```
// Each button randomly cycles through colors.
// Buttons Disabled
// Latch Disabled
// Code from various works, assembled and converted by Braden Licastro
// (c) 2011 Braden Licastro, FullForce Applications. All Rights Reserved


// START DECLARATIONS
#define DATAOUT 11 // MOSI (pin 7 of AD5206)
#define DATAIN 12 // MISO - not used, but part of builtin SPI
#define SPICLOCK 13 // sck (pin 8 of AD5206)
#define SLAVESELECT 10 // removed the slave switching code entirely
#define COLS 4 // x axis
#define ROWS 4 // y axis
#define H 254 // pot high
#define L 64 // pot low


// LED CIRCUIT

// Pins for led column grounding transistors
const byte colpin[COLS] = {
  14,15,16,17}; // Using the analog inputs as digital pins (14=A0,15=A1,16=A2,17=A3)

// The pot register numbers for each of the red, green, and blue channels
// Address map for AD5206:
// Pin bin dec
// 2   101 5
// 11  100 4
// 14  010 2
// 17  000 0
// 20  001 1
// 23  011 3
const byte red[2] = {
  5, 0};
const byte green[2] = {
  4, 1};
const byte blue[2] = {
  2, 3};

byte rGrid[COLS][ROWS] = {
  0};
byte gGrid[COLS][ROWS] = {
  0};
byte bGrid[COLS][ROWS] = {
  0};

byte trajectory[COLS][ROWS] = {
  0};

// Store elapsed time, used to manage animations.
unsigned long time;

```

```

// END DECLARATIONS, BEGIN PROGRAM


// SET UP EVERYTHING
void setup(){
  randomSeed(1);

  // Start Serial Output
  Serial.begin(19200);

  byte i;
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);

  for(byte c = 0; c < COLS; ++c){
    pinMode(colpin[c], OUTPUT); // Initialize rows
    digitalWrite(colpin[c], LOW); // Turn all rows off
  }

  digitalWrite(SLAVESELECT,HIGH); // Disable device

  // SPCR = 01010000
  // Interrupt disabled,spi enabled,msb 1st,master,clk low when idle,
  // Sample on leading edge of clk,system clock/4 (fastest)
  SPCR = (1<<SPE)|(1<<MSTR);
  clr=SPSR;
  clr=SPDR;
  delay(10);

  // Clear all of the pot registers
  for (i=0;i<6;i++)
  {
    write_pot(i,0);
  }

  grid_init();

  delay(10);

  // Milliseconds since applet start - Used to time animation sequence.
  time = millis();
}

// INFINITE LOOP, THE PROGRAM
void loop(){
  always();

  Serial.print(".");

  for(byte c = 0; c < COLS; ++c){
    for(byte r = 0; r < 2; ++r){
      write_pot(red[r],rGrid[c][r]);
      write_pot(green[r],gGrid[c][r]);
      write_pot(blue[r],bGrid[c][r]);
    }

    digitalWrite(colpin[c], HIGH); // Turn one row on
    delayMicroseconds(750); // Display
    digitalWrite(colpin[c], LOW); // Turn the row back off
  }
}


void grid_init(){
  grid_rand();
}

void grid_rand(){
  // Initialize the button grids with random data
  for(byte x = 0; x < COLS; ++x){
    for(byte y = 0; y < ROWS; ++y){
      rGrid[x][y] = random(0,256);
      gGrid[x][y] = random(0,256);
      bGrid[x][y] = random(0,256);
      trajectory[x][y] = random(1,8);
    }
  }
}

void grid_blank(){
  // Initialize the button grids with blank data
  for(byte x = 0; x < COLS; ++x){
    for(byte y = 0; y < ROWS; ++y){
      rGrid[x][y] = 0;
      gGrid[x][y] = 0;
      bGrid[x][y] = 0;
      trajectory[x][y] = random(1,8);
    }
  }
}

byte write_pot(byte address, byte value)
{
  digitalWrite(SLAVESELECT, LOW);
  // 2 byte opcode
  spi_transfer(address % 6);
  spi_transfer(constrain(255-value,0,255));
  digitalWrite(SLAVESELECT, HIGH); // Release chip, signal end transfer
}

char spi_transfer(volatile char data)
{
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
  return SPDR;                    // Return the received byte
}

//Color mixing and fading code
void always(){
  if((long)millis() - (long)time > 10){
    time = millis();
    for(byte x = 0; x < COLS; ++x){
      for(byte y = 0; y < ROWS; ++y){
        rGrid[x][y] = constrain(rGrid[x][y] + ((trajectory[x][y] & B001 ) ? 1 : -1),L, H);
        gGrid[x][y] = constrain(gGrid[x][y] + ((trajectory[x][y] & B010 ) ? 1 : -1),L, H);
        bGrid[x][y] = constrain(bGrid[x][y] + ((trajectory[x][y] & B100 ) ? 1 : -1),L, H);
        if (rGrid[x][y] == ( (trajectory[x][y] & B001) ? H : L ) && gGrid[x][y] == ( (trajectory[x][y] & B010) ? H : L ) && bGrid[x][y] == ( (trajectory[x][y] & B100) ? H : L ) ) {
          trajectory[x][y] = random(1,8);
        }
      }
    }
  }
}
```