void tapCallback() {
  tapDetected = true;
}

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  CircuitPlayground.begin();
  CircuitPlayground.setBrightness(100);
  CircuitPlayground.setAccelRange(LIS3DH_RANGE_8_G);
  CircuitPlayground.setAccelTap(2, TAP_THRESHOLD);

  // Setup tap detection and callback function
  attachInterrupt(digitalPinToInterrupt(7), tapCallback, RISING);

  // Seed the random function with light sensor value
  randomSeed(CircuitPlayground.lightSensor());

  // Initialize the global states
  newRoll = false;
  rolling = false;
  tapDetected = false;
```

```
  tapDetected = false;
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  // Compute total acceleration
  X = 0;
  Y = 0;
  Z = 0;
  for (int i=0; i<10; i++) {
    X += CircuitPlayground.motionX();
    Y += CircuitPlayground.motionY();
    Z += CircuitPlayground.motionZ();
    delay(1);
  }
  X /= 10;
  Y /= 10;
  Z /= 10;

  totalAccel = sqrt(X*X + Y*Y + Z*Z);

  // Check for rolling
  if ((totalAccel > ROLL_THRESHOLD) || tapDetected) {
    rollStartTime = millis();
    newRoll = true;
    rolling = true;
    tapDetected = false;
  }

  // Rolling momentum
  // Keep rolling for a period of time even after shaking has stopped.
  if (newRoll) {
    if (millis() - rollStartTime > 1000) rolling = false;
  }

  // Compute a random number from 1 to 6
  rollNumber = random(1,7);

  // Display status on NeoPixels
  if (rolling) {
    // Make some noise and show the dice roll number
    CircuitPlayground.playTone(random(400,2000), 20, false);
    CircuitPlayground.clearPixels();
    for (int p=0; p<rollNumber; p++) {
      CircuitPlayground.setPixelColor(dicePixels[rollNumber-1][p], DICE_COLOR);
    }
    delay(20);
  } else if (newRoll) {
    // Show the dice roll number
    newRoll = false;
    CircuitPlayground.clearPixels();
    for (int p=0; p<rollNumber; p++) {
      CircuitPlayground.setPixelColor(dicePixels[rollNumber-1][p], DICE_COLOR);
    }
  }
}
```

CircuitPython

The following pages go over creating the D6 Dice using CircuitPython (https://adafru.it/C3t).

CircuitPython only works on the Circuit Playground Express.

Dice Faces

We'll reuse the same approach taken in the Arduino section for representing the dice roll using patterns of NeoPixels.

Storing the Face Patterns

Here's a simple approach for storing the patterns of the NeoPixels for the dice rolls. We can use a dictionary, which is a storage class in CircuitPython that can store any amount of **values** which are accessed by using their associated **keys**. These are called key-value pairs, and a dictionary can hold any number of them. We will use the dice roll number as our key and then a tuple of NeoPixels values associated with the number to represent the dice face patterns.

Like this:

```
dice_pixels = {
 1 : (2,),
 2 : (4, 9),
 3 : (0, 4, 7),
 4 : (1, 3, 6, 8),
 5 : (0, 2, 4, 5, 9),
 6 : (0, 2, 4, 5, 7, 9)
}
```

You'll see this get used later in the final code.

Shake Detect

We can use the same approach for detecting shake as discussed in the Arduino section. See that section for a more complete discussion.

Here's a simple demonstration that will play a tone when shake is detected:

```
# Circuit Playground Express Shake Detect
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
import math
from adafruit_circuitplayground.express import cpx

# Total acceleration threshold
ROLL_THRESHOLD = 30

# Loop forever
while True:
    # Compute total acceleration
    X = 0
    Y = 0
    Z = 0
    for i in range(10):
        x, y, z = cpx.acceleration
        X = X + x
        Y = Y + y
        Z = Z + z
        time.sleep(0.001)
    X = X / 10
    Y = Y / 10
    Z = Z / 10

    total_accel = math.sqrt(X*X + Y*Y + Z*Z)

    # Play sound if rolling
    if total_accel > ROLL_THRESHOLD:
        cpx.play_tone(800, 1)
```

Play around with different values of **ROLL_THRESHOLD**. If it's too low, the detect is too sensitive. If it's too high, you'll break your arm trying to get it to beep. The value of 30 in the above sketch seemed to work OK for me.

Or Just Use The Library

The shake detect approach above is so useful that it was added to the CircuitPython library for Circuit Playground Express. Under the hood, it's doing the exact same thing. But now the code becomes as simple as:

```
# Circuit Playground Express Shake Detect Using Library
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
from adafruit_circuitplayground.express import cpx

# Loop forever
while True:
    # Play sound if rolling
    if cpx.shake():
        cpx.play_tone(800, 1)
```

Running that should work just like the first example. But now the code is much more readable. Sweet.

D6 Dice Code

Here is the CircuitPython version of the D6 dice code. With this code loaded and running on the Circuit Playground Express you can pick it up and shake it to make a roll. You can also just double tap it.

```
# Circuit Playground Express D6 Dice
#
# Roll them bones.
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
import random
from adafruit_circuitplayground.express import cpx

ROLL_THRESHOLD  = 30        # Total acceleration
DICE_COLOR      = 0xEA6292  # Dice digits color

dice_pixels = {
 1 : (2,),
 2 : (4, 9),
 3 : (0, 4, 7),
 4 : (1, 3, 6, 8),
 5 : (0, 2, 4, 5, 9),
 6 : (0, 2, 4, 5, 7, 9)
}

# Configure double tap detection
cpx.detect_taps = 2

# Seed the random function with light sensor value
# (try commenting this out and see if it matters)
random.seed(cpx.light)

# Initialize the global states
new_roll = False
rolling = False

# Loop forever
while True:
    # Check for rolling
    if cpx.shake() or cpx.tapped:
        roll_start_time = time.monotonic()
        new_roll = True
        rolling = True

    # Rolling momentum
    # Keep rolling for a period of time even after shaking stops
    if new_roll:
        if time.monotonic() - roll_start_time > 1:
            rolling = False

    # Compute a random number from 1 to 6
    roll_number = random.randrange(1,7)

    # Display status on NeoPixels
    if rolling:
        # Make some noise and show the dice roll number
        cpx.start_tone(random.randrange(400,2000))
```

```
        cpx.start_tone(random.randrange(400,2000))
        cpx.pixels.fill(0)
        for p in dice_pixels[roll_number]:
            cpx.pixels[p] = DICE_COLOR
        time.sleep(0.02)
        cpx.stop_tone()
    elif new_roll:
        # Show the dice roll number
        new_roll = False
        cpx.pixels.fill(0)
        for p in dice_pixels[roll_number]:
            cpx.pixels[p] = DICE_COLOR
```

Questions and Code Challenges

The following are some questions related to this project along with some suggested code challenges. The idea is to provoke thought, test your understanding, and get you coding!

While the sketches provided in this guide work, there is room for improvement and additional features. Have fun playing with the provided code to see what you can do with it.

Questions

- Why do you think it was necessary to call the `setAccelRange()` function in `setup()`? (hint: 1G = 9.8 m/s^2)
- Can you think of a way to cheat the dice? (hint: think about that seed)

Code Challenges

- Come up with your own NeoPixel patterns for the dice face numbers.
- Change the dice color.
- Change the double tap to a single tap.
- Can you turn the 6-sided die into a *10* sided die?