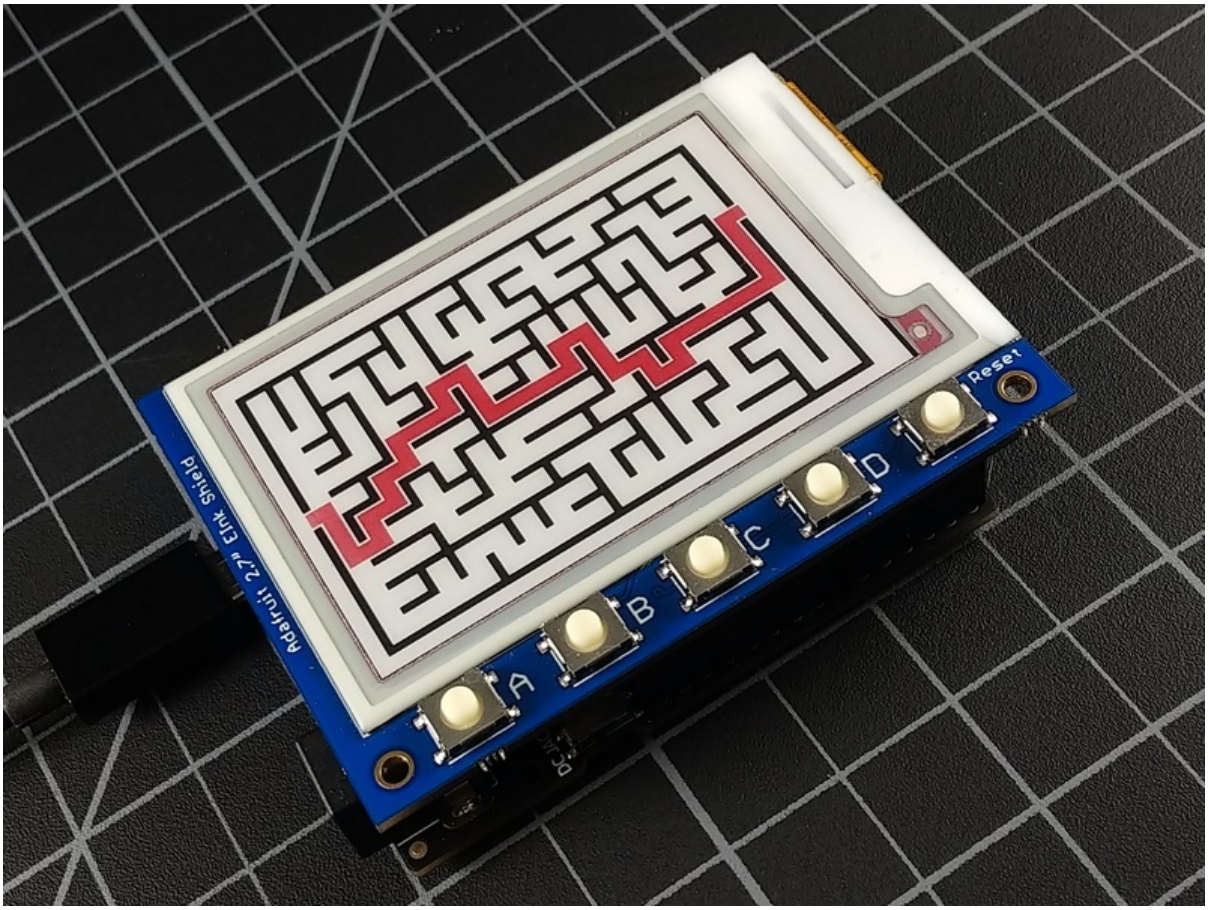
ePaper Maze Maker

Created by Dan Cogliano



https://learn.adafruit.com/epaper-maze-maker

Last updated on 2024-06-03 02:49:26 PM EDT

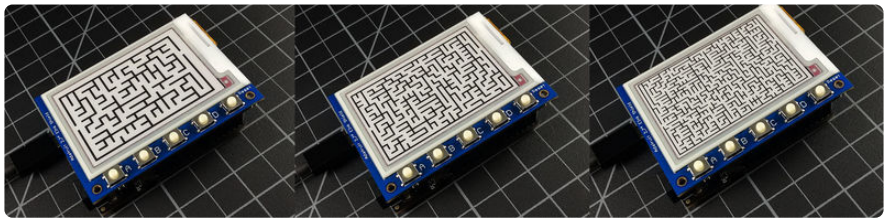
Table of Contents

Overview

This project generates random mazes and their solution using an Adafruit Metro M4 Express and an Adafruit ePaper shield. This is a fun, no-solder project suitable for all ages with three difficulty levels to choose from. You will have some a-**maze**-ing fun with it!

This project does not use WiFi, so it will work with either the Metro M4 Express or the Metro M4 Express Airlift. This project can be made portable by adding a battery, making it a great activity to take on a long car ride to minimize hearing "Are we there yet?" from the back of the car.

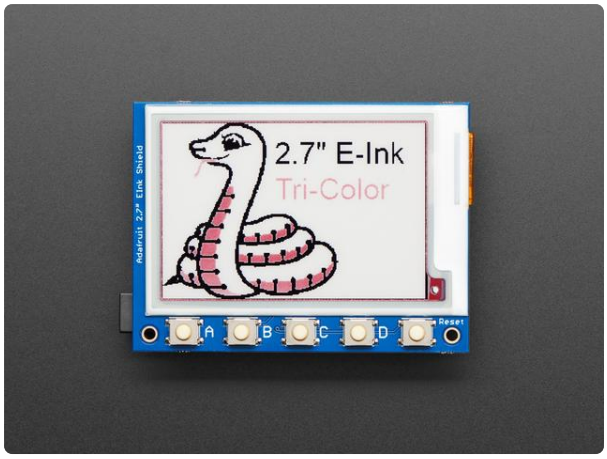
The original maze generator code this project is based on was written over 20 years ago in C for printing random mazes on paper. The use of the ePaper shield in this project brings a modern touch to this classic code.



Parts

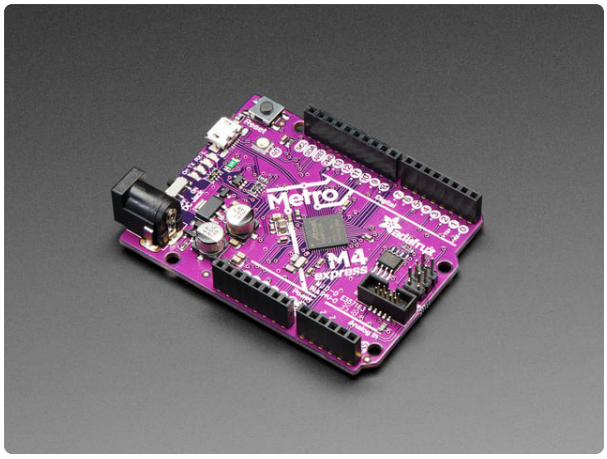
Building this project requires no soldering and uses just two parts: the Adafruit Metro M4 Express and the Adafruit 2.7" Tri-Color eInk / ePaper Shield with SRAM. Either the Metro M4 or Metro M4 Airlift Express can be used. To make this project portable, you could use a 9V battery (http://adafru.it/1321) and adapter (http://adafru.it/67) (shortest battery life) or add a USB battery pack (http://adafru.it/1959) and adapter cable (http://adafru.it/2697) (longer life and rechargeable).

If you are interested in ePaper displays for other projects, check out the entire line of Adafruit's ePaper displays (https://adafru.it/ExU).



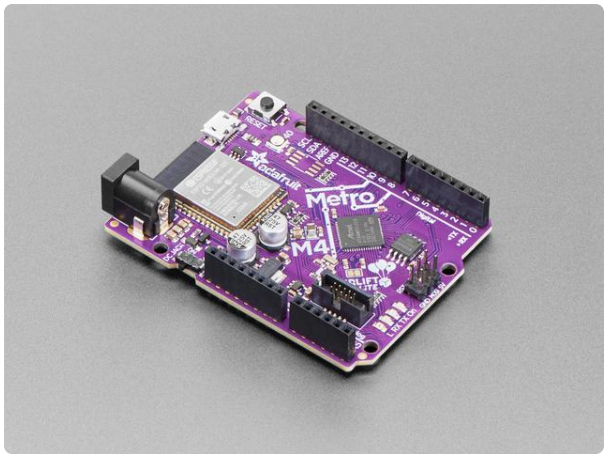
Adafruit 2.7" Tri-Color eInk / ePaper Shield with SRAM

Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those...
https://www.adafruit.com/product/4229



Adafruit Metro M4 feat. Microchip ATSAMD51

Are you ready? Really ready? Cause here comes the fastest, most powerful Metro ever. The Adafruit Metro M4 featuring the Microchip ATSAMD51. This...
https://www.adafruit.com/product/3382



Adafruit Metro M4 Express AirLift (WiFi) - Lite

Give your next project a lift with AirLift - our witty name for the ESP32 co-processor that graces this Metro M4. You already know about the Adafruit Metro...
https://www.adafruit.com/product/4000

USB cable - USB A to Micro-B

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

9V battery holder with switch & 5.5mm/2.1mm plug

This is a 9V battery pack with on/off switch and a pre-attached 5.5mm/2.1mm center-positive barrel plug. Use this to battery-power your Arduino (or other electronic projects) -...
https://www.adafruit.com/product/67

Alkaline 9V Battery

Battery power for your portable project! These batteries are high quality at a good price and work fantastic with any of the kits or projects in the shop that use 9V.These...
https://www.adafruit.com/product/1321

The Maze Algorithm

For those interested in how the program creates a random maze, this section is for you!

There are two algorithms used here, a maze generator and a maze solver. The maze consists of a grid of boxes. Each box is numbered sequentially and initially contains 4 walls surrounding each grid. Using a random number generator, a box is selected from the grid and one of the 4 walls is removed making an opening with an adjacent box. The adjacent box receives the same sequence number as the selected box as well as all the other boxes sharing the same sequence number. This ensures that a maze loop is not created, which will cause a problem with the maze solver algorithm. Also, walls are not removed if the adjacent box has the same sequence number or is on the border. Breaking walls on random boxes continues until all boxes have the same sequence number. This ensures all boxes are connected to each other.

Since the boxes are all connected, any location along the perimeter can act as a start and end location. The algorithm puts the start and end location on opposite sides, but it does not matter which end you start from.

For the maze solver, it traverses through the maze from the start of the maze, making right turns at every intersection, until it reaches the end. This may cause some backtracking when a dead end is encountered, but it eventually finds the end of the maze. After removing the backtracking moves, the solution path is then overlaid on top of the maze.

Arduino Setup

f you don't have it already, you will need the Arduino IDE installed on your computer to upload this sketch to the Metro M4 . You will find information on installing Arduino in this learning guide (https://adafru.it/CfF). You will also need to configure the Metro M4 to work with the Arduino IDE. There are articles for setting up the Metro M4 Express (https://adafru.it/Fkt) and M4 Express Airlift (https://adafru.it/EZh) with Arduino and other environments in the Adafruit Learning Guides site (https://adafru.it/dlu).

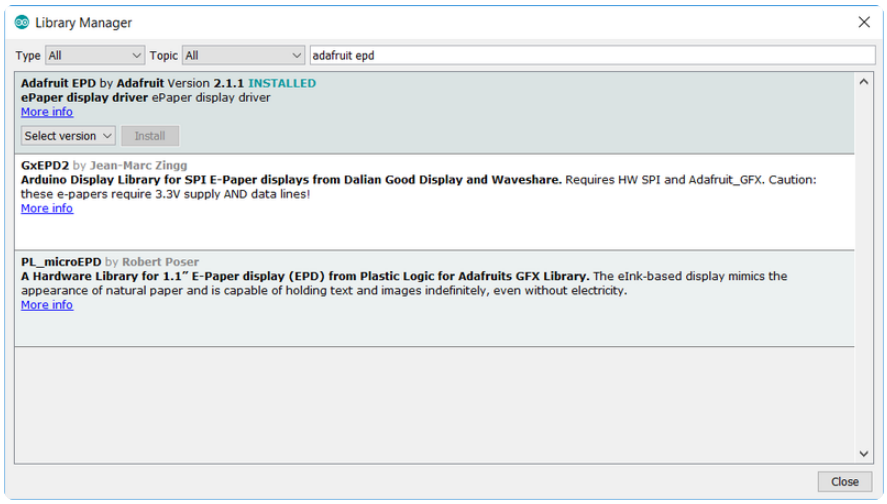
Installing The Libraries

For this sketch, you will need to install these libraries:

- Adafruit EPD (ePaper display) library
- Adafruit GFX library
- Adafruit BusIO library
- Adafruit NeoPixel library

All of these libraries can be installed directly from the IDE. Select the menu item, **Sketch** -> **Include Library** -> **Manage Libraries**. From the Library Manager Window, enter the words **adafruit epd** in the search box and you should see the Adafruit EPD library appear. Click the box where the library appears and then click the "Install" button to install the library. Be sure to install the latest version of the library.

In the same manner, install the **Adafruit GFX** and **NeoPixel** libraries by searching the words **adafruit gfx** and **neopixel,** respectively, to install the latest versions of these libraries.



Once you have the Arduino IDE setup, you are ready to install the ePaper Maze Maker. It is available from GitHub by clicking the links at the top of the code below.

```
// SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/* Program to automatically generate maze puzzles
 *
 * For use with Adafruit Metro M4 Express Airlift or Metro M4 and tricolor e-Paper
Display Shield
 *
 * Adafruit invests time and resources providing this open source code.
 * Please support Adafruit and open source hardware by purchasing
 * products from Adafruit.com!
 *
 * Written by Dan Cogliano for Adafruit Industries
 * Copyright (c) 2019 Adafruit Industries MIT License, all text must be preserved
 */

#include <stdlib.h>
#include <stdio.h>

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_EPD.h>
#include <Adafruit_NeoPixel.h>
#define SRAM_CS     8
#define EPD_CS      10
#define EPD_DC      9
#define EPD_RESET -1
#define EPD_BUSY -1

#define NEOPIXELPIN   40

/* This is for the 2.7" tricolor EPD */
Adafruit_IL91874 gfx(264, 176 ,EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);

Adafruit_NeoPixel neopixel = Adafruit_NeoPixel(1, NEOPIXELPIN, NEO_GRB +
NEO_KHZ800);

#define MEM_MAX    20000

// print character to use when printing to terminal
#define BLOCK_CHAR 'X'

#define BOTTOM  0x01
#define RIGHT 0x02
#define TRUE  1
```

```
#define FALSE 0

// convert block # to x, y coordinate
#define GETX(item,width) (item % width)
#define GETY(item,width) (item / width)
#define COORD(item,width)(String("[") + String(GETX(item,width)) + "," +
String(GETY(item,width)) + String("]"))

/* global variables defined here */
char *maze=NULL;              // pointer to maze wall data
uint16_t *mazepath=NULL;      // pointer to maze path data
uint16_t *mazesolution=NULL;  // pointer to maze solution
uint16_t solutioncount = 0;

int sizex=0, sizey=0;     /* maze size */
int cellsize = 10; // larger the cellsize, easier the puzzle, in pixels
int lwidth = 3; // maze wall width, in pixels

/*
   init_maze(cs, lw) initializes maze memory.
   cs - cell size in pixels (smaller cell = harder maze)
   lw - line width in pixels
   example: init_maze(7,2) will use 7 pixel boxes with 2 pixel maze wall width
     and 3 pixel solution line width (cs - lw - 2)
   Maze size is based on cs value and ePaper screen size
*/
void init_maze(int cs, int lw)
{
  // use max size allowable
  cellsize = cs;
  lwidth = lw;
  sizex = gfx.width()/cellsize;
  sizey = gfx.height()/cellsize;
  Serial.println("maze size: " + String(sizex) + " x " + String(sizey));
  if(sizex < 1 || sizey < 1 || (long)sizex*sizey > MEM_MAX)
  {
    Serial.println("Invalid values entered for maze size\n");
    Serial.println("Maze must be at least 4 x 4\n");
    exit(EXIT_FAILURE);
  }
  long incr;
  for(incr=0; incr < (gfx.width()/5) *  (gfx.height()/5); incr++)
  {
    if(incr==0)
      maze[incr]=0;
    else if(incr < sizex)
      maze[incr]=BOTTOM;
    else if((incr%sizex)==0)
      maze[incr]=RIGHT;
    else
      maze[incr]=BOTTOM|RIGHT;
    mazepath[incr]=incr;
  }
}

/*
 * getDirection(pos1, pos2) get the direction between two points
 * return 1: x direction
 *        2: y direction
 */
int getDirection(uint16_t pos1, uint16_t pos2)
{
  int x1, y1, x2, y2, diffx, diffy;
  x1 = GETX(pos1,sizex);
  y1 = GETY(pos1,sizex);
  x2 = GETX(pos2,sizex);
  y2 = GETY(pos2,sizex);

  diffx = x2 - x1;
```

```
  diffy = y2 - y1;

  //Serial.println( "getDirection: " + COORD(pos1,sizex) + "," + COORD(pos2,sizex)
+ "," + String(diffx) + "," + String(diffy));
  if(diffy == 0)
    return 1;
  if(diffx == 0)
    return 2;
  return 0; //diagonal?
}

/*
 * solve_r() - recursive solve routine, called by solve()
 */
bool solve_r(uint16_t finish, uint16_t pos, uint16_t prevpos, uint8_t dir)
{
  if(pos == prevpos)
  {
    Serial.println("same square, backing up");
    solutioncount--;
    // same square, need to back up from here
    return false;
  }
  mazesolution[solutioncount++] = pos;
  if(pos == finish)
  {
    Serial.println("Solved in " + String(solutioncount) + " moves");
    return true;
  }
  uint16_t posx, posy, newx, newy, newpos;
  /*
   * directions:
   * 0: north
   * 1: west
   * 2: south
   * 3: east
   */
   posy = GETY(pos, sizex);
   posx = GETX(pos, sizex);
   //Serial.println("trying square " + COORD(pos, sizex) + " at direction " + dir);
   switch(dir)
   {
   case 0: // north
      newx = posx;
      newy = posy - 1;
      newpos = newy * sizex + newx;
      if((maze[newpos] & BOTTOM) == 0)
      {
        return solve_r(finish, newpos, pos, 3);
      }
      break;
   case 1: // west
      newx = posx - 1;
      newy = posy;
      newpos = newy * sizex + newx;
      if((maze[newpos] & RIGHT) == 0)
      {
        return solve_r(finish, newpos, pos, 0);
      }
      break;
   case 2: // south
      newx = posx;
      newy = posy + 1;
      newpos = newy * sizex + newx;
      if((maze[pos] & BOTTOM) == 0)
      {
        return solve_r(finish, newpos, pos, 1);
      }
      break;
```

```
   case 3: // east
      newx = posx + 1;
      newy = posy;
      newpos = newy * sizex + newx;
      if((maze[pos] & RIGHT) == 0)
      {
        return solve_r(finish, newpos, pos, 2);
      }
      break;
   }
   dir = (dir + 1) %4;
   //Serial.println("next direction " + String(dir));
   solutioncount--;
   return solve_r(finish, pos, prevpos, dir);
}

/*
 * solve() - solve the maze
 */
void solve()
{
  uint16_t start = 0;
  uint16_t finish = 0;

  for(int i = 1; i < sizex; i++)
  {
    if((maze[i] & BOTTOM) == 0)
    {
      start = i + sizex;// start at row below
      break;
    }
  }
  for(int i = (sizey-1)*sizex + 1; i < (sizey*sizex); i++)
  {
    if((maze[i] & BOTTOM) == 0)
    {
      finish = i;
      break;
    }
  }
  solutioncount = 0;
  //Serial.println("maze start: " + COORD(start, sizex) + ", finish: " +
COORD(finish, sizex));
  mazesolution[solutioncount++] = start;
  solve_r(finish, start, start - gfx.width(), 1);

  // remove dead end moves
  int solutionpos = 0;
  while(solutionpos < solutioncount)
  {
    //Serial.println("solution pos: " + String(solutionpos) +
COORD(mazesolution[solutionpos], sizex)
    //  + ", solution count: " + String(solutioncount));
    for(int i = solutioncount - 1; i > solutionpos; i--)
    {
      if(mazesolution[solutionpos] == mazesolution[i])
      {
        // remove dead end paths
        //Serial.println("removing " + String(i - solutionpos) + " duplicate path
items");
        for(int j = 0; j < (solutioncount - solutionpos); j++)
        {
          mazesolution[solutionpos + j] = mazesolution[i + j];
        }
        solutioncount -= i - solutionpos;
      }
    }
    solutionpos++;
  }
```

```
  Serial.println("Solution reduced to " + String(solutioncount) + " moves");

  /*
  // print out solution
  for(int i = 0; i < solutioncount; i++)
  {
    Serial.println(String(i) + ": " + COORD(mazesolution[i], sizex));
  }
  */
}
/*
 * print_epaper_maze() prints the maze on the ePaper device, optionally showing
solution
 */
void print_epaper_maze(bool showsolution = false)
{
  int xcenter = 0 - cellsize/2;// + (gfx.width() - sizex*cellsize)/2;
  int ycenter = 0 - cellsize/2;// + (gfx.height() - sizey*cellsize) / 2;
  Serial.println("maze centering adjustment: " + String(xcenter)+ ", " +
String(ycenter));
  gfx.powerUp();
  gfx.clearBuffer();
  neopixel.setPixelColor(0, neopixel.Color(0, 255, 0));
  neopixel.show();

  // draw horizontal lines
  for(int incry = 0; incry < sizey; incry++)
  {
    int xstart = -1;
    int xend = -1;
    for(int incrx = 0; incrx < sizex; incrx++)
    {
      if((*(maze+incry*sizex+incrx)&BOTTOM)!=0 && xstart == -1)
      {
        xstart = incrx;
      }
      else if((*(maze+incry*sizex+incrx)&BOTTOM)==0 && xstart != -1)
      {
        xend = incrx;
        gfx.fillRect(xcenter + xstart*cellsize,ycenter + (incry+1)*cellsize,(xend -
xstart)*cellsize+lwidth,lwidth,EPD_BLACK);
        xstart = -1;
        xend = -1;
      }
    }
    if(xstart != -1)
    {
      // finish line
      xend = sizex;
      gfx.fillRect(xcenter + xstart*cellsize,ycenter + (incry+1)*cellsize,(xend -
xstart)*cellsize+lwidth,lwidth,EPD_BLACK);
      xstart = -1;
      xend = -1;
    }
  }

  // draw vertical lines
  for(int incrx = 0; incrx < sizex; incrx++)
  {
    int ystart = -1;
    int yend = -1;
    for(int incry = 0; incry < sizey; incry++)
    {
      if((*(maze+incry*sizex+incrx)&RIGHT)!=0 && incry > 0 && ystart == -1)
      {
        ystart = incry;
      }
      else if((*(maze+incry*sizex+incrx)&RIGHT)==0 && incry > 0 && ystart != -1)
      {
```

```
        yend = incry;
        gfx.fillRect(xcenter + (incrx+1)*cellsize,ycenter + ystart*cellsize,lwidth,
(yend - ystart)*cellsize+lwidth,EPD_BLACK);
        ystart = -1;
        yend = -1;
      }
    }
    if(ystart != -1)
    {
      // finish line
      yend = sizey;
      gfx.fillRect(xcenter + (incrx+1)*cellsize,ycenter + ystart*cellsize,lwidth,
(yend - ystart)*cellsize+lwidth,EPD_BLACK);
    }
  }

  if(showsolution)
  {
    /*
    // mouse droppings version
    for(int i = 0; i < solutioncount; i++)
    {
      gfx.fillRect(xcenter + GETX(mazesolution[i],sizex) * cellsize + lwidth,
ycenter + GETY(mazesolution[i],sizex) * cellsize + lwidth, cellsize - lwidth,
cellsize - lwidth, EPD_RED);
    }
    */

    // line drawn version
    int rectx, recty, rectwidth, rectheight, rectstart, rectend;
    int dir;
    int linestart = mazesolution[0];
    int lineend = mazesolution[1];
    int lastdir = getDirection(linestart,lineend);
    int i = 1;

    // draw startline
    rectx = xcenter + GETX(mazesolution[0],sizex)*cellsize + lwidth + 1;
    recty = 0 - ycenter;
    rectwidth = cellsize - lwidth - 2;
    rectheight = ycenter + cellsize + lwidth + 1;
    gfx.fillRect(rectx, recty, rectwidth, rectheight, EPD_RED);
    //Serial.println("starting rectangle @ " + String(rectx) + "," + String(recty)
+ ": " + String(rectwidth) + " by " + String(rectheight));

    while(i < solutioncount)
    {
      lineend = mazesolution[i];
      if ((dir = getDirection(linestart,mazesolution[i+1])) != lastdir)
      {
        rectstart = linestart;
        rectend = lineend;
        if(((GETX(lineend,sizex) - GETX(linestart,sizex)) < 0) ||
((GETY(lineend,sizex) - GETY(linestart,sizex)) > 0))
        {
          rectstart = lineend;
          rectend = linestart;
        }
        switch(lastdir)
        {
          default:
            rectwidth = rectheight = cellsize - lwidth - 2;
            break;
          case 1: // x direction
            rectwidth = (abs(GETX(lineend,sizex) - GETX(linestart,sizex)) +
1)*cellsize - lwidth - 2;
            rectheight = cellsize - lwidth - 2;
            break;
          case 2: // y direction
```

```
            rectwidth = cellsize - lwidth - 2;
            rectheight = (abs(GETY(lineend,sizex) - GETY(linestart,sizex)) +
1)*cellsize - lwidth - 2;
            break;
        }
        rectx = xcenter + GETX(rectstart,sizex)*cellsize + lwidth + 1;
        recty = ycenter + GETY(rectend,sizex)*cellsize + lwidth + 1;
        gfx.fillRect(rectx, recty, rectwidth, rectheight, EPD_RED);
        //Serial.println("draw line from " + COORD(linestart,sizex) + " to " +
COORD(lineend,sizex));
        //Serial.println("rectangle @ " + COORD(rectstart,sizex) + ": " +
String(rectwidth) + " by " + String(rectheight));
        linestart = lineend;
        //lastdir = dir;
        lastdir = getDirection(linestart,mazesolution[i+1]);
      }
      else
        i++;
    }
    // draw endline
    rectx = xcenter + GETX(mazesolution[solutioncount-1],sizex)*cellsize + lwidth +
1;
    recty = ycenter + GETY(mazesolution[solutioncount-1],sizex)*cellsize + lwidth +
1;
    rectwidth = cellsize - lwidth - 2;
    rectheight = cellsize ;
    gfx.fillRect(rectx, recty, rectwidth, rectheight, EPD_RED);
    //Serial.println("ending rectangle @ " + String(rectx) + "," + String(recty) +
": " + String(rectwidth) + " by " + String(rectheight));
  }
  gfx.display();
  Serial.println("display update completed");
  gfx.powerDown();
  neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
  neopixel.show();
}

/*
   print_block_maze() prints out the maze using the specified block
   character. This was the print routine for the original maze program.
   You can use it to print mazes to the terminal.
*/
void print_block_maze()
{
  int incrx, incry, incr2;
  char buff[5];
  for(incry=0; incry < sizey; incry++)
  {
    for(incr2=0; incr2 < 2; incr2++)
    {
      for(incrx=0; incrx < sizex; incrx++)
      {
        switch(incr2)
        {
        case 0:
          strcpy(buff,"  ");
          if(incry > 0)
          {
          if((*(maze+incry*sizex+incrx)&RIGHT)!=0)
            buff[1]=BLOCK_CHAR;
          }
          Serial.print(buff);
          break;
        case 1:
          strcpy(buff,"  ");
          if((*(maze+incry*sizex+incrx)&BOTTOM)!=0 && incrx > 0)
            buff[0]=BLOCK_CHAR;
          if(((incry < (sizey-1)) &&
            ((*(maze+(incry+1)*sizex+incrx)&RIGHT)!=0))||
```

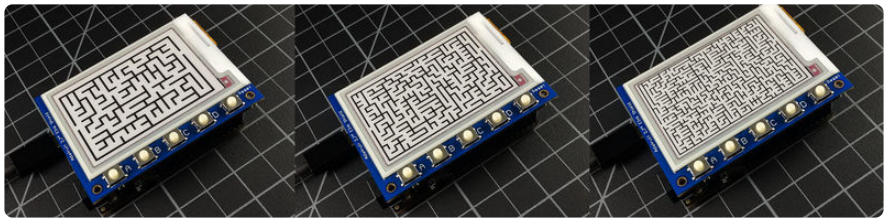
```
            (buff[0]==BLOCK_CHAR)|| (incrx==0) ||
            ((*(maze+incry*sizex+incrx)&RIGHT)!=0) ||
            ((incrx < (sizex-1)) &&
            ((*(maze+incry*sizex+incrx+1)&&BOTTOM)!=0)))
            buff[1]=BLOCK_CHAR;
          Serial.print(buff);
          break;
        }
      }
      Serial.println();
    }
  }
}

/*
   cell_join() joins two cells together, effectively breaking down a wall within
   the maze.
*/
void cell_join(int cell1, int cell2)
{
  int incr,val;

  val=*(mazepath+cell2);
  /* set mazepath value */
  //for(incr=0; incr < sizex*sizey; incr++)
  for(incr = sizex*sizey-1; incr >= 0; incr--)
    if(*(mazepath+incr)==val)
      *(mazepath+incr)=*(mazepath+cell1);
  /* set graphics */
  if(cell1+1 == cell2) /* open right wall */
    *(maze+cell1)=*(maze+cell1)&~RIGHT;
  else /* open bottom wall */
    *(maze+cell1)=*(maze+cell1)&~BOTTOM;
}

/*
   connect() attempts to connect two squares together, returning
   FALSE if the attempt failed
*/
int connect(int cell)
{
  int incr;
  int cellcheck[2]; /* adjacent cell attempts */
  /* check if cell is a border, if so, return false */
  if((cell < sizex) /* top line */
  ||((cell%sizex)==0)) /* left line */
    return(FALSE);
  /* determine order of cell attempts */
  cellcheck[0]=random(2);
  cellcheck[1]=(cellcheck[0]+1)%2;
  /* check cells to see if can be connected */
  for(incr=0; incr < 2; incr++)
  {
    if((GETX(cell,sizex)==(sizex-1))&&(cellcheck[incr]==0))
      continue; // do not attempt to open right edge of maze
    //if((cell > (sizex*(sizey-1)))&&(cellcheck[incr]==1))
    if((GETY(cell,sizex)==(sizey-1))&&(cellcheck[incr]==1))
      continue; // do not attempt to open bottom edge of maze
    if(*(mazepath+cell)!=*(mazepath+cell+1+cellcheck[incr]*(sizex-1)))
    {
      cell_join(cell,cell+1+cellcheck[incr]*(sizex-1));
      return(TRUE);
    }
  }
  return(FALSE);
}

/*
   generate() is the function that generates a random maze.  It calls connect()
```

```
   (which, in turn, calls cell_join()) to generate the maze.
*/
void generate()
{
  int cell,checkcell,incr,complete;
  do
  {
    complete=TRUE;
    /* pick a random cell */
    cell=sizex + random(sizex*(sizey-1));
    /* find the next cell that can be connected */
    for(incr=0; incr < sizex*sizey; incr++)
    {
      checkcell=(incr+cell)%(sizex*sizey);
      if((checkcell < sizex)||((checkcell%sizex)==0))
        continue;
      if(connect(checkcell))
      {
        complete=FALSE;
        break;
      }
    }
  }
  while(!complete);
  /* break walls for start and end of maze, near center */
  cell=sizex/4+(long)random(sizex/2);
  *(maze+cell)=*(maze+cell)&~BOTTOM;
  cell=sizex/4+(long)random(sizex/2)+sizex*(sizey-1);
  *(maze+cell)=*(maze+cell)&~BOTTOM;
}

/*
 * error() display error message and blink red neopixel
 */
void error(const char *err)
{
  Serial.println(err);
  while(1)
  {
    neopixel.setPixelColor(0, neopixel.Color(255, 0, 0));
    neopixel.show();
    delay(400);
    neopixel.setPixelColor(0, neopixel.Color(0, 0, 0));
    neopixel.show();
    delay(100);
  }
}

void setup() {
  Serial.begin(115200);
  //while(!Serial);
  delay(1000);
  Serial.println(
    "XXXXXXX XXXXXXXXXXXXXXXXXXXX\n"
    "X  Maze Generation Program  X\n"
    "X  Written by Dan Cogliano  X\n"
    "X  For Adafruit Industries  X\n"
    "XXXXXXXXXXXXXXXXXXXX XXXXXXX\n\n");

  randomSeed(analogRead(0));

  neopixel.begin();
  neopixel.show();
  gfx.begin();
  Serial.println("ePaper display initialized");
  gfx.clearBuffer();
  gfx.setRotation(1);
  int w = gfx.width();
  int h = gfx.height();
```

```
  // assuming cell size of 5 pixels is the smallest supported maze
  if((maze=(char *)malloc((w/5) *  (h/5) * sizeof(char)))==NULL)
  {
    error("Not enough memory available for maze\n");
  }
  if((mazepath=(uint16_t *)malloc((w/5) *  (h)/5 * sizeof(uint16_t)))==NULL)
  {
    error("Not enough memory available for maze\n");
  }
  if((mazesolution=(uint16_t *)malloc((w/5) *  (h)/5 * sizeof(uint16_t)))==NULL)
  {
    error("Not enough memory available for maze\n");
  }

  init_maze(14,4);
  generate();
  //print_block_maze();
  print_epaper_maze();
}

/*
 * readButtons() to check if a button has been pressed
 */
int8_t readButtons(void) {
  uint16_t reading = analogRead(A3);
  //Serial.println(reading);

  if (reading > 600) {
    return 0; // no buttons pressed
  }
  if (reading > 400) {
    return 4; // button D pressed
  }
  if (reading > 250) {
    return 3; // button C pressed
  }
  if (reading > 125) {
    return 2; // button B pressed
  }
  return 1; // Button A pressed
}

void loop() {
  static bool showsolution = true;
  int button = readButtons();
  if (button == 0) {
    return;
  }
  Serial.print("Button "); Serial.print(button); Serial.println(" pressed");
  if (button == 1) {
    Serial.println("easy maze");
    init_maze(14,4);
    generate();
    //print_block_maze();
    print_epaper_maze();
    showsolution = true;
  }

  if (button == 2) {
    Serial.println("medium maze");
    init_maze(10,3);
    generate();
    //print_block_maze();
    print_epaper_maze();
    showsolution = true;
  }

  if (button == 3) {
    Serial.println("hard maze");
```

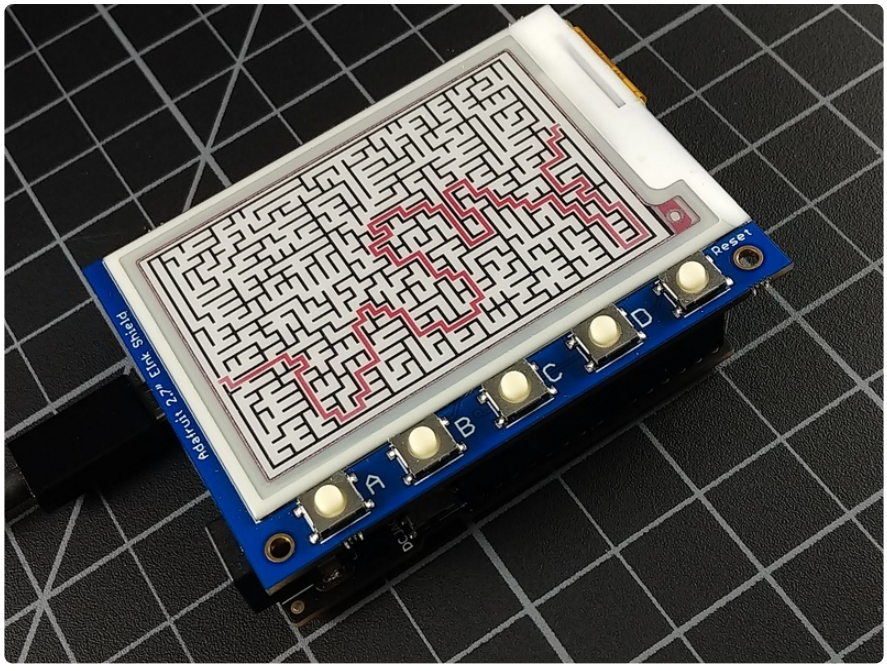
```
    init_maze(7,2);
    //init_maze(5,1); // smallest/hardest maze supported
    generate();
    //print_block_maze();
    print_epaper_maze();
    showsolution = true;
  }

  if (button == 4) {
    Serial.println("solve maze");
    solve();
    print_epaper_maze(showsolution);
    showsolution = !showsolution;
  }
}
```

Use



The ePaper shield contains 4 programmable buttons and a reset button at the bottom of the display. For this sketch, buttons **A**, **B** and **C** select the difficulty level, with **A** being the easiest level, **B** the medium level and **C** the difficult level.

Button **D** toggles the solution on and off in case you get stuck solving it. The solution is overlaid on top of the maze in red ink, thanks to the tri-color feature of the ePaper shield.



The built-in color LED NeoPixel on the Metro M4 Express displays green when the display is being updated for either the maze or the solution. Note that the buttons are disabled while the display is being updated, which could take a few seconds as is true for all ePaper displays.

The maze difficulty is based on the number of squares in the grid. When more squares are used in the grid, the maze becomes more complex and harder to solve. More squares are added to the maze by making them smaller in order to fit them on the fixed size of the ePaper display. Here is the breakdown of the 3 maze sizes:

- Easy Maze: 12 x 18 (14 pixels per box)
- Medium Maze: 17 x 26 (10 pixels per box)
- Hard Maze: 25 x 37 (7 pixels per box)

For example, at 14 pixels per box, this gives enough room on the ePaper shield for a maze size of 12 x 18, allowing for 4 pixels for the maze wall width.

Have fun solving mazes!