```
#include "pitches.h"
float tmp;
int mini(int x, int y) {
 int res = x;
  if (x > y) res = y;
  return res;
}


int maxi(int x, int y) {
  int res = x;
  if (x < y )res = y;
   return res;
}

void limit_angle (int &angle) { // limits angle bw 0 and 360
  // 0-360 conv
  if (angle < 0) angle += 360 * (abs(angle) / 360 + 1);
  if (angle > 360) angle %= 360;
}
int num_points = 0, points[255];
class shapes {
  public:
    int chars[10], i;
    int curr_y, x0, x1;
    int temp;
    // int x[256]; // curr_y intercepts
    shapes() {
      // constructor add stuff
    }
    ~shapes() {
      // desctructor ad stuff if feeling lucky : )
    }

    void cpy2array16 (int a[], int b[], int n) {
      for (i = 0; i < n; i++) a[i] = b[i];
    }

    int sgn(int x) {
      return (x > 0) - (x < 0);
    }


    void circle() {
      // add fill
      // 1- > centx 2-> centy 3-> radius , 4-> start angle , 5_> end angle  .. fill ? : 1/0
      //Serial.println(curr_y);
      tmp = chars[3]chars[3] - (curr_y - chars[2])(curr_y-chars[2]);
      if (tmp >= 0 ) { // lets stay real!
//       tmp = (tmp+temp/tmp)/2;
//       tmp = (tmp+temp/tmp)/2;
//       tmp = (tmp+temp/tmp)/2;
//       tmp = (tmp+temp/tmp)/2;
//       tmp = (tmp+temp/tmp)/2;
        x0 = chars[1] - sqrt(tmp);
        x1 = chars[1] + sqrt(tmp)+1; // calc two intercepts
        //int angle1 = 180.0 / PI * atan((curr_y * 1.0 - chars[2]) / (x0 - chars[1])) ? x0 - chars[1] != 0 : 90 * sgn(curr_y-
chars[2]); // calc angle + catch exploding denominators!!
        //int angle2 = 180.0 / PI * atan((curr_y * 1.0 - chars[2]) / (x1 - chars[1])) ? x1 - chars[1] != 0 : 90 * sgn(curr_y-
chars[2]);

        if (x0 >= 0 && x0 <= 255)
          points[num_points++] =  x0;
        if (x1 >= 0 && x1 <= 255 )
          points[num_points++] =  x1;
```

```
    }
  }

  void linepal() { // line in start-point angle length form
    // * deal with 0 / 90 angle cases **
    // x1 y1 angle(degrees) length

    if (curr_y <= maxi(chars[2] + chars[4]*sin(180.0 * chars[3] / PI), chars[2]) && curr_y >= mini(chars[2] +
chars[4]*sin(180.0 * chars[3] / PI), chars[2]) ) { //current y between two extreme y values
      // check if the current y coordinate falls inside the drawing range                                    // for
drawing purposes and this is it !!
      if (chars[3] != 90 || chars[3] != 0) { // reg non 90 and 0 angle cases
        x0 = chars[1] + (chars[4]) * cos((180.0 * chars[3]) / PI);
        if (x0 <= 255 && x0 >= 0)
          points[num_points++] = x0;
      } // check bounds }

      else if (chars[3] == 90 && mini(chars[2] + chars[4], chars[4]) <= curr_y && curr_y <= maxi(chars[2] + chars[4],
chars[4]) ) { // check if the x falls in the screen y = chars[1] type line
        if (chars[1] <= 255 && chars[1] >= 0)
          points[num_points++] = chars[1];

      }
      else if (chars[3] == 0 && chars[2] == curr_y) // x = chars [2] type line
      {
        i = mini(chars[1], chars[1] + chars[4]);
        i = i ? i >= 0 : 0; // constrain to 0 +
        while (i <= maxi(chars[1], chars[1] + chars[4]) && i <= 255 ) // horizontal line
          points[num_points++] = i;
      }

    }

  }
  void linep2p() { // line in point 2 point form
    // x1 y1 x2 y2
    // 1  2  3  4
    // 30,30,30,10
    // assuming singular intercepts (large angle and dims ) .. ( look  for solns for small angles, quantizations!!!!)

    //Serial.print(chars[2]);
    //Serial.println(chars[4]);

    if (chars[1] == chars[3]){
      // Serial.println("v");// for vertical
      if (mini(chars[2], chars[4]) <= curr_y && curr_y <= maxi(chars[2], chars[4])) // if current y within drawing range
        if (chars[1] <= 255 && chars[1] >= 0) // if x is constrained
          points[num_points++] = chars[1]; // put a point at the specified x
    }
    else if (chars[2] == chars[4] && chars[2] == curr_y){ // horz .. will be triggered at the app instance
        i = mini(chars[1], chars[3]);
        if (i<0)i=0;
        while (i <= maxi(chars[1], chars[3]) && i <= 255 ) // horizontal line
          {points[num_points++] = i;
            i+=2;
          }
    }
    else{
      // y= mx + c         form .. single point non infinity non zero slope
        x0 = chars[1] + (chars[1] * 1.0 - chars[3]) * (curr_y - chars[2]) / (chars[2] - chars[4]); // simple 2 point line form
        if (x0 <= 255 && x0 >= 0 && mini(chars[2], chars[4]) <= curr_y && curr_y <= maxi(chars[2], chars[4]))
          points[num_points++] = x0;
          //Serial.println("n");
      }
```

```
  }

  void read_primitives (int attributes[10], int y)  { // reads the shape characteristics
    // start translation...
    // primitives
    curr_y = y;
    cpy2array16 (chars, attributes, 10);
    if (chars[0] == 0) circle ();
    else if (chars[0] == 1) linepal();
    else if (chars[0] == 2) linep2p();

  }

  int bottom_y(int att[10]) // returns the lower y coordinate of the asked
  {
    int retvar ; // dummy : )
    switch (att[0])
    {
      case 0:
        retvar = att[2] - att[3]; break;
      case 1:
        retvar = mini(att[2], att[4] * sin(180.0 * chars[3] / PI)); break;// if I decide to use signed vars...break;
      case 2:
        retvar = mini(att[2], att[4]); break;
    }
    return retvar;
  }

  int top_y(int att[10]) // returns the lower y coordinate {
  {
    int retvar ; // dummy : )
    switch (att[0])
    {
      case 0:
        retvar = att[2] + att[3]; break;

      case 1:
        retvar = maxi(att[2], att[4] * sin(180.0 * chars[3] / PI)); break;

      case 2:
        retvar = maxi(att[2], att[4]); break;// signed value implementation

    }
    // add for right x and left x
    return retvar;
  }



} s;

 bool req = 0;
 void routine(){
 req = true; }

void setup() {
  // put your setup code here, to run once:
  //SPI.begin();
 // Serial.begin(115200);
  //Serial.print(mini(20,20));

pinMode(PA0,OUTPUT);pinMode(PA1,INPUT);pinMode(PA2,INPUT);pinMode(PA3,OUTPUT);pinMode(PA4,OUTPUT);pin
Mode(PA5,OUTPUT);pinMode(PA6,OUTPUT);pinMode(PA7,OUTPUT);
```

```
pinMode(PB0,OUTPUT);pinMode(PB1,OUTPUT);pinMode(PB2,OUTPUT);pinMode(PB3,OUTPUT);pinMode(PB4,OUTPUT);pinMode(PB5,OUTPUT);pinMode(PB6,OUTPUT);pinMode(PB7,OUTPUT);

pinMode(PA8,OUTPUT);pinMode(PA9,OUTPUT);pinMode(PA10,OUTPUT);pinMode(PA11,OUTPUT);pinMode(PA12,OUTPUT);pinMode(PA13,OUTPUT);pinMode(PA14,OUTPUT);pinMode(PA15,OUTPUT);

pinMode(PB8,OUTPUT);pinMode(PB10,OUTPUT);pinMode(PB12,OUTPUT);pinMode(PB13,OUTPUT);pinMode(PB14,OUTPUT);pinMode(PB15,OUTPUT);
 attachInterrupt(PA2,routine,FALLING);
}


int x, num_objects = 0, frame_objects[50][10] = {0,150,150,50}; // implement double ended queue on frame_objects
int by, ty, i = 0;
int starting = 0, ending, y;
int k = 0;


void sort_objects (int objects[][10], int n) { // ascending
//  int i, j, ex[10], k; // delete objects whose top y < 0 (completely out of drawable frame)
//  for (i = 0; i < num_objects-2; i++) // dumb dumb logic
//    for (j = 0; j < num_objects - 1 - i; i++) {
//      if (s.bottom_y(objects[j]) > s.bottom_y(objects[j + 1])) { // unacceptable !  we shall exchange u for a new one : - )
//        for (k = 0; k < 5; k++) ex[k] = objects[j][k];
//        for (k = 0; k < 5; k++) objects[j][k] = objects[j + 1][k];
//        for (k = 0; k < 5; k++) objects[j + 1][k] = ex[k];
//      }
//    }
 int i,j,mini,ex[5];
    for(i=0;i<n-1;i++){
        mini = i;
        for(j = i+1;j<n;j++)
            if(s.bottom_y(objects[mini]) > s.bottom_y(objects[j]))
                mini = j;

            if(mini!=i){
                for (k = 0; k < 5; k++) ex[k] = objects[i][k];
                for (k = 0; k < 5; k++) objects[i][k] = objects[mini][k];
                for (k = 0; k < 5; k++) objects[mini][k] = ex[k];
             }

    }




}

void sort (int point[], int n, bool dir = 0) { //  by def asc set dir = 1 for desc
  /*int i, j;
  for (i = 0; i < num-2; i++)
    for (j = 0; j < num - i - 1; j++)
      if ((1 - 2 * dir)*point[j] > point[j + 1] * (1 - 2 * dir))
      {
        point[j] += point[j + 1];
        point[j + 1] = point[j] - point[j + 1];
        point[j] = point[j] - point[j + 1];
      }*/
 int i,j,mini,ex;
    for(i=0;i<n-1;i++){
        mini = i;
        for(j = i+1;j<n;j++)
            if(point[j]<point[mini])
                mini = j;
```

```
          if(mini!=i){
              ex = point[mini];
              point[mini] = point[i];
              point[i] = ex;

           }

    }




}

int p = 0; // object counter
bool out = 0,dir =0;
void draw_osc(){
  sort_objects(frame_objects, num_objects); // use s.bottom_y(frame_objects[i])

  starting = 0; // reset object counter
  // drawing part
  delayMicroseconds(100);
  for (y = 0; y <= 255; y++) { // scan vertically .. bottom to top

    num_points = 0; //reset the number of points
    p = starting; // reset ..
    while (p < num_objects) { //scan shapes to fill the current row with their intercepts
        //Serial.println(s.bottom_y(frame_objects[0]));
        //Serial.println(s.top_y(frame_objects[0]));

      if (s.top_y(frame_objects[p]) < y) { // for drawing .. bottom <= y <= top  ... inverse -> bottom > y || top < y
       p++;
       continue;
       // starting++; // the object cannot be read
          // Serial.println("b");
      }
      else if (y < s.bottom_y(frame_objects[p])) { // bottom of the next object above the current y; // pixels for current row
are allocated
         //t
         //Serial.print("..");
       // break;// out = 1; // objects are arranged in ascending order of bottom y's ... hence the next object's bottom y will
obv. be above the current object's bottom y
      }
      // increments only if the next object is supposed to be read

      s.read_primitives(frame_objects[p], y); // generate line_pixels array for current y
      // reverse sorting order after each iteration to effectively half the scan line retractions
      p++; // skip to the next object having higher bottom y coordinate

    }
    sort(points, num_objects,y%2); // sort the generated points array to prevent jitter and improve pointing times
    if (num_points){// skip empty lines
      GPIOA->ODR = (GPIOA->IDR & 0x6FFF) | ((y&0x01)<<12)|((y&0x02)<<14); // automatically picks up the binary eq in
translation// stm32 specific
     GPIOB->ODR = (GPIOB->IDR & 0xFE07) | ((y&0xFC)<<1); // set y // port manipulation
    }
    delayMicroseconds(180);
    for (k = 0; k < num_points; k++) {// sweep x{
     GPIOA->ODR = (GPIOA->IDR & 0xF0FF) | ((points[k]&0xF0)<<4); // automatically picks up the binary eq in
translation// stm32 specific
     GPIOB->ODR = (GPIOB->IDR & 0x0FFF) | ((points[k]&0x0F)<<12); // set x // port manipulation
      delayMicroseconds(180);// wait for capture
    }

  }
    GPIOA->ODR = (GPIOA->IDR & 0x6FFF) ; // reset y
```

```
    GPIOB->ODR = (GPIOB->IDR & 0xFE07) ;
}

void ad(int arr[10], int shapes[][10], int index,int &num){
   int i,j=0;
   for(i=index;i<num;i++)
      for(j=0;j<10;j++)
         shapes[i+1][j] = shapes[i][j];
   for(j=0;j<10;j++)
      shapes[index][j] = arr[j];
   num++;
}
void dl(int shapes[][10], int index, int &num){
   int i,j=0;
   for(i=index;i<num;i++)
      for(j=0;j<5;j++)
         shapes[i][j] = shapes[i+1][j];
  num--;
}
void load(int a[10]){ // loads objects into display array
 for(i=0;i<5;i++)
   frame_objects[num_objects][i] = a[i];
 num_objects++;
}
/*int x, num_objects = 2, frame_objects[][10] = {{2,0,80,100,0},{0,100,100,100}}; // implement double ended queue on
frame_objects
int by, ty, i = 0;
int starting = 0, ending, y;
int k = 0;
 */


 int rad ;
 int num_bulls =0,num_targs = 0,c;
int xlast=0,x1,lasty=0,rmax=0;
int mobjs[50][10] = {{2,0,0,0,0}}; // movable objects
int f,j,bullets[50][10];
int last = 0,last1 = 0, r,xstart = 0,joyx =0,xpos=125;

int turret[][10] = {{2,125,0,135,0},{2,0,225,225,0}};

void loop() {
 // sort _ frame objects and pop those entries which are out of bounds .. use quicksort
// by default 10 bit adc values : 0 - 1023 , 512
 // sort _ frame objects and pop those entries which are out of bounds .. use quicksort
 xpos = constrain(xpos+map(analogRead(PA0)+25,0,1023,-4,4),20,235); // middle of turret
 draw_osc();
 num_objects = 0;
 xlast = 0;
 for(j=0;j<num_targs;j++){ // loads for dispay objects
   load(mobjs[j]);
   mobjs[j][2] -= 1; // update pos for next iteration
   }
  for(j=0;j<num_bulls;j++){ // loads for dispay bullets
   load(bullets[j]);
   bullets[j][2] += 2;// update pos for next iteration
 }


 frame_objects[num_objects][0] = 2;
 frame_objects[num_objects][1] = xpos;
 frame_objects[num_objects][2] = 25;
 frame_objects[num_objects][3] = xpos+20;
 frame_objects[num_objects++][4] = 1;

 frame_objects[num_objects][0] = 2;
```

```
  frame_objects[num_objects][1] = xpos-20; // update horz turret position based on current x cent value
  frame_objects[num_objects][2] = 0;
  frame_objects[num_objects][3] = xpos+20;
  frame_objects[num_objects++][4] = 0;


  frame_objects[num_objects][0] = 2;
  frame_objects[num_objects][1] = xpos-20;
  frame_objects[num_objects][2] = 2;
  frame_objects[num_objects][3] = xpos;
  frame_objects[num_objects++][4] = 25;

// for (j=0;j<2;j++){// load turret based on current controller position
// // load(turret[j]);
//
// }




  if(millis()-last > 15000) //  target spawn/de spawn block
  {
    xstart = 0;
    last = millis();
    r = 15+random(10);
    xstart = 35 + random(150);
    while (xstart + r <= 255){
    mobjs[num_targs][0] = 0;
    mobjs[num_targs][1] = xstart + r;
    mobjs[num_targs][2] = 270; // start out of drawing range for a smooth transition
    mobjs[num_targs++][3] = r;

    xstart += 35 + r + random(150); // calculate the starting pos taking into account the radius of the last circle and the
last xtart
    r = 15+random(10);
    }

  }


for(j=0;j<num_objects;j++){ // cleaning routine
   if ( mobjs[j][2]+mobjs[j][3]<0 && mobjs[j][0] == 0) // clean up circles
       dl(mobjs,j,num_targs);
   for(c=num_bulls-1;c>=0;c--) // target- bullet collision detection
       if (mobjs[j][2]+mobjs[j][3] > bullets[c][2]-bullets[c][3] && mobjs[j][2]-mobjs[j][3] < bullets[c][2]+bullets[c][3]) // iff
           if(mobjs[j][1]+mobjs[j][3] > bullets[c][1]-bullets[c][3] && mobjs[j][1]-mobjs[j][3] < bullets[c][1]+bullets[c][3]){
             dl(mobjs,j,num_targs);
             dl(bullets,c,num_bulls);
              tone(PB9, 70, 100);
                delay(100);
                noTone(PB9);
          }
      if (mobjs[j][2]-mobjs[j][3]<20 && mobjs[j][1]) // collision detection with turret .. check bottom most point
         if(mobjs[j][1]+mobjs[j][3] > xpos && mobjs[j][1]-mobjs[j][3] < xpos){ // turret collision detected
      int melody[] = {
         NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
      };


        int noteDurations[] = {
      4, 8, 8, 4, 4, 4, 4, 4
      };



       for (int thisNote = 0; thisNote < 8; thisNote++) {
```

```
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(PB9, melody[thisNote], noteDuration);


    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);

    noTone(PB9);
  }


          while(1); // stall
          }// call EOG func ... END OF GAME .. // life --

 }

 for (j =0;j<num_bulls;j++)
 if ( bullets[j][2]+mobjs[j][3]>255) // clean up circles
       dl(bullets,j,num_bulls);

  if(millis()-last1 > 2500 && req){ //  spawn bullets if requested if the cool down time has passed
    last1 = millis();
     tone(PB9, 1000, 60);
    bullets[num_bulls][0] = 0;
    bullets[num_bulls][1] = xpos;
    bullets[num_bulls][2] = 28;
    bullets[num_bulls++][3] = 6;
  }
    req =false; // reset req

}
```