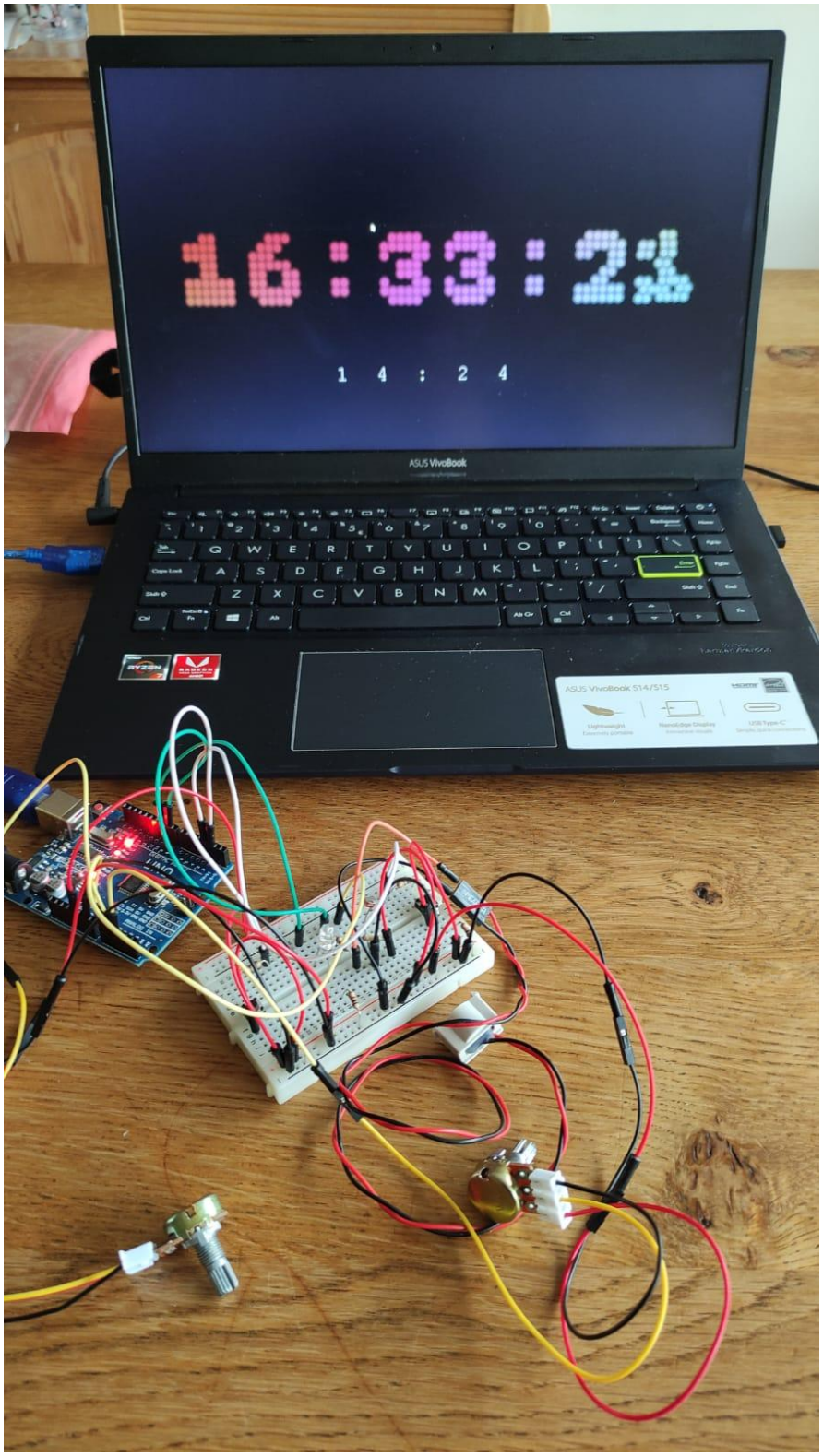
Marinus Bos, s2827603
Denzel Hagen, s1978497
Code credits:
Ball text by Marinus and Nina
3D text by Denzel and Jort

Description

We have made a smart alarm clock. This clock will lower its screen brightness when it is in a dark environment. It also has two knobs with which you can set the hour and minute of when the alarm should go off, and two buttons. One button enables and disables whether the alarm will ring, and the other button will turn off the ringing when the alarm is going off.
For output, there is a buzzer and a light, which will fluctuate in brightness and tone to make sure the alarm is noticed.
The screen is drawn by processing. It contains the clock and the alarm, and when the alarm goes off, the brightness is forced to its maximum and the numbers go wild.
The screen could theoretically be put on a raspberry pi to make it its own device.

Picture



Code

Arduino

```
/*
 * Interface of an alarm clock, requires the accompanying Processing sketch
 * Program made by Marinus Bos and Denzel Hagen
 * Made as final assignment for the course Programming and Physical
Computing at the University of Twente
 * made in module 2 of 2021/2022
 */

// constants
static const int bdrate = 19200;          // baudrate for the Serial Monitor
static const int potThreshold = 5;        // threshold before a change in
the potentiometers is detected.
static const int loopWait = 16;
// pins
static const int lightSensorPin = A0;   // pin for the voltage divider that
senses the light pin
static const int hourPotPin = A1;         // pin for the potentiometer that
controls the hours
static const int minPotPin = A2;          // pin for the potentiometer that
controls the minutes
static const int setButtonPin = 2;  // pin for the button that enables or
disables the alarm
static const int ringingOffPin = 3; // pin for the button that turns off
the alarm when it rings
static const int buzzerPin = 9;           // pin for the buzzer
static const int lightPin = 10;           // pin for the light

// timers
long ringTimer = 0;
long lastLoop = 0;

// variables
int hourPotVal = 0;
int minPotVal = 0;

// alarm set button
bool alarmOn = false;
int alarmOnInt = 0;     // integer version of alarmOn for easier
```

comparison.

```
// alarmRinging settings
bool alarmRinging = false;
int alarmPWM = 0;             // variable pulse width to make sure effects
are not static
bool pwmIncreasing = false;   // keeps track whether the PWM is increasing
or decreasing now
long alarmTimer = 0;          // keeps track of when the PWM is changing.

void setup() {
  Serial.begin(bdrate);
  Serial.setTimeout(10);
  // setting pins
  pinMode(lightSensorPin, INPUT);
  pinMode(hourPotPin, INPUT);
  pinMode(minPotPin, INPUT);
  pinMode(setButtonPin, INPUT);
  pinMode(ringingOffPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(lightPin, OUTPUT);
}

void loop() {
  if(millis() > lastLoop+loopWait) {
      lastLoop = millis();
      readVals();
      alarmSetButton();
      alarmDismissButton();
      readLight();
      readHour();
      readMinute();
      if (alarmRinging) {
      alarmRing();
      }
  }
}

//Receive data from Processing
void readVals() {
  if (Serial.available() > 0) {
      Serial.println("Something's available");
      String input = Serial.readString();
```

```
                input.trim();
      if (input.equals("AA1")) {
      Serial.println("Alarm time!");
      alarmRinging = true;
      }
      else if (input.equals("AA0")) {
      Serial.println("No more alarm");
      ringingOff();
      }
  }
}

void alarmSetButton() {
  int val1 = digitalRead(setButtonPin);
  if (val1 != alarmOnInt) {
      delay(5);
      int val2 = digitalRead(setButtonPin);
      if (val1 == val2) {
      alarmOnInt = val1;
      alarmOn = !alarmOn;
      Serial.print("PA");
      Serial.println(alarmOn);
      }
      if(alarmRinging == true) {
      ringingOff();
      }
  }
}

void alarmDismissButton() {
  int val = digitalRead(ringingOffPin);
  if (val > 0) {
      ringingOff();
  }
}

void readLight() {
      int val = analogRead(lightSensorPin);
      Serial.print("PL");
      Serial.println(val);
}

void readHour() {
```

```
  int val = analogRead(hourPotPin);
  if (abs(val - hourPotVal) > potThreshold) {
      hourPotVal = val;
      Serial.print("PH");
      Serial.println(val);
  }
}

void readMinute() {
  int val = analogRead(minPotPin);
  if (abs(val - minPotVal) > potThreshold) {
      minPotVal = val;
      Serial.print("PM");
      Serial.println(val);
  }
}

// accesses both the buzzer and the alarm LED.
void alarmRing() {
  int alarmTone = 100+alarmPWM*10;
  tone(buzzerPin, alarmTone);
  analogWrite(lightPin, alarmPWM);
  if (pwmIncreasing) {
      if (alarmPWM < 255) {
      alarmPWM++;
      } else {
      pwmIncreasing = false;
      }
  } else {
      if (alarmPWM > 0) {
      alarmPWM--;
      } else {
      pwmIncreasing = true;
      }
  }
}

void ringingOff() {
  alarmRinging = false;
  alarmPWM = 0;
  noTone(buzzerPin);
  analogWrite(lightPin, 0);
  Serial.print("PE");
```

```
  Serial.println(alarmRinging);
}
```

Processing

Main page

```
/*=================
Text ball animation by Marinus and Nina
3D text animation by Denzel and Jort
Clock program made by Marinus Bos and Denzel Hagen
Made as final assignment for the course Programming and Physical Computing
at the University of Twente
made in module 2 of 2021/2022
=================*/

import processing.serial.*;

TimeDisplay timeDisplay;            // displays current time
SerialComm Arduino;                 // connects to Arduino for user input
AlarmDisplay alarm;                 // displays the alarm state
Updater up;                         // custom class that takes care of the
updates.
ScreenBrightness screenBrightness;   // adjusts screen brightness
boolean testing = false;                 // enables keyboard controls in
place of Arduino, used for testing
boolean modifyDisplayBrightness = false;   // if the program should attempt
to change the actual brightness of the display (not recommended)

void setup() {
  //size(1000, 600, P3D);
  fullScreen(P3D);

  PVector field = new PVector(width, height);

  //Attempt to connect to an Arduino
  try {
      Arduino = new SerialComm(new Serial(this, Serial.list()[0], 19200));
  }
  catch(Exception E) {
      println("Arduino not found! (" + E + ")");
  }
```

```
  timeDisplay = new TimeDisplay(field, loadImage("background.jpeg"));
  alarm = new AlarmDisplay();

  if(modifyDisplayBrightness == true) {
      screenBrightness = new ScreenBrightness(127, 64, 255, field, 0, 100);
// initialize screenBrightness modifier to affect backlight brightness and
draw brightness
  } else {
      screenBrightness = new ScreenBrightness(127, 64, 255, field); //
initialize screenBrightness modifier to affect only draw brightness
  }

  up = new Updater(Arduino, alarm, screenBrightness);
}

void draw() {
  background(0);
  // update things
  Arduino.serialRead();
  up.update();
  alarm.updateClock();
  // draw things
  timeDisplay.mainLoop();
  alarm.drawClock();
  screenBrightness.drawBrightness();
}

// used for testing in place of an Arduino
void keyPressed() {
  if(testing){
      switch(key) {
      case 'q':
      // turn up the alarm hour
      Arduino.hour = (abs((alarm.getHour()+1)%24));
      break;

      case 'a':
      // turn down the alarm hour
      Arduino.hour = (abs((alarm.getHour()-1)%24));
      break;

      case 'w':
```

```
      // turn up the alarm minute
      Arduino.minute = (abs((alarm.getMinute()+1)%60));
      break;

      case 's':
      // turn down the alarm minute
      Arduino.minute = (abs((alarm.getMinute()-1)%60));
      break;

      case 'W':
      // turn up the alarm minute quickly
      Arduino.minute = (abs((alarm.getMinute()+1)%60));
      break;

      case 'S':
      // turn down the alarm minute quickly
      Arduino.minute = (abs((alarm.getMinute()-1)%60));
      break;

      case 'e':
      // turn off the ringing
      Arduino.alarmDismiss = true;
      break;

      case 'r':
      // turn onn the ringing
      alarm.startRinging();
      Arduino.triggerAlarm();
      break;

      case 'd':
      // toggle the alarm
      Arduino.alarmToggle = true;
      break;

      case 't':
      // turn up the screen brightness
      Arduino.light = ((screenBrightness.getBrightness()+8)%256);
      break;

      case 'g':
      // turn down the screen brightness
      Arduino.light = ((screenBrightness.getBrightness()-8)%256);
```

```
      break;
      }
  }
}
```

AlarmDisplay

```
/*=================
Displays the alarm time and animates it when the alarm goes off
Also changes color depending on if the alarm is turned on or off

By Denzel, 2022
=================*/

class AlarmDisplay {
  PFont font;
  AlarmLetter[] clock = new AlarmLetter[5];
  int[] xPos = new int[5];
  int hour;
  int minute;
  static final float HEIGHT_FACT = 0.8;
  boolean alarmEnabled;
  boolean isRinging;

  AlarmDisplay() {
      font = loadFont("CourierNewPSMT-48.vlw");
      textFont(font);
      alarmEnabled = false;
      isRinging = false;
      for (int i = 0; i < 5; i++) {
      xPos[i] = width/3+i*width/15;
      // load alarm hours with default "00:00"
      if (i == 2) {
      clock[i] = new AlarmLetter(':', xPos[i], height*HEIGHT_FACT, i);
      } else {
      clock[i] = new AlarmLetter('0', xPos[i], height*HEIGHT_FACT, i);
      }
      }
  }

  void drawClock() {
      for (int i = 0; i < 5; i++) {
```

```
      clock[i].drawLetter();
      }
  }

  void updateClock() {
      for (int i = 0; i < 5; i++) {
      clock[i].updateLetter();
      }
  }

  void toggleAlarm(boolean on) {
      alarmEnabled = on;
      color colour = color(64);
      if (on) {
      colour = color(255);
      }
      for (int i = 0; i < 5; i++) {
      clock[i].setColor(colour);
      }
  }

  void stopRinging() {
      isRinging = false;
      timeDisplay.alarmOff();
      for (int i = 0; i < 5; i++) {
      clock[i].stopRinging();
      }
  }

  void startRinging() {
      isRinging = true;
      timeDisplay.alarmOn();
      for (int i = 0; i < 5; i++) {
      clock[i].startRinging();
      }
  }

  void setHour(int newHour) {
      hour = newHour;
      // find the first and second number of the value, then place them in
their corresponding spot
      char leftHour = (""+(newHour/10)).charAt(0);
      char rightHour = (""+(newHour%10)).charAt(0);
```

```
      clock[0].setLetter(leftHour);
      clock[1].setLetter(rightHour);
  }

  void setMinute(int newMin) {
      minute = newMin;
      // find the first and second number of the value, then place them in
their corresponding spot
      char leftMin = (""+(newMin/10)).charAt(0);
      char rightMin = (""+(newMin%10)).charAt(0);
      clock[3].setLetter(leftMin);
      clock[4].setLetter(rightMin);
  }

  int getHour() {
      return hour;
  }

  int getMinute() {
      return minute;
  }

  boolean isEnabled() {
      return alarmEnabled;
  }

  boolean isRinging() {
      return isRinging;
  }
}
```

AlarmLetter

```
/*=================
Individual letter of the alarm display

by Denzel and Jort
=================*/


class AlarmLetter {
  static final int DEF_SIZE = 60;
```

```
  static final int RING_SIZE = 200;

  char myLetter;
  float xPos, yPos, zPos;
  float xOrigin, yOrigin;
  float angleX, angleY, angleZ;
  color letterColor;
  boolean ringing;
  float rotaX, rotaY, rotaZ; // rotation in the various dimensions
  int num;                   // keeps track which number in the word this
number is
  int textSize;

  AlarmLetter (char letter, float xPos, float yPos, int num) {
      myLetter = letter;
      this.xPos = xPos;
      this.yPos = yPos;
      xOrigin = xPos;
      yOrigin = yPos;
      zPos = 0;
      angleX = 0; //random (2 * PI);
      angleY = 0;
      angleZ = 0;
      ringing = false;
      letterColor = color(255);
      textSize = DEF_SIZE;
      this.num = num;
  }

  void drawLetter() {
      pushMatrix();
      translate(xPos, yPos, zPos);
      fill(letterColor);
      rotateX(angleX);
      rotateY(angleY);
      rotateZ(angleZ);
      textSize(textSize);
      text(myLetter, 0, 0);
      popMatrix();
  }

  // this method updates the letter
  void updateLetter() {
```

```
      if (ringing) {

      angleX = angleX + rotaX;
      angleY = angleY + rotaY;
      angleZ = angleZ + rotaZ;
      }
  }

  // this code is ran to put the letters back into their place.
  void stopRinging() {
      ringing = false;
      rotaX = 0;
      rotaY = 0;
      rotaZ = 0;
      angleX = 0;
      angleY = 0;
      angleZ = 0;
      xPos = xOrigin;
      yPos = yOrigin;
      zPos = 0;
      textSize = DEF_SIZE;
      //letterColor = color(0, 0, 0);
  }

  // this method makes a letter spazz out when the alarm goes off.
  void startRinging() {
      ringing = true;
      xPos = (num*width/5)+RING_SIZE/2;
      yPos = width/3;
      rotaX = random(-0.25, 0.25);
      rotaY = random(-0.25, 0.25);
      rotaZ = random(-0.25, 0.25);
      textSize = RING_SIZE;
  }

  void setColor(color letterColour) {
      letterColor = letterColour;
  }

  void setLetter(char number) {
      myLetter = number;
  }
}
```

Ball

```
/*=================
A basic ball. Pulls color from the background image
and drifts back to its initial position constantly.

Clock changelog:
 - Added ability to change size on the fly
 - Spiral and recoil have their input reworked
 - Alarm function where the ball rotates around the center of the screen
=================*/

class Ball {
  PVector initPos, currPos, velocity;
  int movementType = 0;
  float size;
  PVector field;
  color fillColor;
  PImage background;
  boolean alarmState = false;


  Ball(PVector initPos, float size, PVector field, PImage background) {
      this.initPos = initPos.copy(); //the position to which it returns
      this.currPos = initPos.copy(); //the current position
      this.size = size;
      this.background = background;
      this.fillColor = getColor();
      this.field = field;

      velocity = new PVector(0, 0);
  }

  void move() {
      //when the alarm goes off, start spiraling around the center of the
screen
      if(alarmState) {
      velocity.add(spiral(new PVector(field.x/2, field.y/2), field.x/250));
      }

      velocity.add(gravity()); //apply gravity
      velocity.div(resistance()); //apply resistance
```

```
      currPos.add(velocity); //move the ball
  }

  // Draw the ball with a slight glow
  void display(float seperation) {
      pushMatrix();
      translate(seperation/2, seperation/2);
      ellipseMode(CENTER);
      noStroke();
      fillColor = getColor();
      fill(fillColor, 25);
      float glowRadius = size*2;
      float glowSteps = (glowRadius-size)/10;
      for(; glowRadius >= size; glowRadius -= glowSteps) {
      circle(currPos.x, currPos.y, glowRadius);
      }
      fill(fillColor, 255);
      circle(currPos.x, currPos.y, size);
      popMatrix();
  }

  void alarmOn() {
      alarmState = true;
  }

  void alarmOff() {
      alarmState = false;
  }

  //set a new home location for the balls
  void setHome(PVector newHome) {
      initPos = newHome.copy();
  }

  //set a new size of the ball
  void setSize(float size) {
      this.size = size;
  }

  // Constantly pull the ball back towards the initial position, with the
pull getting stronger as the ball is further away
  PVector gravity() {
      final float gravityAmp = 0.01;
```

```
      return(initPos.copy().sub(currPos).div(1/gravityAmp));
  }
  // Constant resistance to movement
  float resistance() {
      final float resistanceAmp = 0.1;
      return(1+resistanceAmp);
  }

  // Get color from the background image to be used for the ball color
  color getColor() {
      return background.get((int)this.currPos.x, (int)this.currPos.y);
  }

  PVector recoil(PVector gravityPoint, float gravityAmp) {
      float mag = gravityAmp/(sqrt(currPos.dist(gravityPoint)));
      float angle = currPos.copy().sub(gravityPoint).heading();
      return(PVector.fromAngle(angle).setMag(mag));
  }

  PVector spiral(PVector gravityPoint, float gravityAmp) {
      float rotAngle =
currPos.copy().sub(gravityPoint).rotate(PI/2).heading();
      return(PVector.fromAngle(rotAngle).setMag(gravityAmp));
  }
}
```

Letter

```
/*=================
Stores and converts a single letter to an ArrayList of balls
bitmap array credit to dhepper from font8x8 on GitHub
https://github.com/dhepper/font8x8

Clock changelog:
 - Reorganized class by adding the updateBalls() function
 - Changed font of number characters to all use the same amount of balls
 - Modified colon character to be centered
 - Added ability to change character on the fly
 - Added ability to change size on the fly
 - Added alarm state passthrough
=================*/
```

```
class Letter {

  int[][] bitmap;
  char character;
  int charWidth;
  int charHeight;
  float seperation;
  float size;
  PVector charSize;
  PVector charPos;
  PVector field;
  ArrayList<Ball> balls;
  PImage background;
  boolean alarmState;
  int recoilCountdown = 0;

  Letter(char initCharacter, PVector charPos, float seperation, PVector
field, PImage background) {
      this.field = field;
      this.seperation = seperation;
      this.charPos = charPos;
      this.background = background;

      //char width and height in amount of balls
      charWidth = 8;
      charHeight = 8;
      bitmap = new int[charHeight][charWidth];

      balls = new ArrayList<Ball>();

      //initialize the PVector
      charSize = new PVector(0, 0);

      //set the character
      changeChar(initCharacter);
  }

  //calls the main functions of the balls
  void mainLoop() {
      for (Ball ball : balls) {
      ball.move();
      ball.display(seperation);
      }
```

```
  }

  //update the home position and size of the balls from the bitmap and size
variables
  void updateBalls() {
      this.size = seperation*0.8; //the size of the inner ball
      charSize.set(charWidth, charHeight).mult(seperation); //character
size in pixels

      int ballIndex = -1; //keeps track of the amount of the next ball
needing to be updated

      for (int i = 0; i < charWidth*charHeight; i++) { //check through the
entire bitmap
      PVector gridPos = new PVector(i%int(charWidth), i/int(charHeight));
//calculate the current bitmap position
      if (bitmap[int(gridPos.y)][int(gridPos.x)] == 1) { //check if the
current bitmap position contains a ball
      ballIndex++;

      if(alarmState == false) {
            balls.get(ballIndex).initPos =
gridPos.copy().mult(seperation).add(charPos); //update the ball's home
position
            balls.get(ballIndex).alarmOff();
      } else {
            balls.get(ballIndex).alarmOn();
      }

      balls.get(ballIndex).setSize(size); //update the ball's size
      }
      }
  }


  //move the entire character
  void relocate(PVector newLocation) {
      charPos = newLocation;

      updateBalls();
  }

  //resize the character by changing the seperation value and updating the
```

```
size and position of the balls accordingly
  void resizeChar(float seperation) {
      this.seperation = seperation;

      updateBalls();
  }

  //change the character bitmap by updating the amount of balls and giving
them a new home position
  void changeChar(char newCharacter) {
      bitmap = convert(newCharacter); //convert the character code to a
bitmap and replace the old bitmap

      //count the amount of balls in the new bitmap
      int newBallCount = 0;
      for (int i = 0; i < charWidth*charHeight; i++) {
      PVector gridPos = new PVector(i%int(charWidth), i/int(charHeight));
      if (bitmap[int(gridPos.y)][int(gridPos.x)] == 1) {
      newBallCount++;
      }
      }

      //remove or add balls until the right amount is reached for the new
bitmap
      while(newBallCount != balls.size()) {
      //if the old bitmap doesn't have any balls and the new one does,
spawn a ball in the center of the character
      if(balls.size() == 0) {
      balls.add(new Ball(charSize.copy().div(2).add(charPos), size, field,
background));
      }
      //if there are currently balls and you need more, spawn a new ball on
top of an old ball, giving the appearance of them splitting
      if(newBallCount > balls.size()) {
      balls.add(int(random(0, balls.size()-1)), new
Ball(balls.get(int(random(0, balls.size()-1))).currPos.copy(), size, field,
background));
      }
      //if the new bitmap needs less balls, remove a random ball
      else {
      balls.remove(int(random(0, balls.size()-1)));
      }
      }
```

```
      //set the new homes for the balls;
      updateBalls();
  }

  void alarmOn() {
      alarmState = true;
  }

  void alarmOff() {
      alarmState = false;
  }


  //========BITMAP GENERATION========

  //Converts a string to a binary array that visually spells out the text
  int[][] convert(char input) {
      return stringToIntArr(intsToBinaryStr(charToIntMap(input)));
  }

  //Takes each character and looks up the integer representation in the
hexmap
  int[] charToIntMap(char input) {
      int[] convertedHex = new int[charHeight];
      if(input > 0x7F) {
      input = 0x7F;
      }
      convertedHex = hexmap[int(input)];

      return convertedHex;
  }

  //Takes the integer representation of your string and converts them to
binary for the bitmap
  //Also switches from each character being one element to one row of
binary being an element
  String[] intsToBinaryStr(int[] input) {
      String[] converted = new String[charHeight];

      for(int i = 0; i < input.length; i++) {
      converted[i] = intToBinary(input[i]);
      }
```

```
    return converted;
}

//Converts one integer to binary
String intToBinary(int input) {
    String output = "";
    while(input > 0) {
    if(input%2 == 0) {
    output +="0";
    input = input/2;
    } else {
    output +="1";
    input = (input-1)/2;
    }
    }

    while(output.length() < 8) {
    output += "0";
    }
    return output;
}

//Takes the binary strings and converts them to binary arrays
int[][] stringToIntArr(String[] input) {
    int[][] output = new int[input.length][input[0].length()];

    for(int i = 0; i < input.length; i++){
    for(int j = 0; j < input[i].length(); j++){
    output[i][j] = Character.getNumericValue(input[i].charAt(j));
    }
    }

    return output;
}

//Print the letter bitmap (mostly for debugging)
void printBit() {
    for(int i = 0; i < bitmap.length; i++) {
    for(int j = 0; j < bitmap[i].length; j++) {
    print(bitmap[i][j]);
    }
    println();
```

```
        }
    }

    //The map for basic characters
    int hexmap[][] = {
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0000 (nul)
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0001
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0002
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0003
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0004
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0005
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0006
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0007
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0008
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0009
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000A
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000B
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000C
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000D
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000E
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+000F
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0010
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0011
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0012
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0013
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0014
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0015
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0016
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0017
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0018
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0019
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001A
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001B
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001C
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001D
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001E
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+001F
        { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0020
(space)
        { 0x18, 0x3C, 0x3C, 0x18, 0x18, 0x00, 0x18, 0x00},   // U+0021 (!)
        { 0x36, 0x36, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0022 (")
        { 0x36, 0x36, 0x7F, 0x36, 0x7F, 0x36, 0x36, 0x00},   // U+0023 (#)
        { 0x0C, 0x3E, 0x03, 0x1E, 0x30, 0x1F, 0x0C, 0x00},   // U+0024 ($)
```

```
    { 0x00, 0x63, 0x33, 0x18, 0x0C, 0x66, 0x63, 0x00},   // U+0025 (%)
    { 0x1C, 0x36, 0x1C, 0x6E, 0x3B, 0x33, 0x6E, 0x00},   // U+0026 (&)
    { 0x06, 0x06, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0027 (')
    { 0x18, 0x0C, 0x06, 0x06, 0x06, 0x0C, 0x18, 0x00},   // U+0028 (()
    { 0x06, 0x0C, 0x18, 0x18, 0x18, 0x0C, 0x06, 0x00},   // U+0029 ())
    { 0x00, 0x66, 0x3C, 0xFF, 0x3C, 0x66, 0x00, 0x00},   // U+002A (*)
    { 0x00, 0x0C, 0x0C, 0x3F, 0x0C, 0x0C, 0x00, 0x00},   // U+002B (+)
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0C, 0x06},   // U+002C (,)
    { 0x00, 0x00, 0x00, 0x3F, 0x00, 0x00, 0x00, 0x00},   // U+002D (-)
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0C, 0x00},   // U+002E (.)
    { 0x60, 0x30, 0x18, 0x0C, 0x06, 0x03, 0x01, 0x00},   // U+002F (/)
    {0x3C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3C},
    {0x18, 0x1E, 0x1E, 0x18, 0x18, 0x7E, 0x7E, 0x7E},
    {0x3C, 0x7E, 0x66, 0x60, 0x30, 0x18, 0x7E, 0x7E},
    {0x3C, 0x7E, 0x62, 0x38, 0x38, 0x62, 0x7E, 0x3C},
    {0x78, 0x7C, 0x6E, 0x66, 0x66, 0x7E, 0x60, 0x60},
    {0x7E, 0x7E, 0x02, 0x3E, 0x60, 0x60, 0x7E, 0x3C},
    {0x38, 0x1C, 0x0E, 0x3E, 0x66, 0x66, 0x7E, 0x3C},
    {0x7E, 0x7E, 0x70, 0x78, 0x3C, 0x1C, 0x1C, 0x1C},
    {0x3C, 0x66, 0x66, 0x3C, 0x66, 0x66, 0x66, 0x3C},
    {0x3C, 0x7E, 0x66, 0x66, 0x7C, 0x60, 0x3C, 0x1C},
    { 0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x00},   // U+003A (:)
    { 0x00, 0x0C, 0x0C, 0x00, 0x00, 0x0C, 0x0C, 0x06},   // U+003B (;)
    { 0x18, 0x0C, 0x06, 0x03, 0x06, 0x0C, 0x18, 0x00},   // U+003C (<)
    { 0x00, 0x00, 0x3F, 0x00, 0x00, 0x3F, 0x00, 0x00},   // U+003D (=)
    { 0x06, 0x0C, 0x18, 0x30, 0x18, 0x0C, 0x06, 0x00},   // U+003E (>)
    { 0x1E, 0x33, 0x30, 0x18, 0x0C, 0x00, 0x0C, 0x00},   // U+003F (?)
    { 0x3E, 0x63, 0x7B, 0x7B, 0x7B, 0x03, 0x1E, 0x00},   // U+0040 (@)
    { 0x0C, 0x1E, 0x33, 0x33, 0x3F, 0x33, 0x33, 0x00},   // U+0041 (A)
    { 0x3F, 0x66, 0x66, 0x3E, 0x66, 0x66, 0x3F, 0x00},   // U+0042 (B)
    { 0x3C, 0x66, 0x03, 0x03, 0x03, 0x66, 0x3C, 0x00},   // U+0043 (C)
    { 0x1F, 0x36, 0x66, 0x66, 0x66, 0x36, 0x1F, 0x00},   // U+0044 (D)
    { 0x7F, 0x46, 0x16, 0x1E, 0x16, 0x46, 0x7F, 0x00},   // U+0045 (E)
    { 0x7F, 0x46, 0x16, 0x1E, 0x16, 0x06, 0x0F, 0x00},   // U+0046 (F)
    { 0x3C, 0x66, 0x03, 0x03, 0x73, 0x66, 0x7C, 0x00},   // U+0047 (G)
    { 0x33, 0x33, 0x33, 0x3F, 0x33, 0x33, 0x33, 0x00},   // U+0048 (H)
    { 0x1E, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x1E, 0x00},   // U+0049 (I)
    { 0x78, 0x30, 0x30, 0x30, 0x33, 0x33, 0x1E, 0x00},   // U+004A (J)
    { 0x67, 0x66, 0x36, 0x1E, 0x36, 0x66, 0x67, 0x00},   // U+004B (K)
    { 0x0F, 0x06, 0x06, 0x06, 0x46, 0x66, 0x7F, 0x00},   // U+004C (L)
    { 0x63, 0x77, 0x7F, 0x7F, 0x6B, 0x63, 0x63, 0x00},   // U+004D (M)
    { 0x63, 0x67, 0x6F, 0x7B, 0x73, 0x63, 0x63, 0x00},   // U+004E (N)
    { 0x1C, 0x36, 0x63, 0x63, 0x63, 0x36, 0x1C, 0x00},   // U+004F (O)
```

```
    { 0x3F, 0x66, 0x66, 0x3E, 0x06, 0x06, 0x0F, 0x00},   // U+0050 (P)
    { 0x1E, 0x33, 0x33, 0x33, 0x3B, 0x1E, 0x38, 0x00},   // U+0051 (Q)
    { 0x3F, 0x66, 0x66, 0x3E, 0x36, 0x66, 0x67, 0x00},   // U+0052 (R)
    { 0x1E, 0x33, 0x07, 0x0E, 0x38, 0x33, 0x1E, 0x00},   // U+0053 (S)
    { 0x3F, 0x2D, 0x0C, 0x0C, 0x0C, 0x0C, 0x1E, 0x00},   // U+0054 (T)
    { 0x33, 0x33, 0x33, 0x33, 0x33, 0x33, 0x3F, 0x00},   // U+0055 (U)
    { 0x33, 0x33, 0x33, 0x33, 0x33, 0x1E, 0x0C, 0x00},   // U+0056 (V)
    { 0x63, 0x63, 0x63, 0x6B, 0x7F, 0x77, 0x63, 0x00},   // U+0057 (W)
    { 0x63, 0x63, 0x36, 0x1C, 0x1C, 0x36, 0x63, 0x00},   // U+0058 (X)
    { 0x33, 0x33, 0x33, 0x1E, 0x0C, 0x0C, 0x1E, 0x00},   // U+0059 (Y)
    { 0x7F, 0x63, 0x31, 0x18, 0x4C, 0x66, 0x7F, 0x00},   // U+005A (Z)
    { 0x1E, 0x06, 0x06, 0x06, 0x06, 0x06, 0x1E, 0x00},   // U+005B ([)
    { 0x03, 0x06, 0x0C, 0x18, 0x30, 0x60, 0x40, 0x00},   // U+005C (\)
    { 0x1E, 0x18, 0x18, 0x18, 0x18, 0x18, 0x1E, 0x00},   // U+005D (])
    { 0x08, 0x1C, 0x36, 0x63, 0x00, 0x00, 0x00, 0x00},   // U+005E (^)
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF},   // U+005F (_)
    { 0x0C, 0x0C, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+0060 (`)
    { 0x00, 0x00, 0x1E, 0x30, 0x3E, 0x33, 0x6E, 0x00},   // U+0061 (a)
    { 0x07, 0x06, 0x06, 0x3E, 0x66, 0x66, 0x3B, 0x00},   // U+0062 (b)
    { 0x00, 0x00, 0x1E, 0x33, 0x03, 0x33, 0x1E, 0x00},   // U+0063 (c)
    { 0x38, 0x30, 0x30, 0x3e, 0x33, 0x33, 0x6E, 0x00},   // U+0064 (d)
    { 0x00, 0x00, 0x1E, 0x33, 0x3f, 0x03, 0x1E, 0x00},   // U+0065 (e)
    { 0x1C, 0x36, 0x06, 0x0f, 0x06, 0x06, 0x0F, 0x00},   // U+0066 (f)
    { 0x00, 0x00, 0x6E, 0x33, 0x33, 0x3E, 0x30, 0x1F},   // U+0067 (g)
    { 0x07, 0x06, 0x36, 0x6E, 0x66, 0x66, 0x67, 0x00},   // U+0068 (h)
    { 0x0C, 0x00, 0x0E, 0x0C, 0x0C, 0x0C, 0x1E, 0x00},   // U+0069 (i)
    { 0x30, 0x00, 0x30, 0x30, 0x30, 0x33, 0x33, 0x1E},   // U+006A (j)
    { 0x07, 0x06, 0x66, 0x36, 0x1E, 0x36, 0x67, 0x00},   // U+006B (k)
    { 0x0E, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x1E, 0x00},   // U+006C (l)
    { 0x00, 0x00, 0x33, 0x7F, 0x7F, 0x6B, 0x63, 0x00},   // U+006D (m)
    { 0x00, 0x00, 0x1F, 0x33, 0x33, 0x33, 0x33, 0x00},   // U+006E (n)
    { 0x00, 0x00, 0x1E, 0x33, 0x33, 0x33, 0x1E, 0x00},   // U+006F (o)
    { 0x00, 0x00, 0x3B, 0x66, 0x66, 0x3E, 0x06, 0x0F},   // U+0070 (p)
    { 0x00, 0x00, 0x6E, 0x33, 0x33, 0x3E, 0x30, 0x78},   // U+0071 (q)
    { 0x00, 0x00, 0x3B, 0x6E, 0x66, 0x06, 0x0F, 0x00},   // U+0072 (r)
    { 0x00, 0x00, 0x3E, 0x03, 0x1E, 0x30, 0x1F, 0x00},   // U+0073 (s)
    { 0x08, 0x0C, 0x3E, 0x0C, 0x0C, 0x2C, 0x18, 0x00},   // U+0074 (t)
    { 0x00, 0x00, 0x33, 0x33, 0x33, 0x33, 0x6E, 0x00},   // U+0075 (u)
    { 0x00, 0x00, 0x33, 0x33, 0x33, 0x1E, 0x0C, 0x00},   // U+0076 (v)
    { 0x00, 0x00, 0x63, 0x6B, 0x7F, 0x7F, 0x36, 0x00},   // U+0077 (w)
    { 0x00, 0x00, 0x63, 0x36, 0x1C, 0x36, 0x63, 0x00},   // U+0078 (x)
    { 0x00, 0x00, 0x33, 0x33, 0x33, 0x3E, 0x30, 0x1F},   // U+0079 (y)
    { 0x00, 0x00, 0x3F, 0x19, 0x0C, 0x26, 0x3F, 0x00},   // U+007A (z)
```

```
      { 0x38, 0x0C, 0x0C, 0x07, 0x0C, 0x0C, 0x38, 0x00},   // U+007B ({)
      { 0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x18, 0x00},   // U+007C (|)
      { 0x07, 0x0C, 0x0C, 0x38, 0x0C, 0x0C, 0x07, 0x00},   // U+007D (})
      { 0x6E, 0x3B, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // U+007E (~)
      { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}    // U+007F


      //{ 0x3E, 0x63, 0x73, 0x7B, 0x6F, 0x67, 0x3E, 0x00},   // U+0030 (0)
      //{ 0x0C, 0x0E, 0x0C, 0x0C, 0x0C, 0x0C, 0x3F, 0x00},   // U+0031 (1)
      //{ 0x1E, 0x33, 0x30, 0x1C, 0x06, 0x33, 0x3F, 0x00},   // U+0032 (2)
      //{ 0x1E, 0x33, 0x30, 0x1C, 0x30, 0x33, 0x1E, 0x00},   // U+0033 (3)
      //{ 0x38, 0x3C, 0x36, 0x33, 0x7F, 0x30, 0x78, 0x00},   // U+0034 (4)
      //{ 0x3F, 0x03, 0x1F, 0x30, 0x30, 0x33, 0x1E, 0x00},   // U+0035 (5)
      //{ 0x1C, 0x06, 0x03, 0x1F, 0x33, 0x33, 0x1E, 0x00},   // U+0036 (6)
      //{ 0x3F, 0x33, 0x30, 0x18, 0x0C, 0x0C, 0x0C, 0x00},   // U+0037 (7)
      //{ 0x1E, 0x33, 0x33, 0x1E, 0x33, 0x33, 0x1E, 0x00},   // U+0038 (8)
      //{ 0x1E, 0x33, 0x33, 0x3E, 0x30, 0x18, 0x0E, 0x00},   // U+0039 (9)
  };
}
```

ScreenBrightness

```
/*=================
A class which allows adjusting of the screen brightness using two methodes:
 - draw brightness: puts a translucent black overlay over the schetch,
reducing the brightness of the colors.
 - backlight brightness: uses a Windows powershell command to adjust the
actual brightness of the screen.
   Only works on compatible Windows devices, and freezes the program for a
split second for each change. A rate limit can be set to reduce this issue.
Depending on the argument given when initializing the class either draw,
backlight or both methodes will be active.

Backlight changing code is a Processing adaptation of Java code by Darty11
on Stackoverflow
(https://stackoverflow.com/questions/15880547/how-to-change-laptop-screen-b
rightness-from-a-java-application)

By Marinus Bos, 2022
=================*/

import java.io.BufferedReader;
```

```
import java.io.IOException;
import java.io.InputStreamReader;

class ScreenBrightness {
  int brightness = 127; //The screen brightness, between 0 and 255
  int drawBrightnessMin = 0, drawBrightnessMax = 0; //the range of
transparancy of the darkening overlay. Default values disable it as a
fallback. Values between 0 and 255 are allowed.
  PVector field = new PVector(0, 0);
  int backlightBrightnessMin = 0, backlightBrightnessMax = 100; //the range
of screen brightnesses, values between 0 and 100 are allowed
  int backlightChangeWait = 1000; //limits the rate in which the backlight
can be changed
  int lastBacklightChange = 0;
  boolean changeBackgroundBrightness; //sets if the backlight brightness
will be changed

  //initialize to only affect background brightness
  ScreenBrightness(int startBrightness, int backlightBrightnessMin, int
backlightBrightnessMax) {
      this.backlightBrightnessMin = backlightBrightnessMin;
      this.backlightBrightnessMax = backlightBrightnessMax;

      changeBackgroundBrightness = true;

      setBrightness(startBrightness);
  }

  //initialize to only affect draw brightness
  ScreenBrightness(int startBrightness, int drawBrightnessMin, int
drawBrightnessMax, PVector field) {
      this.drawBrightnessMin = drawBrightnessMin;
      this.drawBrightnessMax = drawBrightnessMax;
      this.field = field;

      changeBackgroundBrightness = false;

      setBrightness(startBrightness);
  }

  //initialize to affect both draw and background brightness
  ScreenBrightness(int startBrightness, int drawBrightnessMin, int
drawBrightnessMax, PVector field, int backlightBrightnessMin, int
```

```
backlightBrightnessMax) {
      this.drawBrightnessMin = drawBrightnessMin;
      this.drawBrightnessMax = drawBrightnessMax;
      this.field = field;
      this.backlightBrightnessMin = backlightBrightnessMin;
      this.backlightBrightnessMax = backlightBrightnessMax;

      changeBackgroundBrightness = true;

      setBrightness(startBrightness);
  }

  //call to change the brightness, performs a sanity check and then sets a
variable for drawbrightness and calls the change method for background
brightness
  void setBrightness(int newBrightness) {
      if(newBrightness < 256 && newBrightness > 0) {
      this.brightness = newBrightness;

      if(changeBackgroundBrightness == true) { //set the background
brightness if this is enabled
      setBacklightBrightness(brightness);
      }
      }
      println(brightness);
  }

  int getBrightness() {
      return(brightness);
  }

  //call at end of draw() only if class is initialized to affect draw
brightness
  void drawBrightness() {
      int alpha = int(map(brightness, 0, 255, drawBrightnessMin,
drawBrightnessMax));
      fill(0,0,0,255-alpha);
      rect(0,0,width,height);
  }

  //changes the backlight brightness, with try statement to handle any
errors
  void setBacklightBrightness(int brightness) {
```

```
      if(millis() > lastBacklightChange + backlightChangeWait) {
      lastBacklightChange = millis();
      try {
      brightness = int(map(brightness, 0, 255, backlightBrightnessMin,
backlightBrightnessMax));
      trySetBacklightBrightness(brightness);
      }

      catch(Exception E) {
      println(E);
      }
      }
  }

  //uses Windows PowerShell to change the background brightness. Needs to
be called from a try-statement to handle any errors.
  //Processing adaptation of Java code by Darty11 on Stackoverflow
(https://stackoverflow.com/questions/15880547/how-to-change-laptop-screen-b
rightness-from-a-java-application)
  void trySetBacklightBrightness(int brightness)
      throws IOException {
      //Creates a powerShell command that will set the brightness to the
requested value (0-100), after the requested delay (in milliseconds) has
passed.
      String s = String.format("$brightness = %d;", brightness)
      + "$delay = 0;"
      + "$myMonitor = Get-WmiObject -Namespace root\\wmi -Class
WmiMonitorBrightnessMethods;"
      + "$myMonitor.wmisetbrightness($delay, $brightness)";
      String command = "powershell.exe  " + s;
      // Executing the command
      Process powerShellProcess = Runtime.getRuntime().exec(command);

      powerShellProcess.getOutputStream().close();

      //Report any error messages
      String line;

      BufferedReader stderr = new BufferedReader(new InputStreamReader(
      powerShellProcess.getErrorStream()));
      line = stderr.readLine();
      if (line != null)
      {
```

```
      System.err.println("Standard Error:");
      do
      {
      System.err.println(line);
      }
      while ((line = stderr.readLine()) != null);
      }
      stderr.close();
  }
}
```

SerialComm

```
/*=================
A simple class for Serial communication
Reads data in the following packet structure:
  PA123\r\n
  ^ P indicates that the data is intended for Processing
   ^ A indicates what variable to store the data in
      ^^^ 123 is the value being transmitted
      ^^^^ newline indicates the end of a packet

In this case the following variables are accepted:
 L = light data from the LDR, H = hours, M = minutes (Arduino Uno analog
in, so values between 0 - 1023)
 E = alarm dismissed button, A = alarm toggle button (Digital inputs, so
values between 0 - 1)
=================*/

class SerialComm {
  Serial serial;

  //Store values received from the serial communication
  int light, hour, minute;
  boolean alarmToggle, alarmDismiss;

  SerialComm(Serial serial) {
      // Opening the port
      this.serial = serial;
      serial.bufferUntil('\n');

      // defaults when no Arduino is attached:
```

```
      light = 255;
      hour = 17;
      minute = 30;
      alarmToggle = false;
      alarmDismiss = false;
  }

  void triggerAlarm() {
      serial.write("AA1\r\n");
  }

  void serialRead() {
      String input = "";
      while(serial.available() > 0) {
      try {
      // read the data until the newline n appears
      input = serial.readStringUntil('\n');

      if (input != null) { //if there's data to process
            input = trim(input); //trim whitespace

            if(input.charAt(0) == 'P') { //see if the packet is destined
for Processing
            char type = input.charAt(1); //seperate the variable indicator
from the packet
            int val = int(input.substring(2)); //seperate the value from
the packet

            switch(type) { //store the value in the appropriate variable
            case 'L':
            light = val;
            println("Light: " + light);
            break;
            case 'H':
            hour = val;
            println("Hour: " + hour);
            break;
            case 'M':
            minute = val;
            println("Minute: " + minute);
            break;
            case 'E':
            alarmDismiss = true;
```

```
            println("Alarm dismissed");
            break;
            case 'A':
            alarmToggle = val > 0; //returns false if value is 0, true if
value is 1
            println("Alarm toggled");
            break;
            }
            }
      }
      }

      catch(Exception E) {
      println(E);
      }
      }
  }
}
```

TimeDisplay

```
/*=================
Time display using the letter and ball classes. Holds the letter objects
and actualizes the time displayed by them.
Resizes letters to the field size.
Also passes alarm state to the letters.

By Marinus Bos, 2022
=================*/

class TimeDisplay {
  ArrayList<Letter> characters;
  char[] charDisplay;
  PVector stringSize;
  PVector field;
  float seperation = 1;
  PImage background;

  TimeDisplay(PVector field, PImage background) {
      this.field = field;
      this.background = background;
```

```
      characters = new ArrayList<Letter>();

      //generate initial letters
      for (int i = 0; i < getTimeString().length(); i++) {
      characters.add(new Letter(' ', new PVector(0, 0), 1, field,
background));
      }

      //organise the characters on the field
      organise(field);
      //resize the background image
      background.resize(width, height);
  }

  void mainLoop() {
      updateTime();
      organise(field);
      for (Letter letter : characters) {
      letter.mainLoop();
      }
  }

  //calculate and set the location and size of the letters
  void organise(PVector field) {
      //figure out the strings's displayed size
      stringSize = new PVector(0, 0);
      for(Letter letter : characters) {
      stringSize.x += letter.charSize.x;
      if(letter.charSize.y > stringSize.y) {
      stringSize.y = letter.charSize.y;
      }
      }

      //resize the characters to fit in the screen, with some margins
      seperation = seperation*field.x/(stringSize.x*1.2);
      for(Letter letter : characters) {
      letter.resizeChar(seperation);
      }

      //find where the string needs to start to be centered
      PVector letterPos = field.copy().div(2).sub(stringSize.div(2));

      //move the string to the center
```

```
      float xPos = 0;
      for(Letter letter : characters) {
      letter.relocate(letterPos.copy().add(xPos, 0));
      xPos += letter.charSize.x;
      }
  }

  //update the time displayed
  void updateTime() {
      char[] newCharDisplay = getTimeString().toCharArray();

      if(newCharDisplay != charDisplay) { //only update if the time has
changed
      charDisplay = newCharDisplay;
      for (int i = 0; i < charDisplay.length; i++) {
      characters.get(i).changeChar(charDisplay[i]);
      }
      }
  }

  void alarmOn() {
      for(Letter letter : characters) {
      letter.alarmOn();
      }
  }

  void alarmOff() {
      for(Letter letter : characters) {
      letter.alarmOff();
      }
  }

  //returns a string containing formatted time
  String getTimeString() {
      return String.format("%02d", hour())+":"+String.format("%02d",
minute())+":"+String.format("%02d", second());
  }
}
```

Updater

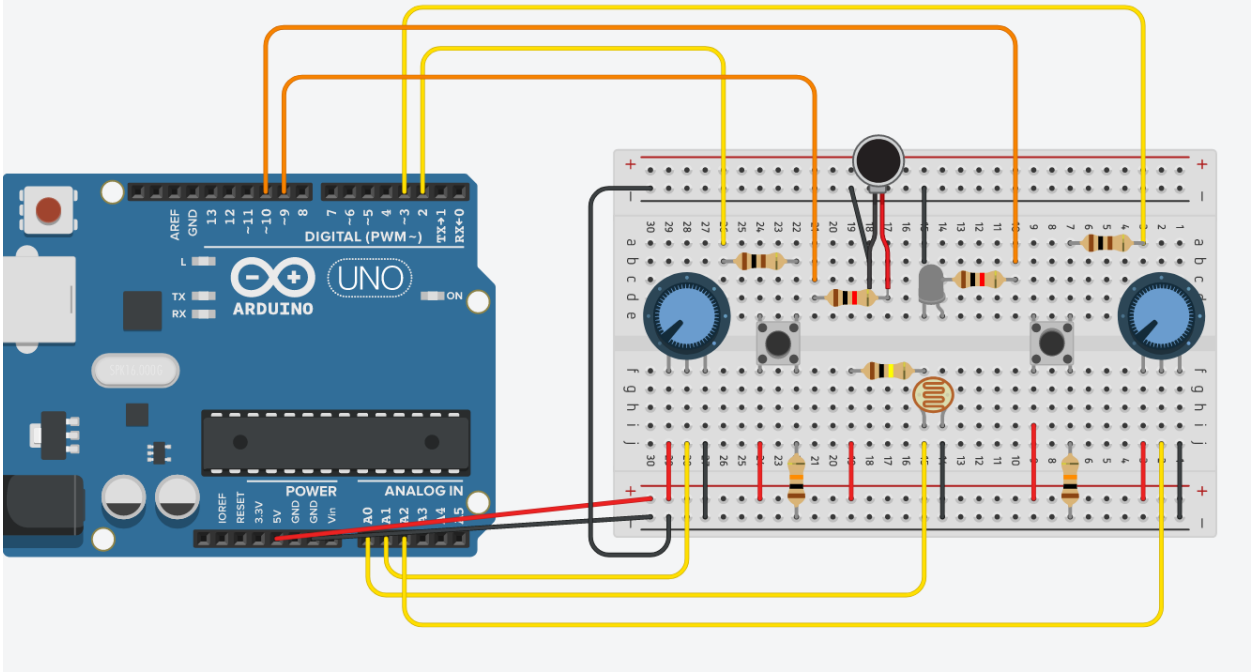
```
/*=================
```

```
Prepares variables to be used by functions, and calls these functions when
the variables have changed.
=================*/


class Updater {
  SerialComm s;
  AlarmDisplay alarm;
  ScreenBrightness screen;
  boolean alarmHasBeenToggled = false; //makes sure each button press only
toggles the alarm once

  Updater(SerialComm serial, AlarmDisplay a, ScreenBrightness scr) {
      s = serial;
      alarm = a;
      screen = scr;
  }

  //call relevant functions to update parts of the program based on
variables
  void update() {
      updateHour();
      updateMinute();
      updateAlarm();
      updateRinging();
      checkAlarm();
      updateBrightness();
  }

  //activate the alarm at the correct time
  void checkAlarm() {
      if (hour() == alarm.getHour() && minute() == alarm.getMinute() &&
alarm.isEnabled() && !alarm.isRinging() && second() == 0) {
      alarm.startRinging();
      s.triggerAlarm();
      }
  }

  //when the toggle alarm button has been pressed, toggle the alarm state
  void updateAlarm() {
      if(s.alarmToggle) {
      if(alarmHasBeenToggled == false) {
      if(alarm.isEnabled()) {
```

```
            alarm.toggleAlarm(false);
            alarmHasBeenToggled = true;
      } else {
            alarm.toggleAlarm(true);
            alarmHasBeenToggled = true;
      }
      }
      } else {
      alarmHasBeenToggled = false;
      }
  }

  //dismiss the alarm when the relevant button has been pressed
  void updateRinging() {
      if (s.alarmDismiss) {
      s.alarmDismiss = false;
      alarm.stopRinging();
      }
  }

  void updateHour() {
      int hour = int(map(s.hour, 0, 1000, 0, 23)); //map the analog in
values to hours, with some margins for the upper value

      //ensure that the values are in the correct range
      if(hour > 23) {
      hour = 23;
      } else if(hour < 0) {
      hour = 0;
      }

      //if the value has been updated, call the relevant function
      if (hour != alarm.getHour()) {
      alarm.setHour(hour);
      }
  }

  void updateMinute() {
      int min = int(map(s.minute, 0, 1000, 0, 59)); //map the analog in
values to minutes, with some margins for the upper value

      //ensure that the values are in the correct range
      if(min > 59) {
```

```
      min = 59;
      } else if(min < 0) {
      min = 0;
      }

      //if the value has been updated, call the relevant function
      if (min != alarm.getMinute()) {
      alarm.setMinute(min);
      }
  }

  void updateBrightness() {
      int brightness = int(map(s.light, 512, 32, 0, 255)); //map the analog
in values to a byte, with large margins

      println("Mapped light: " + brightness);

      //if the value has been updated, call the relevant function
      if (brightness != screenBrightness.getBrightness()) {
      screenBrightness.setBrightness(brightness);
      }
  }
}
```

Schematic



Insightful mistakes

#1: Attempting to change the actual screen brightness of the computer

This one didn't quite pan out as hoped. I found some Java code online that uses Windows PowerShell to change the brightness of the screen, but it ended up just being too much. Every time the brightness changed, the program stuttered for a bit, and the code ended up messy and unorganized. We ended up settling for a translucent black overlay instead.

#2: Poor planning

It is 21:30 on the day of the deadline as I write this, and I really want to go to bed. This is something that seems to happen on every major project I work on, and fixing it would be outside of the scope of PPC. It is presumably due to us having to divide time over both PPC and SE projects, and the SE project has more people behind it and more time invested in it, so it usually gets priority.

#3: Simulation Versus Reality

I worked on the basic Arduino code, but I didn't have time to set up the circuitry physically, so I used Tinkercad's simulation feature. It is a bit clunky, but it mostly works. The only problem I had with this is that a button that was working as intended (like a latch) on the simulation, but didn't work in the real setup. In the future we should test this sooner.