

**Philosophische** Fakultät III

Sprach- , Literatur- und Kulturwissenschaften

Institut für Information und Medien, Sprache und Kultur (I:IMSK)  
Lehrstuhl für Medieninformatik

Sketching with Hardware

Modul: MEI\_M 10.3

SS 2017

Leitung: Raphael Wimmer

**Dokumentation Projektarbeit – Tangible Games**

Nina Hösl

Matr.-Nr.: 1827558

3. Semester B.A. Medieninformatik / Informationswissenschaft

E-Mail: nina.hoesl@stud.uni-regensburg.de

Christian Winkler

Matr.-Nr.: 1827558

6. Semester B.A. Medieninformatik / Informationswissenschaft

E-Mail: @stud.uni-regensburg.de

Abgegeben am [Abgabetermin der Arbeit]

Inhalt

[1 Konzept 2](#_Toc490635435)

[2 Schaltung 4](#_Toc490635436)

[3 Komponenten 4](#_Toc490635437)

[3.1.1 Code 4](#_Toc490635438)

Konzept (Idee und Nutzung klar beschrieben?)

Schaltung (klar dokumentiert?)

Komponenten (genügend Details?)

Optik (schöne Bilder, korrekte Formatierung)

Rechtschreibung

Stil (sachlich, Quellenangaben, kein Nominalstil)

Reproduzierbarkeit (alle Details zum Nachbau / zur

Inbetriebnahme dokumentiert?)

# Konzept

Für das Projekt soll ein Stofftier angelehnt an das Tamagotchi-Prinzip umgebaut und erweitert werden, so dass ein berührbares Spieleerlebnis mit verschiedene Interaktions- und Feedbackmöglichkeiten möglich ist.

Abhängig von den Werten Health, Hunger, Happiness und Energy gibt das Stofftier aktustisches und haptisches Feedback an den Nutzer, um anzuzeigen, sobald einer der Werte ein kritisches Maß unterschreitet. Der Nutzer hat verschiedene Möglichkeiten, die Werte wieder aufzuladen (siehe Tabelle 1).

Der Wert Health soll dabei ein übergeordneter Wert sein, der von den anderen drei Werten abhängt: sinkt einer der Werte stark ab, sinkt auch der Health-Wert und lässt sich auch nur wieder aufladen, indem der gesunkene Wert angehoben wird. Der Health-Wert soll den allgemeinen Gesundheitszustand wiedergeben.

Um das Stofftier bei niedrigem Hunger-Wert zu füttern, wird ein mitgelieferter spezieller Löffel an den Mund geführt. Jeder Löffel hebt den Wert um einen festen Wert an und ist wiederholbar, bis der Hunger-Wert wieder sein Maximum erreicht hat.

Der Happiness-Wert wird durch Knuddeln, also fstes Drücken des Stofftieres, aufgeladen.

Der Energy-Wert sinkt durch den wachen Zustand des Stofftieres. Wach ist das Tier, wenn es sich in einer hellen Umgebung aufhält. Man kann es schlafen legen, indem man es für eine längere Zeitdauer in einer dunklen Umgebung belässt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribut** | **Entladen durch** | **Aufladen durch** | **Feedback bei niedrigem Wert** |
| Health |  |  | Weinen |
| Hunger | Vergangene Zeit | Füttern | Magenknurren  Weinen |
| Happiness | Vergangene Zeit ohne Aktionen | Knuddeln | Traurige Augen  Weinen |
| Energy | Vergangene wache Zeit | Schlafen | Müde Augen  Weinen |

Tabelle 1

Zusätzlich zu den hapitschen und akustischen Feedbackmöglichkeiten hat der Nutzer die Möglichkeit, die vier Zustände visuell abzufragen. Jedem Wert ist eine Pfote des Stofftieres zugeordnet, welche gedrückt werden kann. Auf den Druck folgt ein Leuchtstreifen auf dem Rücken des Stofftieres, der in verschiedenen Farben den Ladezustand des jeweiligen Wertes anzeigt. Dies ist besonders bei Zukunftsplanungen wichtig, da es z.B. ungünstig ist, das Tier schlafen zu legen, obwohl es demnächst hungrig werden könnte.

Zusätzlich zu den Basisfeatures soll das Tier verschiedene Entwicklungsstufen durchlaufen. Die Entwicklung beginnt im Ei, welches ausgebrütet werden muss. Wurde diese Phase erfolgreich absolviert, schlüpft das Stofftier als Säugling aus dem Ei und muss entsprechend gepflegt werden, um zu einem Erwachsenen heranzureifen. Der Zustand des Erwachsenwerdens wird durch zusätzlich leuchtende Körperteile symbolisiert.

Das Ausbrüten des Eis erfolgt durch eine hohe Wärmezufuhr über einen längeren Zeitraum hinweg. Der Nutzer packt das Stofftier selber aus dem Ei aus, sobald das Stofftier soweit ist. Packt der Nutzer es zu früh aus, verstirbt das Stofftier automatisch.

Um das Stofftier während der Baby-Zeit zu pflegen, werden zum Füttern spezielle Milchflaschen benötigt. Die Ent- und Aufladung er einzelnen Werte erfolgt schneller, weshalb ein erhöhter Pflegeaufwand mit kürzeren Zeitabständen der Interaktionen notwendig ist.

Als Erwachsener ist das Tier leichter zu pflegen, da die einzelnen Werte langsamer absinken und kein hoher Betreuungsaufwand mehr nötig ist, da die Abstände zwischen den nötigen Betreuungsaktionen steigen.

# Komponenten

Je nach benötigter Interaktions- oder Feedbackmethode wurden unterschiedliche Sensoren und Aktoren eingebaut (siehe Tabelle 2).

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribut** | **Feedback** | **Eingabe** | **Zustandsabfrage** |
| Health | Lautsprecher  LED-Matrix | (Temperatursensor) | Drucksensor  LED-Streifen |
| Hunger | Lautsprecher  Vibrationsmotor | RFID-Sensor | Drucksensor  LED-Streifen |
| Happiness | Lautsprecher  LED-Matrix | Drucksensoren | Drucksensor  LED-Streifen |
| Energy | Lautsprecher  LED-Matrix | Photosensor | Drucksensor  LED-Streifen |

Tabelle 2

Der allgemeine Gesundheitszustand ist abhängig von allen anderen drei Werten und bildet einen Mittelwert aus diesen. Sinkt allerdings einer der anderen Werte unter 50%, wird der kleinste Wert stärker gewichtet, sinkt einer der drei Werte unter 30%, sinkt der Gesundheitswert auf den gleichen Wert und mit dem gleichen Tempo weiter bis auf null, wenn keine Interaktion des Nutzers erfolgt. Bei sinkendem Gesundheitszustand wird über den Lautsprecher ein Weinen ausgegeben. Zudem nehmen die Augen zunehmen einen traurigen Ausdruck an. An der Brust des Tieres zeigt eine LED-Matrix ein Herz, das mit sinkender Gesundheit immer kleiner wird und beim Tod des Tieres ausgeht. Der Nutzer kann den genauen Wert der Gesundheit über einen Drucksensor in den Pfoten abfragen und an einem LED-Streifen im Rücken des Tieres ablesen. Dies ist auch für alle anderen drei Werte möglich. Der LED-Streien zeigt unterschiedliche Werte in unterschiedlichen Farben.

Der Hungerzustand wird bei sinkendem Niveau durch Lautsprecherausgaben mit Weinen und durch einen Vibrationsmotor, der einen knurrenden Magen simulieren soll, dargestellt. Der Nutzer kann den Hunerwert auffüllen, indem er einen Löffel (im Zustand des Säuglings eine Flasche), in dem ein RFID-Chip verbaut ist, dem Maul annähert, in dem sich der Sensor befindet. Als zusätzliches Feedback wird über den Lautsprecher Schmatzen ausgegeben.

Der Zustand Happiness soll das Wohlempfinden des Tieres darstellen und gibt bei sinkendem Niveau ebenfalls Feedback über den Lautsprecher. Die LED-Matrizen, die die Augen darstellen, zeigen einen traurigen Ausdruck. Die Anzeige kann aufgefüllt werden, indem das Tier „geknuddelt“ wird. Dafür wird das Tier fest gedrückt, was durch Drucksensoren im Bauch registriert wird. Beim Knuddeln gibt das Tier über die Lautsprecher Lachen aus.

Wie bei allen anderen Zuständen wird bei niedrigem Energy-Wert ein Weinen über den Lautsprecher ausgegeben. Die LED-Matrizen an den Augen zeigen müde, also immer kleinere Augen, die immer wieder zufallen. Um den Wert der Energy zu messen wird ein Photosensor verwendet, der misst, wie lange das Stofftier einer hellen Umgebung ausgesetzt ist. Zum Auffüllen muss es sich entsprechend lange in einer dunklen Umgebung aufhalten.

Zusätzlich zu den einzelnen Gesundheitswerten befindet sich das Stofftier auch noch in unterschiedlichen Entwicklungszuständen, die durch unterschiedliche Sensoren abgefragt und repräsentiert werden (siehe Tabelle 3).

|  |  |  |
| --- | --- | --- |
| **Entwicklungsstufe** | **Stufenanzeige** | **Eingabe zur Zustandsänderung** |
| Ei | Pappmache-Ei | Temperatursensor  Photosensor |
| Säugling |  |  |
| Erwachsen | LED-Streifen |  |

Tabelle 3

Die einzelnen Entwicklungsstufen werden allerdings nicht nur durch einzelne Sensoren dargestellt, sondern auch durch reale Bestandteile. Der Zustand des Eis wird durch ein Ei aus Pappmache, in dem sich das Stofftier befindet, dargestellt. Das Ausbrüten des Eis wird durch einen Temperatursensor überwacht. Wird das Ei vor dem vollendeten Ausbrüten geöffnet, reagiert ein eingebauter Photosensor darauf und sorgt dafür, dass das Stofftier sofort stirbt.

Die Entwicklungsstufe des Säuglings hat keine zusätzliche Anzeigefunktion, das Tier wird im Normalzustand angezeigt und fordert die normalen Interaktionenoptionen ein. Allerdings werden besonders bei Interaktion mit RFID-Chip andere Gegenstände angefordert als später im erwachsenen Zustand.

Sobald das Tier eine gewisse Zeit im Zustand des Säuglings verbracht hat und ausreichend gut versorgt wurde, wird es erwachsen. Dieser Zustand wird durch einen LED-Streifen im Horn angezeigt, der mit dem Erwachsenwerden anfängt zu leuchten.

# Schaltung

### Für die Schaltung wurden sämtliche Sensoren und Aktoren um den Arduino angeordnet und in korrekter Schaltung verbaut.

### C:\Users\ninah\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Steckplan_Steckplatine.jpgC:\Users\ninah\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Steckplan_Schaltplan.jpg

# Code

/\*

\*\*\*\*\*\*\*\*\*\* TAMAGUINO \*\*\*\*\*\*\*\*\*\*\*

Tamagotchi clone for Arduino

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "Arduino.h"

#include "SoftwareSerial.h"

#include <Wire.h>

#include <Adafruit\_NeoPixel.h>

#include <Adafruit\_GFX.h>

#include "Adafruit\_LEDBackpack.h"

#include "DFRobotDFPlayerMini.h"

#include "anim.h"

#ifdef \_\_AVR\_\_

#include <avr/power.h>

#endif

/\* ------- DFPLAYER MINI ------- \*/

SoftwareSerial dfPlayerSerial(10, 11); //RX, TX

DFRobotDFPlayerMini dfPlayer;

// Sound IDs

int soundStart = 1;

int soundCry = 2;

int soundLaugh = 3;

int soundEat = 4;

int soundBurp = 5;

int soundSnore = 6;

int soundLost = 7;

int soundGameOver = 8;

int soundNeigh = 9;

int soundGasp = 10;

/\* ------- LED STRIPS ------- \*/

Adafruit\_NeoPixel backStrip = Adafruit\_NeoPixel(5, 7, NEO\_GRB + NEO\_KHZ800);

Adafruit\_NeoPixel hornStrip = Adafruit\_NeoPixel(11, 8, NEO\_GRB + NEO\_KHZ800);

/\* ------- LED MATRICES ------- \*/

Adafruit\_8x8matrix leftEye = Adafruit\_8x8matrix();

Adafruit\_8x8matrix rightEye = Adafruit\_8x8matrix();

Adafruit\_8x8matrix heart = Adafruit\_8x8matrix();

int modeLength = 0;

int blinkDelay = 100;

int blinkChance = 10;

int blinkChanceResult;

int animSelection = 0;

/\* ------- FORCE SENSITIVE RESISTORS ------- \*/

int fsrFrontRight = A0; // front right (A0)

int fsrFrontLeft = A1; // front left (A1)

int fsrBackRight = A2; // back right (A2)

int fsrBackLeft = A3; // back left (A3)

/\* ------- RFID TAGS ------- \*/

int tag\_yellow[14] = {2, 48, 56, 48, 48, 48, 65, 56, 57, 55, 51, 70, 56, 3};

int tag\_red[14] = {2, 48, 55, 48, 48, 69, 51, 52, 48, 49, 67, 66, 56, 3};

int newtag[14] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // used for read comparisons

int rfidData = 0;

int rfidCheck = -1;

/\* ------- VIBRATION MOTOR ------- \*/

int vibrationMotorPin = 6;

/\* ------- COPPER PIN ------- \*/

int copperPin = 4;

/\* ------- PHOTOCELL ------- \*/

int photocellPin = A4;

/\* ------- PET STATS ------- \*/

float scalingFactor = 10;

float hunger = 100;

float happiness = 100;

float energy = 100;

float health = 100;

float age = 0;

bool dead = false;

bool sleeping = false;

static const unsigned long REFRESH\_INTERVAL\_STATS = 50; // 0.05s

static const unsigned long REFRESH\_INTERVAL\_LED = 5000; // 5s

static const unsigned long REFRESH\_INTERVAL\_SOUND = 20000; // 20s

static const unsigned long REFRESH\_INTERVAL\_VIBRATION = 1500; // 1.5s

static const unsigned long REFRESH\_INTERVAL\_DEBUG = 10000; // 10s

static unsigned long previousMillisStats = 0;

static unsigned long previousMillisLed = 0;

static unsigned long previousMillisSound = 0;

static unsigned long previousMillisVibration = 0;

static unsigned long previousMillisDebug = 0;

unsigned long currentMillis;

boolean resetBackLed = false;

void setup() {

// Initialize DFPlayer Mini

dfPlayerSerial.begin(9600);

dfPlayer.begin(dfPlayerSerial);

Serial1.begin(9600);

Serial.begin(9600);

// Initialize all pixel strips to 'off'

backStrip.begin();

backStrip.show();

hornStrip.begin();

hornStrip.show();

// Set pins

pinMode(copperPin, INPUT);

pinMode(vibrationMotorPin, OUTPUT);

// Pass the led-matrix addresses

leftEye.begin(0x70);

rightEye.begin(0x71);

heart.begin(0x72);

// Set volume value (0 - 30)

dfPlayer.volume(25);

dfPlayer.play(soundStart);

}

void loop() {

currentMillis = millis();

if (currentMillis - previousMillisStats >= REFRESH\_INTERVAL\_STATS) {

previousMillisStats = currentMillis;

/\* ------- MODIFY PET STATS ------- \*/

if (!dead) {

if (sleeping) {

hunger -= 0.0005 \* scalingFactor;

if (happiness - 0.001 > 0) {

happiness -= 0.001 \* scalingFactor;

}

energy += 0.1 \* scalingFactor;

} else {

hunger -= 0.0025 \* scalingFactor;

if (happiness - 0.002 > 0) {

happiness -= 0.002 \* scalingFactor;

}

energy -= 0.001 \* scalingFactor;

}

age += 0.00025 \* scalingFactor;

// Weighting the pet stats

if (hunger <= 30 || happiness <= 30 || energy <= 30) {

health = min(hunger, min(happiness, energy));

} else {

health = (hunger + happiness + energy) / 3;

}

}

}

if (currentMillis - previousMillisDebug >= REFRESH\_INTERVAL\_DEBUG) {

previousMillisDebug = currentMillis;

debugLog();

}

// STATUS: threshold exceeded

if ((hunger > 29.99975 && hunger < 30.00025) ||

(happiness > 29.9998 && happiness < 30.0002) ||

(health > 28 && health < 30) ||

(energy > 29.9998 && energy < 30.0002)) {

Serial.println("YOUR PET IS CRYING");

dfPlayer.play(soundCry);

}

// STATUS: good

if (hunger > 20 && happiness > 20 && health > 20 && energy > 20) {

animationTwinkle();

// Play random sound

if (currentMillis - previousMillisSound >= REFRESH\_INTERVAL\_SOUND) {

previousMillisSound = currentMillis;

playRandomSound();

}

}

// STATUS: critical

if (hunger <= 20 || happiness <= 20 || health <= 20 || energy <= 20) {

Serial.println("CRITICAL STATUS");

animationAngry();

}

//STATUS: hungry

if (hunger <= 20) {

analogWrite(vibrationMotorPin, 110);

if (currentMillis - previousMillisVibration >= REFRESH\_INTERVAL\_VIBRATION) {

previousMillisVibration = currentMillis;

analogWrite(vibrationMotorPin, 0);

}

} else {

analogWrite(vibrationMotorPin, 0);

}

// STATUS: dead

if (hunger <= 0 || health <= 0 || happiness <= 0 || energy <= 0) {

dead = true;

animationDead();

animationSad();

dfPlayer.play(soundLost);

delay(10000);

softReset();

}

// Cuddling

if (digitalRead(copperPin) == 1) {

happiness += 5;

dfPlayer.play(soundLaugh);

}

// Check if the pet is sleeping

if (analogRead(photocellPin) <= 40) {

sleeping = true;

} else {

sleeping = false;

}

if (sleeping) {

animationSleepy();

dfPlayer.play(soundSnore);

delay(2000);

}

// Check hitting on paws

if (analogRead(fsrFrontRight) > 500) {

for (uint16\_t i = 0; i < getPetStatus(health); i++) {

backStrip.setPixelColor(i, 255, 0, 0);

backStrip.show();

resetBackStrip();

}

}

if (analogRead(fsrFrontLeft) > 500) {

for (uint16\_t i = 0; i < getPetStatus(happiness); i++) {

backStrip.setPixelColor(i, 255, 255, 0);

backStrip.show();

resetBackStrip();

}

}

if (analogRead(fsrBackRight) > 500) {

for (uint16\_t i = 0; i < getPetStatus(energy); i++) {

backStrip.setPixelColor(i, 0, 0, 255);

backStrip.show();

resetBackStrip();

}

}

if (analogRead(fsrBackLeft) > 500) {

for (uint16\_t i = 0; i < getPetStatus(hunger); i++) {

backStrip.setPixelColor(i, 0, 255, 0);

backStrip.show();

resetBackStrip();

}

}

// Resets the LED strip on the back

if (resetBackLed) {

if (currentMillis - previousMillisLed >= REFRESH\_INTERVAL\_LED) {

resetBackLed = false;

previousMillisLed = currentMillis;

for (uint16\_t i = 0; i < 5; i++) {

backStrip.setPixelColor(i, 0, 0, 0);

backStrip.show();

}

}

}

readTags();

animateHorn();

}

boolean compareTag(int aa[14], int bb[14]) {

boolean ff = false;

int fg = 0;

for (int cc = 0 ; cc < 14 ; cc++) {

if (aa[cc] == bb[cc]) {

fg++;

}

}

if (fg == 14) {

ff = true;

}

return ff;

}

// Compares each tag against the tag just read

void checkTags() {

// this variable helps decision-making,

// if it is 1 we have a match, zero is a read but no match,

// -1 is no read attempt made

rfidCheck = 0;

if (compareTag(newtag, tag\_red) == true || compareTag(newtag, tag\_yellow) == true) {

rfidCheck++;

}

}

void readTags() {

rfidCheck = -1;

if (Serial1.available() > 0) {

delay(100);

for (int z = 0 ; z < 14 ; z++) {

rfidData = Serial1.read();

newtag[z] = rfidData;

}

Serial1.flush(); // stops multiple reads

// do the tags match up?

checkTags();

}

// now do something based on tag type

// > 0 -> match

// = 0 -> no match

if (rfidCheck > 0) {

delay(100);

Serial.println("RFID ACCEPTED");

hunger += 5;

dfPlayer.play(soundEat);

rfidCheck = -1;

}

else if (rfidCheck == 0) {

Serial.println("RFID REJECTED");

rfidCheck = -1;

}

}

float getPetStatus(float value) {

value = map(value, 0, 100, 1, 5);

return value;

}

// Resets the LED Strip

void resetBackStrip() {

if (!resetBackLed) {

resetBackLed = true;

previousMillisLed = currentMillis;

}

}

// Animate the horn with rainbow colors

// Taken from Adafruit NeoPixel Examples (Strandtest)

void animateHorn() {

uint16\_t i, j;

for (j = 0; j < 256 \* 5; j++) {

for (i = 3; i < hornStrip.numPixels(); i++) {

hornStrip.setPixelColor(i, Wheel(((i \* 256 / hornStrip.numPixels()) + j) & 255));

}

hornStrip.show();

}

}

// Input a value 0 to 255 to get a color value.

// The colours are a transition r - g - b - back to r.

uint32\_t Wheel(byte WheelPos) {

WheelPos = 255 - WheelPos;

if (WheelPos < 85) {

return hornStrip.Color(255 - WheelPos \* 3, 0, WheelPos \* 3);

}

if (WheelPos < 170) {

WheelPos -= 85;

return hornStrip.Color(0, WheelPos \* 3, 255 - WheelPos \* 3);

}

WheelPos -= 170;

return hornStrip.Color(WheelPos \* 3, 255 - WheelPos \* 3, 0);

}

void eyeUpdate() {

leftEye.writeDisplay();

rightEye.writeDisplay();

leftEye.clear();

rightEye.clear();

blinkAnim();

animationHeart();

}

// Slight chance of eye blinking

void blinkAnim() {

blinkChanceResult = random(blinkChance);

if (blinkChanceResult > 8) {

leftEye.drawBitmap(0, 0, eyesclosed\_bmp, 8, 8, LED\_ON);

leftEye.writeDisplay();

leftEye.clear();

rightEye.drawBitmap(0, 0, eyesclosed\_bmp, 8, 8, LED\_ON);

rightEye.writeDisplay();

rightEye.clear();

delay(blinkDelay);

}

}

void animationTwinkle() {

animSelection = random(9);

if (animSelection == 0) {

rightEye.drawBitmap(0, 0, eyes1\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes1\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 1) {

rightEye.drawBitmap(0, 0, eyes2\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes2\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 2) {

rightEye.drawBitmap(0, 0, eyes3\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes3\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 3) {

rightEye.drawBitmap(0, 0, eyes4\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes4\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 4) {

rightEye.drawBitmap(0, 0, eyes5\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes5\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 5) {

rightEye.drawBitmap(0, 0, eyes6\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes6\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 6) {

rightEye.drawBitmap(0, 0, eyes7\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes7\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 7) {

rightEye.drawBitmap(0, 0, eyes8\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes8\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 8) {

rightEye.drawBitmap(0, 0, eyes9\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, eyes9\_bmp, 8, 8, LED\_ON);

}

eyeUpdate();

}

void animationAngry() {

animSelection = random(3);

if (animSelection == 0) {

rightEye.drawBitmap(0, 0, angry1\_right\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, angry1\_left\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 1) {

rightEye.drawBitmap(0, 0, angry2\_right\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, angry2\_left\_bmp, 8, 8, LED\_ON);

}

if (animSelection == 2) {

rightEye.drawBitmap(0, 0, angry3\_right\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, angry3\_left\_bmp, 8, 8, LED\_ON);

}

eyeUpdate();

}

void playRandomSound() {

animSelection = random(2);

if (animSelection == 0) {

dfPlayer.play(soundNeigh);

}

if (animSelection == 1) {

dfPlayer.play(soundGasp);

}

}

void animationDead() {

rightEye.drawBitmap(0, 0, dead\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, dead\_bmp, 8, 8, LED\_ON);

leftEye.writeDisplay();

rightEye.writeDisplay();

leftEye.clear();

rightEye.clear();

}

void animationSleepy() {

rightEye.drawBitmap(0, 0, sleepy\_bmp, 8, 8, LED\_ON);

leftEye.drawBitmap(0, 0, sleepy\_bmp, 8, 8, LED\_ON);

leftEye.writeDisplay();

rightEye.writeDisplay();

leftEye.clear();

rightEye.clear();

}

void animationHeart() {

blinkChanceResult = random(blinkChance);

heart.clear();

heart.drawBitmap(0, 0, heart\_big\_bmp, 8, 8, LED\_ON);

heart.writeDisplay();

if (blinkChanceResult >= 5) {

heart.clear();

heart.drawBitmap(0, 0, heart\_small\_bmp, 8, 8, LED\_ON);

heart.writeDisplay();

}

}

void animationSad() {

heart.drawBitmap(0, 0, sadface\_bmp, 8, 8, LED\_ON);

heart.writeDisplay();

heart.clear();

}

// Used for debugging stats

void debugLog() {

Serial.println("HEALTH:");

Serial.print(health);

Serial.println();

Serial.println("HUNGER:");

Serial.print(hunger);

Serial.println();

Serial.println("HAPPINESS:");

Serial.print(happiness);

Serial.println();

Serial.println("ENERGY:");

Serial.print(energy);

Serial.println();

Serial.println("AGE:");

Serial.print(age);

Serial.println();

Serial.println("-------------");

Serial.println();

}

// Restarts the sketch

void softReset() {

asm volatile (" jmp 0");

}