```
//MONDRISONIC: Processing Code
//All audio was rendered from Supercollider and imported as audio files
//The data from the Insight EEG headset was through simulated keystrokes
//provided by the proprietary software EmotivXavierEmoKey and the EEG training
//was undertaken using the EmotivXavierControlPanel

//SETUP AND OVERALL SYSTEM CONTROL
import ddf.minim.*; //<>//
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
import ddf.minim.signals.*;
import ddf.minim.spi.*;
import ddf.minim.ugens.*;

//main globals
int boxes = 1;
int maxBoxes = 40;
int colours = 5;
int border = 20;
int min = 40;
float fadeVal=0;
float strokeWeight=10;
float fadeSpeed = 2;
int[][] score;

//globals for Mover function
int whichSide=int(random(4)); //initialised direction
int whichDir = (int(random(2))==1) ? 1 : -1;
int checkAlso = whichSide;
float speed=2; //sets speed of animation
int moveProb=5;
int x = border;
int y = border;
int gap = 1;
int x1 =0;
int y1 =0;
int area = (width*(border*2))+(height*(border*2));
int[] ratio = {50,50};
//int[] colRatio = {10/colours,10/colours,10/colours,200/colours,270/colours};
int[] randomNumbers = {2,4,7,2,1,5,6,9};

//globals for handeling colour
int[] colRatio = {100/colours,100/colours,100/colours,100/colours,100/colours};
color [][] col = {{#354489,#E23B02,#E0CF0F,#3C4343,#E7E7E7},
                  {#A22A01,#621A01,#E23B02,#230900,#C83402},
                  {#004A7F,#4CB3FF,#0093FF,#47687F,#0076CC},
                  {#9FA000,#606000,#DEE000,#212100,#FDFF00},
                  {#8F99AB,#9FA8BD,#AEBBD1,#8A94A6,#C2CFE5}};
int[] colWeight = {0,0,0,0,0};
int pallet = 0;
int palletNum = 5;
int activeBox = int(random(boxes));
```

```
//Audio globals
Minim minim;
AudioPlayer blankAP;
AudioPlayer track0, track1, track2, track3, track4, dummyTrack;
String[][] sounds = {{"p1-1.wav","p1-2.wav","p1-3.wav","p1-4.wav","p1-5.wav"},
                     {"p2-1.wav","p2-2.wav","p2-3.wav","p2-4.wav","p2-5.wav"},
                     {"p3-1.wav","p3-2.wav","p3-3.wav","p3-4.wav","p3-5.wav"},
                     {"p4-1.wav","p4-2.wav","p4-3.wav","p4-4.wav","p4-5.wav"},
                     {"p5-1.wav","p5-2.wav","p5-3.wav","p5-4.wav","p5-5.wav"}};

float [] vol = {0.0,0.0,0.0,0.0,0.0};
int soundsPerBank=5;
int soundBankNum=5;
int soundBank = 0;
int minVol = 40;
boolean turnSoundOn = true;
PImage title;

//globals for Timer function
//top row is startime and second row is duration in seconds
// 1-newsection, 2-changecolour 3-changeAciveBox 4- stop
int numOfSections=5;
float minPerSec = 2;
float [][] times = {{0,0,0,0},
                    {minPerSec*60,30,3,minPerSec*60*numOfSections+60}};
//global start stop on keystroke 3
boolean GO = false;
int startTime = 0;
int section = 0;
int dbLevel = 5;
boolean brain = false;

void setup() {
  db("setup",1);
  minim = new Minim(this);
  //loadAudio();
  //size(600,600);
  //size(1920,3600);
  //size(480,900);
  fullScreen(2);
  title = loadImage("title.png");
  noCursor();
  //setup variables
  boxes = 2;
  activeBox = int(random(boxes));
  pallet = 0;
  soundBank = 4;
 // loadAudio();
  speed = 2;
  moveProb = 5;
  fadeVal=255;
  fadeSpeed = 4;
  createScore();
```

```
}
void draw(){
    db("draw",1);
    background(155);
    drawScore();
 if(GO){
    if(timer(1)){//move to new section (pallet, soundBank and score)
      section++;
      switch(section){

      case 1: //Section 2 Red Pallet
        fadeVal=0;
        fadeSpeed = 1;
        brain=true;
        boxes = 12;
        activeBox = int(random(boxes));
        gap=1;
        pallet = 1;
        soundBank = 1;
        loadAudio();
        speed = 2;
        moveProb = 2;
        createScore();
        break;
      case 2: //Section 3 Blue Pallet
         fadeVal=0;
         fadeSpeed = 1;
         boxes = 50;
         activeBox = int(random(boxes));
         gap=1;
         pallet = 2;
         soundBank = 2;
         loadAudio();
         speed = 12;
         moveProb = 1;
         createScore();
         break;
      case 3: //Section 4 Yellow Pallet
         fadeVal=0;
         fadeSpeed = 4;
         boxes = 5;
         activeBox = int(random(boxes));
         gap=10;
         pallet = 3;
         soundBank = 3;
         loadAudio();
         speed = 2;
         moveProb = 2;
         createScore();
         break;
      case 4://Section 5 Complete Pallet
         fadeVal=0;
         fadeSpeed = 4;
         boxes = 15;
```

```
        activeBox = int(random(boxes));
        gap=1;
        pallet = 0;
        soundBank = 4;
        loadAudio();
        speed = 10;
        moveProb = 5;
        createScore();
        break;
      case 5://End
       fadeAudio();
       speed = 0;
       break;
      }
      db("Section: "+section,5);
      db("GO: "+GO+" activeBox = "+activeBox+" speed = "+speed+" moveProb = "+moveProb+" whichSide =
"+whichSide+" boxes = "+boxes+" pallet no = "+pallet+" soundBank = "+soundBank+" turnSoundOn
="+turnSoundOn,5);
    }

    if(timer(2)){//turn on brain responsiveness
      brain=true;
    }

    if(timer(3)){ //change active box
      activeBox = int(random(boxes));
    }

    if(timer(4)){//start and stop
      GO = (GO) ? false : true;
      db("GO: "+GO+" turnSoundOn: "+turnSoundOn,5);

    }
    if(GO){
      mover();
      if (section != 5) mixAudio();
      calcColourArea();
      for(int i=0; i<soundsPerBank;i++){
        db(lookUpColour(0,i)+" : "+colWeight[i]/1000+" vol"+i+" : "+vol[i],2);
      }
    }
}else{
    //imageMode(CENTER);
    //image(title, width/2, height/3,title.width/4,title.height/4);
    //image(title, width/2, height/2,width-(border*20),height-(border*40));
}
}
```

```
//GRAPHIC SCORE CREATION AND IMPLEMENTATION

void createScore(){
  db("createScore - In",1);
  if(boxes<1) boxes=1;
  if(boxes>=maxBoxes) boxes=maxBoxes;
  int[][] score1 = new int[boxes][];
  score = score1;
  colorMode(HSB);
  int colourNum = 5;
  background(150);
  //fadeVal=0;
  x1 = width-border;
  y1 = height-border;
  area = x1 * y1;
  strokeWeight(10);
  score[0] = new int[] {x,y,x1,y1,col[pallet][randColour()],x1*y1,int(random(strokeWeight))};
  for(int i=0;i<boxes-1;i++){
    db("createScore - In",1);
    // ramdonly choose a box from number of boxes generated and copy it to a new entry on the list
      boolean flag = true;
      int index = 0;
      int chooseBox = 0;
      int newX = 0, newY = 0;
      while (flag) {
        db("createScore - While1",1);
        index = boxPicker(i);
        //index = int(random(i));
        score[i+1] = new int[] {score[index][0],score[index][1],score[index][2],score[index][3],col[pallet]
[randColour()],score[index][5],int(random(strokeWeight))};

      if (((score[index][2] - score[index][0])>min) && ((score[index][3] - score[index][1])>min)) {
        //if (((score[index][2] - score[index][0])>min) || ((score[index][3] - score[index][1])>min)) {
          chooseBox =  (score[i][2]-score[i][0] > score[i][3]-score[i][1]) ? 0:1;
          switch (chooseBox){
            case 0: //x axis
                // choose a new X boundary from the choosen box, makeing sure it's greater than the minimum size
                newX = int(random(score[index][0] + 2*min, score[index][2]-score[index][0])) - min;
                score[index][2] = newX;
                score[i+1][0] = newX;
                ratio[0] -= 10;
                ratio[1] += 10;
                break;
            case 1: //y axis
                // choose a new Y boundary from the choosen box, makeing sure it's greater than the minimum size
                newY = int(random(score[index][1] + 2*min,score[index][3]-score[index][1])) - min;
                score[index][3] = newY;
                score[i+1][1] = newY;
                ratio[1] -= 10;
                ratio[0] += 10;
                break;
            }
            flag = false;
```

```
        }
        score[i+1][5]=(score[i+1][2]-score[i+1][0])*(score[i+1][3]-score[i+1][1]);
        score[index][5]=(score[index][2]-score[index][0])*(score[index][3]-score[index][1]);
     }
   }
      for(int i=0;i<boxes;i++){
        db("box"+i+" "+score[i][0]+" "+score[i][1]+" "+score[i][2]+" "+score[i][3]+" ",3);
   }
   db("createScore - Out",1);
}

void drawScore(){
   db("drawScore - In",1);
   //int[] tempColWeight = {0,0,0,0,0};
   fill(155);
   rect(border,border,width-(border*2),height-(border*2));
   fadeVal =(fadeVal<=255) ? fadeVal+=fadeSpeed : 255;
   for(int i=0;i<boxes;i++){
      fill(score[i][4],fadeVal);
      strokeWeight(strokeWeight);
      // random borders strokeWeight(score[i][6]);
      stroke(0,fadeVal);
//      stroke(0,(fadeVal<=255) ? fadeVal+=fadeSpeed : 255);
      // added a 'r' reducer value to make the squares smaller to debug but it looks quite nice
      //rect(score[i][0],score[i][1],score[i][2]-score[i][0],score[i][3]-score[i][1]);
      rect(score[i][0]+gap,score[i][1]+gap,score[i][2]-score[i][0]-(gap*2),score[i][3]-score[i][1]-(gap*2));
   } //<>//
 db("drawScore - Out",1);
 }
```

```
//TIMING AND SCHEDULING OF EVENTS

 boolean timer(int whichTimer){
    if((millis()-startTime) - times[0][whichTimer-1] >times[1][whichTimer-1]*1000){
      times[0][whichTimer-1] = millis()-startTime;
      return true;
    }else{
      return false;
  }
}

void mover(){
 db("mover-in",1);
// if(int(random(100))<moveProb) whichDir = (int(random(2))==1) ? 1 : -1;
 if(int(random(100))<moveProb) whichDir *= -1 ;
 while(!checkSide(activeBox)){
    whichDir = (int(random(2))==1) ? 1 : -1;
    whichSide = int(random(4));
    activeBox = int(random(boxes));
 }

 //move adjecent box
 boolean moveBox = false;
 for (int i=0; i<boxes; i++) {
   if(i!=activeBox){
     //check same side
     if (score[i][whichSide] == score[activeBox][whichSide]){
       score[i][whichSide]+=whichDir*speed;
       moveBox=true;
       db("HIT box same side: "+i+" new value: "+score[i][whichSide],3);
     }
     //check opposite side
     if (score[i][checkAlso] == score[activeBox][whichSide]){
       score[i][checkAlso]+=whichDir*speed;
       moveBox=true;
       db("HIT box op side: "+i+" new value: "+score[i][checkAlso],3);
     }
     score[i][5]=(score[i][2]-score[i][0])*(score[i][3]-score[i][1]);
   }
   db("i: "+i+" whichside: "+whichSide+" whichDir: "+whichDir+" activeBox: "+activeBox+" checkAlso:
"+checkAlso+" moveBox: "+moveBox+" moveProb: "+moveProb,4);
 }
 if(moveBox) score[activeBox][whichSide]+=whichDir*speed;
 calcColourArea();
 db("mover-out",1);
}

boolean checkSide(int whichBox){
 db("checkSide-In",1);
 boolean noMatch=true;
 switch(whichSide){
  case 0: //x left
  if(score[whichBox][0]<=border) noMatch= false;
```

```
 if((score[whichBox][2]-score[whichBox][0])<=min && whichDir==1)noMatch= false;
 if(score[whichBox][0]>=width-border && whichDir==1) noMatch= false;
 if(score[whichBox][0]<border) score[whichBox][0]=border;
 if(score[whichBox][0]>width-border) score[whichBox][0]=width-border;
 break;
 case 1: //y top
 if(score[whichBox][1]<=border) noMatch= false;
 if((score[whichBox][3]-score[whichBox][1])<=min && whichDir==1)noMatch= false;
 if(score[whichBox][1]>=height-border && whichDir==1) noMatch= false;
 if(score[whichBox][1]<border) score[whichBox][1]=border;
 if(score[whichBox][1]>height-border) score[whichBox][1]=height-border;
 break;
 case 2: //x1 right
 if(score[whichBox][2]>=width-border) noMatch= false;
 if((score[whichBox][2]-score[whichBox][0])<=min && whichDir==-1)noMatch= false;
 if(score[whichBox][2]<=border && whichDir==-1) noMatch= false;
 if(score[whichBox][2]<border) score[whichBox][2]=border;
 if(score[whichBox][2]>width-border) score[whichBox][2]=width-border;
 break;
 case 3: //y2 bottom
 if(score[whichBox][3]>=height-border) noMatch= false;
 if((score[whichBox][3]-score[whichBox][1])<=min && whichDir==-1)noMatch= false;
 if(score[whichBox][3]<=border && whichDir==-1) noMatch= false;
 if(score[whichBox][3]<border) score[whichBox][3]=border;
 if(score[whichBox][3]>height-border) score[whichBox][3]=height-border;
 break;
 }

 //int checkAlso=whichSide;
 switch (whichSide){
 // x left
 case 0: checkAlso=2;
        break;
 // y top
 case 1: checkAlso=3;
        break;
 // x right
 case 2: checkAlso=0;
        break;
 // y bottom
 case 3: checkAlso=1;
        break;
}
 //check adjecent boxs
 for (int i=0; i<boxes; i++) {
   if(i!=activeBox){
     //check same side
     if ((score[i][whichSide] == score[activeBox][whichSide])||(score[i][checkAlso] == score[activeBox][whichSide])){
       if(abs(score[i][whichSide]-score[i][checkAlso])<=min) noMatch= false;
     }
   }
 }
 db("checkSide: whichBox : "+whichBox+" whichSide: "+whichSide+" whichDir: "+whichDir+" moveProb:
"+moveProb+" noMatch: "+noMatch,4);
```

```
 db("checkSide-Out",1);
 return noMatch;
}

void calcColourArea(){
  //add box area to colour area
 int areaTotal = 0;
 int[] tempColWeight = {0,0,0,0,0};
 for(int i=0;i<boxes;i++){
  score[i][5]=(score[i][2]-score[i][0])*(score[i][3]-score[i][1]);
  score[i][5] = (score[i][5]<0) ? -score[i][5] : score[i][5];
  areaTotal +=score[i][5];
  }
 for(int j=0;j<colWeight.length; j++){
  for(int i=0;i<boxes;i++){
    if(score[i][4]==col[pallet][j]){
      tempColWeight[j]+=score[i][5];
      //println("J:"+j+" "+lookUpColour(score[i][4],0)+" "+tempColWeight[j]);
    }

 }
  vol[j]=(tempColWeight[j]/7000)-minVol;
 }
 colWeight=tempColWeight;
 db("areaTotal: "+areaTotal,1);

}
```

```
//GENERAL PROCESSING OF KEYSTROKE EVENTS AND COLOUR HANDLING

public int randColour() {
  int chooseCol = int(random(100));
  int count = 0;
  boolean colFound = false;
  int colourValue = 0;
  for (int i = 0; i < colours; i++) {
    count += colRatio[i];
    if (!colFound && (chooseCol < count)) {
      colourValue = i;
      colRatio[i] -= 4;
      colFound = true;
      if (colRatio[i]<0)
        colRatio[i] = 0;
    }
    else {
      colRatio[i]++;
    }
  }
  return colourValue;
}

public String lookUpColour( color s, int n ) {
 if(s!=0){
 switch ( s ) {
    case #354489: return "Blue";
    case #E23B02: return "Red";
    case #E0CF0F: return "Yellow";
    case #3C4343: return "Black";
    case #E7E7E7: return "White";
    case #006516: return "Green";
    case #014552: return "Dark Blue";
    case #850D00: return "Dark Red";
    //case col[1][3]: return "Black";
    //case col[1][4]: return "White";
    default: return "didn't match";
    }
 }else{
   if(pallet == 0){
     switch ( n ) {
        case 0: return "Blue";
        case 1: return "Red";
        case 2: return "Yellow";
        case 3: return "Black";
        case 4: return "White";
        default: return "didn't match";
     }
   }else{
     switch ( n ) {
        case 0: return "Green";
        case 1: return "Dark Blue";
        case 2: return "Dark Red";
```

```
        case 3: return "Black";
        case 4: return "White";
        default: return "didn't match";
     }
    }
  }
}

public int lookupColourNum( color s ) {
   //color [] col = {#354489,#E23B02,#E0CF0F,#020202,#E7E7E7};
 switch ( s ) {
    case #354489: return 0;
    case #E23B02: return 1;
    case #E0CF0F: return 2;
    case #3C4343: return 3;
    case #E7E7E7: return 4;
    case #006516: return 0;
    case #014552: return 1;
    case #850D00: return 2;
    //case col[1][3]: return "Black";
    //case col[1][4]: return "White";

    default: return 99;
    }

}

public int boxPicker(int i) {
  boolean boxFound = false;
  int count = 0;
  int j = 0;
  int rand = int(random(area));
  while (!boxFound && j<=i) {
   count += score[j][5];
   if (rand <= count)
     boxFound = true;
   j++;
  }
  return j-1;
}

void db(String message, int level){
  if(level==dbLevel) println(message);
}

void keyPressed() {
  if(!GO && !brain) rect(width-border,height-border, border/4,border/4); //createScore();


  switch(key){
    case 'g': //start and stop
      if(GO){
        GO=false;
        stopAudio();
```

```
    brain=false;
   }else{
    GO= true;
    loadAudio();
    //brain=true;
    noCursor();
   }
   startTime = millis();
   db("GO: "+GO+" section: "+section+" startTime: "+startTime,5);

   break;
 case 'b': //brain trigger
   if (brain==true){
    switch(section){
      case (0):  //full pallet
                boxes++;
                fadeVal=255;
                createScore();
                drawScore();
                break;
      case (1):  //Red
                score[int(random(boxes))][4] = col[pallet][int(random(5))];
                speed = (speed>=20) ? 20 : speed+1;
                fadeVal=255;
                if(speed%5 == 0) createScore();
                drawScore();
                break;
      case (2):  //Blue
                score[int(random(boxes))][4] = col[pallet][int(random(5))];
                speed = (speed<=1) ? 1 : speed-1;
                fadeVal=255;
                createScore();
                gap = (gap>=15) ? 15 : gap+1;
                drawScore();
                break;
      case (3):  //Yellow
                score[int(random(boxes))][4] = col[pallet][int(random(5))];
                gap = (gap<=0) ? 0 : gap-1;
                boxes++;
                fadeVal=255;
                createScore();
                drawScore();
                break;
      case (4):  //Full Score
                score[int(random(boxes))][4] = col[pallet][int(random(5))];
                gap = (gap<=0) ? 0 : gap-1;
                speed = (speed<=1) ? 1 : speed-1;
                fadeVal=255;
                moveProb = (moveProb<=0) ? 0 : moveProb-1;
                //gap+=10;
                if (boxes>2){
                  boxes--;
                  activeBox = int(random(boxes));
                  createScore();
```

```
                drawScore();
               }
               break;
     }
    }
    db("BRAIN: "+brain,5);
    break;
  case 'q': //moveProb increment
    moveProb = (moveProb>=100) ? 100 : moveProb+1;
    db("moveProb = "+moveProb,5);
    break;
  case 'w': //moveProb decrement
    moveProb = (moveProb<=0) ? 0 : moveProb-1;
    db("moveProb = "+moveProb,5);
    break;
  case 'e': //side change
    whichSide = int(random(4));
    db("whichSide = "+whichSide,5);
    break;
  case 'r': //direction change
    whichDir = (whichDir==-1) ? 1 : -1;
    db("whichDir = "+whichDir,5);
    break;
  case 't': //gap increment
    gap++;
    //createScore();
    db("gap = "+gap,5);
    break;
 case 'y': //gap decrement
    gap = (gap==0) ? 0 : gap-1;
    //createScore();
    db("gap = "+gap,5);
    break;
  case 'a': //new random box
    activeBox = int(random(boxes));
     db("activeBox = "+activeBox,5);
    break;
  case 's': //speed increase
    speed++;
    db("speed = "+speed,5);
    break;
  case 'd': //speed decrease
    speed = (speed==0) ? 0 : speed-1;
    //speed--;
    db("speed = "+speed,5);
    break;
  case 'f': //new score
    createScore();
    db("New score created",5);
    break;
 case 'z': //increase boxes
    boxes++;
    createScore();
    db("boxes = "+boxes,5);
```

```
  break;
 case 'x': //decrease boxes
  boxes = (boxes<=2) ? 2 : boxes-1;
  createScore();
  db("boxes = "+boxes,5);
  break;
 case 'c': //colour change
  score[int(random(boxes))][4] = col[pallet][int(random(5))];
  db("random colour change",5);
  break;
 case 'v': //change pallet
  pallet = (pallet==palletNum-1)? 0 : pallet+1;
  createScore();
  db("pallet no = "+pallet,5);
  break;
 case 'n' : //cursor off
  noCursor();
  db("Cursor Off ",5);
  break;
  case 'm' : //cursor on
  cursor(ARROW);
  db("Cursor On ",5);
  break;
 case '1': //change sound bank
  stopAudio();
  soundBank= 0;
  loadAudio();
  db("soundBank = "+soundBank,5);
  break;
 case '2':
  stopAudio();
  soundBank= 1;
  loadAudio();
  db("soundBank = "+soundBank,5);
  break;
 case '3':
  stopAudio();
  soundBank= 2;
  loadAudio();
  db("soundBank = "+soundBank,5);
  break;
 case '4':
  stopAudio();
  soundBank= 3;
  loadAudio();
  db("soundBank = "+soundBank,5);
  break;
 case '5':
  stopAudio(); //<>//
  soundBank= 4;
  loadAudio();
  db("soundBank = "+soundBank,5);
  break;
 case '0': //new score
```

```
    stopAudio();
    db("turnSoundOn ="+turnSoundOn,5);
    break;
  }
  db("section: "+section+" activeBox: "+activeBox+" speed: "+speed+" moveProb: "+moveProb+" whichSide:
"+whichSide+" boxes: "+boxes+" pallet no: "+pallet+" soundBank: "+soundBank+" turnSoundOn: "+turnSoundOn+"
gap: "+gap,5);
}
```

//MondriSonic â€“ KEYSTROKES FOR TESTING

// q : increase moveProb
// w : decrease moveProb
// e : change side movement
// r : change direction
// t : increase gap
// y : decrease gap
// a : new random box
// s : increase speed
// d : decrease speed
// f : new score
// g : start and stop
// z : increase number of boxes
// x : dencrease number of boxes
// c : colour change
// v : change pallet
// b : brain signal
// n : cursor on
// m : cursor off
// 1 : soundbank 1
// 2 : soundbank 2
// 3 : soundbank 3
// 4 : soundbank 4
// 5 : soundbank 5
// 0 : kill audio

//screen 1920 wide x 3600
//epoc version 2.0.0.21