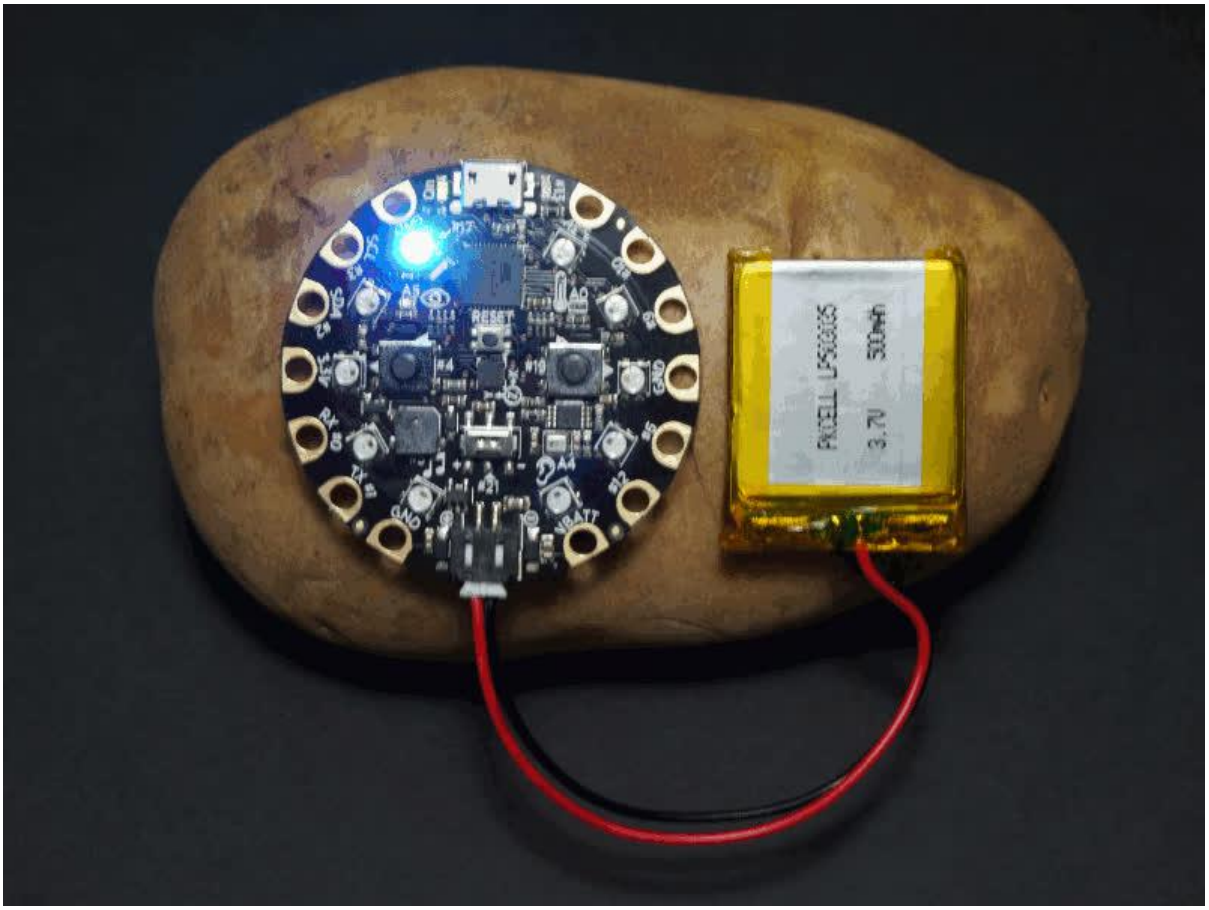
Circuit Playground Hot Potato

Created by Carter Nelson



https://learn.adafruit.com/circuit-playground-hot-potato

Last updated on 2023-08-29 03:20:55 PM EDT

Table of Contents

Overview

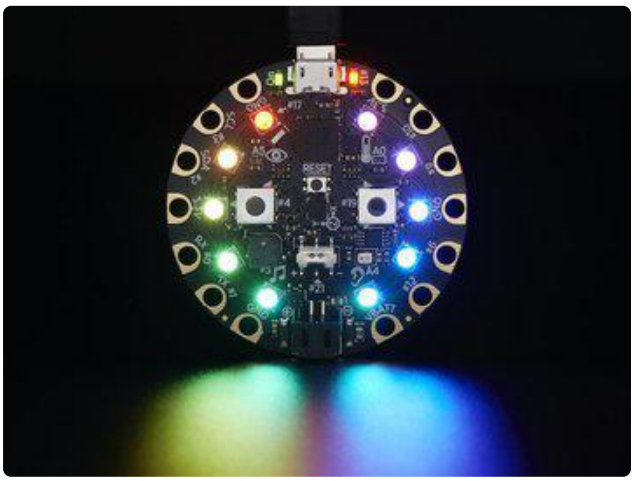
In this guide we will use our Circuit Playground to create a fun and simple game you can play with your friends. It's an old timey game called Hot Potato. Yep. People used to make games with potatoes.

No soldering required! (also no potato required)

Please don't use a real hot potato. That could really hurt you.

Required Parts

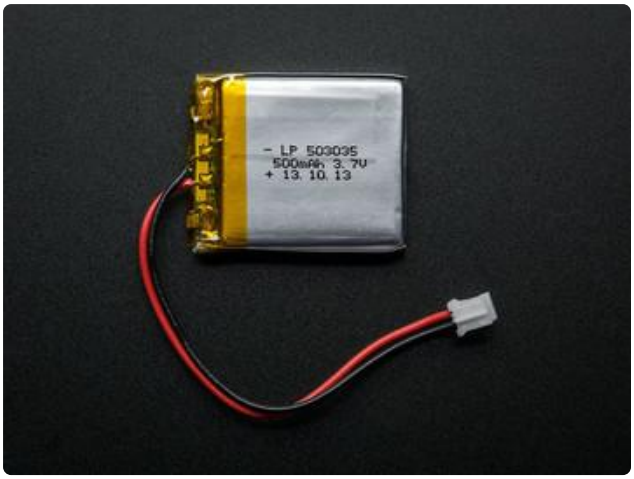
In addition to a Circuit Playground, you will need some form of battery power so you can play the game without being attached to a computer. Choose an option that works for how you plan to make your 'potato'.



Circuit Playground
Classic (http://adafru.it/3000)
Express (http://adafru.it/3333)

3 x AAA Battery Holder with On/Off Switch and 2-Pin JST (http://adafru.it/727)

(also need AAA batteries)



Lithium Ion Polymer Battery - 3.7v 500mAh (http://adafru.it/1578)

(or similar)

If you go with the LiPo battery, be sure you have a way to charge it ().

Before Starting

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

Circuit Playground Classic

- Overview ()
- Lesson #0 ()

Circuit Playground Express

- Overview ()

Game Play

The game Hot Potato has been around so long, no one is really sure where it comes from, not even the all knowing wikipedia ().

> The origins of the hot potato game are not clear.

But it does come from a time way before there was an internet and even reliable electricity. So people had to get creative. As Grandpa Simpson might say, "Back in my day all we had were taters. And we loved it!"

The game is played like this:

1. Gather a bunch of friends together and stand in circle.
2. Someone starts playing a melody.
3. The 'hot potato' is then tossed from person to person (order doesn't matter).
4. At some random time, the melody stops playing.
5. Whoever is holding the 'hot potato' at that point is out.

We will use our Circuit Playground to create our 'hot potato'. And since the Circuit Playground has a built in speaker, we can use it to play the melody. All we need to do is write a program to play a melody and stop it after a random period of time.

Let's see how we can do this.

Arduino

The following pages develop the Hot Potato game using the Arduino IDE.

Playing a Melody

The Circuit Playground has a very simple speaker. It basically can just play one tone at a time. However, simple melodies can be created by stringing together multiple tones of different frequencies. This approach is covered here:

- Arduino ToneMelody ()
- Circuit Playground Sound and Music ()

In each of these examples, two arrays are used to store the melody. The first holds the actual notes and the second specifies the duration for each note.

For example, from the Arduino ToneMelody example:

```
// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int tempo[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};
```

To play this back, we simply loop through the array and play each note. For the Circuit Playground, that would look something like this:

```
for (int n=0; n&lt;numberOfNotes; n++) {
  int noteDuration = 1000 / tempo[n];
  CircuitPlayground.playTone(melody[n], noteDuration);
  delay(0.3*noteDuration);
}
```

We first compute the actual note duration based on the tempo value. Then we play it. A small delay is added between each note to make them distinguishable.

Pitch Definitions

Note how in the melody array we used values that looked like `NOTE_C4`. This isn't some kind of built in Arduino magic. It is simply a variable that has been defined with the frequency value of the note C.

This is discussed further in the links provided above. The key thing to remember here is that you will need to make sure to include the pitches.h file with your sketch. That is where all of these values are defined. It is nothing but a file with a bunch of lines that look like:

```
#define NOTE_C4  262
```

And that's the line that defines `NOTE_C4` as having a value of `262`.

Sample Melody Sketch

Here's a sketch which plays the simple melody from the Arduino example whenever either Circuit Playground button is pressed.

PlayMelody.zip

It's a zip file which contains both the sketch along with the pitches.h file that defines the notes. So download it, unzip it, and load the sketch to the Circuit Playground. You'll know you got it working when you hear Shave and a Haircut.

Stopping a Melody

So now we know how to play a melody. For our Hot Potato game, we will want to let this melody play for a random amount of time and then stop it. How do we do that?

There is a very simple way we can do this. Instead of thinking in terms of time, just think in terms of number of notes. We'll just pick a random number of notes and play only that many.

Stop Melody 1

Here is a modified version of Mike Barela's chiptune sketch () which only plays a random number of notes from the melody.

StopMelody1.zip

The modifications are minor. The majority of the new lines deal with seeding the pseudo-random number generator to increase actual randomness. It is the same approach as taken in the Circuit Playground D6 Dice () guide. Here are the lines:

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

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

  randomSeed(seed);
```

With that taken care, we then just compute a random number for how many notes to play and modify the loop control appropriately.

```
    int numNotesToPlay = random(numNotes);
    for (int thisNote = 0; thisNote &lt; numNotesToPlay; thisNote++) { // play
notes of the melody
```

Try pressing the right button multiple times and hear how the length of the melody varies.

Stop Melody 2

The one problem with the previous approach is it will play the melody at most only once. For our Hot Potato game, we may want the melody to play longer (multiple times) to make the game more exciting. So we want to be able to specify a random number bigger than the number of notes in the melody. Say the melody had 10 notes. A loop of 10 would play it once. But we want to be able to loop 20 times to play it twice. Or 17 times to play it once, and then the first 7 notes again. Etc.

But we can't simply increase the size of our random number and use the same loop structure. Once the loop number exceeds the number of notes in the melody, bad things will happen. For example, there is no value for `melody[numNotes+1]`. In fact, because of zero indexing, the highest defined note is only `melody[numNotes-1]`.

So how do we deal with this? One answer is to simply use two variables. The first will be our random loop variable that will be the total number of notes to play. The second will be used to actually play the melody note. It will be incremented manually inside the loop, and once it exceeds the size of the melody, it will be set back to zero. That way the melody will loop.

Here's a second version of the sketch that makes these changes.

StopMelody2.zip

The two variables are setup before the loop:

```
    int numNotesToPlay = random(numNotes,3*numNotes);
    int noteToPlay = 0;
    for (int thisNote = 0; thisNote &lt; numNotesToPlay; thisNote++) { // play
notes of the melody
```

We've increased the range of `numNotesToPlay` (up to 3 times the length of the song). Then, the new variable `noteToPlay` is used to actually play the note:

```
      int noteDuration = 1000 / noteDurations[noteToPlay];
      CircuitPlayground.playTone(melody[noteToPlay], noteDuration);
```

And it is incremented and checked at the end of the loop:

```
      // increment and check note counter
      noteToPlay++;
      if (noteToPlay &gt;= numNotes) noteToPlay = 0;
```

Try again pressing the right button and play the melody multiple times to see how the length varies. This time however, the melody will play through at least once and then stop randomly at some point after that.

Shake to Start

OK, so now we can play a melody and have it stop after a random amount of time (number of notes). But how do we start the whole thing and get the game rolling? Using the buttons on the Circuit Playground would be a great way to do this. However, as you will see when we make the 'potato', the buttons may not be easily accessible. Instead, let's use the accelerometer. Then we'll just shake to start.

To do this, we can just re-use the 'shake detect' method discussed in the Circuit Playground D6 Dice () guide. We just need a function that returns the total acceleration value:

```
float getTotalAccel() {
  // Compute total acceleration
  float X = 0;
  float Y = 0;
  float Z = 0;
  for (int i=0; i&lt;10; i++) {
    X += CircuitPlayground.motionX();
    Y += CircuitPlayground.motionY();
    Z += CircuitPlayground.motionZ();
    delay(1);
  }
  X /= 10;
  Y /= 10;
  Z /= 10;

  return sqrt(X*X + Y*Y + Z*Z);
}
```

Then to wait for shaking, we just check the value and do nothing until it exceeds a preset value. Yes, this is pretty boring.

```
  // Wait for shaking
  while (getTotalAccel() &lt; SHAKE_THRESHOLD) {
    // do nothing
  }
```

Hot Potato Code

OK, we are ready to put this all together into our Circuit Playground Hot Potato sketch. Here's the final result:

HotPotato.zip

Download it, unzip, open it in the Arduino IDE, and upload it to your Circuit Playground.

Note that the definition of the melody has been moved to the file melody.h. This was done mainly to clean up the main sketch. But it also makes it a little more modular, making it easier to swap out the melody for a different one.

How To Play

With the Hot Potato sketch loaded and running on the Circuit Playground you're ready to play. Here's how:

The NeoPixels will be all red at the start (and end) of the game.

SHAKE TO START!

The melody will play and random NeoPixels will light up.

ZOMG! START TOSSING THE HOT POTATO!!!!

When the melody stops, all the lights will turn red again.

GAME OVER.

(shake to play again)

CircuitPython

The following pages develop the Hot Potato game using CircuitPython ().

Playing a Melody

We will use the speaker on the Circuit Playground Express to create a melody by playing a series of simple tones. We will store the actual notes of the melody in one tuple and the length of each note in another.

For example, here's a short little melody:

```
        # notes in the melody:
melody = (
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
)

# note durations: 4 = quarter note, 8 = eighth note, etc.:
tempo = (
  4, 8, 8, 4, 4, 4, 4, 4
)
```

To play this back, we simply loop through the tuple and play each note. For the Circuit Playground, that would look something like this:

```
    for i in range(len(melody)):
    note_duration = 1 / tempo[i]
    note = melody[i]
    if note == 0:
        time.sleep(note_duration)
    else:
        cpx.play_tone(note, note_duration)
```

We first compute the actual note duration based on the tempo value. We then check if the note is 0, which indicates a rest. If it is, we don't play anything and just sleep. Otherwise, we play the note.

Pitch Definitions

Note how in the melody array we used values that looked like `NOTE_C4`. This isn't some kind of built in CircuitPython magic. It is simply a variable that has been defined with the frequency value of the note C.

This information is stored in a separate file called pitches.py which you will need to place on your Circuit Playground Express. It is included in the zip file with the rest of the code. That is where all of these values are defined. It is nothing but a file with a bunch of lines that look like:

```
NOTE_C4  = 262
```

And that's the line that defines `NOTE_C4` as having a value of `262`.

Sample Melody

Here's a set of files which plays the simple melody from the example above whenever either Circuit Playground button is pressed.

play_melody.zip

It's a zip file which contains both the main program play_melody.py along with the pitc hes.py file that defines the notes. So download it,unzip it, and copy both files to your Circuit Playground Express. You'll know you got it working when you hear Shave and a Haircut.

Be sure to change the name of play_melody.py to main.py so the program will run when you press reset.

Stopping a Melody

So now we know how to play a melody. For our Hot Potato game, we will want to let this melody play for a random amount of time and then stop it. How do we do that?

There is a very simple way we can do this. Instead of thinking in terms of time, just think in terms of number of notes. We'll just pick a random number of notes and play only that many.

Stop Melody 1

Here is a modified version of the previous program which only plays a random number of notes from the melody. It also moves the melody definition to a separate file called melody.py.

stop_melody1.zip

Stop Melody 2

The one problem with the previous approach is it will play the melody at most only once. For our Hot Potato game, we may want the melody to play longer (multiple times) to make the game more exciting. So we want to be able to specify a random number bigger than the number of notes in the melody. Say the melody had 10 notes. A loop of 10 would play it once. But we want to be able to loop 20 times to play it twice. Or 17 times to play it once, and then the first 7 notes again. Etc.

But we can't simply increase the size of our random number and use the same loop structure. Once the loop number exceeds the number of notes in the melody, bad things will happen. For example, there is no value for `melody[number_of_notes+1]`. In fact, because of zero indexing, the highest defined note is only `melody[number_of_notes-1]`.

So how do we deal with this? One answer is to simply use two variables. The first will be our random loop variable that will be the total number of notes to play. The second will be used to actually play the melody note. It will be incremented manually inside the loop, and once it exceeds the size of the melody, it will be set back to zero. That way the melody will loop.

Here's a second version of the sketch that makes these changes.

stop_melody2.zip

Shake to Start

OK, so now we can play a melody and have it stop after a random amount of time (number of notes). But how do we start the whole thing and get the game rolling? Using the buttons on the Circuit Playground would be a great way to do this. However, as you will see when we make the 'potato', the buttons may not be easily accessible. Instead, let's use the accelerometer. Then we'll just shake to start.

To do this, we can just re-use the 'shake detect' method discussed in the Circuit Playground D6 Dice () guide. We just need a function that returns the total acceleration value:

```
def get_total_accel():
    # Compute total acceleration
    X = 0
    Y = 0
    Z = 0
    for count 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

    return math.sqrt(X*X + Y*Y + Z*Z)
```

Then to wait for shaking, we just check the value and do nothing until it exceeds a preset value. Yes, this is pretty boring.

```
    # Wait for shaking
    while get_total_accel() &lt; SHAKE_THRESHOLD:
        pass  # do nothing
```

Hot Potato Code

OK, we are ready to put this all together into our Circuit Playground Express Hot Potato sketch. Download and unzip the following file:

hot_potato.zip

You should get the following 3 files:

- hot_potato.py - the main game
- melody.py - defines a melody as a series of notes

- pitches.py - defines tones for each note

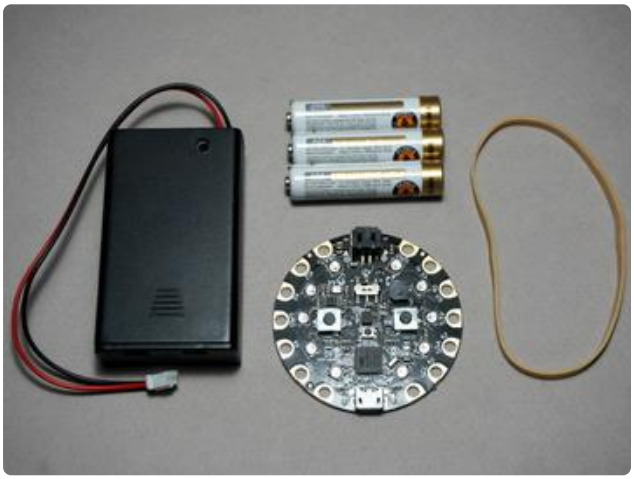
Place all 3 files on your Circuit Playground Express. To have the code run when you press reset, rename hot_potato.py to main.py.

How To Play

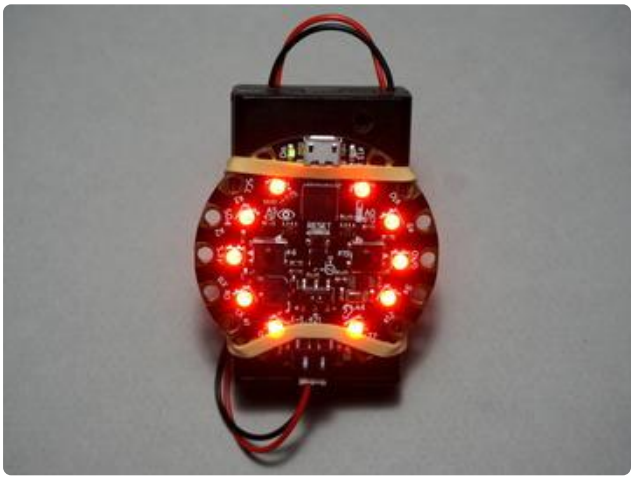
With the Hot Potato sketch loaded and running on the Circuit Playground you're ready to play. See the Arduino section for details.

Building the Potato

We got the code, now we need the potato. This doesn't need to be fancy. In fact, you can make one with not much more than a rubber band.



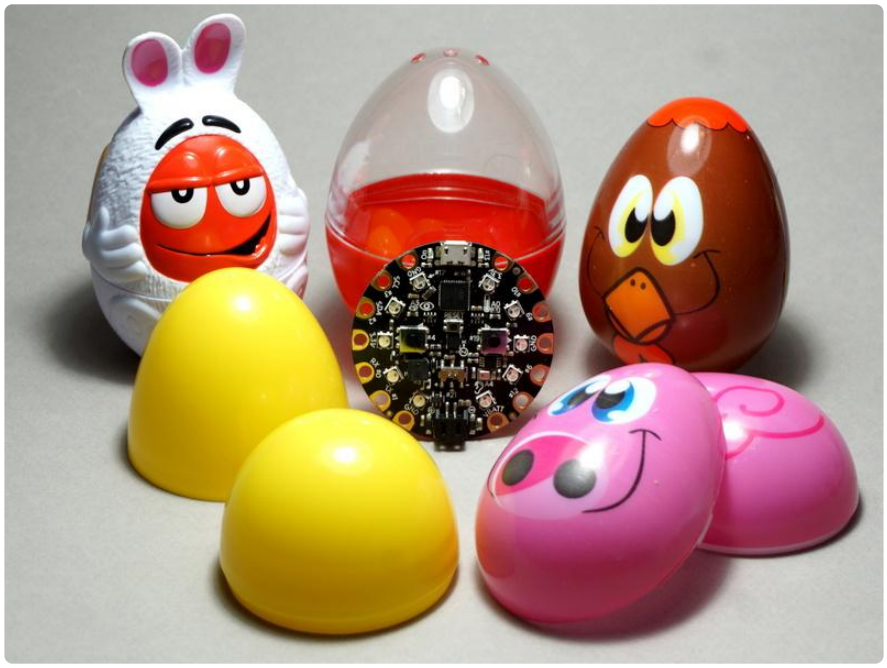
Use a rubber band to hold the Circuit Playground to the 3 x AAA Battery Holder.



Ready for action. Give it a shake and start tossing!

Eggy Eggy

Just so happens that this guide is being written near Easter time. This provides us the opportunity to make our potato out of a currently readily available item...



PLASTIC EASTER EGGS!

I just went to the store and looked around. I ended up finding several styles of plastic Easter eggs that were big enough for the Circuit Playground to fit in. Some of these had candy in them, so I had to eat that first. The one with the clear top had a toy inside. Others were just empty.

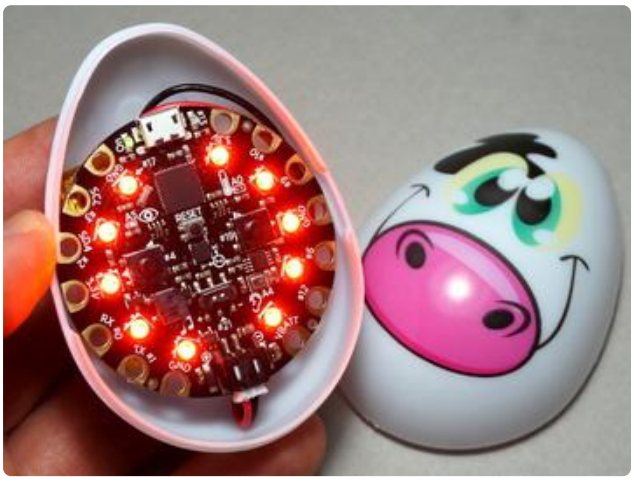
Battery Fit Check

This egg style had room for a small LiPo battery.

However, the 3 x AAA battery pack would not fit. :(

This egg style could fit the 3 x AAA battery pack, as well as the small LiPo battery. As a bonus, it had a clear top! :)

Eggy One



Here's the first egg with the Circuit Playground and the small LiPo battery installed. Ready to be closed up!

Eggy Two

Here's the second egg with the Circuit Playground and the 3 x AAA battery pack installed. Good to go!

Battery Cushion

Since the HPE (Hot Potato Egg) is going to be tossed around a lot during game play, it is a good idea to add some cushioning to the inside. You can use anything soft and stuffable, like bubble wrap.

Reuse some bubble wrap from a previous shipment. Some of the Adafruit shipping envelopes are even made of bubble wrap.

Get it all packed in there nice a tight (but not too tight).

Close it up and you're good to go!

YOU'RE READY TO PLAY. HAVE FUN!

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. Or do something new and exciting!

Questions

- What is the maximum length (in number of notes) of the game for the provided sketch?
- Why is a `delay(2000)` added to the end of the loop? (hint: read the code comment)

Code Challenges

- Change the melody that is played during the game.
- Make the NeoPixels do something different during game play.
- Use the accelerometer to check how 'softly' players are catching the egg. If they catch it too hard, they 'break' the egg and the game is over.