code

SLQ Wiki Fabrication Lab 2026/05/31 22:06

First version of the Neopixel workshop code:

```
//Ne0n00bp1x3l l1br4ry
// Bodged together by Byron for TheEdge's Arduino 101 'neopixel' workshop
// With help from Vince

//Code from:
//Adafruit's 'strandtest' example
//https://www.tweaking4all.com/hardware/arduino/adruino-led-strip-effects/
//https://forum.arduino.cc/index.php?topic=418923.msg2886563#msg2886563
//And other places

// This line makes the AdaFruit NeoPixel library available to this sketch.
//  The NeoPixel library handles the gory details of writing a high speed,
finicky set of commands to the pixels.
#include <Adafruit_NeoPixel.h>

// This line makes the word "PIN_LEDS" look like the number '6' to the
computer. Your strip should be attached to pin 6 on your Arduino!
// We could just use the number '6' instead, but using a named constant
describes what the number is FOR, and makes your code easier to read and
fix.
#define PIN_LEDS 6

// How many LEDs are in the string
// Each segment is 30 LEDs long. A meter is 60 LEDs. A full spool is 5m
long, so 300 LEDs
// Don't set this any longer than your strip.
// 30 LEDs is about as much as a single normal USB2 port can power. If you
want to light up longer strips, you'll need an external power supply
//  (or you might blow up your USB port or computer).
#define NUM_LEDS 30

// Configure the neopixel library with the correct settings for the strips
we are using
// in order: how many LEDs in the connected strip, which Arduino pin the
strip is connected to, the RGB ordering our pixels use (can vary), and the
speed.
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, PIN_LEDS, NEO_GRB +
NEO_KHZ800);

// The Arduino builtin function setup() runs once, when the microcontroller
starts up. Put any once-off setup code in setup()
void setup() {
  // Turn on strip
  strip.begin();

  // Writes LED values to strip, then tells strip to turn ON.
```

```
  // Changes you make to the LEDs won't display until you call show() !!
  strip.show();

 // The overall brightness of all the LEDs can be adjusted using
setBrightness().
 //This takes a single argument, a number in the range 0 (off) to 255 (max
brightness).
 //For example, to set a strip to 1/2 brightness:
 strip.setBrightness(128);
}  // end setup()

// The Arduino runs the loop() function in an infinite repeat, until it is
turned off. Put your program code in here
void loop() {

  // setAllRGB will fill the entire strip with a single color - specified as
Red, Green, Blue components
  // Because of the way eyes work, a mix of different amounts of red, green
and blue light can look like other colors - eg yellow
  // Because of this, you'll see colors specified a lot as a red, green,
blue (RGB) triplet.
  // This sets all three channels to zero (off).
  setAllRGB(0,0,0);
  // Now, update the strip
  showStrip();

/*
 * Setting LED colours individually.
 * setPixelRGB() changes a single pixel in the strip
 * First number is the position of the pixel to set. Pixel zero is the FIRST
pixel in the string, and pixel 1 is the SECOND, and so on.
 * The next three parameters are the 'red', then 'green' then 'blue' values.
 * Each channel can be from 0 to 255, but not greater, and not negative.
Also, no fractional numbers (eg 2.5)
*/

  setPixelRGB(0, 255,0,0);  // Set the FIRST pixel in the string to full
brightness RED
  showStrip();   // Update change to the strip
// The Arduino builtin delay() function pauses program execution for the
indicated number of milliseconds. There are 1000 mS in one second.
  delay(200);  // Wait 200mS

  setPixelRGB(1,0,255,0);  // Set the SECOND pixel to full brightness GREEN
  showStrip();   // Update change to the strip
  delay(200);  // Wait 200mS

  setPixelRGB(2,0,0,255);  // Set the third pixel to full brightness BLUE
```

```
  showStrip();   // Update change to the strip
  delay(500);  // Wait 500mS

  setPixelRGB(0,0,0,0);  // First pixel off
  setPixelRGB(1,0,0,0);  // Second pixel off
  setPixelRGB(2,0,0,0);  // Third pixel off
  showStrip();   // Update change to the strip
  delay(1000);  // Wait 1000mS

/*
 * Instead of using RGB values, you can 'pack' them into a single value (a
big number) and store that number in a named container, called a VARIABLE.
 * This lets you pass around just one value, instead of three.
 * When you use a variable's name in your code, the Arduino retrieves the
contents of that variable and uses it right there.
 * This allows you to change the colour without changing your whole pattern.
 * The first time you refer to a variable, the Arduino creates it. When you
create a variable, you need to tell the Arduino what 'shape' it is. This is
what the 'uint32_t' bit is for.
 * 'uint32_t' defines the variable as 'big enough to hold an Unsigned
INTeger of 32 bits length', which can hold a single whole number between 0
and 4.2 billion.
 * You only need this the first time you refer to a variable.
 */

// Full RED and GREEN make YELLOW looking light
uint32_t mycolour1 = RGBtoPacked(255,255, 0);

// Full RED and BLUE make MAGENTA (reddish purple) looking light
uint32_t mycolour2 = RGBtoPacked(255, 0, 255);

// Full GREEN and BLUE make CYAN (light blue) light
uint32_t mycolour3 = RGBtoPacked(0, 255, 255);

// Setting everything to 0 turns off the pixel entirely, for 'black'
uint32_t mycolour4 = RGBtoPacked(0, 0, 0);

// Setting RED, GREEN and BLUE to equal amounts gives a WHITE appearing
light
uint32_t mycolour5 = RGBtoPacked(255, 255, 255);

// You can assign variables to other variables... and using descriptive
names makes your code easier to use, and fix!
uint32_t color_White = mycolour5;

// Changing mycolour5 won't change color_White. Assignment does not 'link'
variables, the data is copied from the second to the first and now lives in
two places.
```

```
mycolour5 = RGBtoPacked(0, 64, 0); // careful of spelling!!

// What color is in mycolour5 now?
// Lots of other colours are possible by choosing different mixes of red,
green and blue!

  setPixelPacked(0, mycolour1);  // Set first pixel to the color stored in
mycolor1
  setPixelPacked(2, mycolour1);  // Set third pixel to the color stored in
mycolor1 also
  showStrip();
  delay(500);
  setPixelPacked(1, mycolour2);  // Second pixel
  setPixelPacked(4, mycolour2);  // Fifth pixel
  showStrip();
  delay(500);

  // let's define a variable for the pixel position, too.
  uint8_t pixelPosition = 3;
  setPixelPacked(pixelPosition, mycolour3);  // Fourth pixel
  pixelPosition = 5;
  setPixelPacked(pixelPosition, mycolour3);  // Sixth pixel
  showStrip();
  delay(1000);

  // And now, turn all 6 pixels off, by assigning the same color ('black')
to each
  setPixelPacked(0, mycolour4);
  setPixelPacked(1, mycolour4);
  setPixelPacked(2, mycolour4);
  setPixelPacked(3, mycolour4);
  setPixelPacked(4, mycolour4);
  setPixelPacked(5, mycolour4);
  showStrip();
  delay(1000);

  // You can use a code structure called a 'FOR loop' to do repetitious
things. if used, this should have the same effect as the six setPixelPacked
calls above:
  /*
   *  for ( uint8_t position = 0; position < 6; ++position ) {
   *    setPixelPacked(position, mycolour4);
   *  }
   *  showStrip();
   *  delay(1000);
   */

/* Predefined pretty animation patterns, for you to play with. The code of
```

```
each function is at the bottom of this file - you can change it if you like
 *  ... but keep a backup, in case you break it!
 *
 *  colorWipe() is a function that will animate progressively filling the
entire strip with a single color.
 *    You can write to only every second, third, ... pixel by changing the
second parameter.
 * First parameter is a color, in PackedColor format (32 bit unsigned
integer).
 *  Second: set N, to update every Nth pixel ( 1 = all pixels 2 = every
second 3 = every third, and so on)
 *  third: starting pixel position - skips updating all pixels before this
 *  fourth: delay (in mS) between each frame of the fill animation
 */
  colorWipe(RGBtoPacked(255, 0, 0), 1, 0, 50); // Every Red
  delay(500);
  colorWipe(RGBtoPacked(0, 255, 0),  2, 0, 50); // Every second green
  delay(500);
  colorWipe(RGBtoPacked(0, 0, 255),  2, 1, 50); // Replace just the red with
Blue
  delay(500);
  colorRetract(RGBtoPacked(255, 255, 0),  3, 1, 100); // Yellow backwards,
every third slower
  delay(500);
  colorWipe(RGBtoPacked(0, 0, 0),  2, 0, 20); // Quickly half off forward
  colorRetract(RGBtoPacked(0, 0, 0),  2, 0, 20); // Quickly half  off
backwards
  delay(1000);

/*
 * FadeIn() evenly brightens the entire strip from black to full intensity
of the supplied colour. FadeOut() fades entire strip from full brightness to
dark.
 *  First parameter is packed colour, second is mS per frame of animation.
 */
  FadeIn(RGBtoPacked(255, 255, 255), 10); // white
  FadeOut(RGBtoPacked(255, 255, 255), 0); // white
  delay(1000);

/*
 * Fade from one colour to another, using two sets of Red Green Blue values
 * Note: This won't produse the most accurate colours, but it's simple.
 */
//  RGBcrossFade(RGBtoPacked(10, 10, 255),RGBtoPacked(255, 0, 100),10);
//  delay(500);
  RGBcrossFade(RGBtoPacked(0, 0, 255),RGBtoPacked(255, 0, 0),10);
  delay(1000);
```

```
/* You can combine the two!
 * This uses colours defined eairler
 */
  FadeIn(mycolour1, 10);                          // Fade up first colour
  RGBcrossFade(mycolour1,mycolour3,10);           // Go from first to
second
  RGBcrossFade(mycolour3,mycolour2,10);           // Go from second to
third
  FadeOut(mycolour2, 10);                         // Fade down third colour
  delay(1000);


/*
 * Scanner style examples:
 * (PackedColor, EyeSize, SpeedDelay, ReturnDelay);
 */

  RightToLeft(mycolour2, 4, 10, 200);
  LeftToRight(mycolour2, 4, 10, 200);
  OutsideToCenter(mycolour1, 2, 10, 200);
  CenterToOutside(mycolour1, 2, 10, 200);
  delay(1000);


/*
 * Cylon aka. Larsen scanner
 * Cylon() function takes 4 parameters, where the first is your desired
colour. Packed
 * The 4th parameter (EyeSize) determines how many LEDs run around, or: the
width of the “eye” (outer 2, faded, LEDs not counted).
 * The 5th parameter (SpeedDelay) influences how fast the eye moves, higher
values means slow movement.
 * The last parameter (ReturnDelay) sets how much time it should wait to
bounce back.
*/
// BROKEN
// CylonBounce(RGBtoPacked(255, 0, 0), 4, 10, 50);
// delay(1000);

/*
 * Strobe:
 * The first is PackedColour  2nd parameter (StrobeCount) indicates how many
flashes you’d like to see.
 * Parameters 3 (FlashDelay) delay between each individual flash.
 * EPILEPTICS BEWARE!!
 */
//Strobe(mycolour1, 20, 50);
//Strobe(mycolour2, 20, 50);
```

```
//Strobe(mycolour3, 20, 50);
//Strobe(mycolour4, 20, 50);
//delay(1000);

/*
 * Twinkle
 * first is Packed Colour
 * The 2th parameter (Count) determines how many pixels will be done in one
run,
 * 3rd 5th parameter determines how much time will be paused between
individual pixels (speed).
 * The 4th parameter (OnlyOne) should be true if you want to see only one
LED at a time. If it’s set to false then all “Count” number of LEDs will be
visible (added one at a time).
 */
 Twinkle(RGBtoPacked(10, 10, 255), 10, 100, false);
 delay(1000);

/*
 * This is a variation on the Twinkle() effect.
 * Difference is that the colors are now randomly generated, and therefor
color parameter is no longer needed.
 * The first parameter (Count) determines how many pixels will be done in
one run
 * The second parameter determines how much time will be paused between
individual pixels (speed).
 * The last parameter (OnlyOne) should be true if you want to see only one
LED at a time.
 * If it’s set to false then all “Count” number of LEDs will be visible
(added one at a time).
 */
  TwinkleRandom(20, 100, true);
  delay(1000);


/*
 * Randomly blink LEDs
 * Takes PackedColour as first parameter, delay between pixels as second
 */
  Sparkle(RGBtoPacked(255,255,255), 300);


/*  An alternative to specifying numbers by R,G,B is to specify them by Hue
(color), Saturation (hcolor intensity. 0 = grayscale 255 = rainbow) and
Value (how bright)
 *   This is referred to as HSV (sometimes HSL where L = Luminance). Using
this representation, you can independently control the color, saturation and
brightness of your colours easily.
```

```
 *   The Neopixel library does not understand HSV colors, so they still need
to be converted to RGB for the strip pixels to use, but we have a function
for that!
 *   The h12sv2rgb() function converts a Hue, Saturation, Value color into a
packed RGB one that you can set pixels to.
 */

// write a rainbow to the strip
uint32_t aHue;
for (  pixelPosition = 0; pixelPosition < NUM_LEDS; ++pixelPosition) {
  aHue = (1536 * pixelPosition) / NUM_LEDS;
  setPixelPacked(pixelPosition, hsvToPacked(aHue, 255, 255));
}
showStrip();
delay(5000);

// this function writes all 1536 hues to the strip
for( uint16_t hue = 0; hue < 1536; ) {
  for ( uint8_t pixelPosition = 0; pixelPosition < NUM_LEDS;
++pixelPosition) {
    setPixelPacked(pixelPosition, hsvToPacked(hue, 255, 255));
    hue = hue + 1;
  }
  showStrip();
  delay(100);
}

delay(250);
setAllRGB(0,0,0);



/*
 *      More complicated and thus less generel effect
 *      Some of these take a long time to run or have other problems
 *      One or two might not be able to be combined with other effects
 */


/*
 *  Rainbow Cycle
 * cycles rainbow colors, where the only parameter is the speed delay.
 * Completes after it goes through all colours
 */
//rainbowCycle(20);

/*
 * This effect looks best when hanging your LED strip vertical and it
```

```
simulates a one LED wide “fire”,
 * and is adapted from an example in FastLED, which is adapted from work
done by Mark Kriegsman (called “Fire2012”).
 * This function takes 3 parameters.
 * The first one (Cooling) indicates how fast a flame cools down. More
cooling means shorter flames, and the recommended values are between 20 and
100. 50 seems the nicest.
 * The Second parameter (Sparking), indicates the chance (out of 255) that a
spark will ignite. A higher value makes the fire more active. Suggested
values lay between 50 and 200, with my personal preference being 120.
 * The last parameter (SpeedDelay) allows you to slow down the fire activity
… a higher value makes the flame appear slower.
*/
//Fire(55,120,15);


 //byte colors[3][3] = { {0xff, 0,0},  {0xff, 0xff, 0xff}, {0   , 0   ,
0xff} };
 //BouncingColoredBalls(3, colors);

}   // End loop()



//  ---------------------------------  Functions and other messy stuff below
here! ------------------------


// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t every, uint8_t offset, int wait) {
  if(every<1){every=1;}
  for(uint16_t i=offset; i<NUM_LEDS; i=i+every) {
    setPixelPacked(i, c);
    showStrip();
    delay(wait);
  }
}

// Same as colorWipe but backwards
void colorRetract(uint32_t c,  uint8_t every, uint8_t offset, uint8_t wait)
{
  if(every<1){every=1;}
  for(int i=NUM_LEDS+1-offset; i>-1; i=i-every) {
    setPixelPacked(i, c);
    showStrip();
    delay(wait);
  }
}
```

```
//https://forum.arduino.cc/index.php?topic=418923.msg2886563#msg2886563
void RGBcrossFade(const uint32_t startColor, const uint32_t endColor, int
speed) {
  byte startRed = (startColor >> 16) & 0xff;
  byte startGreen = (startColor >> 8) & 0xff;
  byte startBlue = startColor & 0xff;

  byte endRed = (endColor >> 16) & 0xff;
  byte endGreen = (endColor >> 8) & 0xff;
  byte endBlue = endColor & 0xff;

  // for each step in the cross-fade
  for (int step = 0; step < 256; step++) {
    byte red = map(step, 0, 255, startRed, endRed);
    byte green = map(step, 0, 255, startGreen, endGreen);
    byte blue = map(step, 0, 255, startBlue, endBlue);
    setAllRGB(red,green,blue);
    delay(speed);
  }
}


void rainbowCycle(int SpeedDelay) {
  uint16_t i, j;

  for(j=0; j<256; j++) { // cycle all colors on wheel
    for(i=0; i< NUM_LEDS; i++) {
      setPixelPacked(i, Wheel(((i * 256 / NUM_LEDS) + j) & 255));
    }
    showStrip();
    delay(SpeedDelay);
  }
}

void Strobe(const uint32_t c, int StrobeCount, int FlashDelay){
  for(int j = 0; j < StrobeCount; j++) {
    setAllPacked(c);
    showStrip();
    delay(FlashDelay);
    setAllRGB(0,0,0);
    showStrip();
    delay(FlashDelay);
  }
}


void CenterToOutside(const uint32_t c, int EyeSize, int SpeedDelay, int
ReturnDelay) {
```

```
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  for(int i =((NUM_LEDS-EyeSize)/2); i>=0; i--) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    setPixelRGB(NUM_LEDS-i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(NUM_LEDS-i-j, red, green, blue);
    }
    setPixelRGB(NUM_LEDS-i-EyeSize-1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
  }
  delay(ReturnDelay);
}

void OutsideToCenter(const uint32_t c, int EyeSize, int SpeedDelay, int
ReturnDelay) {
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  for(int i = 0; i<=((NUM_LEDS-EyeSize)/2); i++) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    setPixelRGB(NUM_LEDS-i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(NUM_LEDS-i-j, red, green, blue);
    }
    setPixelRGB(NUM_LEDS-i-EyeSize-1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
  }
  delay(ReturnDelay);
}

void LeftToRight(const uint32_t c, int EyeSize, int SpeedDelay, int
ReturnDelay) {
```

```
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  for(int i = 0; i < NUM_LEDS-EyeSize-2; i++) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
  }
  delay(ReturnDelay);
}

void RightToLeft(const uint32_t c, int EyeSize, int SpeedDelay, int
ReturnDelay) {
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  for(int i = NUM_LEDS-EyeSize-2; i > 0; i--) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
  }
  delay(ReturnDelay);
}

void CylonBounce(byte red, byte green, byte blue, int EyeSize, int
SpeedDelay, int ReturnDelay){

  for(int i = 0; i < NUM_LEDS-EyeSize-2; i++) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
```

```
  }

  delay(ReturnDelay);

  for(int i = NUM_LEDS-EyeSize-2; i > 0; i--) {
    setAllRGB(0,0,0);
    setPixelRGB(i, red/10, green/10, blue/10);
    for(int j = 1; j <= EyeSize; j++) {
      setPixelRGB(i+j, red, green, blue);
    }
    setPixelRGB(i+EyeSize+1, red/10, green/10, blue/10);
    showStrip();
    delay(SpeedDelay);
  }
  delay(ReturnDelay);
}


void FadeIn(const uint32_t c, int delayval){
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;

  float r, g, b;
  for(int k = 0; k < 256; k=k+1) {
    r = (k/256.0)*red;
    g = (k/256.0)*green;
    b = (k/256.0)*blue;
    setAllRGB(r,g,b);
    showStrip();
    delay(delayval);
  }
}

void FadeOut(const uint32_t c, int delayval){
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  float r, g, b;
  for(int k = 255; k >= 0; k=k-2) {
    r = (k/256.0)*red;
    g = (k/256.0)*green;
    b = (k/256.0)*blue;
    setAllRGB(r,g,b);
    showStrip();
    delay(delayval);
```

```
  }
}


void TwinkleRandom(int Count, int SpeedDelay, boolean OnlyOne) {
  setAllRGB(0,0,0);
  for (int i=0; i<Count; i++) {
setPixelRGB(random(NUM_LEDS),random(0,255),random(0,255),random(0,255));
     showStrip();
     delay(SpeedDelay);
     if(OnlyOne) {
       setAllRGB(0,0,0);
     }
   }
   delay(SpeedDelay);

}

void Twinkle(const uint32_t c, int Count, int SpeedDelay, boolean OnlyOne) {
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  setAllRGB(0,0,0);
  for (int i=0; i<Count; i++) {
     setPixelRGB(random(NUM_LEDS),red,green,blue);
     showStrip();
     delay(SpeedDelay);
     if(OnlyOne) {
       setAllRGB(0,0,0);
     }
   }
  delay(SpeedDelay);
}



void BouncingColoredBalls(int BallCount, byte colors[][3]) {
  float Gravity = -9.81;
  int StartHeight = 1;
  float Height[BallCount];
  float ImpactVelocityStart = sqrt( -2 * Gravity * StartHeight );
  float ImpactVelocity[BallCount];
  float TimeSinceLastBounce[BallCount];
  int   Position[BallCount];
  long  ClockTimeSinceLastBounce[BallCount];
  float Dampening[BallCount];
  for (int i = 0 ; i < BallCount ; i++) {
```

```
    ClockTimeSinceLastBounce[i] = millis();
    Height[i] = StartHeight;
    Position[i] = 0;
    ImpactVelocity[i] = ImpactVelocityStart;
    TimeSinceLastBounce[i] = 0;
    Dampening[i] = 0.90 - float(i)/pow(BallCount,2);
  }

  while (true) {
    for (int i = 0 ; i < BallCount ; i++) {
      TimeSinceLastBounce[i] =  millis() - ClockTimeSinceLastBounce[i];
      Height[i] = 0.5 * Gravity * pow( TimeSinceLastBounce[i]/1000 , 2.0 ) +
ImpactVelocity[i] * TimeSinceLastBounce[i]/1000;
      if ( Height[i] < 0 ) {
        Height[i] = 0;
        ImpactVelocity[i] = Dampening[i] * ImpactVelocity[i];
        ClockTimeSinceLastBounce[i] = millis();
        if ( ImpactVelocity[i] < 0.01 ) {
          ImpactVelocity[i] = ImpactVelocityStart;
        }
      }
      Position[i] = round( Height[i] * (NUM_LEDS - 1) / StartHeight);
    }
    for (int i = 0 ; i < BallCount ; i++) {
      setPixelRGB(Position[i],colors[i][0],colors[i][1],colors[i][2]);
    }
    showStrip();
    setAllRGB(0,0,0);
  }
}


void Fire(int Cooling, int Sparking, int SpeedDelay) {
  static byte heat[NUM_LEDS];
  int cooldown;
  // Step 1.  Cool down every cell a little
  for( int i = 0; i < NUM_LEDS; i++) {
    cooldown = random(0, ((Cooling * 10) / NUM_LEDS) + 2);
    if(cooldown>heat[i]) {
      heat[i]=0;
    } else {
      heat[i]=heat[i]-cooldown;
    }
  }
  // Step 2.  Heat from each cell drifts 'up' and diffuses a little
  for( int k= NUM_LEDS - 1; k >= 2; k--) {
    heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2]) / 3;
  }
```

```
  // Step 3.  Randomly ignite new 'sparks' near the bottom
  if( random(255) < Sparking ) {
    int y = random(7);
    heat[y] = heat[y] + random(160,255);
    //heat[y] = random(160,255);
  }

  // Step 4.  Convert heat to LED colors
  for( int j = 0; j < NUM_LEDS; j++) {
    setPixelHeatColor(j, heat[j] );
  }

  showStrip();
  delay(SpeedDelay);
}

void setPixelHeatColor (int Pixel, byte temperature) {
  // Scale 'heat' down from 0-255 to 0-191
  byte t192 = round((temperature/255.0)*191);

  // calculate ramp up from
  byte heatramp = t192 & 0x3F; // 0..63
  heatramp <<= 2; // scale up to 0..252

  // figure out which third of the spectrum we're in:
  if( t192 > 0x80) {                     // hottest
    setPixelRGB(Pixel, 255, 255, heatramp);
  } else if( t192 > 0x40 ) {             // middle
    setPixelRGB(Pixel, 255, heatramp, 0);
  } else {                               // coolest
    setPixelRGB(Pixel, heatramp, 0, 0);
  }
}

void Sparkle(const uint32_t c, int SpeedDelay) {
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
  int Pixel = random(NUM_LEDS);
  setPixelRGB(Pixel,red,green,blue);
  showStrip();
  delay(SpeedDelay);
  setPixelRGB(Pixel,0,0,0);
}

/*
byte * Wheel(byte WheelPos) {
```

```
  static byte c[3];
  if(WheelPos < 85) {
   c[0]=WheelPos * 3;
   c[1]=255 - WheelPos * 3;
   c[2]=0;
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   c[0]=255 - WheelPos * 3;
   c[1]=0;
   c[2]=WheelPos * 3;
  } else {
   WheelPos -= 170;
   c[0]=0;
   c[1]=WheelPos * 3;
   c[2]=255 - WheelPos * 3;
  }
  return c;
}
*/

uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return RGBtoPacked(255 - WheelPos * 3, 0, WheelPos * 3);
   }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return RGBtoPacked(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return RGBtoPacked(WheelPos * 3, 255 - WheelPos * 3, 0);
}


void showStrip() {
 #ifdef ADAFRUIT_NEOPIXEL_H
   // NeoPixel
   strip.show();
 #endif
 #ifndef ADAFRUIT_NEOPIXEL_H
   // FastLED
   FastLED.show();
 #endif
}

// RGB sets
void setPixelRGB(int Pixel, byte red, byte green, byte blue) {
 #ifdef ADAFRUIT_NEOPIXEL_H
```

```
   // NeoPixel
   strip.setPixelColor(Pixel, strip.Color(red, green, blue));
 #endif
 #ifndef ADAFRUIT_NEOPIXEL_H
   // FastLED
   leds[Pixel].r = red;
   leds[Pixel].g = green;
   leds[Pixel].b = blue;
 #endif
}

void setAllRGB(byte red, byte green, byte blue) {
  for(int i = 0; i < NUM_LEDS; i++ ) {
    setPixelRGB(i, red, green, blue);
  }
  showStrip();
}

// Packed sets. Thanks to: https://forum.arduino.cc/index.php?topic=512914.0

// Convert separate R,G,B into packed 32-bit RGB color.
// Packed format is always RGB, regardless of LED strand color order.
uint32_t RGBtoPacked(uint8_t r, uint8_t g, uint8_t b) {
  return ((uint32_t)r << 16) | ((uint32_t)g <<  8) | b;
}

void setPixelPacked(int Pixel, uint32_t c) {
 #ifdef ADAFRUIT_NEOPIXEL_H
   // NeoPixel
   strip.setPixelColor(Pixel, c);
 #endif
 #ifndef ADAFRUIT_NEOPIXEL_H
   // FastLED
   uint8_t *p,
   red = (uint8_t)(c >> 16),
   green = (uint8_t)(c >>  8),
   blue = (uint8_t)c;
   leds[Pixel].r = red;
   leds[Pixel].g = green;
   leds[Pixel].b = blue;
 #endif
}

void setAllPacked(uint32_t c) {
  for(int i = 0; i < NUM_LEDS; i++ ) {
    setPixelPacked(i, c);
  }
  showStrip();
```

```
}

void printColorPacked(uint32_t c){
  uint8_t *p,
  red = (uint8_t)(c >> 16),
  green = (uint8_t)(c >>  8),
  blue = (uint8_t)c;
  Serial.print("C:");  Serial.print(c,HEX);
  Serial.print(" R:");  Serial.print(red);
  Serial.print(" G:"); Serial.print(green);
  Serial.print(" B:"); Serial.println(blue);
}

/* H12SV - HSV algorithm with 12 bit Hue component (1536 possible hues)
 * Created by: Vince Sutton
 * Modified from 8 bit HSV -> RGB algorithms.
 * Not final version, may contain bugs :) YMMV!
 * Note only lowest 12 bits of hue are used (value 0 - 1535) - if you assign
higher numbers, the final hue will be something strange.
 */
uint32_t hsvToPacked(uint16_t hue, uint8_t sat, uint8_t val) {

  if ( sat == 0     // no chroma component, R G B even
       || val == 0) // 0% bright, return black
  {
    return RGBtoPacked(val, val, val);
  }

  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;
  uint16_t tmp = sat * val;
  uint8_t chroma = tmp / 255;
  uint16_t hue2 = (hue % 1536);
  uint8_t chroma2 = hue2 % 256;

  if( hue2 > 1279 ) {  // magenta -> red
    r = chroma;
    b = ((255 - chroma2) * chroma) / 255;
  }
  else if (hue2 > 1023 ) {
    r = (chroma2 * chroma) / 255;
    b = chroma;
  }
  else if ( hue2 > 767) {
    g = ((255 - chroma2) * chroma) / 255;
    b = chroma;
  }
```

```
  else if ( hue2 > 511) {
    g = chroma;
    b = (chroma2 * chroma) / 255;
  }
  else if ( hue2 > 255 ) {
    r = ((255 - chroma2) * chroma) / 255;
    g = chroma;
  }
  else { // hue < 255
    r = chroma;
    g = (chroma2 * chroma) / 255;
  }
  uint8_t sv = val - chroma;
  return RGBtoPacked(
      r + sv, g + sv, b + sv);
}
```