Hintergrund

Was ist Conways Game of Life?

Game of Life ist ein 0-Player-Game zur Umsetzung der Automaten-Theorie von Stanisław Marcin Ulam, entworfen von dem Mathematiker John Conway. Wenige, einfache Regeln führen dabei zu verblüffend komplexem Verhalten.

Regeln

Das Feld ist aufgeteilt in einzelne, quadratische Zellen, die einfach so nebeneinander angeordnet sind, dass jede Zelle 8 Nachbarn hat (wie auf einem Karo-Papier). Jede Zelle kann entweder lebendig oder tot sein, dieser Status hängt vom Anfangszustand und dem Zustand der 8 Nachbarn ab, die die Zelle „sehen“ kann. In jeder Runde (=Generation) zählt jede Zelle, wie viele ihrer Nachbarn lebendig ist und ändert dann je nach Anzahl den eigenen Lebensstatus nach folgenden Regeln:

- Zellen mit weniger als 2 Nachbarn sterben (an „Vereinsamung“)
- Zellen mit genau 2 Nachbarn ändern ihren Status nicht
- Zellen mit genau 3 Nachbarn werden lebendig
- Zellen mit mehr als 3 Nachbarn sterben (an „Überbevölkerung“)

Vision

Im Rahmen unseres Projektes möchten wir die Funktionsweise von Conway's Game of Life mit Hilfe einer interaktiven leuchtenden Kunstinstallation erfahrbar machen. Um ein intuitives Verständnis der Thematik zu fördern, wollen wir ein manuell verstellbares und programmierbares Display bauen.

Ziele

- Feld aus 3×3 Einzelzellen bauen
- Interaktivität durch Hall-Sonde & Magnet am „Zauberstab“
- Aufbau soll cell-based sein: Jede Zelle sieht nur ihre Nachbarn, es gibt also kein großes Programm im Hintergrund, das alle Zellen ansteuert
- modulare Aufbauweise: Das Feld soll in verschiedenen Konfigurationen aufgebaut werden können, um Strukturen verschiedenen Ausmaßes darstellen zu können (dafür idealerweise jede Zelle einzeln herausnehm- und zusammensteckbar)
- das Projekt soll vollständig dokumentiert und open-source sein
- Nebenziel: Erweiterung der Installation zur Visualisierung komplexerer Strukturen durch nebenstehenden PC
- Traumziel: wenn Zeit und Resourcen da sind auf ein ca. 20$\times$20 Feld erweitern

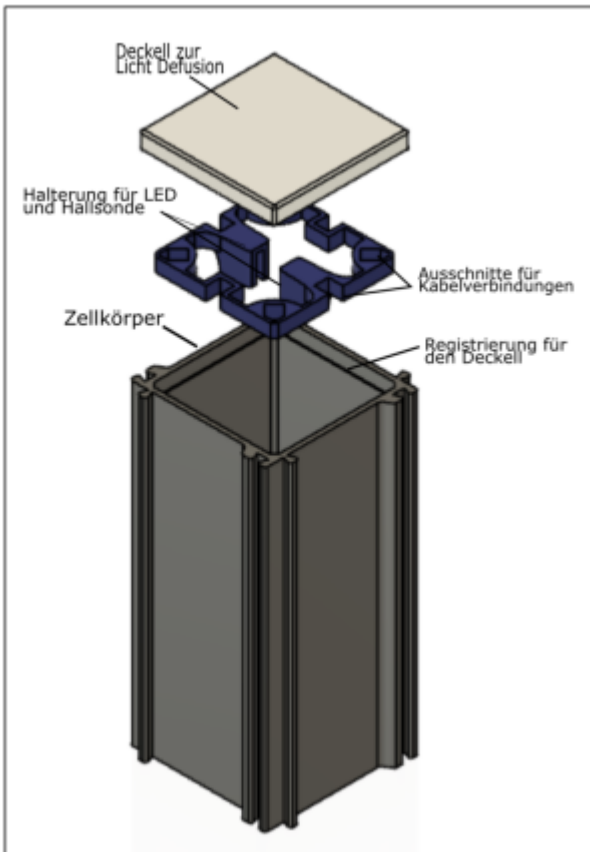
Arbeitsablauf

Anfangsüberlegungen

Zellform: Wir hatten kurz mit dem Gedanken gespielt, die Felder achteckig zu gestalten, da man dort aber zwangsläufig Zwischenräume hat, die überbrückt werden müssen, sind wir doch zu den üblichen

Quadraten zurückgekehrt.
lebendig/tot darstellen: Die Ursprüngliche Idee war ein Flippdot-Display selbst zu bauen, das schien in der Planung aber zu großer Aufwand zu werden und auch zu fragil für die Nutzung durch Personen, die sich nicht genau damit auskennen, daher haben wir uns dazu entschieden, den Lebensstatus der Zellen über eine LED sichtbar zu machen. Eine leuchtende Zelle symbolisiert also eine lebendige Zelle, eine nicht leuchtende eine tote.
Welcher Sensor?: Um die Interaktivität zu gewährleisten, also direkt am Feld verschiedene Anfangsbedingungen einstellen zu können, brauchen wir eine Art Knopf. Ein simpler Druckknopf in jeder Zelle schien uns zu unhandlich, gerade bei größeren Feldern, deswegen haben wir uns letztendlich für eine Lösung mit einer Hall-Sonde (Magnetsensor) in jeder Zelle entschieden, sodass man den Lebensstatus der Zelle durch Berührung mit einem „Zauberstab“ (fancy Stab mit Magnet an der Spitze) geändert werden kann. Weitere Ideen, die wir vor allem auf Grund der Störanfälligkeit durch äußere Einflüsse verworfen haben, waren ein Kapazitätsberührungssensor und weiter Berührungssensoren, die über die Erschütterung einer kleinen Feder ausgelöst werden.

Zelldesign



Jede Zelle besteht aus einem 3D-gedruckten Zellkörper , der so gestaltet ist, dass man die Zellen einfach ineinander schieben kann und somit ein solides Feld erhält. darin müssen die Hall-Sonde, eine LED, ein Arduino Nano und einiges an Kabeln Platz finden, deswegen haben wir die Zellkörper 7cm tief gedruckt. Die LED und die Hall-Sonde werden im ebenfalls 3D-gedruckten Deckel befestigt (s. Bild rechts), damit das Licht von außen gut sichtbar ist und alle Zellen einheitlich leuchten und ebenso einheitlich auf den Magneten reagieren. Außerdem gibt es Aussparungen für die Steckverbindungen der Kabel, die die Kommunikation der Zellen ermöglicht, im Deckel, sodass diese gut sichtbar sind und auch im zusammengesteckten Zustand leicht geändert werden können. Dies hat sowohl den praktischen Grund, dass es wohl die einfachst mögliche Lösung der Kommunikation ist, als auch pädagogische, denn so kann man im laufenden Betrieb eine Verbindung unterbrechen und darüber demonstrieren, wie die Zellen einander „sehen“. Abgedeckt ist die 3d-gedruckte Deckelrahmenkonstruktion mit milchigem Acrylglas, das mit dem Laser-Cutter geschnitten wurde. Die Rückseite ist mit einem ebenfalls 3D-gedruckten Deckel verschlossen, in dem Aussparungen für die Stromverbindung (und Taktgebung) sind.
Die 3D-gedruckten Teile wurden mit einem SLA-Drucker gedruckt, da dieser an sich genauer drucken kann als ein FDM-Drucker. Dadurch, dass die Teile nach dem Drucken aber gehärtet werden müssen, besteht bei unseren recht dünnen Wänden eine Neigung zum Neigen. Deswegen drucken wir die Teile doch lieber mit einem FDM-Drucker.

Software

Jede Zelle trägt das gleiche kleine Programm in sich, dass im Prinzip nur die Anzahl der lebendigen Nachbarn zählen und danach entscheiden muss, wie ihr Lebensstatus sein soll. Das größte Problem hierbei ist die Taktung, denn alle Zellen müssen zunächst den Status ihrer Nachbarn auslesen, bevor sie ihren eigenen ändern, ansonsten würde es zu Fehlern kommen. Recht leicht lösbar wäre diese Herausforderung, wenn man alle benachbarten Zellen mit jeweils zwei Kabeln verbinden würde. Dann würde an einem Zelle A ihren Lebensstatus schreiben, was Zelle B lesen könnte und an dem anderen umgekehrt. Das kann allerdings schnell zu einem riesigen Kabelsalat führen und die Anzahl der Pins an den Nanos würde nicht reichen, daher wollten wir eine Lösung, in der zwischen zwei benachbarten Zellen immer nur ein Kabel verläuft. Dafür dürfen zwei benachbarte Zellen nicht versuchen, gleichzeitig ihren Lebensstatus über das gleiche Kabel zu vermitteln.
Unsere erste Idee war, die Zellen im Schachbrettmuster auslesen zu lassen, also in zwei Gruppen (angeordnet wie schwarze und weiße Felder auf einem Schachbrett), in denen sich die Zellen mit lesen und schreiben abwechseln. Allerdings lesen sich die diagonal zueinander liegenden Zellen so immer noch gleichzeitig aus, mit diesem Ansatz funktioniert es also nicht.
Die zweite und auch verwendete Idee ist, die Arduinos in acht Schritten in einer Art „Uhr“ ihre Nachbarn auslesen zu lassen. Dafür lesen zuerst alle Arduinos den Nachbarn aus, der über ihnen angeordnet ist und schreiben dann logischerweise auf den passenden Nachbarn, also den, der unter ihnen angeordnet ist. Dann geht es im Uhrzeigersinn weiter: als nächstes wird rechts oben gelesen und links unten geschrieben und so weiter, bis die Uhr einmal herum ist und alle acht Nachbarn ausgelesen und beschrieben wurden. Für diese Umsetzung ist eine zentrale Taktung notwendig, damit wirklich alle Zellen genau gleichzeitig das gleiche tun. Dafür kam die Idee auf, einen zentralen Taktgeber-Arduino zu verwenden, der über die analog-Pins die verschiedenen Schritte über unterschiedlich hohe Spannung angeben sollte. Das erwies sich in der Umsetzung als komplizierter als gedacht und auch als komplizierter als nötig. Es reicht nämlich völlig, einen Taktgeber zu haben, der die ganze Zeit Impulse an alle Zellen-Arduinos schickt und diese dann zählen zu lassen.

Möchte man mit dem Magneten den Lebensstatus der Zellen ändern, also eine neue Startkonfiguration setzen, so muss dieser Prozess unterbrochen werden. Dafür gibt es einen einfachen Kippschalter, der entweder auf „automatik“ (das Programm läuft wie oben beschrieben durch) oder auf „manuell“ geschaltet sein kann. Schaltet man auf manuell um, so wird der aktuelle Taktungsdurchlauf noch abgeschlossen, danach sendet der Taktgeber aber keine weiteren Signale mehr. In diesem Zustand kann man nun mit dem Magneten den Lebensstatus der Zellen verändern. In dem manuellen Modus kann man außerdem über einen Druckknopf einzeln die Generationen durchgehen, was nützlich zum erklären der grundsätzlichen Regeln ist.
Der Code der Zellen besteht also grob aus folgenden Abschnitten:

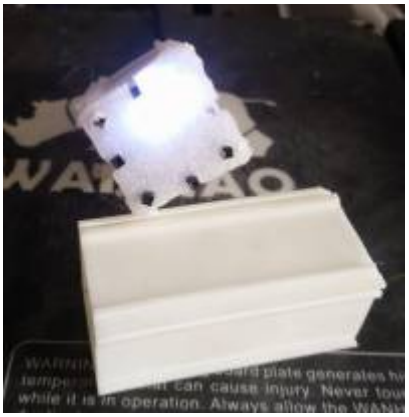
1. Möglichkeit, mit Magnet den Status zu ändern
2. Nachbarn zählen (nach „Uhr“)
3. je nach Anzahl der Nachbarn über eigenen Lebensstatus entscheiden

Für Details und Code des Taktgebers siehe den ordentlich auskommentierten Code.

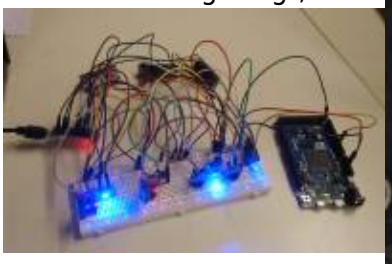
Tipp: Speist man mehreren verbundenen Arduinos nur über eine Verbindung zum Laptop, so kann es passieren, dass die Stromstärke zu gering ist und merkwürdiges, unerklärliches Verhalten entsteht, also die Stromversorgung besser über ein Netzteil regeln.

Ergebnis

Die fertige Zelle sieht am Ende wie folgt aus:



Im Idealfall sollten deutlich kürzere Kabel verwendet werden, die dann an den Arduino gelötet werden, damit die Zellkörper nicht so vollgestopft werden. Darauf haben wir bis jetzt allerdings verzichtet, damit alle Materialien wiederverwendet werden können. Ein komplettes 3×3 Feld konnten wir noch nicht testen, da dafür noch Materialien fehlen, aber ein Test auf einem Steckbrett hat gezeigt, dass alles grundsätzlich funktioniert, wie es soll.



Materialien

Algemein benötigte Materialien:	
Beliebiger Arduino (auch für Stromversorgung verantwortlich)	1
Kippschalter	1
Druckknopf	1

Benötigte Materialien pro Zelle:	
Arduino nano	1
LEDs	1
Halleffekt-Sensoren	1
Kabel:	
male-male (Kommunikation)	4 (ein 3×3 Feld braucht insgesammt nur 27)
Kommunikations anschlüsse (x-female)	8
Kabel für interne Elektronik	3 (Hallsonde) + 2 (LED)
Strom Kabel	2
Takt Kabel	1

Code

Jeder Arduino nano, der in eine Zelle verbaut ist, benötigt folgenden Code:

CellGenome.ino

```
#define LED 11                          //LED/STATUS pin dieser Zelle.
```

```
Gesteuert durch interne Logik
#define sensorPIN A4                //pin zur Senor-Auswerung
#define sensorVALUE analogRead(A4)  //sensor wert
#define stepPIN A5                  //pin zur koordinierung. Gesteuert
durch externe Logik

#define readPIN (stepSTATE+3)%8+2   //formel zur berechnung der Pins,
an denen die Nachbarzellen gelesen werden
#define writePIN stepSTATE+1        //formel zur berechnung der Pins,
an denen Nachbarzelen angeschrieben werden

#define stepINPUT digitalRead(A5)
int stepSTATE = 0;                  //extern angegebener Counter für
den Lese/Schreibe Zyklus
boolean stepTOGGLE = false;         //toggle für den stepSTATE
int jobSTATE = 0;                   //interner counter für den
Lese/Schreibe Zyklus

boolean liveSTATE = false;          //"Lebens"status dieser Zelle
int liveNEIGHBOUR = 0;              //Anzahl der "Lebenden" Nachbar
Zellen

const int threshold = 450;          //Grenzwert ab dem der Sensor
anschlagen soll
boolean sensorTOGGLE = false;       //"sensor zustand"

void setup() {
  //Serial.begin(9600);
  pinMode(LED, OUTPUT);             //LED Pin
  pinMode(sensorPIN, INPUT);        //Pin wo die hall-sonde ausgelesen
wird
  pinMode(stepPIN, INPUT);          //Pin wo durch der Lese/schreibe
Zyklus reinkommt
  pinMode(12, OUTPUT);
  pinMode(12, LOW);
}

void loop() {
  digitalWrite(LED, liveSTATE);                                  //Die LED
spiegelt den Lebensstatus der Zelle wieder



  if(stepTOGGLE == false && stepINPUT == true){                  //wurde
bis jetzt noch kein genarationsimpuls regestriert, geht aber grad einer
ein
    stepTOGGLE = true;                                           //wird
vermerkt das ein impuls eingeht
    stepSTATE += 1;                                              //und
eine Generation hochgegangen
    }
```

```
  if(stepTOGGLE == true && stepINPUT == false){                //ist
vermerkt das ein Impuls registriert wurde, liegt aber keiner an
    stepTOGGLE = false;                                        //wird
dies vermerkt
    }

  if (stepSTATE == 0){                                        //geht die
Zelle nicht die genarationen durch, kann mit dem magneten der status
geändert werden:
    if (sensorVALUE <= threshold && sensorTOGGLE == false){  //Ist der
Magnet nah genug dran und wars bis jetzt aber noch nicht
      sensorTOGGLE = true;                                     //wird
vermerkt das der Magnet nah genug dran ist
      liveSTATE = !liveSTATE;                                  //der
Lebenstatus geändert
      Serial.println(sensorVALUE);
      }
    if (sensorVALUE > threshold){                              //ist der
Magnet wieder weiter weg
      sensorTOGGLE = false;                                    //wird
vermerkt das grad kein Magnet da ist
     }
  }


  if (stepSTATE > 0 && stepSTATE < 9 && jobSTATE == stepSTATE - 1){
//werden die schritte durchgegangen:
    if (stepSTATE == 1){
//wird erst gecheckt ob dies der erste schritt ist
      liveNEIGHBOUR = 0;
//ist dies der fall wird der Nachbar counter auf null gesetzt
    }
    pinMode(writePIN,OUTPUT);
//es wird der Erste pin in der folge als out-put definiert
    digitalWrite(writePIN,liveSTATE);
//und darüber der lebensstatus an den entsprechenden Nachbarn
vermittelt
    pinMode(readPIN, OUTPUT);
    digitalWrite(readPIN,LOW);
//der entsprechend gegenüberliegende pin wird resetet,
    pinMode(readPIN,INPUT);
//als input definiert
    liveNEIGHBOUR += digitalRead(readPIN);
//und sofern dort eine lebende nachbarzelle entdeckt wird, wird dies
vermerkt
    jobSTATE++;
//es wird vermerkt das der erste schritt fertig ist
    Serial.print(liveNEIGHBOUR);
    Serial.print(" Nachbarnzahl, die nach dem pin ");
    Serial.print(readPIN);
```

```
    Serial.println(" gelesen wurde");
  }

  else if (stepSTATE ==9 && jobSTATE == stepSTATE - 1){     //wenn
angefordert und fertig mit Job 2 werden die Regeln angewendet:
    if (liveSTATE == false && liveNEIGHBOUR == 3){          //tote
Zelle mit genau drei Lebenden Nachbarn
      liveSTATE = true;                                     //wird
lebendig
    }
    else if (liveSTATE == true && liveNEIGHBOUR < 2){       //Lebende
Zelle mit weniger als zwei Lebenden Nachbarn
      liveSTATE = false;                                    //Stirbt
    }
    else if (liveSTATE == true && liveNEIGHBOUR > 3){       //Lebende
Zelle mit mehr als 3 Lebenden Nachbarn
      liveSTATE = false;                                    //Stirbt
    }

    Serial.println();
    Serial.print("Gesammt zahl lebender Nachbarn:");
    Serial.println(liveNEIGHBOUR);
    Serial.println();
    jobSTATE = 0;                                            //Fertig
mit allen Jobs,
    stepSTATE = 0;                                           //wartet
auf erste Aufgabe
    }

}
```

Der Taktgebende Arduino wird mit folgendem Code bespielt:

controllpanell.ino

```
#define button 8                                    //Der druckknopf
für's schritweise fortschreiten der Genarationen wird an diesen Pin
angeschlossen
#define Switch 7                                    //Der Kippschalter,
mit dem zwischen "Manuell" und "Automatisch" gewechselt werden kann,
wird an diesen Pin angeschlossenangeschlossen
#define PulseOut A5                                 //Output Pin für die
Pulse
#define Delay 5                                    //verzögerung zwischen
den einzelnen Pulsen (ein Genarationswechsel braucht 16*Delay
millisekunden)

boolean stepTOGGLE = false;                         //toggle mit dem ein
Knopfdruck eine funktion bloß einmal ausführt, bis der knopf das
nächste mal gedrückt wird
```

```
void setup() {
  Serial.begin(9600);
  pinMode(PulseOut, OUTPUT);
  pinMode(button, INPUT);
  pinMode(7, INPUT);
}

void loop() {
  if(digitalRead(Switch)== true){                //wenn der Schalter
umgelegt ist
    for (int i = 1; i <= 9; i++ ){               //wird acht mal
      digitalWrite(PulseOut,HIGH);               //der Takt pin an,
      delay(Delay);                              //nach kurzem Delay
      digitalWrite(PulseOut,LOW);                //aus geschaltet.
      delay(Delay);                              //dies wird nach
kurzem delay wiederholt
    }
  }

  else{                                          //ist der schalter
nicht umgelegt
    if(stepTOGGLE == false && button == true){   //wird nach der selben
Logik wie im code "CellGenome" getestet ob gerade ein Signal
eingegangen ist (der knopf gedrückt wurde)
      stepTOGGLE = true;
      for (int i = 1; i <= 9; i++ ){             //und 8 impulse mit
entsprechender länge und entsprechenden Pausen abgegeben.
      Serial.println(i);
      digitalWrite(PulseOut,HIGH);
      delay(Delay);
      digitalWrite(PulseOut,LOW);
      delay(Delay);
      }
    }

    if(stepTOGGLE == true && button == false){
      stepTOGGLE = false;
      }
    }
```

Verbesserungsideen

- Kabel anders lösen (großer Kabelsalat)
- nicht jede Zelle mit einem Nano, sondern eigenen Chips designen
- Taktung eleganter lösen (z.B. mit interrupt-Funktion der Nanos)