Sense Your Heart

R. Donselaar, J.J. Wubbels, I.B.I. Ayoola, J.J. Siekmans
18th of March, 2011

TU/e Technische Universiteit Eindhoven University of Technology

Sense Your Heart | TU/e

R. Donselaar | J.J. Wubbels, | I.B.I. Ayoola | J.J. Siekmans

18th of March, 2011

Contents

1 | introduction

In order to benefit a design with the use of ECG (electrocardiogram), this module instructs on how to engineer such a device. By using a PPG system (photoplethysmograph), data from the heart beat can easily be obtained. Although the device is relatively simple, the design opportunities that are created are fast and as complex as the designer's wishes.

This report shows an example of what designers can do with a PPG that was made during this module. By combining the PPG with an Arduino programs can be made to let an interaction happen between the data extracted from the heart and a program.

We would like to thank the following coaches that offered their support and knowledge during this module:

- prof.dr. S. Bambang Oetomo

- prof.dr.ir. L.M.G. Feijs

- W. Chen

- dr. J. Hu PDEng MEng

- dr.ir. G.R. Langereis

- G.J.A. van den Boomen

2 | PPG

2.1 how it works

PPG stands for photoplethysmograph, and makes volumetric measurements of an organ. This is done by illuminating the skin and measuring the changes in light absorption. The change in volume within the organ resulting from heart's contractions can provide data on the heartbeats. From this different calculations can be made for further application purposes as is what we try to explore during this module.

This PPG delivers two different types of data set, digital and analog. Analog can be used to better understand the signal coming from the heart. The digital signal allows a computerized system to easily work with the gathered data. Both signals can be seen in the next paragraph.

2.2 testing

The PPG can be tested in three ways to determine whether it works or not. The first being the LED that is located on the board. This LED already shows the beats of the user's heart when it works properly. However this readout is not very accurate and cannot be used for further calculations. It can also be the case that the Piezometer should be adjusted as the functionality of the system can differ per person.

Therefore the PPG should be tested with the oscilloscope in order to get a better idea whether the system works as required. When it does, a clear pattern can be seen of apparent sinus function that represents the heartbeats. Although the amplitude can easily differ, this does not matter considering the signal is going to be used digital later on.

As can be seen we use the sensor of the PPG on the finger, as a clear feedback is gained when the heartbeat is felt in the finger. This can then directly be visible relayed and compared to the LED or oscilloscope.

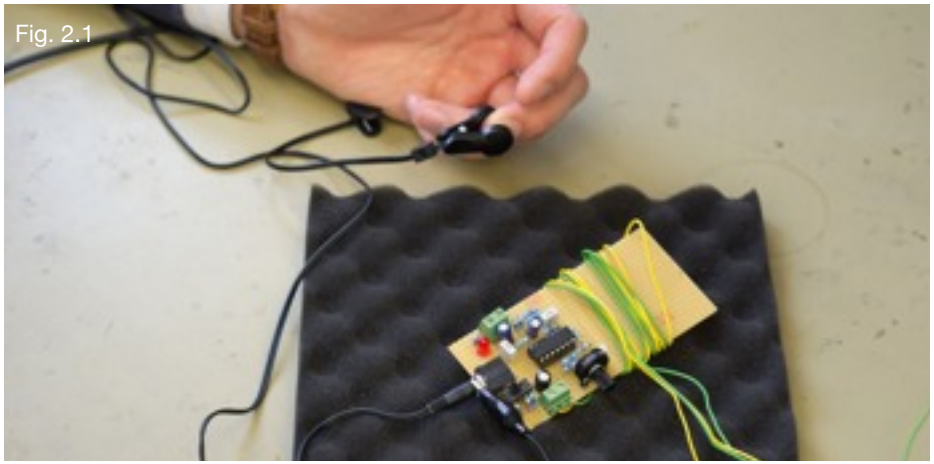


Fig. 2.1

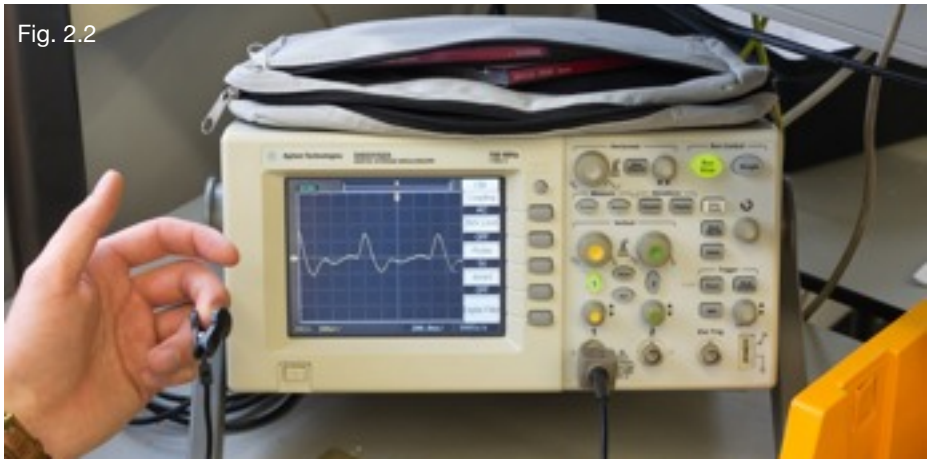


Fig. 2.2

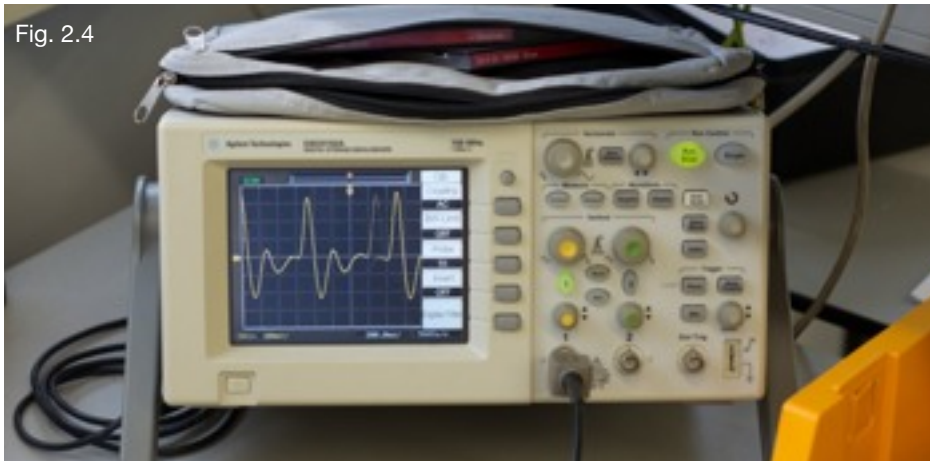
The final confirmation of that the PPG is working is done with the help of the computer. First the Arduino has to be updated to have the Arduino sensor program installed on it. Hooking up the PPG to the Arduino then allows the computer to gather the data from the patient. Here the different values are calculated by using the heartbeat data. The following image shows what the result from this program is.

The software shows three calculation in a interface. The green diagram at the bottom shows the beat to beat interval. The top blue diagram shows the heart rate variability (HRV). The HRV means the variability of the interval between the heartbeats. The centre circle plot is a more aesthetic composition that uses formulas linked to specific values to make the reading easier. This later option also allows for more contrast that could be beneficial to a design later in this module.

Fig. 2.3

Below the analog signal is seen in figure 2.3, this time with a greater amplitude then in figure 2.2. On the right, the digital signal from the PPG is shown. Note that the digital signal transfers the peaks to an absolute value. This way the software can easily count the heartbeats from the patient, as most of the interference is filtered out.

A PPG system cannot indicate as much detail as a ECG does, however some medical issues can be detected like with kid patients and more general heart related problems.



3 | idea generation

3.1 orientation

A we design using a single modality, being the data coming from the PPG, the design will be created from a bottom up approach. In order to plot the possibilities and boundaries for a design that just uses measurement data, a number of points have been set out. The measurements are divided into what can actually be measured, and what we think can be indirectly measured.

Table. 3.1

Measure directly	**Measure indirectly**
Heart rate	Stress / relaxation
Beat to beat	Breathing
Heart rate variability	Arousal
	Stamina / condition
	Mood
	Clinical deviations

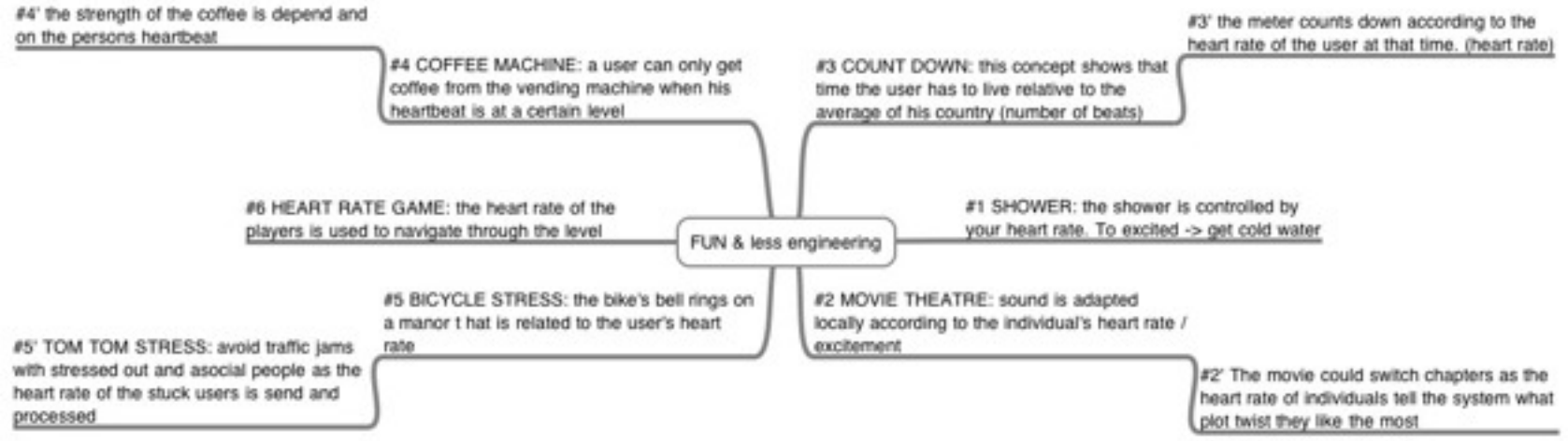
Besides these directional points, the brainstorm also stimulated to set up points of interest. Considering other products made by using a PPG, a strong sensation of engineering was experienced. The products seem to be very focussed on function and result, instead of perhaps fun and even meaninglessness. This seemingly unexplored terrain could be an interesting take for this design process.

Furthermore only one example presented in the presentations involved multiple users. This also could be an interesting point for development as a human's heart rate is commonly unknown to the public. Bringing the individual heart rate into context could give rise to some interesting scenarios. The relative slow adaptation time that is inherent to the heart rate is the difficulty.

3.2 brainstorm

Below the main topics of the brainstorm is shown. Some ideas have derivatives and are indicated by ('). From this brainstorm the best idea was selected, namely the game. The game offers the possibility to use the data from the PPG in a fun and playful way. Furthermore the game allows multiple users to be brought into the design.

Fig. 3.1



4 | game prototype

4.1 introduction

As was stated earlier, the game allows for users to play together, and has a more fun and playful core then other designs on the market that use a PPG. The program is written using an Arduino in combination with Processing. The code that was delivered with this module allowed us to further expand a game construct around it. The game concept is relative simple considering the controls are more or less crude especially when compared to modern game controllers.

4.2 concept

The game starts with the user being hooked up to the PPG. Data about the player's heart is subtracted, from which the heart rate is calculated. Elevating the heart rate would make an image of the heart go up, and lowering the heart rate makes this same image move downwards. The meaning of the game is to move the heart up and down and collect as many red blood cells as possible. Because the red blood cells move towards the heart, the player has to try to move the heart up and down in order to score points. When a player has gathered XXX points, the red blood cells start moving faster what would indicate the next level. The faster moving cells are harder to hit, and should have an impact on the player's heart rate. This then creates another difficulty aspect to the game. keeping in control of your heart rate and breathing is essential to be able to score well in the game.

When a second players joins the game, they can compete for the red blood cells. Both their scores will be counted. When one player gathers more points then the other, and get to the next level, it becomes even harder for the lesser player to overtake the winner. This friction will create a dynamic situation that in it's turn would make it even harder for both players to regulate their heart rate.

If a players fails to receive a significant number of cells, this player will loose the game. Loosing is caused by the counter that also registers missed blood cells, that will be subtracted from the players total points.

4.3 programming

All the programs delivered to the students during this module are designed to work with just one user. As the game concept becomes richer with at least two players, all the programming had to be

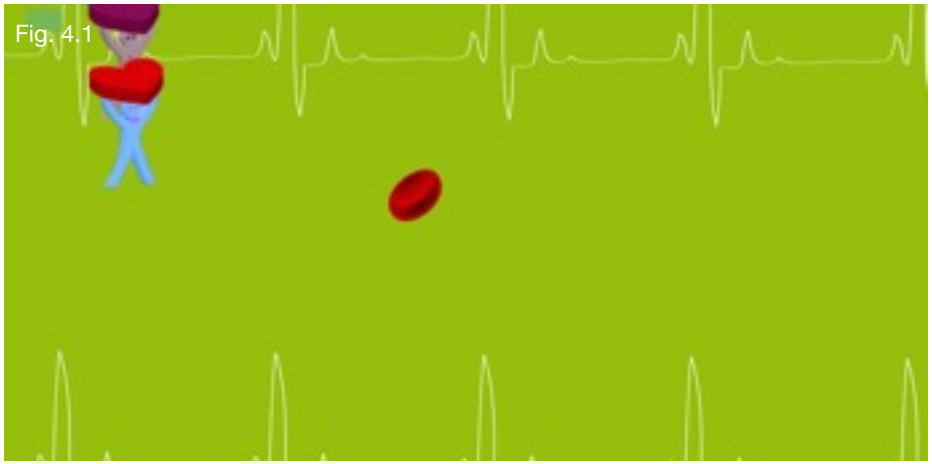


Fig. 4.1

Fig. 4.2

adapted to support two PPG's. By adjusting the Arduino's sensor program, and changing the RRport file in order to process the two different sets of data, it became possible to accept two players in the game. By making the Arduino send a key from which PPG it gets its signal, the computer understands that there are two sets of information coming in. Firstly, in the Arduino codes an additional hardware interrupt was attached to the sketch on digital port 3. An handshake protocol was used to reliably communicate with the game program in processing. Arduino sends in an ID number appended with the RR sample. On serial event in processing, the processing file sorts the data and push the relevant values into the allocated buffers.

The data from the heart, by using a PPG, is not easily transcribed to fit the rather fast control that is needed for a game. The heart rate needs time to recover from an activity, and is therefore less reactive then for example a regular game controller. We had to take this into account when we programmed the navigation. By limiting the heart's data to control just the up and down movement, a manageable and fun navigation is established.

Thoughts on making the blood cells move faster or slower depending on the heart rate we're dismissed as the speed has greater value coupled to the levels.

In the delivered program multiple calculations are done with the data retrieved from the PPG. We tried the three calculations that were offered to control the game. However we found that the heart rate is the most usable value to use for control. Nevertheless we did have to adapt the handling of it a bit. As when the heart rate is used directly, the control is not so smooth and confusing. The user cannot directly understand what effect his heart rate has on the movement of the heart figure in the game. Therefore we made it dependent on one average value. When the heart rate goes up relative to the preset average, the players moves up and when the heart rate goes down relative to the preset average, the player goes down, both in a smooth motion. This smooth motion is set by using a set value of movement that relates to whether the value is above the preset average or below.

Lack of time left the possibility to explore using patterns linked to decreases / increases of the heart beat data untested. Using patterns was used with the "spiral program" that is showed in figure XXX. This could have an advantage to create a more usable control of the game using the heart rate.

As the game runs now, it can continue onwards for as long as the system is powered. Therefore we had to make a Game Over screen in order to set an end to the game. The game over image appears when the players fails to gain enough red blood cells. The player with the most red blood cells wins.

Few other tryouts were attempted to directly map a convenient maximum and minimum Heart-rate, or heart-rate-variability, or RR intervals to the movement of the player icon were not successful because they rapidly changed. This also explains why the final control method was adopted.

5 | conclusion

Using the PPG requires a designer to be thoughtful of the limited control one has over the heart rate. The innate slow reaction time of changing the heart rate can make it difficult to us it as a direct stimulus for example a game. Therefore this game has been designed to compress the data from the heart into a more black and white processing system.

When the players moves the onscreen player he or she has to control their heart rate and breathing. A combination of both can be used to move up and down, and so collect the red blood cells.

When both players are more or less equally in skill, the game will let the blood cells move faster thus making it even harder to compete.

Testing with the different data manipulations that were offered, and created by ourselves showed that control cannot be gained by using sensor input directly.

6 | reflections

By Johan Siekmans

Considering we had to design from a single modality, being the PPG, this module has parallels with the “sensual dynamics” module. It also worked on my train of thought in a similar way, as it can be difficult to find proper design solutions that fit just a single modality.

Therefore we looked at the strengths of this modality and tested with it. By simply doing, and testing the usability of such a design approach becomes more apparent. This is what is very valuable to me to know, because I used to have trouble conceptualizing from a single modality. But now I have a way of approaching this.

The practical bit of the module, where the PPG sensors were soldered should not have faced me with difficulty, considering I have experience in this. Nevertheless I made the mistake of trying to work to fast. By soldering the PPG for a second time, in a whole, I got a proper working one. This shows me that I should slow down at times. The lifestyle of a designer can be in a fast mode continuously, but apparently gearing down has its benefits at times.

Although the medical side of this module, and the practical experience with the ECG was not new to me. It was interesting to look at them in a designer’s point of view. Before I specialized in the medical field of Mechanical Engineering, and therefore have my share of experience. However I rarely looked at these kinds of data sets in such a way. Therefore, for me, the game was even more fun considering it moves so far away from an engineering mindset.

Although we had a working game, I am still of the opinion that combining modalities that work great together is what can be described as the designers greatest asset. Therefore with a full project this concept should have been improved and expanded. Nevertheless it was fun to see how a game can be created that is fun and innovative just by controlling it with two options, up and down.

By Jordy Wubbels

At the beginning of this module I didn’t really know what to expect. My overall intention was to get familiar with the design possibilities offered by heart measurement sensors. I was unaware of how this would be done and what the possibilities were. Even though my general interest is not in healthcare design, I was interested in finding out about the possibilities for new interaction opportunities.

During the first part of the module we spent a day building a working PPG-sensor. Though I understand that this sensor was needed for the module, spending a full day on making this sensor wasn’t very valuable. It might be considerd supplying people with a (highly accurate) working sensor if the costs involved would allow for this (one per team). There would be a big advantage in having an extra day for exploration, tests, prototyping and more.

Nevertheless once we had the sensor up and running it was interesting to learn about the possibilities of heart monitoring. There seems to be something magical about being able to see your own heart rate. Unfortunately it soon turned out that the sensors were very sensitive for picking up noise, for example caused by movement. As we found out later they couldn’t be used for precise and reliable measurements in combination with the Arduino.

During the exploration with the sophisticated heart monitoring system I learned that even this high end equipment was very sensitive to small changes, like movement. This taught me that systems which rely on heart measurements for input and require precise control at the same time aren’t very suitable for applications that involve movement, or other noise introducing factors. It was nice to get a little bit familiar with the possibilities of professional heart measurement equipment. I can see this equipment being used for user testing where peoples hart can give valuable information, for example on excitement. Though my interests are more with the smaller, low cost, sensors that we build ourselves since I’m more interested in design and interaction implementation rather than precise and health related monitoring.

Choosing to create a game for this module, provided the opportunity to explore the limits and possibilities of the sensor in situations that dó in fact introduce movement and noise. Being especially interesting for interactive design, as this hardly ever involves people in a static situation.

I learned from this that these sensors can actually be used in more dynamic situations, as long as they don't require very precise control, or involve continuous movements without more steady intervals. As a rough input or maybe combined with other signals the sensors are certainly useable in more dynamic situations, for precise monitoring related to health, not so much. There might be opportunities for extra filtering through software, which could make the sensor/Arduino combination more precise and reliable.

Overall I found it interesting to become aware of the possibilities of using heart monitoring in design. However, for me the learning opportunities during this module were limited as it consisted largely of programming our application. This was a good refreshment of my programming skills, but I didn't really learn anything new here and I felt that more people experienced the same. I therefore mainly see this module as an awareness-raiser and a step-up for possible future uses of heart measurements in my designs.

For future editions of this module it might be interesting for participants to learn more about software filtering and to go more in depth on how the Arduino program works and for example how to connect multiple sensors to one Arduino. This would introduce more new knowledge and skills to participants. Maybe by using the sensors that were created this module, (one from each group) time could be saved on the first day, which could then later be used for the above mentioned exploration in software. This is of course just a suggestion and might not be suitable in the timeframe of one week.

By Rik van Donselaar

My identity as a designer identifies healthcare related project. I believe that design should come form a specific need or desire which is closely related to the end user. Ideally, users who are really requiring help. My areas broadly varies from healthcare themes to developing countries. At the moment I am conducting a research project on measuring oxygen saturation for neonates in the healthcare theme as well. I have chosen the module because I found it very interesting to investigate the same principle in another context. The technology used for measuring ECG is also used for measuring SpO2. In that sense I could translate acquired knowledge to my own project later.

In the module I learned how to build an analog Photoplethysmograph (PPG) sensor that measures changes in volume. More specific, the builded sensor consisted of an amplifier, a low and high pass filter. I experienced that a PPG signal is weak and sensitive to motion artifacts. Therefore filters were used to reduce signal noise.
For building the sensor all resources and equipment were available, in term of that building the sensor was not really challenging because I have already a lot soldering skills achieved from previous projects. However, at the end it turned out that my sensor did not work. I had to start debugging. Because of this debugging process I managed to specifically figure out how the schematic precisely worked. At the end it was just one simple mistake in the wiring but eventually I knew how the sensor and filters work. As a point for improvement it might be interesting to provide fully assembled sensors. In a one week module, time is limited, very limited. And building a sensor for a full day is a lot. So if there is an opportunity to provide a fully assembled sensor per group, groups will have one day more to work on concepts and explorations.

In our idea generation we strive to combine the possibilities of the sensor with fun aspects. In this creative phase I gained a clear understanding of what you can indirectly measure with PPG, for example heart rate variability or beat-to-beat rate(RR), which can be used as an inspiration for other projects.

In our concept I aimed to design something valuable for other people and not just to create a certain experience. The game should create a

better understanding of your heart beat and what influences your heart beat. It is about learning to control and regulate your own heart. As with most prototypes the most valuable experience is in details. The game itself worked but there was too little time to really fine tune the sensor part to demonstrate the full experience of the game.

The gained knowledge in this module was very valuable for me. I gained a clear understanding of PPG sensors and theoretically background of ECG and how it is measured. Through developing the game concept I experimented what can be indirectly measured with PPG. In my current project I want to use the gained knowledge as well. Instead of measuring heart beat I want to measure SpO2 which is a bit more difficult. Measuring SpO2 requires an IR LED, Red Led and a photodiode.

By Idowu Ayoola

The reason for participating in the Sense Your Heart module was to gain knowledge in physiological sensing like the heart beat. Along the curse of this study, I leant certain techniques and theories for building electrical circuitries and I understood the way the human body works and how the heart is influenced. Finally, I experimented in calculating indicators like heart-rate-variability and observed its behavior especially in a game context.

As I understand, the heart does not function linearly however, it is continually regulated by the nervous systems like the Sympathetic or Parasympathetic and it could be influenced by breathing. This is one of the reason why it was experientially impossible to directly map heart measurements to controls within a game application. In as much as we can observe emotional indications by sensing the heart, it is not a robust way of detecting actual emotions of humans due to the frequent rise and fall signals.

The principle of Photoplethysmography (PPG) works by change in volume of the blood. When the heart beats, the circulation of blood aggregates forcing a change in volume. A red or infra red light source is placed opposite to a photo transistor and the change in light transmission is monitored as the volume of blood changes. A similar sensor can be built by placing the components side by side to measure the reflection of light. I learnt to build such a system to work with an “Arduino” as a sampling hardware. These samples are then sent to “Processing” for advanced processing or application.

I built a PPG sensor system with low and high pass filters to eliminate noise from signal. LM234, an integrated circuit was

used to amplify the signal into a readable range of 0 - 5 volts for optimal analog to digital conversion in Arduino. Eventually I've spend more time in debugging the electronics than the time in coupling the component together. I learnt few tricks to be more effective next time; I would build the circuit from left to right to enable me to debug each section built and not to wait and hope it works after all components are coupled.

8 | sources

1. Game example used for inspiration for the program; http://www.local-guru.net/blog/2009/06/19/processing-tutoria
2. All the provided programs delivered with this module. (see appendices)

9 | appendices

I Reflections
II Arduino data sampler
III RR heartbeat game
IV RR Parser
V RR port
VI Blood cells
VII Music
VIII Players

Arduino data sampler

```
// This program can either sample an analog signal on "Analog-
Input_Pin, or act as an interval

// timer on pin "Digital-Input_Pin". The selection is made directly after
reset. If the reply on

// the prompt ">" is "I", then the interval mode is selected, otherwise
the A/D mode.

// In both cases the program is interrupt driven:

//   - Interval mode: A hardware interrupt is generated on a low-to-high
transition on digital

//     input pin "Digital-Input_Pin". The interrupt service routine
"Send_Interval()" sends the

//     elapsed time with respect to the previous transition as a
hexadecimal string over the

//     serial bus. This is done with a 4us resolution.

//   - A/D mode: A timer interrupt is generated every 2ms (=500Hz).
The interrupt service

//      routine samples pin "Analog_Input_Pin" and sends the value
over the serial bus as a byte.

//

//                                                Geert Langereis, february
2010


// Edited 17th, March 2011.

// After the initial prompt of '>', the responses from processing are
depending of the requested data in:

// 'D' initialize arduino for digital sampling from external interrupt 0 and
1.

// 'A' initialize ardino  for analoge sampling with a set ISR

// 'I' initialize arduino as in 'D' and 'A'.

// Arduino sends comma seperated values (CSV) through the serial
bus to processing.

// For digital sampling, element sent are in the order of PIN_TYPE,
PIN_ID, VALUE.

// For Analoge sampling, element sent are in the order of PIN_TYPE,
VALUE1, VALUE2.


const int Baudrate = 19200;
const int Analog_Input_Pin = 0;      //Analog pin for internal ISR ;
const int Analog_Input_Pin2 = 1;
const int Digital_Input_Pin1 = 2;    // Digital pin for external interupt 1;
const int Digital_Input_Pin2 = 3;    // Digital pin for external interupt 2;

void setup()
{
  Serial.begin(Baudrate);          // For sending data to the computer
over USB

  Serial.print(">");

  while (Serial.available()==0);

  byte response = Serial.read();

  if (response == 'D' || response == 'I')
  {

    pinMode(Digital_Input_Pin1, INPUT);        // Set pin direction
    pinMode(Digital_Input_Pin2, INPUT);
```

```
    attachInterrupt(0,Send_Interval1, RISING); // Attach to interupt to
pin 2
    attachInterrupt(1,Send_Interval2, RISING); // Attach to interrupt to
pin 3

  }

  if (response == 'A' || response == 'I')
  {

    // The following settings are for a Duemillanove with a 16Mhz
ATMega328

    cli();                  // disable interrupts while messing with their
settings

    TCCR1A = 0x00;          // clear default timer settings, this kills the
millis() function

    TCCR1B = 0x00;          // timer in normal mode

    TCCR1B |= (1 << WGM12);   // Configure timer 1 for CTC mode

    TCCR1B |= (0 << CS12);    // Set timer prescaling by setting 3 bits

    TCCR1B |= (1 << CS11);    // 001=1; 010=8, 100=256, 101=1024

    TCCR1B |= (1 << CS10);

    TIMSK1 |= (1 << OCIE1A);  // Enable CTC interrupt with OCF1A flag
in TIFR1

    OCR1A = 124;            // Turn interrupts back on

    sei();                  // Enable interrupts

  }

}
```

```
void loop() {

  // nothing to do, its all in the interrupt handlers!

}


unsigned long LastTime, NewTime, j;
unsigned long LastTime2, NewTime2;

void Send_Interval1()

{

  NewTime = micros();       // Resolution 4us, only one overrun (=error
in 70 minutes)

  Serial.print("D");       // Encode PinType to differenciate between
digital and analogue sampling.
  Serial.print(",");
  Serial.print(0);          // ID for interrupt 0
  Serial.print(",");
  Serial.println(NewTime-LastTime, HEX);    // Encode value in HEX
figure.

  LastTime = NewTime;         // update lastTime

}
void Send_Interval2()

{

  NewTime2 = micros();       // Resolution 4us, only one overrun (=error
in 70 minutes)

  Serial.print("D");          // Encode PinType to differenciate between
digital and analogue sampling.
  Serial.print(",");
  Serial.print(1);            // ID for interrupt 1
  Serial.print(",");
```

```
  Serial.println(NewTime2-LastTime2, HEX);    // Encode value in HEX
figure.

  LastTime2 = NewTime2;          // update lastTime

}

ISR(TIMER1_COMPA_vect)       // when timer counts down it fires this
interrupt for A/D-mode

{

  int val = analogRead(Analog_Input_Pin);      // Sample analog Pin
  int val2 = analogRead(Analog_Input_Pin2);

  Serial.print("A");            // Encode PinType to differenciate between
digital and analogue sampling.
  Serial.print(",");
  Serial.print( (val >> 2));      // Push value1
  Serial.print(",");
  Serial.println( (val2 >> 2));    // push value2

}
```

RR heart beat game

```
import processing.serial.*;
import ddf.minim.*;

RRport myport;
RRparser myparser;
boolean settedup = false;

AudioPlayer player;
Minim minim;

Music music;
Player1 player1;
Player1 player2;

Bloodcells bloodcells;

PImage bg;
PImage go;
PImage player1wins;
PImage player2wins;
PFont fontA;

int beamx = 1050;

void setup() {
  size(1024,800);
  bloodcells = new Bloodcells();

  // Load external images //

  PImage simg = loadImage( "player2.png" );
  player1 = new Player1( 100, height/2, 3, simg );

  PImage simg2 = loadImage( "player1.png" );
  player2 = new Player1( 100, height/2, 3, simg2 );

  bg = loadImage("background.png");
  go = loadImage("gameover.png");
  player1wins = loadImage("player1wins.png");
  player2wins = loadImage("player2wins.png");

  // Load external font //
  fontA = loadFont("KozGoPro-ExtraLight-48.vlw");
  textFont(fontA, 32);

  // Load sound using Minim library //
  minim = new Minim(this);
  player = minim.loadFile("BD.mp3", 2048);
  music = new Music();


  frameRate( 25 );
  smooth();
  noStroke();

  //INITIATE STUFF FOR ARDUINO
  myport = new RRport(this);
  myparser = new RRparser();

  settedup=true;  //flag setup completed
}

void draw() {
  background(bg);

  int newY1;
  int newY2;

  if (RR[0] > pRR[0] + 20) player1.down();
  else if (RR[0] < pRR[0] - 20) player1.up();

  if (RR[1] > pRR[1] + 20) player2.down();
  else if (RR[1] < pRR[1] - 20) player2.up();


  player1.draw();   // draw player1
  player2.draw();  // draw player2
  bloodcells.draw();  // draw bloodcells
```

```
  fill(136,12,85);
  text("HR " + HR[0], 900, 50);
  fill(0,120,252);
  text("HR " + HR[1], 900, 110);
}

// void for manually control players //

void keyPressed() {
  if ( keyCode == UP ) {
    player1.up();
  }
  else if ( keyCode == DOWN ) {
    player1.down();
  }
  if ( keyPressed == true) {
    if ( key == 'w' ) {
      player1.up();
    }
    else if ( key == 's' ) {
      player1.down();
    }
  }
}



int [] RRavg = {
  1200, 1200
};
int [] RRstd = {
  50, 50
};
int [] HR = {
  60, 60
};
int [] RR = {
  0, 0
};
int [] pRR = {
  0, 0
};

void serialEvent(Serial p) {
  if (settedup) {
    //print("!");
    //zat allemaal eerst in draw
    myport.step();
    myparser.step();
    //if (time/10 > 120) stop();
    for (int id = 0; id < 2; id++) {
      if (myparser.event(id)) {
        pRR[id] = RR[id];
        RR[id] = myparser.val[id];
        print(" RR"+id+"=");
        print(RR[id]);
        //output.println(RR);
        RRavg[id] = int((23 * RRavg[id] + 1*RR[id])/24);
        //
        print(" AVG"+id+"=");
        print(RRavg[id]);
        RRstd[id] = int((15.0 * RRstd[id] + abs(RR[id] - RRavg[id])) / 16.0);
        //
        print(" STD"+id+"=");
        print(RRstd[id]);//niet met kwadraat ivm outlyers
        //
        print(" HR"+id+"=");
        print(HR[id]);
        HR[id] = int(60000 / (((60000 / HR[id]) + RR[id])*0.5));
        //
        print(" Beats=");
        //
        println();
        //
      }
      println();
    }
  }
}
```

```
public void stop() {
  println("THANK YOU, GOODBYE");

  super.stop();
  exit(); // Stops the program
}
```

RR Parser

```
class RRparser {
  public int [] val = new int [myport.numberOfSensors];
  private int avg = 1000;
  private int tmp=0;
  private boolean []flag = new boolean[myport.numberOfSensors];


  RRparser() {
    for (int i = 0; i < myport.numberOfSensors; i++) {
      val[i] = 0;
      flag[i] = false;
    }
  }

  boolean event(int id) {
    if (flag[id]) {
      flag[id] = false;
      return true;
    }
    else return false;
  }

  void step() {
    int b;
    //print('.');
    for (int id = 0; id < myport.numberOfSensors; id++)
    {
      while (myport.bufcnt[id]>0) {
        b=myport.getbuf(id);
        //print(char(b));
        //if (b==13) println("CR");//TZT WEGECOOMENTEREN
        //if (b==10) println("LF");//TZT  ERUITCOMMENT
        if ((tmp>0)&&(b==10||b==13)) {
          flag[id] = true;
          val[id] = tmp/1000;//ms ipv us
          if (val[id]>600 && val[id] < 1600) avg = 9*(avg/10) + val[id]/10;   //
fake feedback
          if (val[id] < 0.5*avg) {
            print(" adjusting sensor ");
            println(id);
            //output.println(0);
            flag[id] = false;
          }
          if (val[id] > 1.5*avg) {
            print(" ADJUSTING SENSOR ");
            println(id);
            //output.println(00);
            flag[id] = false;
          }
        }
        if ((b==10||b==13)) tmp=0;
        if (b!=10&&b!=13)   tmp=16*tmp + hex2int(b); //msb first
      }
    }
  }

  private int hex2int(int c) {
    if (c=='0') return 0;
    if (c=='1') return 1;
    if (c=='2') return 2;
    if (c=='3') return 3;
    if (c=='4') return 4;
    if (c=='5') return 5;
    if (c=='6') return 6;
    if (c=='7') return 7;
    if (c=='8') return 8;
    if (c=='9') return 9;
    if (c=='A') return 10;
    if (c=='B') return 11;
    if (c=='C') return 12;
    if (c=='D') return 13;
    if (c=='E') return 14;
    if (c=='F') return 15;
    println("!@#$%"); //TZT ERUIT
    return -1;
  }
}// RRparser
```

RR Port

```
class RRport
{
  private Serial myPort;
  final int numberOfSensors = 2;
  public int[][] buffer;
  public int[] bufcnt;

  private int linefeed = 10;

  RRport(PApplet parent) {
    String portName = Serial.list()[0];
    //com7, arduino, lower usb right (loe's laptop)
    //of bovenste, front, loe's pc hg3.53
    println(Serial.list());
    myPort = new Serial(parent, portName, 19200);
    myPort.bufferUntil(this.linefeed); //buffer until linefeed --> Linefeed
in ASCII is 10

    while (myPort.available()==0) {
    }
    if (myPort.read()=='>') {
      myPort.write('D');
      println(">I");
    }
    else println("ARDUINO NON > PROMPT ERROR");

    buffer = new int[numberOfSensors][100];
    bufcnt = new int[numberOfSensors];
    for (int p = 0; p < numberOfSensors; p++) {
      bufcnt[p] = 0;
    }
  }

  void putbuf(int id, int c) {
    if(bufcnt[id]  >= 100-1) println("OVERFLOW buffer ERROR");
    buffer[id][bufcnt[id]++] = c;
  }

int  getbuf(int id) {
  if (bufcnt[id] <= 0) println("UNDERFLOW buffer ERROR");

  int v = buffer[id][0];

  //works like FIFO, shift rest again
  int i = 0;
  while (i < bufcnt[id]-1) {
    buffer[id][i] = buffer[id][i+1];
    i++;
  }     //alternative: cyclic buffer, not now

  bufcnt[id]--;
  return v;
  //return 0;
}

void step()
{
  //first value is an ID
  //second value is the number of millis
  String [] inByte;

  // read the serial buffer:
  String myString = this.myPort.readStringUntil(linefeed);

  // if you got any bytes other than the linefeed:
  if (myString != null) {

    myString = trim(myString);

    // split the string at the commas
    // and convert the sections into integers:

    inByte = split(myString, ',');

    // print out the values you got:
```

```
      //      for (int sensorNum = 0; sensorNum < numberOfSensors;
sensorNum++) {
      //        print("Value " + sensorNum + ": " + inByte[sensorNum] +
"\t");
      //      }

      // add a linefeed after all the sensor values are printed:
      //      println();


      //DO SOMETHING WITH THE VALUE
      String type = inByte[0];

      if (type.equals("D")) {
        int id = int (inByte[1]);
        String val = inByte[2];

        for (int l=0; l < val.length(); l++) {
          if (id >= 0 && id < numberOfSensors) putbuf(id, val.charAt(l));
          else println("Bad ID");
        }

        putbuf(id, 13); // push CR to buffer to mark end of data;
      }

      else if(type.equals("A")) {
        int s1 = int (inByte[1]);
        int s2 = int (inByte[2]);
        print("s1 " + s1 + " :  s2 " + s2 + "\t");
      }
    }
  }
} //     RRport
```

```
public class Bloodcells {

  int random_y;

  int movespeed = 2; //determines the speed with which cells move from right to left, substracting movespeed from their x-position with every move.
  int speedscore = 0; // keeps track of the speedscore, every time a blood cell has passed by, speedscore is raised by 1
  int speedscoreincreaselevel = 3; // this value determines after how many passings of bloodcells the movespeed is increased, in this case after speedscore has reached 3.

  int viewscore = 11; // represents the displayed score for player 1, now starting at 11 but any desired starting value could be set.
  int viewscore2 = 11; // represents the displayed score for player 2

  // the opacity settings below are used to create a 'blinking' visual of the score when a player gets to score 10 or lower to warn them of an approaching 'game over' scenario.
  int opacity = 255;
  int opacity2 = 255;
  boolean opacitydecrease = true;
  boolean opacitydecrease2 = true;

  boolean fieldfree = true; // this boolean is set to true once a bloodcell has passed from right to left, allowing a new bloodcell to appear.
  boolean hit; // set to true if either player 1 or player 2 catches the bloodcell

  public Bloodcells() {
  }

  void draw() {
    drawscore();
    playfield();
    compare();
  }

  void playfield() {
    // check whether the previous bloodcell is out of the screen, if so spawn a new one at a random Y-location.
    if(fieldfree == true) {

      float rnumber = random(100,750);
      random_y = int(rnumber);

      fieldfree = false;
    }

    bloodcell();
    moverightleft();
  }

  // creation/insert image of bloodcell and positioning
  void bloodcell() {

    PImage cell;
    cell = loadImage("cell.png");
    image(cell,cell_x-15,random_y-10,65,65);
  }

  // move bloodcell right to left //
  void moverightleft() {

    cell_x = cell_x - movespeed;

    if(cell_x <= 0) // if the bloodcell passes both players ( x coordinate <= 0 ), -1 score 1 and 2, cell_x is set to starting position (the width of the screen)
    {
      cell_x = width;
      fieldfree = true;
      hit = false;
      score();
      score2();
    }
  }
```

```
  // calculates 'collision' between bloodcell and player1/player2
  void compare() {

    // collision player1
    if ((cell_x - player1.x < player1.playerwidth) &&
      (cell_x - player1.x > - 30) &&
      (random_y - player1.y < player1.playerheight) &&
      (random_y - player1.y > -55) ) {

      cell_x = width;
      hit = true;
      music.play(); // triggers playing of heartbeat sound (or any other
sound depending on the coupled sound-file, see 'music')
      fieldfree = true;
      score();
    }

    // collision player 2
    if ((cell_x - player2.x < player2.playerwidth) &&
      (cell_x - player2.x > - 30) &&
      (random_y - player2.y < player2.playerheight) &&
      (random_y - player2.y > -55) ) {

      cell_x = width;
      hit = true;
      music.play();
      fieldfree = true;
      score2();
    }
  }

  // here the score is increased and/or decreased depending on hit
true/false
  // 'score' targets player 1 and 'score2' targets player 2
  void score() {
    if (hit == true) {
      if (viewscore < 100) {
        viewscore++;
        speedscore++;
        viewscore2--; // when player 1 hits the bloodcell this means
player 2 has missed it and so 1 is substracted from player 2's score.
      }
    }
    if (hit == false) {
      viewscore = viewscore - 1;
      if(viewscore <= 0) {
        gameover();  // a score below 0 triggers the game-over screen
      }
    }

    // increase speed after interval speedscoreincreaselevel, standard
value is 3 so speed increases after 3 hits
    if (speedscore == speedscoreincreaselevel ) {
      movespeed = movespeed + 1;
      speedscore = 0;
    }
  }

  // see explanation above for 'void score'
  void score2() {
    if (hit == true) {
      if (viewscore2 < 100) {
        viewscore2++;
        speedscore++;
        viewscore--;
      }
    }

    if (hit == false) {
      viewscore2--;
      if(viewscore2 <= 0) {
        gameover();
      }
    }

    if (speedscore == speedscoreincreaselevel ) {
      movespeed = movespeed + 2;
      speedscore = 0;
    }
```

```
}


// draws the score to be vissible for the players
void drawscore() {

  textSize(25);
  // mechanism for pulsating score signal if the score gets below 10
  if(viewscore <= 10)
  {
    if(opacity > 75 && opacitydecrease == true) {
      opacity = opacity - 10;
    }
    else {
      opacitydecrease = false;
      opacity = opacity + 10;
      if(opacity > 254) {
        opacitydecrease = true;
      }
    }

    fill(136,12,85,opacity);
    text("PLAYER ONE " + viewscore + "%",25,50);
    rect(25,55,viewscore*5,25);
  }
  else {
    fill(136,12,85);
    text("PLAYER ONE " + viewscore + "%",25,50);
    rect(25,55,viewscore*5,25);
  }

  textSize(25);
  // mechanism for pulsating score2 signal if the score2 gets below
10
  if(viewscore2 <= 10)
  {
    if(opacity2 > 75 && opacitydecrease2 == true) {
      opacity2 = opacity2 - 10;
    }
    else {
      opacitydecrease2 = false;
      opacity2 = opacity2 + 10;
      if(opacity2 > 254) {
        opacitydecrease2 = true;
      }
    }


    fill(0,120,252,opacity2);
    text("PLAYER TWO " + viewscore2 + "%",25,110);
    fill(119,184,opacity2);
    rect(25,115,viewscore2*5,25);
  }
  else {
    fill(0,120,252);
    text("PLAYER TWO " + viewscore2 + "%",25,110);
    fill(119,184,255);
    rect(25,115,viewscore2*5,25);
  }
 }



 void gameover() {
   // triggered when one of the players goes below '0' and loads the
correct screen

   if(viewscore<=0) {
     background(player2wins);
     noLoop();
   }
   else if(viewscore2<=0) {
     background(player1wins);
     noLoop();
   }
 }
}
```

Music

```
public class Music {

  public Music() {
  }
  void play()
  {
    player.play();
    player.rewind();
  }
}
```

Players

```
public class Player1 {
  PImage img;
  int speed;
  int x;
  int y;
int playerwidth = 82; //x
int playerheight =125; //y


  public Player1( int x, int y, int speed, PImage img ) {
    this.speed = speed;
    this.x = x;
    this.y = y;
    this.img = img;
  }

  void draw() {
    fill(255,255,0,0);
    rect (x,y,playerwidth,playerheight);


    pushMatrix();
    translate ( x+15, y+25 );
    image( img, -img.width/2, -img.height/2,100,150 );
    popMatrix();
  }

  void up() {
    y -= speed;
    if (y <= 0) y = 0;
  }

  void down() {
    y += speed;
    if (y>=480) y = 480;
  }

  public void up(int yPos) {
    y = yPos;
  }
  public int getObjH() {
    return img.height;
  }
}
```


TU/e
Technische Universiteit
Eindhoven
University of Technology