LEUCHTENSCHWARM

Raumapparate | *Hier spricht das Mobiliar!* | SS 2009 | Katja Knecht

Abb. 1: Installation des Leuchtenschwarms in der Ausstellung Raumapparate „Hier spricht das Mobiliar!“

Der Leuchtenschwarm ist eine Installation, die im Sommersemester 2009 innerhalb des Projekts Raumapparate “Hier spricht das Mobiliar” der Professuren Interface Design und Gestaltung medialer Umgebungen an der Bauhaus-Universität entstanden ist und die im Rahmen der jährlich stattfindenden Summaery vom 10. bis 12. Juli 2009 ausgestellt wurde. Die folgende Dokumentation gibt einen Einblick in das Projekt und seine Grundidee sowie in die technische Umsetzung und Ausführung.

PROJEKTBESCHREIBUNG

Der Leuchtenschwarm ist eine Licht-Raum-Installation, die im Spannungsfeld steht zwischen Vergangenheit und Erinnerung, dem Hier und Jetzt und dem, was sein kann. Die Leuchten sind Relikte einer vergangenen Zeit, eines vergangenen Orts und eines vergangenen Lebens. In ihnen ruht die Vergangenheit, die Aura der Erinnerung umgibt sie ohne jedoch zu konkret zu werden. Die Installation lässt dem Besucher den Freiraum, sich selbst zu erinnern und Assoziationen zu entwickeln.

Darüber hinaus steht Licht als Symbol für Leben, Hoffnung und Erneuerung. Es bestimmt unseren Tagesrhythmus und ist verankert in der Gegenwart, unserem Hier und Jetzt. Gleichzeitig verweist es in seiner Vergänglichkeit auf das Kommende, jenseits alles Gewesenen. Auch die Leuchten sind hier nicht nur Träger der Erinnerung. Befreit von ihren herkömmlichen Zwängen entwickeln sie eine neuartige Form des Daseins. Herausgelöst aus ihrem ursprünglichen Umfeld bilden sie eine neue Organisations- und „Lebensform", den Schwarm.

Abb. 2: Illustration des Konzepts

Der Leuchtenschwarm reagiert auf die physische Präsenz des Besuchers. Die Bewegungen der Personen im Raum beeinflussen das Leuchtverhalten seiner Elemente. Es entsteht dabei ein Lichtraum, der sich kontinuierlich durch die körperliche Interaktion des Besuchers ändert. Der Besucher kann so sein Raumerlebnis maßgeblich mitbestimmen.

Die Installation arbeitet mit einem Bewegungs-Tracking, wobei der Abstand des Besuchers zur Leuchte deren Helligkeit bestimmt. Das Tracking und die Steuerung der Leuchten werden durch die Verknüpfung von Processing und Arduino realisiert, die in einem späteren Kapitel genauer beschrieben wird.

INSPIRATIONSQUELLEN

Vorbilder aus verschiedenen Bereichen inspirierten die Idee zum Leuchtenschwarm und seine Umsetzung.

Das Schwarm- und Leuchtverhalten lehnt sich einem Beispiel der Natur an, dem Blinkverhalten von Leuchtkäfern, oder auch Glühwürmchen (Lampyridae) genannt. (WIKIPEDIA, 18.07.2009) Die Glühwürmchen können Lichtsignale zur Kommunikation mit ihren Artgenossen aussenden. Einige dieser Arten synchronisieren ihre Blinksignale in Abhängigkeit ihrer Umgebung. Mehrere Käfer entwickeln ein gemeinsames, selbstorganisiertes Blinkverhalten, so dass ganze Busch- oder Baumreihen im gleichen Takt blinken. (KÄSMACHER, 2008, S. 50)

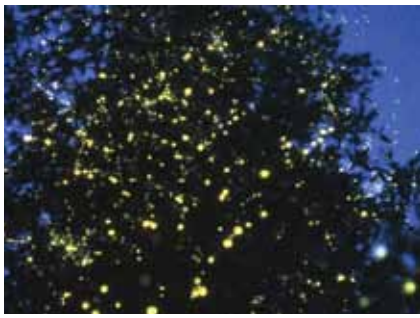

Abb. 3: Vorbild aus der Natur – der Glühwürmchenschwarm

Auch die aus ihrem ursprünglichen Dasein herausgelösten Leuchten transzendieren zu einer solchen, neuen Organisationsform. Die Elemente des Schwarms entwickeln eine neue Verhaltensweise und reagieren auf ihre Nachbarn, ihre Umgebung und auf die Bewegung der Besucher im Raum.

Weitere Inspiration lieferten künstlerische Projekte, die mit Licht und insbesondere Leuchten im Raum arbeiten. Hierzu gehört die Installation Space Invaders von Rainer Kehres und Sebastian Hungerer, die 2006

am Zentrum für Kunst und Medientechnologie (ZKM) in Karlsruhe im Rahmen der Ausstellung Lichtkunst aus Kunstlicht ausgestellt wurde. (ZKM, 06.09.2009)

Space Invaders ist eine Lichtinstallation bestehend aus 176 Lampen verschiedener Jahrzehnte. Die Leuchten wurden hier streng geometrisch und im selben Abstand innerhalb einer Metall-Konstruktion angeordnet. Die Anordnung selbst erfolgte hier nach dem ästhetischen Gesamteindruck den die Künstler während dem Prozess des Aufhängens hatten. Mit Abmessungen von 11 mal 14 Metern war die Arbeit raumgreifend und raumbestimmend.

Ebenfalls von Rainer Kehres und Sebastian Hungerer ist die Arbeit Chorus, die 2008 in St. Louis, USA, ausgestellt wurde und den Space Invaders ähnelt. (HUNGERER KEHRES, 06.09.2009)

Für das ausgebrannte Dach der Kirche in St. Louis sammelten die Künstler Lampen von den Gemeindemitgliedern. Die Leuchten waren jeweils mit einer eigenen Geschichte und Erinnerungen dieser Menschen verknüpft. Insgesamt wurden für die Lichtinstallation 289 Lampen verwendet. Die Geschichten zu diesen Leuchten wurden im Internet veröffentlicht.

Der Leuchtenschwarm wurde außerdem von dem Projekt Lumen der Künstlergruppe Electroland inspiriert, das die Bewegungen von Besuchern trackt und in Licht überführt. (ELECTROLAND, 06.09.2009)

Lumen ist eine ortsspezifische Installation, die 2006 im Cooper Hewitt National Design Museum in New York ausgestellt wurde. Die Wände der Haupttreppe des Gebäudes wurden mit einer transparenten Hülle versehen. Das Hinauf und Hinuntergehen der Besucher führt zu einer Licht- und Klangreaktion der Installation, die den Besuchern in ihrer Bewegung folgt.

Elemente dieses Projekts sowie der vorher beschriebenen Beispiele, finden sich in anderer Form im Leuchtenschwarm wieder. Der Leuchtenschwarm verbindet diese Einflüsse und Ideen zu einer neuen Form des Ausdrucks von Erinnerung, Bewegung, Licht und Reaktion.

Abb. 4: Space Invaders im ZKM Karlsruhe

Abb. 5: Chorus

Abb. 6: Lumen von Electroland

DIE LEUCHTEN

Die Installation besteht insgesamt aus einer Mischung aus 26 Decken-, Wand-, Steh- und Tischlampen, die zu einem Schwarm verdichtet im Raum angeordnet wurden. Die Leuchten wurden nahezu komplett aus

dem Haus einer verstorbenen Verwandten entnommen, das kurz vor der Renovierung stand. Ursprünglich waren sie auf zwei Stockwerke und damit zwei Wohnungen verteilt. Die ursprüngliche Verteilung und Anordnung wurde bei der Anordnung der Lampen in der späteren Installation berücksichtigt, wobei die beiden Wohnungsgrundrisse überlagert wurden.

Abb. 7: Verteilung der Leuchten auf Basis des Grundrisses

Die Leuchten stammen aus verschiedenen Jahrzehnten. Sie verweisen an sich auf den Ort ihrer Herkunft, auf eine vergangene Zeit und sind damit ein Gegenstand konkreter Erinnerung. Durch die Umsiedlung und Neuorganisation erhält auch die Erinnerung eine neue Bedeutung. Die Erinnerung wird universelles Gut. Gleichzeitig ist sie jedoch immer auch persönlich, da jeder Betrachter Orte, Zeiten und Personen seiner eigenen Vergangenheit mit dem Objekt assoziiert und diese Erinnerungen im Augenblick der Betrachtung wachruft.

TECHNISCHES KONZEPT

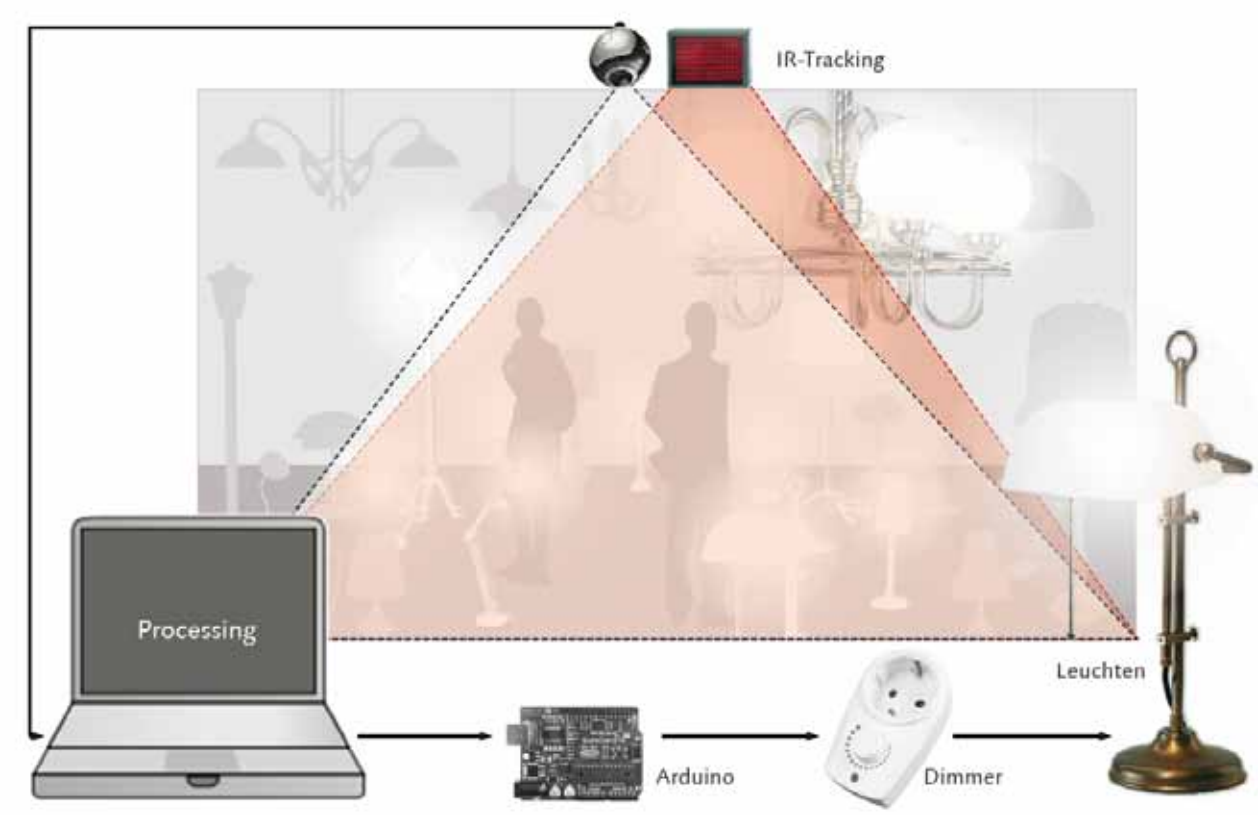

Abb. 8: Technisches Konzept in der Übersicht

Kurzbeschreibung
Die Reaktion des Leuchtenschwarms auf die Bewegungen der Besucher wird durch ein Differenzbildtracking realisiert. Da die Installation für

einen dunklen Raum gedacht ist, wird der Raum zunächst mit Infrarotlicht ausgeleuchtet. Im Zentrum der Installation wird eine infrarotempfindliche Webcam angebracht, die den Raum überwacht. Das Videobild wird in Processing ausgelesen und weiterverarbeitet. Die dabei getrackten Bewegungen sind Ausgangspunkt für die Helligkeitsberechnung der einzelnen Leuchten.

Diese Helligkeitswerte werden anschließend über die serielle Schnittstelle an ein Arduino-Board übermittelt. Für jede der Lampen wird eine weiße LED an einen PWM-Ausgang des Arduino angesteuert. Der aus Processing erhaltene Helligkeitswert wird dann an den entsprechenden Ausgang an eine LED übertragen.

Über die LEDs wird die Helligkeit von Steckdosendimmern gesteuert, an welche wiederum die Leuchten angeschlossen sind. Hierzu wurde in die Dimmer ein photosensitiver Widerstand eingebaut. Der Widerstand reagiert auf die Helligkeitswerte der LED und steuert den Grad der Dimmung der eingesteckten Leuchten entsprechend.

Abb. 9: Microsoft LiveCam VX-3000

Beschreibung des Technischen Ablaufs

Schritt 1: Überwachung des Aktionsraums
Das ursprüngliche Konzept sieht vor den Aktionsraum der Installation mit Infrarotlicht auszuleuchten und mit einer infrarotempfindlichen Webcam zu überwachen. Aufgrund der im Ausstellungsraum herrschenden, wechselhaften Lichtverhältnisse und dem starken Tageslichteinfall wurde der Leuchtenschwarm jedoch ohne Infrarotunterstützung umgesetzt. Die Videobilder werden anschließend per USB an den angeschlossenen Computer übertragen und in Processing weiterverarbeitet.

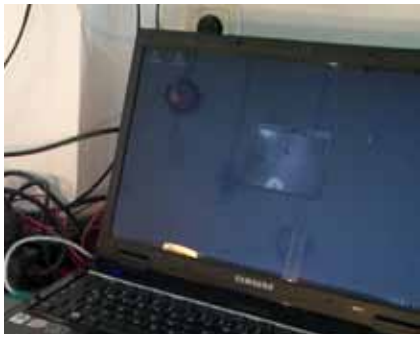

Abb. 10: Screenshot des Trackings

Schritt 2: Bewegungstracking
Durch das Auslesen der Videobilder werden Bewegungen im Raum getrackt. Das Tracking selbst erfolgt durch Differenzbild, d.h. die Veränderung von Pixeln im Vergleich zu einem Referenzbild wird berechnet. Die sich daraus ergebenden Blobs im Bild werden als Ausgangspunkt für die Berechnung der Helligkeitswerte verwendet.

Schritt 3: Abstands- und Helligkeitsberechnung
Den Leuchten wurden im Bezug zum Videobild virtuelle Standorte zugewiesen, über die die Abstände zu den Blobs im Raum berechnet werden konnten. Jede Leuchte besitzt dabei einen festgelegten Aktionsradius innerhalb dessen sie auf Bewegungen reagiert. Wird innerhalb dieses Aktionsradius eine Bewegung erkannt, so wird der Helligkeitswert der Lampe im Verhältnis zum Abstand des Bewegungsmittelpunkts berechnet.

Abb. 11: Helligkeit der Lampen in Relation zum Standort des Betrachters

Schritt 4: Reaktion der Nachbarlampen
Die Leuchten reagieren nicht nur auf die Bewegung im Raum sondern auch auf die Helligkeitsunterschiede ihrer Nachbarlampen. Innerhalb eines ebenfalls festgelegten, zweiten Aktionsradius registrieren sie das Leuchtverhalten der anderen Leuchten und reagieren darauf. Abstands- und Helligkeitsberechnung werden nach dem gleichen Prinzip wie in Schritt 3 durchgeführt.

Schritt 5: Übermittlung der Helligkeitswerte
Der jeweilige Helligkeitswert wird über die serielle Schnittstelle von Processing aus an das Arduino-Board übertragen. Durch das Anschliessen von zwei TLC 5940 – Chips wurde die Zahl der PWM-Ausgänge hier auf 32 erhöht. Für jede Leuchte wird ein Ausgang festgesetzt, an den je eine LED angeschlossen wird. Der in Processing ermittelte Helligkeitswert und die Ausgangsnummer der Leuchte wird als Hexadezimalzahl kodiert über die serielle Schnittstelle an das Arduino übertragen.

Abb. 12: Arduino und TLC-Chips

Schritt 6: Verarbeitung der Helligkeitswerte
Die über die serielle Schnittstelle ausgelesenen Daten werden zunächst in Arduino dekodiert und dann an den angegebenen Ausgang und die dort angeschlossene LED übermittelt. Die LED leuchtet im angegebenen Wert in einer Helligkeitstufe zwischen 0 (aus) und 4098 (an).

Schritt 7: Übertragung der Helligkeitswerte an die Leuchten
Über die Helligkeit der LED wird ein mit einem photosensitiven Widerstand ausgestatteter Dimmer gesteuert. Hierzu wurde das Potentiometer des Dimmers, über den sich manuell per Drehregler die Dimmung einstellen ließ, durch einen photosensitiven Widerstand ersetzt. Die LED wurde innerhalb des Gehäuses, in der Nähe des Widerstands, platziert. Das Gehäuse ist abgedunkelt, um störenden Lichteinfall von außen zu minimieren. Der photosensitive Widerstand reagiert dann auf die Helligkeit der in den Dimmer eingesteckten LED und übersetzt sie in die Dimmung. Die in den Steckdosendimmer eingesteckten Leuchten bzw. deren Leuchtmittel nehmen den vorgesehenen Helligkeitswert an.

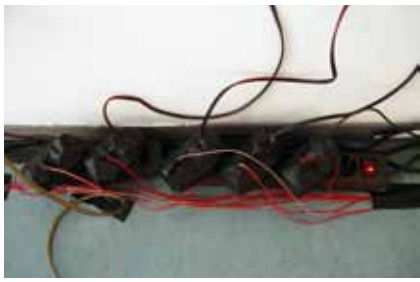

Abb. 13: Steckdosendimmer

INSTALLATION

Der Leuchtenschwarm wurde zur Summaery 2009 über die gesamte Breite des zur Verfügung stehenden Ausstellungsbereichs in der Kaufstraße 11 in Weimar installiert und führte die Besucher in die weitere Ausstellung hinein.

Die Bilder und Impressionen der Installation auf der nächsten Seite sind gleichzeitig Abschluss dieser Dokumentation.

LEUCHTENSCHWARM | Raumapparate | *Hier spricht das Mobiliar!* | SS 2009 | Katja Knecht

Abb. 14 bis 22: Impressionen der Ausstellung

ABBILDUNGSVERZEICHNIS

Abb. 1, 2: © Katja Knecht
Abb. 3: © Nature Publishing Group, http://www.nature.com/nature/journal/v437/n7057/fig_tab/437325a_F1.html
Abb. 4: © ONUK, http://hosting.zkm.de/lichtkunst/stories/storyReader$161
Abb. 5: © Rainer Kehres, Sebastian Hungerer, http://www.commonlights.com/seite2.html
Abb. 6: © Electroland, http://electroland.net/projects/lumen
Abb. 7, 8: © Katja Knecht
Abb. 9, 10, 11, 12: Video-Standbilder, © Max Neupert
Abb. 13: © Katja Knecht
Abb. 14, 17, 22: © Katja Knecht
Abb. 15, 16, 18, 19, 20, 21: © Daniel Huhndt / Karsten Kleinert

LITERATUR

DALIBOR, Andrijevic; GRILL, Andreas: Simulation der Synchronisation von Glühwürmchen/Neuronen; Universität Wien, 2009;
Abgerufen am 22. 07 2009 von http://www.unet.univie.ac.at/~a0400086//hpmoi/content/protokolle/biophysik/glwu.pdf

ELECTROLAND: Lumen.
Abgerufen am 06.09.2009 von http://electroland.net/projects/lumen

HUNGERER, Sebastian; KEHRES, Rainer: Chorus.
Abgerufen am 06.09.2009 von http://www.commonlights.com/seite2.html

KÄSMACHER, Dr. Horst: Von Glühwürmchen und Menschenschwärmen; Zeitschrift Humane Wirtschaft, 03/2008
Abgerufen am 28.07.2009 von http://www.humane-wirtschaft.de/03-2008/kaesmacher_gluehwuermchen.pdf

WIKIPEDIA: Leuchtkäfer.
Abgerufen am 18.07.2009 von http://de.wikipedia.org/wiki/Leuchtk%C3%A4fer

Zentrum für Kunst und Medientechnologie Karlsruhe (ZKM): Lichtkunst aus Kunstlicht.
Abgerufen am 06.09.2009 von http://hosting.zkm.de/lichtkunst/stories/storyReader$161

CODE

Processing

Leuchtenschwarm_kknecht

```
float threshold= 50;
float maxBright = 255;

class Firefly // Klassendefinition
{
  //---------------- Eigenschaften der Klasse Firefly ------------------

  float radius = 5; // Radius des Glühwürmchens
  float envRadius = 100; // Überwachungsradius für Bewegung
  float envRadSwarm = 100; // Überwachungsradius f. Helligkeitsänderungen innerhalb des Schwarms
  float xPos; // x-Position
  float yPos; // y-Position
  int intensity=0; // Glühintensität
  //int oldIntensity = 0; // vorangegangener Glühwert

  //------------------ Konstruktor der Klasse Firefly ------------------

  Firefly(float xp, float yp)
  {
    xPos = xp; // Werteübergabe von lokalen zu globalen Variablen
    yPos = yp;
  }

  //------------------ Methoden der Klasse Firefly ---------------------

  // 1. Schritt: GLOBS - Überwachung des Umfelds und Distanzberechnung
  //--------------------------------------------------------------------

  int run (int [][] globList, int sumD)  // überwacht den Interaktionsradius
  {
    float distance;
    float minDistance= 1000;
    int xTot=0, yTot=0, cnt=0;
    int val=0;

    for (int p = 0; p < globList.length; p++) {
      int x = globList[p][0];
      int y = globList[p][1];

      distance = dist(xPos, yPos, x, y);

      if (x!=0 || y!=0) {
        xTot += x;
        yTot += y;
        //cnt++;
      }

      if (distance < minDistance) {
        minDistance = distance;
      }
    }

    if (minDistance < envRadius) {
      val = round(maxBright * (envRadius - minDistance)/envRadius);
    }
    else if (sumD == 0){
      val = 0;
    }
```

```
    else
   {
    val = run2();
   }

    draw(color(val),radius);
    intensity = val;

    return val;
  }

  // 2. Schritt: LEUCHTEN - Überwachung des Umfelds und Distanzberechnung
  //---------------------------------------------------------------------

int run2 (Firefly [] swarm, int idx, int sumDif)  // überwacht den
  {                                                Interaktionsradius
    float distance2 = 800;
    float minDistance2= 1000;
    int xTot2=0, yTot2=0, cnt2=0;
    int val2=0;
    int minIdx2=0;

    for (int b = 0; b < swarm.length; b++) {
      if (b != idx){
        int x2 = int(swarm[b].xPos);
        int y2 = int(swarm[b].yPos);

        distance2 = dist(xPos, yPos, x2, y2);

        if (x2!=0 || y2!=0) {
          xTot2 += x2;
          yTot2 += y2;
          //cnt++;
        }
      }

      if (distance2 < minDistance2) {
        minDistance2 = distance2;
        minIdx2 = b;
      }
    }

    if (minDistance2 < envRadSwarm) {
      val2 = round(swarm[minIdx2].intensity * (envRadius - minDistance2)/envRadSwarm);
    }
    else if (sumDif == 0){
      val2 = 0;
    }
    else if (minDistance2 > envRadSwarm) {
      val2 = 0;
    }
    else
   {
    val2 = intensity;
   }

    draw(color(val2),radius);
    intensity = val2;

    return val2;
  }
```

```
  // Zeichenfunktionen

  void draw (color cl, float r)  // zeichnet das Glühwürmchen
  {
    stroke(100,100,100,150);
    fill (cl);
    ellipseMode(CENTER); // setzt Kreismittelpunkt
    ellipse(xPos,yPos,r*2,r*2); // zeichne Ellipse
  }
}
```

Main

```
import JMyron.*;
import processing.serial.*;

JMyron m;
Serial port;

int tol; // Toleranzwert

int firefliesNumber = 17; // Anzahl der angeschlossenen Leuchten
float maxRad = 5;

int[] oldValue = new int[firefliesNumber];
int[] newValue = new int[firefliesNumber];

int numPixels;
int[] backgroundPixels;

Firefly[] myFirefly = new Firefly[firefliesNumber]; // deklarieren des Arrays

void setup ()  // Programm-Setup
{
  size(320,240);
  m = new JMyron();
  m.start(width,height);

  numPixels = width * height;

  // Array zur Speicherung des Hintergrundbilds
  backgroundPixels = new int[numPixels];

  for(int s = 0; s < numPixels; s++)
  {
    backgroundPixels[s] = 255;
  }

  tol = 600;
  m.findGlobs(1);
  m.trackColor(200,200,200,tol);
  m.adaptivity(20);
  m.adapt();
  loadPixels();
  rectMode(CENTER);
  smooth();
  noFill();
  stroke(255,255,0);
  frameRate(15);
  int i = 0;

  // Verortung der Leuchten; Koordinatenangabe in Bezug zum Videobild
  myFirefly[0] = new Firefly(320, 100);
  myFirefly[1] = new Firefly(300, -50);
```

```
  myFirefly[2] = new Firefly(280, 270);
  myFirefly[3] = new Firefly(260, -100);
  myFirefly[4] = new Firefly(240, -100);
  myFirefly[5] = new Firefly(220, -100);
  myFirefly[6] = new Firefly(220, 0);
  myFirefly[7] = new Firefly(200, -100);
  myFirefly[8] = new Firefly(180, -100);
  myFirefly[9] = new Firefly(160, 240);
  myFirefly[10] = new Firefly(150, -20);
  myFirefly[11] = new Firefly(140, -100);
  myFirefly[12] = new Firefly(120, 100);
  myFirefly[13] = new Firefly(100, -100);
  myFirefly[14] = new Firefly(80, 0);
  myFirefly[15] = new Firefly(40, 40);
  myFirefly[16] = new Firefly(20, 240);

  for(int k = 0; k < firefliesNumber; k++) // Anfangswerte werden auf Null gesetzt
  {
    oldValue[k] = 0;
  }

  //println(Serial.list());
  String arduinoPort = Serial.list()[1];
  port = new Serial(this, arduinoPort, 9600); // serielle Schnittstelle zum Arduino
}

void draw ()
{
  color c;
  String cs;

  background(0);
  m.update();
  m.imageCopy(pixels);
  int presenceSum = 0;
  for (int r = 0; r < numPixels; r++) { // Für jeden Pixel des Videobilds
      // wird die aktuelle Farbe an dieser Stelle, sowie auch die Farbe
      // des Hintergrunds in diesem Punkt ermittelt
      color currColor = pixels[r];
      color bkgdColor = backgroundPixels[r];
      int diff = abs(currColor - bkgdColor);
      presenceSum += diff;
    }

  updatePixels();
  displayGlobs(m.globCenters());

  for(int j = 0; j < firefliesNumber; j++)
  {
    newValue[j] = myFirefly[j].run(m.globCenters(),presenceSum);  // Aufruf der
    //Funktion run() zur Überwachung des Umfelds und zur Abstands- bzw.
    //Helligkeitsberechnung
  }

  for(int l = 0; l < firefliesNumber; l++)
  {
    if (oldValue[l] != newValue[l]){
      c = color(l, newValue[l], 0);
      cs = „#“ + hex(c,6); // Vorbereitung der Zeichenkette
      port.write(cs); // Daten werden über die serielle Schnittstelle an das Arduino gesendet
      oldValue[l] = newValue[l];
    }
    else if (presenceSum == 0){
      cs = „#000000“;
      port.write(cs); // Daten werden über die serielle Schnittstelle an das Arduino gesendet
```

```
    }
  }
}

void displayGlobs(int [][] lst) // Globs anzeigen
{
  stroke(255,255,0);
  int xTot=0, yTot=0;
  for (int q = 0; q < lst.length; q++)
  {
    int lx = lst[q][0];
    int ly = lst[q][1];
    if (lx!=0 || ly!=0)
    {
      xTot += lx;
      yTot += ly;
      stroke(255,0,0);
      rect(lx,ly,5,5);
    }
  }
}

void stop() {
  m.stop();
  super.stop();
}

void keyPressed() // Speichert ein Referenzbild auf Tastendruck
{
  loadPixels();
  arraycopy(pixels, backgroundPixels);
  for(int u = 0; u < firefliesNumber; u++)
  {
    color clr = color(u, 0, 0);
    String cString = „#“ + hex(clr,6); // Vorbereitung der Zeichenkette
    port.write(cString); // Daten werden über die serielle Schnittstelle an das Arduino gesendet
  }
}
```

Arduino

TLC5940_setup

```
#include „Tlc5940.h“

byte inByte = 0;      // Variable zur Speicherung der eingehenden seriellen Daten
char buffer[7] ;
int pointer = 0;

byte pin = 0;
byte value = 0;
byte x = 0;

void setup()
{
  Tlc.init(); // Set-up des TLC-Chips
  Serial.begin(9600); // Initialisierung der seriellen Kommunikation
}

/* Der Loop überträgt die über die serielle Schnittstelle gesendeten Helligkeitswerte
   an die angegebenen TLC Ausgänge. */

void loop()
```

```
{
  if (Serial.available() > 0) {
    // das älteste byte wird im Buffer gespeichert:
    inByte = Serial.read();

    // Nach dem Auslesen des Markers folgen die codierten Daten
    if (inByte == ,#') {
      while (pointer < 6) { // 6 char-Werte auslesen
        buffer[pointer] = Serial.read(); // und im Buffer speichern
        pointer++; // die Stelle um 1 erhöhen
      }

      // Die Daten sind als Hexadezimalzahl gespeichert
      // und müssen wieder decodiert werden
      pin = hex2dec(buffer[1]) + hex2dec(buffer[0]) * 16; // Nummer des TLC-Ausgangs
      value = (hex2dec(buffer[3]) + hex2dec(buffer[2]) * 16)*16; // zu übertragender Helligkeitswert
      x = hex2dec(buffer[5]) + hex2dec(buffer[4]) * 16; // nicht benutzte Stellen

      pointer = 0; // Zurücksetzen des Zeigers
    }
  }

  /* Tlc.set(channel (0-15), value (0-4095)) setzt den Graustufenwert für einen
       Channel (15 entspricht OUT15 auf der ersten TLC, bei mehreren TLCs
       hintereinander entspricht channel = 16 OUT0 der zweiten TLC, etc.).

       Der Wert liegt zwischen aus (0) und immer an (4095).

       Diese Funktion stellt lediglich die Daten bereit, Tlc.update()
       sendet die Daten. */

  Tlc.set(pin, value);

  /* Tlc.update() schickt die Daten an die TLCs. Hier wird sich die Anzeige
       der LED ändern. */

  Tlc.update();

  delay(10); // wartet 10 ms zwischen jedem Senden
}

int hex2dec(byte c) // Konvertiert einen Hexadezimalwert in eine Zahl
{
    if (c >= ,0' && c <= ,9') {
      return c - ,0';
    }
    else if (c >= ,A' && c <= ,F') {
      return c - ,A' + 10;
    }
}
```

tlc_config.h

```
/*  Copyright (c) 2009 by Alex Leone <acleone ~AT~ gmail.com>

    This file is part of the Arduino TLC5940 Library.

    The Arduino TLC5940 Library is free software: you can redistribute it
    and/or modify it under the terms of the GNU General Public License as
    published by the Free Software Foundation, either version 3 of the
    License, or (at your option) any later version.

    The Arduino TLC5940 Library is distributed in the hope that it will be
    useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with The Arduino TLC5940 Library.  If not, see
    <http://www.gnu.org/licenses/>. */

#ifndef TLC_CONFIG_H
#define TLC_CONFIG_H

#include <stdint.h>

/** \file
    Configuration for the Arduino Tlc5940 library.  After making changes to
    this file, delete Tlc5940.o in this folder so the changes are applied.

    A summary of all the options:
    - Number of TLCs daisy-chained: NUM_TLCS (default 1)
    - Enable/Disable VPRG functionality: VPRG_ENABLED (default 0)
    - Enable/Disable XERR functionality: XERR_ENABLED (default 0)
    - Should the library use bit-banging (any pins) or hardware SPI (faster):
        DATA_TRANSFER_MODE (default TLC_SPI)
    - Which pins to use for bit-banging: SIN_PIN, SIN_PORT, SIN_DDR and
        SCLK_PIN, SCLK_PORT, SCLK_DDR
    - The PWM period: TLC_PWM_PERIOD (be sure to change TLC_GSCLK_PERIOD
        accordingly!)

    How to change the pin mapping:
    - Arduino digital pin 0-7  = PORTD, PD0-7
    - Arduino digital pin 8-13 = PORTB, PB0-5
    - Arduino analog pin  0-5  = PORTC, PC0-5 */

/** Bit-bang using any two i/o pins */
#define TLC_BITBANG        1
/** Use the much faster hardware SPI module */
#define TLC_SPI            2

/* ------------------------ START EDITING HERE ---------------------------- */

/** Number of TLCs daisy-chained.  To daisy-chain, attach the SOUT (TLC pin 17)
    of the first TLC to the SIN (TLC pin 26) of the next.  The rest of the pins
    are attached normally.
    \note Each TLC needs it's own IREF resistor */

#define NUM_TLCS    2 // ANZAHL DER VERBUNDENEN TLC-CHIPS

/** Determines how data should be transfered to the TLCs.  Bit-banging can use
    any two i/o pins, but the hardware SPI is faster.
    - Bit-Bang = TLC_BITBANG
    - Hardware SPI = TLC_SPI (default) */
#define DATA_TRANSFER_MODE    TLC_SPI

/* This include is down here because the files it includes needs the data
   transfer mode */
#include „pinouts/chip_includes.h"

/* Set DATA_TRANSFER_MODE to TLC_BITBANG and change the pins below if you need
   to use different pins for sin and sclk.  The defaults are defined in
   pinouts/ATmega_xx8.h for most Arduino's. */

#if DATA_TRANSFER_MODE == TLC_BITBANG
/** SIN (TLC pin 26) */
#define SIN_PIN        DEFAULT_BB_SIN_PIN
#define SIN_PORT       DEFAULT_BB_SIN_PORT
#define SIN_DDR        DEFAULT_BB_SIN_DDR
```

```
/** SCLK (TLC pin 25) */
#define SCLK_PIN        DEFAULT_BB_SCLK_PIN
#define SCLK_PORT       DEFAULT_BB_SCLK_PORT
#define SCLK_DDR        DEFAULT_BB_SCLK_DDR
#endif


/** If more than 16 TLCs are daisy-chained, the channel type has to be uint16_t.
    Default is uint8_t, which supports up to 16 TLCs. */
#define TLC_CHANNEL_TYPE    uint8_t

/** Determines how long each PWM period should be, in clocks.
    \f$\displaystyle f_{PWM} = \frac{f_{osc}}{2 * TLC\_PWM\_PERIOD} Hz \f$
    \f$\displaystyle TLC\_PWM\_PERIOD = \frac{f_{osc}}{2 * f_{PWM}} \f$
    This is related to TLC_GSCLK_PERIOD:
    \f$\displaystyle TLC\_PWM\_PERIOD =
       \frac{(TLC\_GSCLK\_PERIOD + 1) * 4096}{2} \f$
    \note The default of 8192 means the PWM frequency is 976.5625Hz */
#define TLC_PWM_PERIOD    8192

/** Determines how long each period GSCLK is.
    This is related to TLC_PWM_PERIOD:
    \f$\displaystyle TLC\_GSCLK\_PERIOD =
       \frac{2 * TLC\_PWM\_PERIOD}{4096} - 1 \f$
    \note Default is 3 */
#define TLC_GSCLK_PERIOD    3

/** Enables/disables VPRG (TLC pin 27) functionality.  If you need to set dot
    correction data, this needs to be enabled.
    - 0 VPRG is not connected.  <em>TLC pin 27 must be grounded!</em> (default)
    - 1 VPRG is connected
    \note VPRG to GND inputs grayscale data, VPRG to Vcc inputs dot-correction
          data */
#define VPRG_ENABLED    0

/** Enables/disables XERR (TLC pin 16) functionality to check for shorted/broken
    LEDs
    - 0 XERR is not connected (default)
    - 1 XERR is connected
    \note XERR is active low */
#define XERR_ENABLED    0

/*  You can change the VPRG and XERR pins freely.  The defaults are defined in
    the chip-specific pinouts:  see pinouts/ATmega_xx8.h for most Arduino's. */

#if     VPRG_ENABLED
/** VPRG (TLC pin 27) */
#define VPRG_PIN        DEFAULT_VPRG_PIN
#define VPRG_PORT       DEFAULT_VPRG_PORT
#define VPRG_DDR        DEFAULT_VPRG_DDR
#endif

#if     XERR_ENABLED
/** XERR (TLC pin 16) */
#define XERR_PIN        DEFAULT_XERR_PIN
#define XERR_PORT       DEFAULT_XERR_PORT
#define XERR_DDR        DEFAULT_XERR_DDR
#define XERR_PINS       DEFAULT_XERR_PINS
#endif

/* ------------------------- STOP EDITING HERE ---------------------------- */

#if DATA_TRANSFER_MODE == TLC_SPI
/** SIN (TLC pin 26) */
#define SIN_PIN        TLC_MOSI_PIN
```

```
#define SIN_PORT       TLC_MOSI_PORT
#define SIN_DDR        TLC_MOSI_DDR
/** SCLK (TLC pin 25) */
#define SCLK_PIN       TLC_SCK_PIN
#define SCLK_PORT      TLC_SCK_PORT
#define SCLK_DDR       TLC_SCK_DDR
#endif

#if !(DATA_TRANSFER_MODE == TLC_BITBANG \
 || DATA_TRANSFER_MODE == TLC_SPI)
#error „Invalid DATA_TRANSFER_MODE specified, see DATA_TRANSFER_MODE“
#endif

/* Various Macros */

/** Arranges 2 grayscale values (0 - 4095) in the packed array format (3 bytes).
    This is for array initializers only: the output is three comma seperated
    8-bit values. */
#define GS_DUO(a, b)    ((a) >> 4), ((a) << 4) | ((b) >> 8), (b)


#if VPRG_ENABLED
/** Arranges 4 dot correction values (0 - 63) in the packed array format.
    \see setDCtoProgmemArray */
#define DC_QUARTET(a, b, c, d)   ((a) << 2) | ((b) >> 4), \
                                  ((b) << 4) | ((c) >> 2), \
                                  ((c) << 6) | (d)
#endif

#endif
```