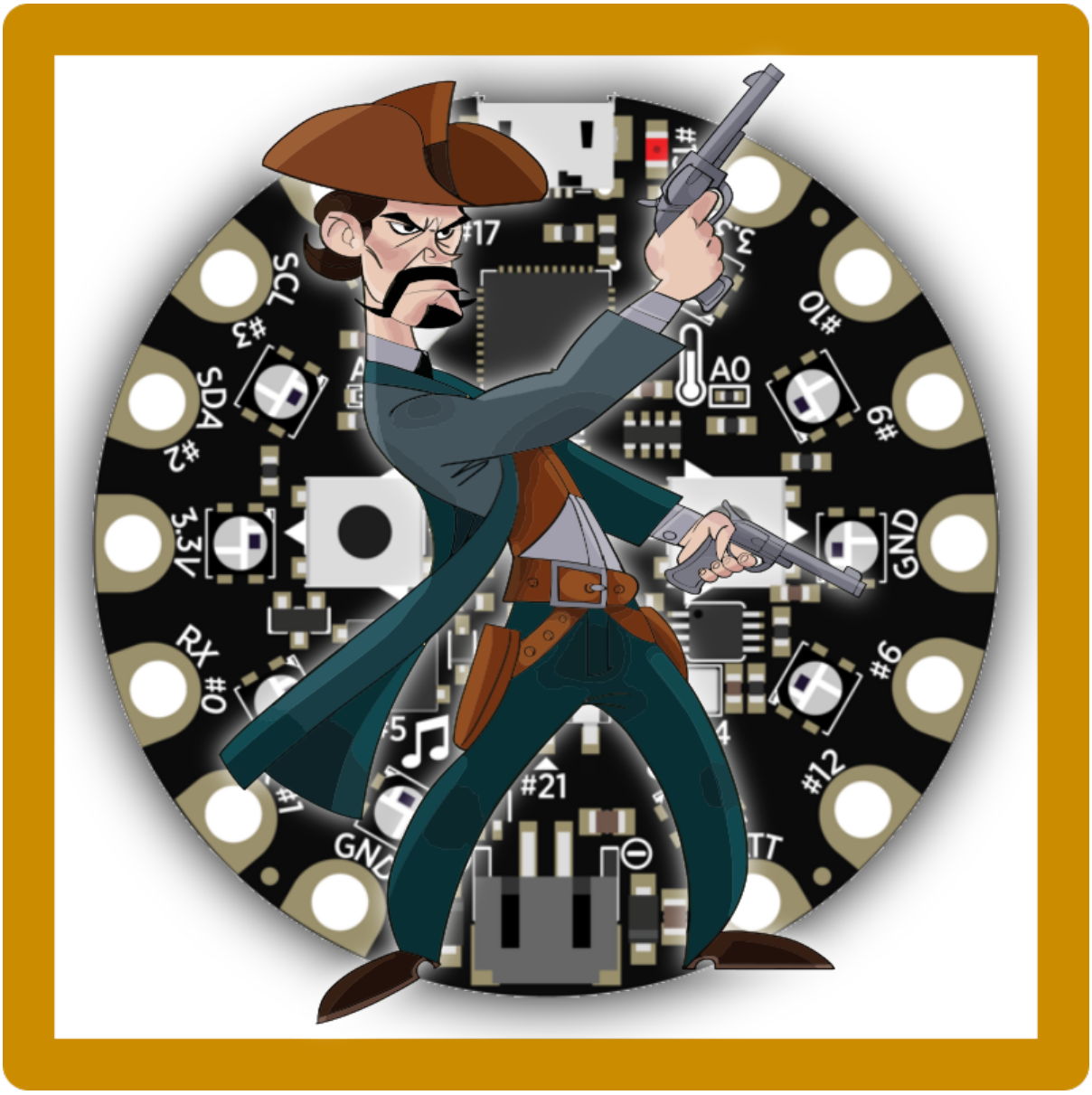
Circuit Playground Quick Draw

Created by Carter Nelson



https://learn.adafruit.com/circuit-playground-quick-draw

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

Table of Contents

Overview 3

- Required Parts
- Before Starting
- Circuit Playground Classic
- Circuit Playground Express

The Town Clock 4

Game Design 4

- Game Logic
- Player Buttons
- Countdown NeoPixels
- DRAW!
- Player NeoPixels
- PLAYER 1 MISDRAW!
- PLAYER 2 MISDRAW!
- PLAYER 1 WON!
- PLAYER 2 WON!

Arduino 7

Randomest Random 7

- Random Isn't Random

The Countdown 10

Show Outcome 10

Code Listing 12

CircuitPython 14

The Countdown 14

Show Outcome 15

Code Listing 16

Questions and Code Challenges 17

- Questions
- Code Challenges

Overview

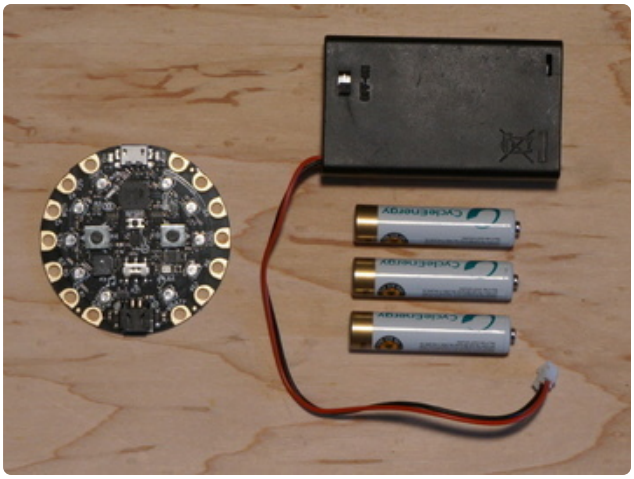
You know the scene. Old West town. Two gunslingers face each other on a dusty street. Tumble weed rolls by. Everyone is eying the town clock. Tick. Tick. 'Cause when it strikes high noon, the gunslingers.....**DRAW!**

Pew! Pew! Who was the Quickest Draw?

Well put your guns away pardnah. Let's just use these two buttons we got here on our Circuit Playground. This here is a two person show down game to see who can press their button the quickest.

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)

The Town Clock

In the classic gunslinger show down portrayed in numerous movies, the town clock was often used as the 'go' or 'draw' signal for the two gunslingers. High noon or some other on-the-hour time was used so that the minute hand was the main 'go' indicator. As soon as it pointed straight up, it was time to draw.

For our Circuit Playground Quick Draw game, we'll use the NeoPixels instead. They will initially be all off. The two players should then be at the ready. Then, after a random period of time, we will turn on all of the NeoPixels. This is the 'go' signal at which point the two players press their buttons as quick as they can. The winner is whoever pressed their button the quickest.

Game Design

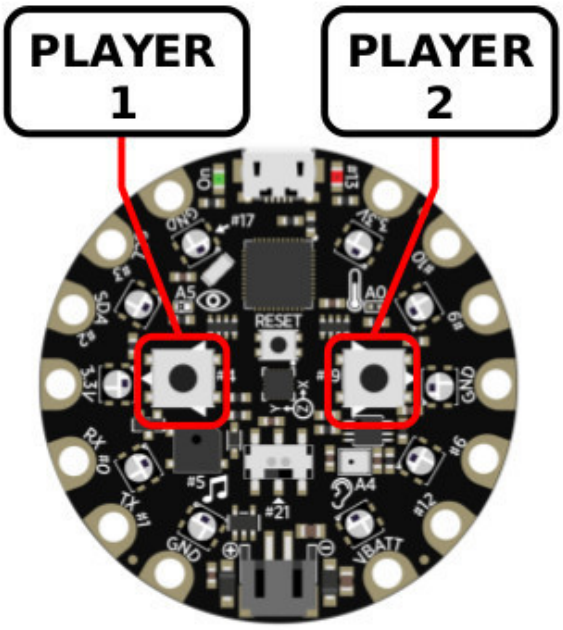
Game Logic

Once we have our random countdown time figured out, the game logic is very simple:

1. Turn off all of the NeoPixels.
2. Wait the determined countdown time.
3. If a player presses a button during this time, they drew too soon (misdraw).
4. Once countdown time has elapsed, turn on all of the NeoPixels.
5. Look for the first (quickest) button press.
6. Which ever button was pressed first is the Quick Draw winner.

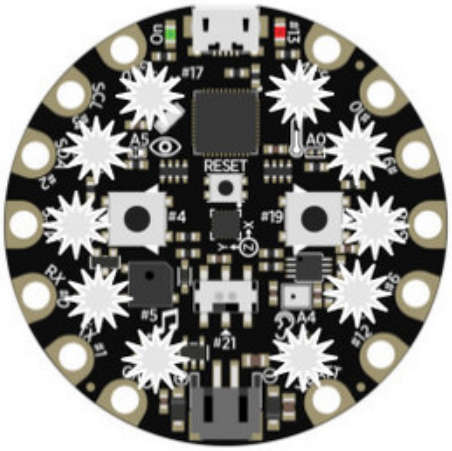
Player Buttons

This is pretty straight forward. We've got two players, we've got two buttons. So we can assign them as shown in the figure below.



Countdown NeoPixels

This could be anything, but to keep it simple we'll just turn on all the NeoPixels to white when the countdown completes.

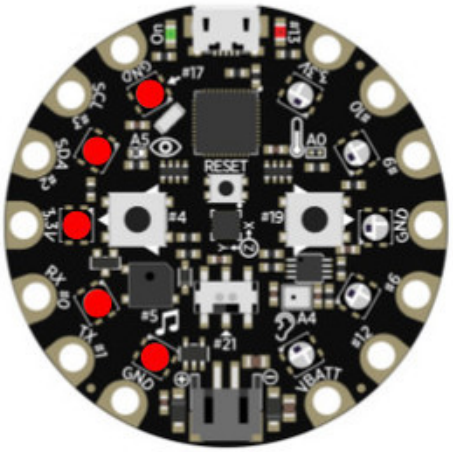


DRAW!

When all of the lights come on (all white), press your button as fast as you can.

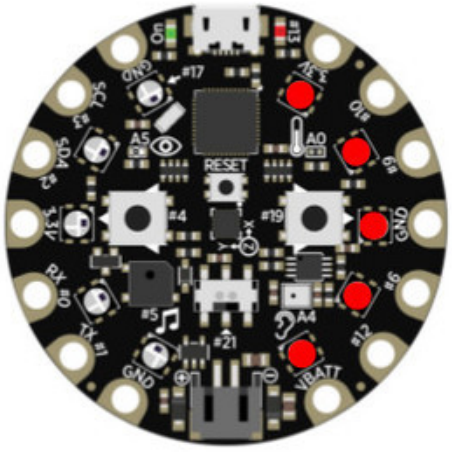
Player NeoPixels

We can use the NeoPixels on the left to indicate Player 1's outcome, and the NeoPixels on the right to indicate Player 2's outcome. There are two possible outcomes: a misdraw if a player draws too soon, or a game with a winning outcome.



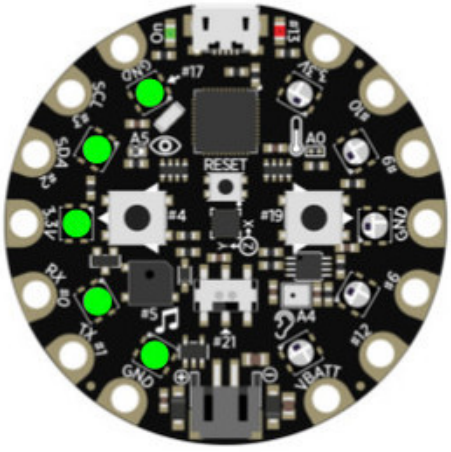
PLAYER 1 MISDRAW!

If all of the lights on the Player 1 side turn red, Player 1 misdrew (pressed the button too soon).



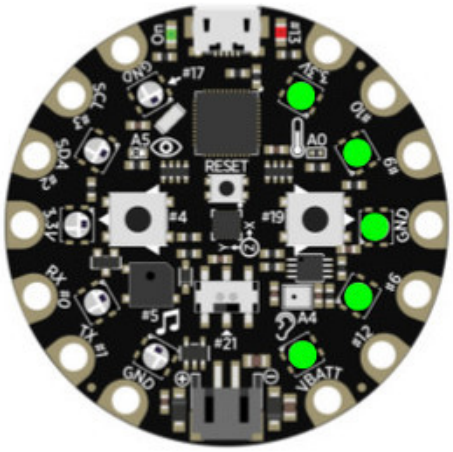
PLAYER 2 MISDRAW!

If all of the lights on the Player 2 side turn red, Player 2 misdrew (pressed the button too soon).



PLAYER 1 WON!

If all of the lights on the Player 1 side turn green, Player 1 was the quickest.



PLAYER 2 WON!

If all of the lights on the Player 2 side turn green, Player 2 was the quickest.

Arduino

The following pages go over creating the Quick Draw game using the Arduino IDE.

Randomest Random

Random Isn't Random

We can create the random period of time needed for the countdown using the `random()` function available in the Arduino library (https://adafru.it/t9A). However, it may not behave like you think it behaves. Let's take a look.

Try running the simple sketch below.

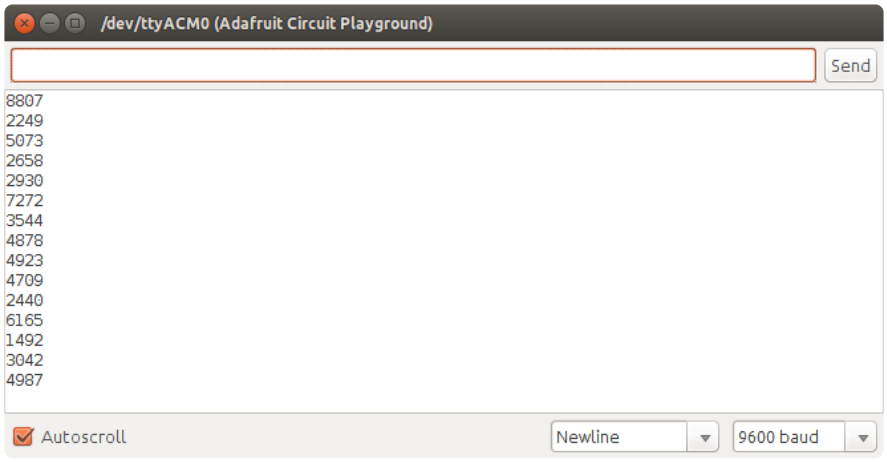
```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Random Demo
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define SHORTEST_DELAY  1000    // milliseconds
#define LONGEST_DELAY   10000   //      "

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

  CircuitPlayground.begin();
}

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

```
  }

  // Print a random number
  Serial.println(random(SHORTEST_DELAY, LONGEST_DELAY));

  // 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 **SHORTEST_DELAY** to **LONGEST_DELAY** 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. This is called seeding the function and the value is called the seed. But where can we come up with a random seed value? One way is to use some of the (hopefully) unconnected pads on the Circuit Playground and read in their analog values. Since the pads are not connected, the value returned by a call to `analogRead()` will contain noise. Noise is random, and that's what we want.

Here's a new version of the code that includes a call to `randomSeed()` in the `setup()`. This seed is generated by reading all four of the available analog inputs.

The analog pads shown are for the Circuit Playground Classic, but the code will still run on the Express.

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

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define SHORTEST_DELAY  1000    // milliseconds
#define LONGEST_DELAY   10000   //      "

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

  CircuitPlayground.begin();

  // Seed the random function with noise
  int seed = 0;

  seed += analogRead(12);
  seed += analogRead(7);
  seed += analogRead(9);
  seed += analogRead(10);

  randomSeed(seed);
}

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

  // Print a random number
  Serial.println(random(SHORTEST_DELAY, LONGEST_DELAY));

  // 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 amount of time needed for our countdown timer.

The Countdown

We could just use the `delay()` function to wait the random amount of time determined for the countdown. Something like this:

```
// Wait a random period of time
unsigned long countTime = random(SHORTEST_DELAY, LONGEST_DELAY);
delay(countTime);
```

However, we need to monitor the buttons during the countdown to make sure one of the players did not cheat (misdraw). We can do this by using a `while()` loop instead of `delay()` for the countdown and polling the buttons inside the loop. That would look something like the following, note that there is now no use of `delay()`.

```
// Wait a random period of time
unsigned long countTime = random(SHORTEST_DELAY, LONGEST_DELAY);
unsigned long startTime = millis();
while (millis() - startTime &lt; countTime) {
  // Check if player draws too soon.
  if (CircuitPlayground.leftButton()) showOutcome(1, false);
  if (CircuitPlayground.rightButton()) showOutcome(2, false);
}
```

But what's the `showOutcome()` function? Well, we'll talk about that next.

Show Outcome

The `showOutcome()` function will be created to, as the name implies, show the outcome of the game. The first parameter will be the player who's button was pressed. The second parameter is a boolean to indicate if the button press was a winning press (true) or a misdraw (false).

By generalizing the outcome code, to be able to show the proper NeoPixels for either player and for either outcome, we make our code more compact. Otherwise we would need to duplicate a lot of code in more than one place. It also makes it easier to change the behavior of the code in the future.

The complete code for the `showOutcome()` function will be shown at the end. Here, we'll go through it in pieces. First, we need to declare it:

```
void showOutcome(int player, bool winner) {
```

We aren't going to return anything, so the return parameter is `void`. We take in two parameters: the player who's button was pressed as an `int`, and whether this was a winning game or not as a `bool`.

Next, we create a couple of variables to be used locally within the function.

```
  int p1, p2;
  uint32_t color;
```

Then we turn off all of the NeoPixels, just to insure a known state.

```
  // Turn them all off
  CircuitPlayground.clearPixels();
```

Then we set the pixel color depending on the game outcome, green for a winning game, red for a misdraw.

```
  // Set pixel color
  if (winner) {
    color = 0x00FF00;
  } else {
    color = 0xFF0000;
  }
```

Then we set the range of NeoPixels to be lit up based on which payer pressed their button.

```
  // Set pixel range for player
  switch (player) {
    case 1:
      p1 = 0;
      p2 = 4;
      break;
    case 2:
      p1 = 5;
      p2 = 9;
      break;
    default:
      p1 = 0;
```

```
    p2 = 9;
}
```

Now we have a color for the NeoPixels, and which ones to turn on, so do that.

```
// Show which player won/lost
for (int p=p1; p&lt;=p2; p++) {
  CircuitPlayground.setPixelColor(p, color);
}
```

And why not play a little tune. A happy one if this was a winning game, a more error sounding one if it was a misdraw.

```
// Play a little tune
if (winner) {
  CircuitPlayground.playTone(800, 200);
  CircuitPlayground.playTone(900, 200);
  CircuitPlayground.playTone(1400, 200);
  CircuitPlayground.playTone(1100, 200);
} else {
  CircuitPlayground.playTone(200, 1000);
}
```

And we are done with the game, so we'll just sit here forever until the reset button is pressed to start a new game.

```
// Sit here forever
while (true) {};
```

And don't forget the closing curly bracket to finish off the `showOutcome()` function.

```
}
```

OK. Let's put it all together.

Code Listing

Here's the complete code listing for the Quick Draw game.

```
////////////////////////////////////////////////////////////////////////
// Circuit Playground Quick Draw
//
// Who's faster?
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

#define SHORTEST_DELAY  1000    // milliseconds
#define LONGEST_DELAY   10000   //      "
```

```
////////////////////////////////////////////////////////////////////////
void showOutcome(int player, bool winner) {
  int p1, p2;
  uint32_t color;

  // Turn them all off
  CircuitPlayground.clearPixels();

  // Set pixel color
  if (winner) {
    color = 0x00FF00;
  } else {
    color = 0xFF0000;
  }

  // Set pixel range for player
  switch (player) {
    case 1:
      p1 = 0;
      p2 = 4;
      break;
    case 2:
      p1 = 5;
      p2 = 9;
      break;
    default:
      p1 = 0;
      p2 = 9;
  }

  // Show which player won/lost
  for (int p=p1; p&lt;=p2; p++) {
    CircuitPlayground.setPixelColor(p, color);
  }

  // Play a little tune
  if (winner) {
    CircuitPlayground.playTone(800, 200);
    CircuitPlayground.playTone(900, 200);
    CircuitPlayground.playTone(1400, 200);
    CircuitPlayground.playTone(1100, 200);
  } else {
    CircuitPlayground.playTone(200, 1000);
  }

  // Sit here forever
  while (true) {};
}

////////////////////////////////////////////////////////////////////////
void setup() {
  // Initialized the Circuit Playground
  CircuitPlayground.begin();

  // Turn off all the NeoPixels
  CircuitPlayground.clearPixels();

  // Seed the random function with noise
  int seed = 0;

  seed += analogRead(12);
  seed += analogRead(7);
  seed += analogRead(9);
  seed += analogRead(10);

  randomSeed(seed);

  // Wait a random period of time
  unsigned long countTime = random(SHORTEST_DELAY, LONGEST_DELAY);
```

```
  unsigned long startTime = millis();
  while (millis() - startTime &lt; countTime) {
    // Check if player draws too soon.
    if (CircuitPlayground.leftButton()) showOutcome(1, false);
    if (CircuitPlayground.rightButton()) showOutcome(2, false);
  }

  // Turn on all the NeoPixels
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 0xFFFFFF);
  }
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  if (CircuitPlayground.leftButton()) showOutcome(1, true);
  if (CircuitPlayground.rightButton()) showOutcome(2, true);
}
```

CircuitPython

The following pages go over creating the Quick Draw game in CircuitPython (https://adafru.it/C3t).

CircuitPython only works on the Circuit Playground Express.

The Countdown

We could just use the `time.sleep()` function to wait the random amount of time determined for the countdown. Something like this:

```
# Wait a random amount of time
count_time = random.randrange(SHORTEST_DELAY, LONGEST_DELAY+1)
time.sleep(count_time)
```

However, we need to monitor the buttons during the countdown to make sure one of the players did not cheat (misdraw). We can do this by using a `while` loop instead of `time.sleep()` for the countdown and polling the buttons inside the loop. That would look something like the following, note that there is now no use of `time.sleep()`.

```
# Wait a random amount of time
count_time = random.randrange(SHORTEST_DELAY, LONGEST_DELAY+1)
start_time = time.monotonic()
while time.monotonic() - start_time &lt; count_time:
    # Check if player draws too soon
    if cpx.button_a:
        show_outcome(1, False)
    if cpx.button_b:
        show_outcome(2, False)
```

But what's the `show_outcome()` function? Well, we'll talk about that next.

Show Outcome

The `show_outcome()` function will be created to, as the name implies, show the outcome of the game. The first parameter will be the player who's button was pressed. The second parameter is a boolean to indicate if the button press was a winning press (true) or a misdraw (false).

By generalizing the outcome code, to be able to show the proper NeoPixels for either player and for either outcome, we make our code more compact. Otherwise we would need to duplicate a lot of code in more than one place. It also makes it easier to change the behavior of the code in the future.

The complete code for the `show_outcome()` function will be shown at the end. Here, we'll go through it in pieces. First, we need to declare it:

```
def show_outcome(player, winner):
```

We take in two parameters: the player who's button was pressed and whether this was a winning game or not.

Then we turn off all of the NeoPixels, just to insure a known state.

```
    # Turn them all off
    cpx.pixels.fill(0)
```

Then we set the pixel color depending on the game outcome, green for a winning game, red for a misdraw.

```
    # Set pixel color
    if winner:
        color = 0x00FF00
    else:
        color = 0xFF0000
```

Now we have a color for the NeoPixels, so turn them on for the correct player.

```
    # Show which player won/lost
    for p in PLAYER_PIXELS[player]:
        cpx.pixels[p] = color
```

And why not play a little tune. A happy one if this was a winning game, a more error sounding one if it was a misdraw.

```
    # Play a little tune
    if winner:
        cpx.play_tone(800, 0.2)
        cpx.play_tone(900, 0.2)
        cpx.play_tone(1400, 0.2)
        cpx.play_tone(1100, 0.2)
    else:
        cpx.play_tone(200, 1)
```

And we are done with the game, so we'll just sit here forever until the reset button is pressed to start a new game.

```
    # Sit here forever
    while True:
        pass
```

OK. Let's put it all together.

Code Listing

Here is the complete CircuitPython version of the Quick Draw code.

```
# Circuit Playground Express Quick Draw
#
# Who's faster?
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
import random
from analogio import AnalogIn
import board
from adafruit_circuitplayground.express import cpx

SHORTEST_DELAY = 1  # seconds
LONGEST_DELAY  = 10 #    "
PLAYER_PIXELS = {
  1 : (0,1,2,3,4),
  2 : (5,6,7,8,9)
}

def show_outcome(player, winner):
    # Turn them all off
    cpx.pixels.fill(0)

    # Set pixel color
    if winner:
        color = 0x00FF00
    else:
        color = 0xFF0000

    # Show which player won/lost
    for p in PLAYER_PIXELS[player]:
        cpx.pixels[p] = color

    # Play a little tune
    if winner:
        cpx.play_tone(800, 0.2)
        cpx.play_tone(900, 0.2)
        cpx.play_tone(1400, 0.2)
```

```
        cpx.play_tone(1100, 0.2)
    else:
        cpx.play_tone(200, 1)

    # Sit here forever
    while True:
        pass

# Seed the random function with noise
a4 = AnalogIn(board.A4)
a5 = AnalogIn(board.A5)
a6 = AnalogIn(board.A6)
a7 = AnalogIn(board.A7)

seed  = a4.value
seed += a5.value
seed += a6.value
seed += a7.value

random.seed(seed)

# Wait a random amount of time
count_time = random.randrange(SHORTEST_DELAY, LONGEST_DELAY+1)
start_time = time.monotonic()
while time.monotonic() - start_time &lt; count_time:
    # Check if player draws too soon
    if cpx.button_a:
        show_outcome(1, False)
    if cpx.button_b:
        show_outcome(2, False)

# Turn on all the NeoPixels
cpx.pixels.fill(0xFFFFFF)

# Check for player draws
while True:
    if cpx.button_a:
        show_outcome(1, True)
    if cpx.button_b:
        show_outcome(2, True)
```

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

Questions

- Does it matter which order the functions `leftButton()` and `rightButton()` are called in?
- Can you think of a way to cheat the countdown timer? (hint: seed)

Code Challenges

- Put the code that does the random seeding into a function called `initRandom()`.
- Change the color of the NeoPixels for the countdown.
- Come up with a different way to start a new game, instead of using reset button.