Gymnasium im Schloss
Schlossplatz 13
38304 Wolfenbüttel

Facharbeit im Rahmen des Seminarfaches „Technische Schaltung“
im 2. Halbjahr des Schuljahres 2020/2021

Oberthema: **Nützliche Automaten mit Mikrocontrollern in Zeiten der Corona-Pandemie**

Thema der Arbeit:

Realisierung einer modellhaften Eingangskontrolle für Restaurants mit einem Mikrocontroller

Verfasser: Kilian Arendt, Daniel Gluch, Nelly Schrader
Fachlehrer: StD Frithjof Hummes/ StD Torsten Micheels
Abgabetermin: 19.03.2021

Inhaltsverzeichnis

1.Einleitung

Das Oberthema dieser Facharbeit ist „praktische Hilfsmittel in der Corona-Krise“. Wir haben uns im Detail vorgenommen, mit unserer Facharbeit den Ablauf beim Betreten von gastronomischen Betrieben hinsichtlich der zeitlichen Dauer und der Einfachheit so effizient wie möglich zu gestalten.

Im Folgenden werden wir unsere Motivation und unsere Zielsetzung näher bringen, gefolgt von der Planung und der Durchführung dieses Projekts.
Genauer gesagt beginnen wir nach unserer Motivation mit einem kleinen Einblick in unsere Gedanken bei der Planung des Projekts und mit einer Auflistung der Bauteile kombiniert mit dem Grund, warum genau wir welches Bauteil gewählt haben. Anschließend werden wir auf die tatsächliche Realisierung eingehen und unsere Durchführung dokumentieren.
Zum Schluss ziehen wir noch ein Fazit, um mögliche Verbesserungen und Probleme näher zu bringen.

1.1 Motivation

Wir leben in einer Pandemie, die von dem einen Tag auf den anderen unser Leben, beziehungsweise unseren Lebensstil verändert hat. Wir selbst haben gemerkt, dass Geschäfte im Moment nicht wirklich wissen, wie sie ihre Eingangskontrolle gestalten sollen, da sie dokumentieren müssen, wann die verschiedenen Kunden angekommen sind, wie viele Personen schon im Geschäft sind und wie die Kunden heißen, beziehungsweise wie man sie während der Coronazeit erreichen kann. Deshalb haben wir uns vorgenommen, etwas daran zu ändern und den betroffenen Betrieben in dieser schwierigen Zeit zu helfen und gleichzeitig den Restaurantbesuch für Kunden so angenehm wie möglich zu gestalten. Aus diesem Grund haben wir uns dazu entschlossen ein System zu entwerfen, das all das tut: Das „**G**lobal **A**utomatic **T**echnical **E**ntrance“ – System, kurz **G.A.T.E.**. Es soll den Kundinnen und Kunden sowie den Geschäftsinhabern dabei helfen, sich so normal wie möglich zu fühlen und ihnen ein bisschen Entlastung im Alltag zu ermöglichen.

1.2 Zielsetzung

Wir wollen am Ende der sechswöchigen Arbeit ein System entwickelt haben, welches die Kontaktdaten sowie die Zeit, in der sich die Person in dem Gebäude aufgehalten hat, speichert und darauf achtet, dass auch nur eine gewisse Anzahl an Personen sich im Gebäude befindet und dabei das Risiko für eine Übertragung des Virus minimiert. Die Läden sollen das **G.A.T.E.** einfach nur in ihren Eingang stellen und sich dann um nichts mehr kümmern müssen.

2.Planung

2.1 Dokumentation der Planungsphase

Als das Thema des Eingangskontrollsystems gefunden war, haben wir uns Gedanken über die Realisierung gemacht. Zuerst haben wir das Projekt in mehrere kleinere Abschnitte unterteilt:

Registrierung; Anmeldung / Abmeldung; Tür-System

Anschließend haben wir uns im Detail diese einzelnen Teilsysteme angeschaut und die Anforderungen dieser Systeme notiert:

Registrierung:

Es sollen alle Benutzer permanent identifizierbar und kontaktierbar gespeichert werden. Außerdem soll jedem Benutzer, der sich registriert, eine Art einzigartiger Schlüssel gegeben werden, mit dem er sich anmelden kann.

Damit würden wir das Problem lösen, dass man seine Daten immer wieder neu angeben müsste. Sobald man sich einmal registriert hat, wären die Daten permanent im System.

Anmeldung / Abmeldung:

Man soll in der Lage sein über die Eingabe seines einzigartigen Schlüssels, die Anmeldung / Abmeldung durchzuführen. Nach der Eingabe sollen mit dem Schlüssel einige logische Tests durchgeführt werden, unter anderem, ob der Benutzer überhaupt im System existiert und Weiteres. Außerdem sollen die infektionsschutztechnischen Bedingungen überprüft werden, wie z. B. die Personenanzahl im Gebäude.

Damit würde man die „Türsteher“ umgehen, die immer aktuell am Eingang stehen müssen und darauf achten, dass nicht zu viele Personen auf einmal im Restaurant sind. Natürlich soll dieses Anmeldesystem auch dazu beitragen, den Aufwand der Datenerfassung zu minimieren. Wie schon gesagt, muss man statt seine gesamten Daten neu anzugeben, nur noch den bei der Registrierung erhaltenen Schlüssel eingeben.

Tür-System:

Sobald eine erfolgreiche Anmeldung / Abmeldung durchgeführt wurde, soll eine Tür, bzw. hier modellhaft eine Schranke, geöffnet werden, die sich nach dem Passieren idealerweise auch noch von alleine schließt. Auch wollen wir den Eingang vom Ausgang trennen, also separate Schranken installieren.

Das würde dabei helfen, die Übertragungschancen des Virus zu minimieren, da physische Kontakte mit der Tür eliminiert werden. Durch das Trennen von Ausgang und Eingang würde eine Art Einbahnstraße realisiert werden, die weiter zum Infektionsschutz beitragen würde.

2.2 Welche Materialien / Bauteile oder auch softwareseitige Funktionen werden benötigt? Warum gerade diese?

ESP32:

Zur Realisierung des Registrierungssystems wollen wir eine Website hosten, auf die man über ein lokales Netzwerk zugreifen können soll. Da wir das Projekt auf Basis von Mikrocontrollern realisieren sollen, bietet sich hierfür die ESP Plattform an, da diese alle Anforderungen dafür erfüllt und auch in der Lage ist, mit weiteren Mikrocontrollern zu kommunizieren:

> *„Die ESP32-Familie weist diverse Peripherieschnittstellen auf, unter anderem stehen mehrere UARTs, SPI-, CAN- und I²C-Schnittstellen, ein integriertes Wireless Local Area Network (WLAN nach IEEE 802.11 b/g/n), das auch Bluetooth unterstützt, und eine Ethernet-Schnittstelle über das Media Independent Interface (MII) zur Verfügung.“*[1]

DS3031 (RTC):

Um zu speichern, wann ein Kunde im Geschäft war (damit wir im Nachhinein feststellen können, wer im Falle einer Infektion mit dem Infizierten im Gebäude war) wird eine RTC (Real Time Clock) benötigt. Wir haben genau diese genommen, da sie in den Arduino-Kits, die wir vorher bestellt haben, dabei war.

AZDelivery MicroSD Card Adapter + MicroSD Karte:

Zur tatsächlichen Speicherung der Daten wird eine MicroSD Karte per Adapter benutzt. Wir haben uns für diese Speichermethode entschieden,

1 Autor unbekannt, https://de.wikipedia.org/wiki/ESP32, Zugriff am 17.03.2021

da wir keine direkte Möglichkeit hatten, permanent viele Daten auf den Mikrocontrollern zu speichern. Eine SD-Karte mit Adapter zu benutzen, war für uns dann die nächste Lösung. Außerdem haben wir dazu viele Ressourcen und Hilfen im Netz gefunden, was oft darauf hindeutet, dass die Methode gut geeignet ist.

5x4x4 Matrix Eingabefeld:

Diese Zahlenfelder werden benötigt, um den Schlüssel bei der Anmeldung auszulesen. Wir benutzen diese, da sie auch schon in unseren Starter-Kits vorhanden waren.

Arduino Nano:

Diese Mikrocontroller benutzen wir, um die Eingaben der Zahlenfelder auszulesen. Wir können die Zahlenfelder nämlich nicht direkt an den ESP anschließen, da sie zu viele Pins benutzen. Deswegen lassen wir sie von den Nanos auslesen und fragen das Ergebnis dann über I²C ab. Darüber hinaus dienen sie als Steuereinheiten für die Servomotoren.

LCD-Display:

Für ein besseres visuelles Feedback zeigen wir die aktuelle Eingabe auf LCD-Displays an. Die Displays, die bei uns in den Starter-Kits waren, benötigen aber sehr viele Pins, um angesteuert zu werden und da wir 2 benötigen müssen wir diese anderweitig betreiben.

LCD I²C Backpack:

Diese Adapter reduzieren die benötigten Pins, im Grunde genommen von 16 komplett belegten Pins, auf 4 Anschlüsse an Kommunikations- und Strompins, die auch noch für weitere Verbindungen verwendet werden können, was das Anschließen sehr viel einfacher gestaltet.

Servomotoren:

Um die Schranken zu öffnen, wollen wir Servomotoren benutzen. Wir hatten zu Beginn die Auswahl zwischen Stepper-Motoren und Servomotoren, die beide in den Arduino-Kits waren. Da die Servomotoren einfacher anzusteuern waren, haben wir uns für diese entschieden.

HC-SR04 (US-Sensor):

Um zu merken wann der Kunde die Schranke passiert hat, wollen wir einen Ultraschall-Abstandssensor benutzen. Auch hier benutzen wir dieses Modell, da es schon in unserem Besitz war.

Schranken:

Nach einer kurzen Bedenkzeit haben wir den Entschluss gefasst, die Schranken 3D drucken zu lassen. Das hat den Hintergrund, dass der Kunststoff leichter ist als andere Materialien, wodurch wir uns erhoffen, dass die Servomotoren stark genug sind, diese zu bewegen.

Softwaretechnische Funktionen:

Zuerst einmal benötigen wir einen Code-Generator, der die einzigartigen Schlüssel für die Benutzer generiert. Dieser soll außerdem kontrollieren, dass der generierte Code nicht schon existiert.

Außerdem wird eine Implementierung unseres Automaten benötigt. Daneben müssen wir eine Website auf dem ESP32 hosten und uns ein „Datenbank“-Modell auf der SD-Karte zur Datenspeicherung überlegen.

2.3 Wie / wo haben wir uns was besorgt?

Wie oben schon mehrfach angesprochen, hatten wir schon viele der Bauteile aus den Arduino-Kits, die wir uns zu Beginn des Semesters anschaffen sollten. Den ESP32, das microSD-Card Modul, die Arduino Nanos sowie die LCD-Adapter haben wir uns zu Beginn über Amazon bestellt. Das Holz für das Gehäuse und die Farbe haben wir im nächsten Baumarkt gekauft, was sich tatsächlich als relativ schwierig herausgestellt hat, da Baumärkte zu dieser Zeit nur für Gewerbetreibende geöffnet hatten.

3.Realisierung

3.1 Was haben wir, wie, umgesetzt ?

Allgemein ist das System in verschiedene Mikrocontroller aufgeteilt. So hat man einmal einen ESP32, welcher als „Master“ gilt und zusätzlich noch zwei Arduino Nanos, welche als deren „Slaves“ agieren. Der ESP32 ist dabei für die ganze Zusammenarbeit zwischen den Systemen

zuständig, während die Nanos zum Auslesen von Daten und dem Durchführen von Aktionen benötigt werden.

3.1.1 Registrierung

Website:

Das Eintragen der Personen läuft über einen auf dem ESP32 gehosteten asynchronem Webserver, über den man sich per vom ESP32 ausgehenden WLAN (ESP32-Acces-Point) verbinden kann. Auf der dann aufrufbaren Website kann man anschließend seine Daten angeben, welche umgehend in einer eigens dafür entworfenen Datenbank mit einem für jede Person einzigartig generierten Code gespeichert werden.
Dieser Code wird nach der Registrierung der Person auf der Website angezeigt.

```
// HTML-Code der Website
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
  <title>ESP Input Form</title>
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  </head><body>
  <form action="/get">
    Vorame: <input type="text" name="name"><br>
    Nachname: <input type="text" name="nname"><br>
    EMail: <input type="text" name="email"><br>
    <input type="submit" value="Submit">
  </form><br>
</body></html>)rawliteral";


  // Bereitstellen der Website
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });
```

SD-Karte:

Die Datenbank wird auf einer SD-Karte angelegt, welche über einen Adapter mit dem ESP32 über SPI verbunden ist, so dass dieser direkt die SD-Karte beschreiben und auslesen kann. Zusätzlich war geplant, dass in dieser Datenbank per Unixzeit vermerkt wird, wann die Person zuletzt und für wie lange sie im Restaurant war und ob Sie sich gerade im Gebäude befindet.

```
    // Auf SD-Karte schreiben
    write_string_to_EOF(current_msg);

// Diese Funktion nimmt eine Zeichenkette als Parameter und
schreibt diese
```

```
// ans Ende der aktuellen Datenbankdatei
void write_string_to_EOF(String to_write) {
  File myFile;
  // Welche der Dateien existiert gerade
  if (SD.exists("/data.txt")) {
    myFile = SD.open("/data.txt", FILE_WRITE);
  }
  else if (SD.exists("/data2.txt")) {
    myFile = SD.open("/data2.txt", FILE_WRITE);
  }
  else {
    myFile = SD.open("/data.txt", FILE_WRITE);
  }
  // Schreiben
  if (myFile) {
    myFile.seek(myFile.size());
    myFile.println(to_write);
    myFile.close();
  }
}
```

Code:

Jeder Code besteht aus 5 Ziffern, wobei die letzte Ziffer als Kontrollziffer dient. Bei einem Registrierungsprozess wird jedes Mal eine Methode aufgerufen, die einen neuen Code generiert, der noch keinem Benutzer zugeordnet ist. In dieser werden die ersten vier Ziffern per Zufall ermittelt, anschließend wird die Kontrollziffer erzeugt, indem so lange die Quersumme der ersten Ziffern gebildet wird, bis diese schließlich einstellig ist. Zuletzt wird jeder existierende Code in der Datenbank mit dem erzeugten Code verglichen, um die Einzigartigkeit der Codes zu gewährleisten. Sollte nämlich der gerade erzeugte Code schon existieren, wiederholt sich der Prozess, bis ein noch unbenutzter Code gefunden wurde.

```
// Diese Funktion generiert einen theoretisch möglichen Code der
eine korrekte Quersumme hat
long genCode() {
  long tenthousand = 10000;
  long random1 = random(1, 10);
  long random2 = random(1, 10);
  long random3 = random(1, 10);
  long random4 = random(1, 10);
  long finalnumber = random1 + random2 + random3 + random4;
  while(finalnumber > 9) {
    int finalhilfe = finalnumber /10;
    finalnumber = finalnumber - (finalhilfe*10);
    finalnumber = finalnumber + finalhilfe;
  }

  long codehilf = random1*tenthousand + random2*1000 + random3*100
+ random4*10 + finalnumber;
  return codehilf;
}
```

```
  // Erster Code (für die nächste Registrierung) wird schonmal
generiert
  randomSeed(analogRead(0));
  do {
     number_to_compare = (String)genCode();
  } while(code_already_exists(number_to_compare));
  current_msg = "" + number_to_compare + "|";
```

Datenbanksystem:

Das genaue System zur Datenspeicherung besteht eigentlich nur aus einer Text-Datei, in der in jeder Zeile ein Nutzer mit all seinen Daten vermerkt ist. Dabei sieht die Formatierung so aus:

Code|Vorname|Nachname|E-Mail|Letztes Betreten|Letztes Verlassen|Aktuell Drinnen?

Dabei werden natürlich all diese Felder mit den entsprechenden Daten ausgefüllt. Die „|“ dienen hier als Trennzeichen, dass man sozusagen weiß, wann ein neuer Abschnitt beginnt.

```
    // Kompletten Datenbank-String zusammenbauen
    current_msg += inputvName;
    current_msg += '|';
    current_msg += inputnName;
    current_msg += '|';
    current_msg += inputEmail;
    current_msg += '|';
    current_msg += (String)zeit;
    current_msg += '|';
    current_msg += (String)zeit;
    current_msg += "|0";
```

Code:

Der bei der Registrierung erhaltene einzigartige Schlüssel zur Identifizierung jedes einzelnen Benutzers. Er wird später zur Anmeldung verwendet.

Vorname, Nachname, Email:

Die bei der Registrierung angegebenen persönlichen Daten.

Letztes Betreten / - Verlassen:

Die Zeitpunkte, wann der Benutzer zuletzt die Eingangs- bzw. Ausgangsschranke passiert hat. Nach der Registrierung ist der Startwert erst einmal gleich dem Zeitpunkt der Registrierung. Diese Daten sollen, im Falle einer Infektion, in diesem Restaurant später

dazu benutzt werden, alle Personen ausfindig zu machen, die zeitgleich mit der infizierten Person im Gebäude waren.

Aktuell Drinnen:

Ein Wahrheitswert, ob der Nutzer sich gerade im Restaurant befindet oder nicht.

3.1.2 Anmeldeprozess

Code-Eingabe:

Um als Kunde nun ins Restaurant zu gelangen, muss man seinen Code eingeben. Dies erfolgt über ein Tastenfeld. Die Eingaben werden parallel auf einem LCD-Display visualisiert.

Abb. 1: Tastenfeld und LCD-Display

Das Tastenfeld ist dabei an einem Arduino Nano angeschlossen, welcher die eingegeben Zahlen erkennt und in einem Array speichert. Dieses Array wird alle 0.5 Sekunden von dem ESP32 abgefragt und dann vom Arduino Nano an diesen übermittelt. Mit dem Druck der „*-Taste“ lässt sich das letzte Zeichen löschen und mit dem Benutzen der „#-Taste“ werden die Daten für den Esp32 zum Verarbeiten freigegeben.

```
    char lastCode[6] = {'#', '#', '#', '#', '#', 'n'};


  // Letzte Tastenfeld-Eingabe auslesen
  char key = keypad.getKey();
  if (key) {
    // Wenn Buchstabentasten, ignorieren
    if (key == 'A' || key == 'B' || key == 'C' || key == 'D') { //
                                                    Unbelegte Keys
      ;
    } else if (key == '#') {                    //wenn Eingabe key(#)
                                                  gedrückt
      //Eingabe wird überprüft:
      lastCode[5] = 'y';
```

```
    } else if (key == '*') {          //wenn löschen-key(*) gedrückt
                                        wird
      //löscht letzte Eingabe
      for (int i = 0; i < 5; i++) {
        if (eingabe[i] == '#' && i != 0) { //löscht die letzte
                                             eingegebene Zahl
          eingabe[i - 1] = '#';
          break;
        }
        if (i == 4) {                  //wenn alles belegt ist,
                                        löscht es die letzte zahl
          eingabe[i] = '#';
          arrayvoll = false;
        }
      }

    } else {                          //wenn Zahl eingegeben wurde:
      if (arrayvoll) {                //löscht das komplette array
                                        wenn es voll war
        cleararray();
      }

      for (int i = 0; i < 5; i++) { //setzt die zahl an nächste
                                        freie Stelle
        arrayvoll = true;
        if (eingabe[i] == '#') {
          eingabe[i] = key;
          if (i != 4) {
            arrayvoll = false; // array ist noch nicht komplett
                                  gefüllt
          } else {
            arrayvoll = true;  //array ist nun komplett gefüllt
          }
          break;
        }
      }
    }
  }
  for (int i = 0; i < 5; i++) {
    lastCode[i] = eingabe[i];
  }
  delay(50);
```

Die Freigabe erfolgt dabei über ein Anhängsel im Array, so bedeutet der Anhang „n“, dass das Array noch nicht zur Überprüfung freigegeben ist (der Benutzer also gerade noch am Eingeben ist) und der Anhang „y“, dass die Daten so wie sie sind überprüft werden sollen (der Kunde also die „#-Taste“ gedrückt hat). Die Anzeige auf dem LCD-Display erfolgt allerdings über den ESP32, da der Arduino als „Slave“ fungiert und nur abgefragte Daten übermitteln kann.

Zusammengefasst fragt der ESP32 die Daten des Arduinos ab und überträgt diese dann auf die Displays. Die Kommunikation erfolgt über das Inter Integrated Circuits Protokoll (I^2C).

Code-Validierung:

Wurde nun die „#-Taste“ gedrückt und die Daten wurden zur Verarbeitung freigegeben, wird über die implementierte Turing-Maschine (siehe Abschnitt 3.3) zuerst überprüft, ob der Code so überhaupt stimmen kann, also ob die Prüfziffer zu den ersten 4 Ziffern passt.

```
// Prüfung durch Turingmaschine
if (tm(temp)) {
```

Kann der Code stimmen, wird auf der Datenbank geschaut zu welcher Person der Code gehört, bzw. ob er überhaupt schon vergeben ist.

```
// Prüfung ob der Nutzer existiert
if (code_already_exists(received_number.substring(0, 5))) {
```

Anschließend wird überprüft, ob sich die Person momentan schon im Restaurant befindet oder nicht.

```
// Prüfung ob der Nutzer schon im Gebäude ist
if (!is_inside(received_number.substring(0, 5))) {
```

Ist die Person nicht im Restaurant und es ist auch noch Platz im Gebäude

```
// Prüfung ob noch Platz im Gebäude ist
if (person_counter < MAX_PERSONS) {
```

wird ein Signal vom ESP32 an den entsprechenden Arduino geschickt, damit dieser seine Schranke öffnet. Außerdem wird der letzte Eintrag in der Datenbank beim entsprechenden Benutzer von falsch (0) auf wahr (1) aktualisiert: Er befindet sich jetzt offiziell im Gebäude.

```
//Schranken öffnen
digitalWrite(digComEingang, HIGH);
delay(1000);
digitalWrite(digComEingang, LOW);
delay(1000);


// Datenbank aktualisieren
toggle_user(received_number.substring(0, 5));
```

Schrankensystem:

Die Schranke ist dabei per Servomotor an den Arduino angeschlossen. Bekommt der Arduino das Signal vom Esp32, öffnet er die Schranke und wertet die Daten eines angeschlossenen Abstandssensors aus. Der Abstandssensor sitzt dabei direkt unter der Schranke. Fällt der gemessene Wert des Abstandssensors unter 30cm, bedeutet dies, dass die Person die Schranke passiert hat, wodurch die Schranke 2 Sekunden später schließt.

```
// Gucken ob der ESP will dass die Schrank aufgemacht wird
if (digitalRead(comPin)) {
  turauf = true;
}
if (turauf) {
  opendoor();
}
```

Möchte man das Gebäude später wieder verlassen, ist der Vorgang ähnlich, nur dass nun nach der Code-Eingabe auf dem inneren Zahlenfeld in der Datenbank überprüft wird, ob die Person schon draußen ist, oder nicht.

3.2 Probleme und ihre Lösungen

Während der Realisierung des Projekts sind wir auf mehrere Probleme gestoßen. Im Folgenden erläutern wir diese und wie wir sie letztendlich gelöst haben.

Nachdem die Implementierung der Website eigentlich problemlos verlaufen ist, ging es um die Speicherung der auf der Website erfassten Daten. Zu diesem Zeitpunkt hatten wir noch geplant, einen Arduino Uno als „Master“ ins Zentrum der ganzen Schaltung zu setzen. Dann hätte der ESP32 die erfassten Nutzerdaten auf die SD-Karte geschrieben und der Uno sie, wenn nötig, gelesen. Das Problem dabei war, dass dieses SD-Karten Modul keine geteilte Nutzung der Speicherkarte erlaubt. Um das realisieren zu können, hätte man irgendwie zwischen den Mikrocontrollern zusätzlich noch kommunizieren müssen, wann und wer mit der Karte reden darf, da wir sonst immer die Pins und die Stromzufuhr für den Adapter deaktivieren müssen. Das erschien uns aber dann zu komplex für dieses Projekt. Deshalb (und auch noch wegen einem anderen Problem, welches im Folgenden noch erwähnt wird) haben wir uns dazu entschieden, den Arduino Uno komplett raus zunehmen und den ESP32 sozusagen zum Kern des Systems zu machen. So muss nur der ESP auf die SD-Karte zugreifen und damit war das Problem gelöst.

Ein weiteres Problem hing mit der Aufgabenteilung auf die Arduino Nanos zusammen. Wie schon vorher erwähnt, mussten wir das Einlesen der Anmeldedaten auf je einen Arduino Nano pro Eingabefeld aufteilen, da die Tastenfelder beide sehr viele Pins zum Anschließen benötigen. Um diese Daten dann aber nutzbar verarbeiten zu können, mussten wir sie auf den

Kern-Controller des Systems bringen. Das wollten wir auch über eine I²C Verbindung machen, da sowieso schon einiges darüber kommuniziert wird. Zu diesem Zeitpunkt wollten wir auch noch den Arduino Uno als „Master“ benutzen. Uns ist dann aber aufgefallen, dass dieses System generell nicht in Kombination mit dem ESP32 umsetzbar ist, da dieser hardwaretechnisch eine Einbindung als „Slave“ im I²C-Protokoll verbietet. Dann hätten wir aber 2 „Master“ gehabt, was selten und auch nur schwer funktioniert (es ist, wie gesagt, nur auf einen ausgelegt). Das war dann der zweite Grund, warum wir uns dazu entschieden haben, den Arduino Uno nicht mehr zu verwenden und eigentlich die komplette Logik auf den ESP32 zu packen.

Später sind wir dann noch dem Problem begegnet, dass unsere geplanten Servomotoren nicht genug Kraft hatten, um die extra für das Projekt 3D-gedruckten Schranken zu bewegen. Deshalb wollten wir probieren, auf die in unseren Starter-Kits enthaltenen Stepper-Motoren zu wechseln. Diese hatten aber einen fast runden Anschluss, was sie nicht wirklich nutzbar machte. Wir konnten sie nicht befestigen, weil wir auch keinerlei Aufsatz hatten. Also sind wir wieder zurück zu den Servomotoren gewechselt. Diesmal 2 pro Schranke, um endlich das Gewicht zu schaffen. Das hat zwar dann tatsächlich funktioniert, aber dann ist uns nach einiger Zeit beim Testen einer der Servos kaputt gegangen. Deshalb mussten wir die Befestigung noch einmal komplett neu bauen, um das Gewicht der Schranke jetzt doch mit nur einem Servomotor zu bewegen. Nachdem wir den Winkel angepasst hatten und die Befestigung verstärkt war, hat es dann doch mit nur einem Servomotor funktioniert. Das haben wir dann auch auf beide Seiten übertragen, damit es symmetrisch war, obwohl wir eigentlich noch 3 Servomotoren zur Verfügung hatten.

Im weiteren Verlauf hatten wir dann noch Probleme mit einzelnen Anschlüssen, die aber einfach an einer leicht falschen Verkabelung, aufgrund von unklaren Beschriftungen am ESP32 lagen. Die Beschriftungen der digitalen Pins war an einer Stelle um eine Position verschoben. Das herauszufinden hat uns gute 4 Stunden gekostet, da wir uns nicht erklären konnten, was nicht funktioniert (weil wir ja auch eigentlich alles richtig gemacht hatten). Aber es war dann letztendlich leicht zu beheben, nachdem wir wussten, was das Problem war.

Zwischenzeitlich kam auch die Frage auf, wie wir denn die Daten auf der SD-Karte bearbeiten wollen, wenn, zum Beispiel der Status einer Person (ob sie im Gebäude ist oder nicht) aktualisiert werden soll, da man eigentlich nur lesen und schreiben und nicht direkt Zeichen ersetzen / bearbeiten kann. Unsere Lösung war dann, bei jeder Aktualisierung die Datei neu zu schreiben. Dafür „wechseln“ wir sozusagen immer zwischen zwei verschiedenen Dateien hin und her. Wenn nur Datei 1 existiert, kopieren wir den gesamten Inhalt in Datei 2 und nehmen die gewünschten Änderungen vor. Anschließend wird die alte Datei 1 gelöscht, sodass man beim nächsten Bearbeitungsprozess weiß, dass man (in diesem Fall) aus Datei 2 Datei 1 machen soll.

3.3 Modellierung als Automat

3.3.1 Entwicklung in AutoEdit

Der Automat bildet bei uns nicht das ganze Projekt ab, da dies viel zu komplex und auch nicht im Sinne der Aufgabenstellung wäre. Deshalb bilden wir nur einen Teil ab. Hier: Die Kontrolle, ob ein eingegebener Code überhaupt richtig sein kann. Die Codes haben nämlich einen bestimmten Aufbau und sind nicht komplett zufallsgeneriert. Zuerst kommen zwar 4 willkürliche Ziffern, doch dann folgt die Quersumme aus diesen 4 Ziffern (sollte diese Quersumme zweistellig sein, bilden wir wieder die Quersumme aus ihr). Unser Automat prüft, ob wirklich 5 Ziffern eingegeben wurden und gleicht dann die Quersumme mit der fünften Ziffer ab. Wenn alles stimmt, erreichen wir einen Endzustand, sonst nicht.

Um den optimalen Typ für den Automaten zu finden. gab es folgende Überlegungen:

- Der Automat muss nicht mit mehreren Möglichkeiten arbeiten und soll mit den gleichen Eingaben immer auf ein- und dasselbe Ergebnis kommen. Also fallen alle *nicht-deterministische* Typen weg.
- Außerdem muss der Automat den Code irgendwie speichern können, um damit zu arbeiten. Damit dabei nicht extrem viele Zustände benötigt werden, bleiben nur noch der *Kellerautomat* und die *Turingmaschine*

- Zuletzt müssen wir Zahlen addieren, was effektiv nicht mit einem Kellerautomaten machbar ist. Also bleibt noch die *Turingmaschine*.

Dadurch, dass wir die *Turingmaschine* benutzen müssen, können wir aber die Zahlen nicht „live“ einlesen, da die Turingmaschine schon direkt die komplette Eingabe benötigt, um dann damit zu arbeiten. Das heißt, wir werden die physische Eingabe des Codes später separat machen müssen und dann, beispielsweise wenn die „Enter“-Taste gedrückt wird, einfach die letzten 5 Ziffern nehmen und sie dem Automaten zur Verfügung stellen.

Jetzt sieht der grobe Ablauf des Automaten so aus:

1. Wir prüfen erst, ob da wirklich 5 Ziffern stehen.
2. Die Quersumme der ersten 4 Stellen wird gebildet.
3. Die Quersumme wird mit der 5ten Stelle verglichen.

Im Detail sieht das Ganze so aus:
Um zu prüfen, ob wirklich genau 5 Ziffern eingegeben wurden, gehen wir einfach 5 Übergänge lang nach rechts, unter der Bedingung, dass da eine der Ziffern 0-9 steht und erwarten dann am Ende ein „#“ (unser Band-Vorbelegungszeichen). Wird dieses „#“ da gefunden schreiben wir stattdessen ein „|“ als Trennzeichen für später.

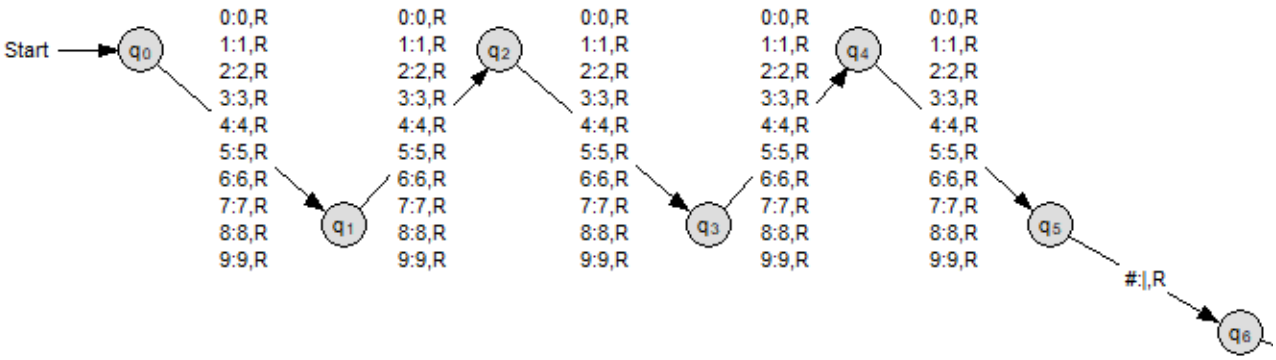

Abb. 2: Validierung der Länge der Eingabe

Nun soll die Quersumme aus den ersten 4 Ziffern gebildet werden. Wir haben uns dazu entschieden, die Quersumme zuerst in binär zu bilden, da wir uns damit auch schon mehr im Informatik-Unterricht beschäftigt hatten und es uns trivialer erschien. Damit wir das Ergebnis in binär speichern können bereiten wir hier erst einmal einen 6-stelligen Bereich vor (6 Stellen, da die maximale Quersumme 9+9+9+9=36 ist und da $36_{10} = 100100_2$ Wir benötigen also maximal 6 Stellen)

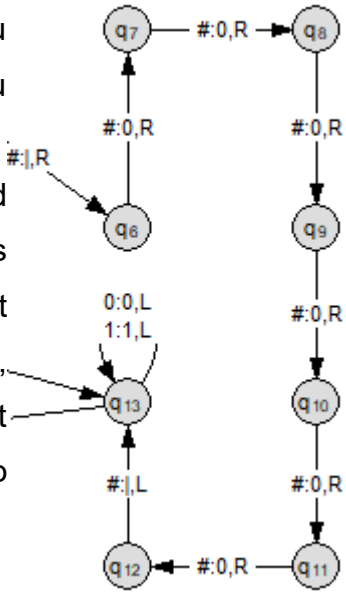

Abb. 3: Bereich für die binäre Quersumme vorbereiten

Um einen kleinen Überblick zu geben, hier ein Bild von einem beispielhaften Band der Turingmaschine bisher:

#	1	2	3	4	1	|	0	0	0	0	0	0	|	#	#

Abb. 4: Beispiel 1 von dem Band der Turingmaschine

Jetzt beginnt die Bildung der Quersumme. Dafür wird die Leseposition auf dem Band bis zum Trennzeichen zwischen Code und Summe bewegt. Dann wird die fünfte Ziffer übersprungen, da sie für die Bildung der Quersumme egal ist. Anschließend wird von der folgenden Zahl 1 subtrahiert. Wenn dort eine 0 steht, gehen wir weiter zur nächsten Zahl usw. Nachdem die Ziffer um den Wert 1 dekrementiert wurde, wird die Leseposition nach ganz rechts zum äußeren Trennzeichen bewegt. Dann bewegen wir uns nach links durch die Binärzahl durch und erhöhen sie dabei um 1. Schlussendlich wird die Leseposition wieder ans mittlere Trennzeichen bewegt. Dieser Prozess wird so lange wiederholt, bis die Stellen 1 bis 4 im Code zu 0 geworden sind.
Hier der Ausschnitt aus dem tatsächlichen Automaten:

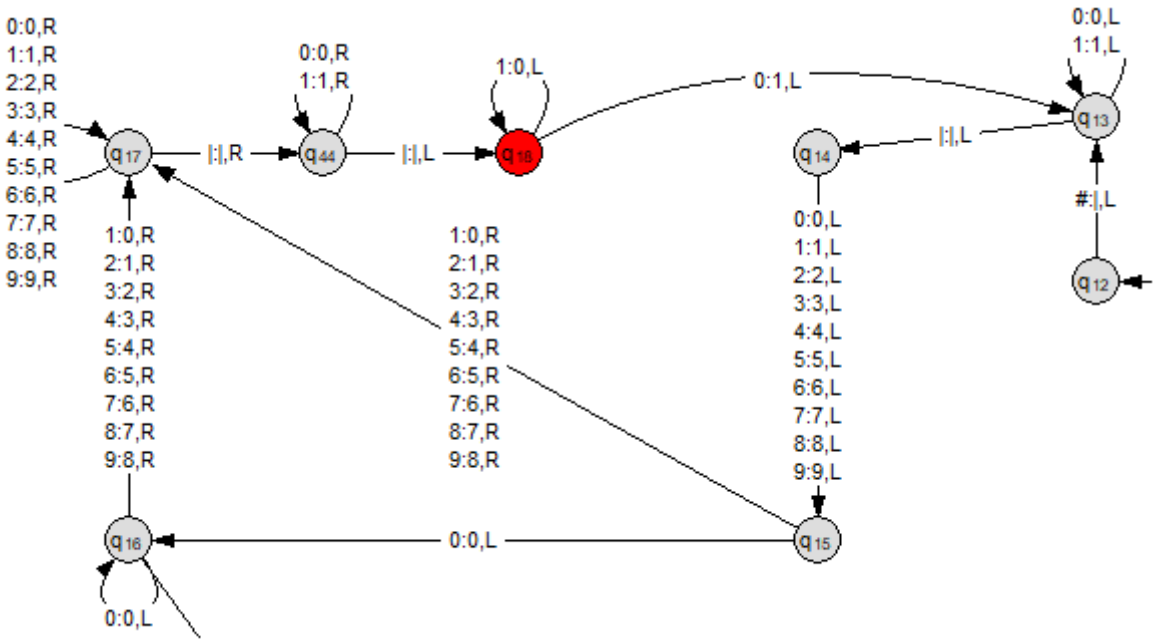

Abb. 5: Bildung der binären Quersumme

Das beispielhafte Band mit der Eingabe „12341“ sähe jetzt so aus:

#	0	0	0	0	1	|	0	0	1	0	1	0	|	#	#

Abb. 6: Beispiel 2 von dem Band der Turingmaschine

Um sicherzustellen, dass die erhaltene Quersumme einstellig ist, müssen wir sie jetzt wieder zurück ins Dezimalsystem konvertieren. Dafür bewegen wir den „Cursor“ erst mal ganz nach rechts, um dort Platz für die zu bauende Dezimalzahl zu schaffen.

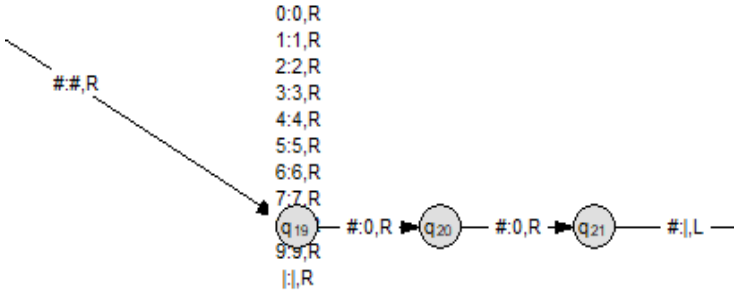

Abb. 7: Bereich für die dezimale Quersumme vorbereiten

#	0	0	0	0	1	|	0	0	1	0	1	0	|	0	0	|	#

Abb. 8: Beispiel 3 von dem Band der Turingmaschine

Für die tatsächliche Umrechnung wird die Leseposition bis zum Trennzeichen zwischen Binär- und Dezimalsumme bewegt. Dann subtrahieren wir immer 1 von der Binärzahl und addieren im Anschluss 1 zur Dezimalzahl, bis der Bereich der Binärzahl nur noch mit 0 gefüllt ist.

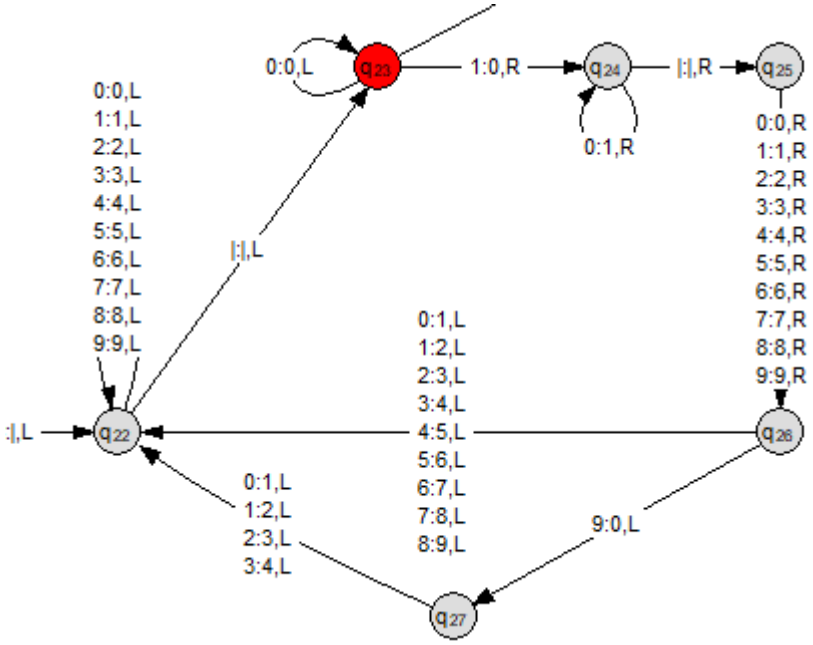

Abb. 9: Umwandlung der binären Quersumme ins Dezimalsystem

Auf dem Band sieht man jetzt im dritten Zahlenbereich die 10 als Quersumme von 1234:

#	0	0	0	0	1	|	0	0	0	0	0	0	|	1	0	|	#

Abb. 10: Beispiel 4 von dem Band der Turingmaschine

Der letzte Schritt ist, hier jetzt nochmal die Quersumme aus der eventuell zweistelligen Quersumme zu bilden. Dafür wird, solange die Zehnerstelle nicht 0 ist, immer 1 von der Quersumme subtrahiert und anschließend 1 zur Einerstelle addiert. Sobald die Zehnerstelle 0 ist, haben wir in der Einerstelle die fertige Prüfziffer.

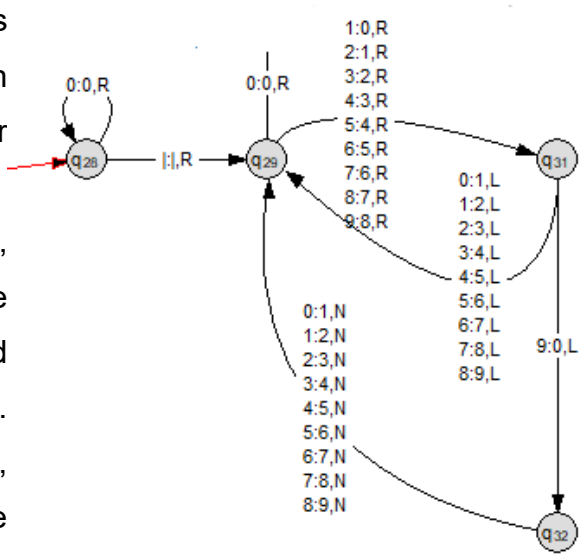

Abb. 11: Sicherstellen, dass das Ergebnis einstellig ist

#	0	0	0	0	1	|	0	0	0	0	0	0	|	0	1	|	#

Abb. 12: Beispiel 5 von dem Band der Turingmaschine

Zu guter Letzt erfolgt der Vergleich der beiden Prüfziffern. Hierfür bewegt sich der „Cursor“ immer zwischen den beiden hin und her und subtrahiert abwechselnd von ihnen immer je 1. Sollte die eingegebene Prüfziffer zu früh 0 erreichen, bricht der Automat ab → der Code kann gar nicht richtig sein. Wenn die ausgerechnete Prüfziffer 0 erreicht, wird als allerletztes überprüft, ob auch die eingegebene Prüfziffer im gleichen Schritt 0 geworden ist. Ist das der Fall, kann der Code stimmen und der Automat erreicht einen Endzustand.

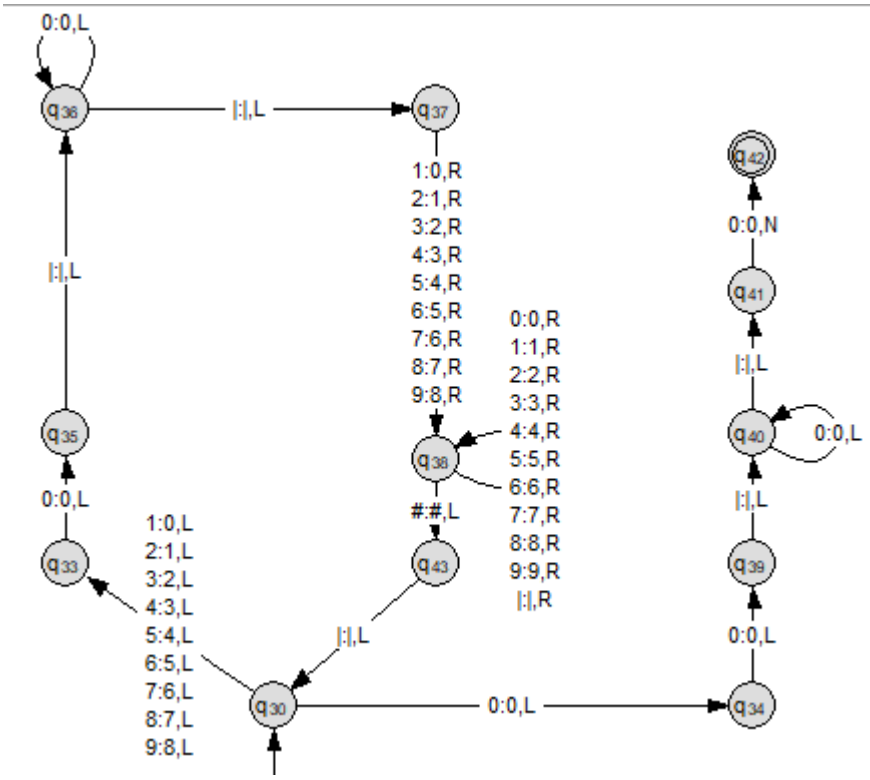

Abb. 13: Die Prüfziffer wird überprüft

In unserem Beispiel mit der Eingabe 12341 ist die 1 die richtige Prüfziffer, weshalb auf dem Band zur gleichen Zeit vorne und hinten Nullen entstehen:

#	0	0	0	0	0	|	0	0	0	0	0	0	|	0	0	|	#

Abb. 14: Beispiel 6 von dem Band der Turingmaschine

Testphase: Der Automat funktioniert, er erkennt also, theoretisch richtige Codes, als richtig und theoretisch falsche Codes, als falsch.
Achtung: Ab einer bestimmten Komplexität braucht der Automat länger als 1000 Schritte. Da AutoEdit ab 1000 Schritten abbricht, funktioniert die Simulation hier nur bis zu einem bestimmten Punkt. Die Implementierung auf dem Arduino besitzt aber die Kapazitäten, alle möglichen Codes zu testen.

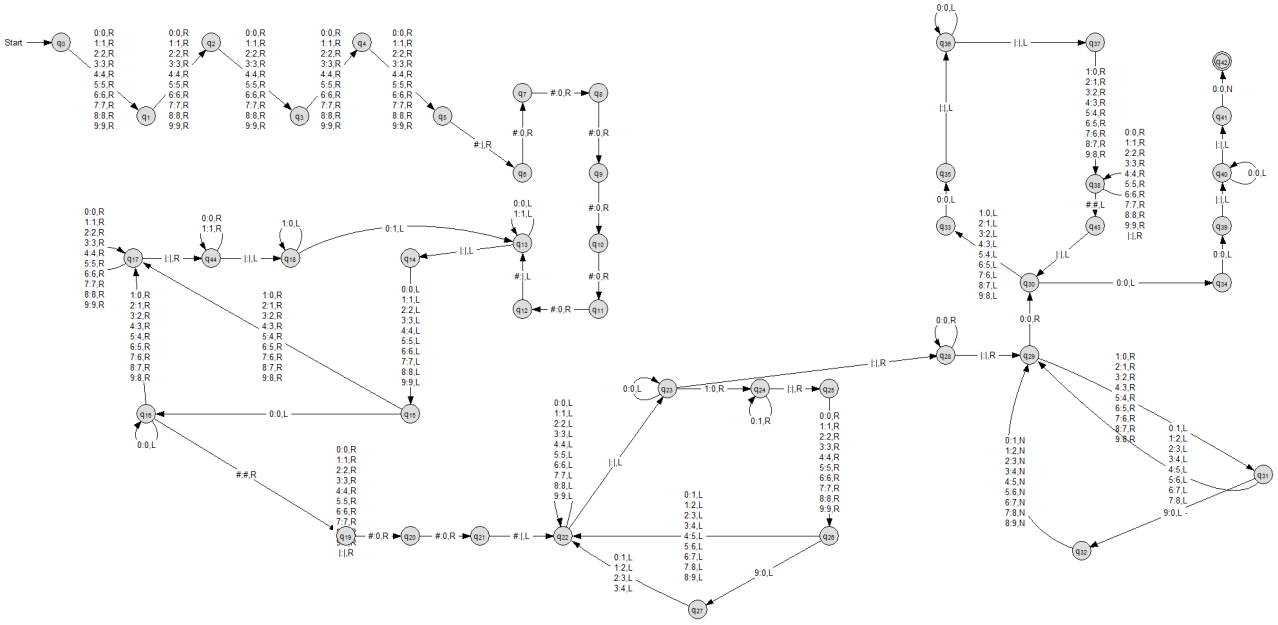

Abb. 15: Gesamter Automat

3.3.2 Übertragung zu Arduino

Nachdem wir den Automaten fertig in AutoEdit geplant und entworfen hatten, mussten wir ihn auch noch in Arduino implementieren, damit wir ihn später tatsächlich benutzen können. Der Aufbau sieht dabei wie folgt aus: Wir haben ein Char-Array, mit dem das Band des Automaten simuliert werden soll. Außerdem gibt es eine Zähler-variable „Position“, die sich modellhaft merkt, an welcher Stelle sich der „Lesekopf“ des Automaten befindet.

```
char band[] = {
        '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
        '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'
        };
short position = 1;
```

Der Hauptaufruf der Turing-Maschine erfolgt dann über die Methode tm(). Diese erstellt ein paar weitere interne Variablen und betritt dann eine while Schleife. Diese bildet den Hauptteil der Implementierung. In ihr werden immer wieder nacheinander:

- das aktuelle Zeichen auf dem Band ausgelesen

- anhand des aktuellen Zustands und dem gelesenen Wert der zu schreibende Wert ermittelt

- anhand des aktuellen Zustands und dem gelesenen Wert den nächsten Zustand ermittelt

- anhand des aktuellen Zustands und dem gelesenen Wert die nächste Bewegung auf dem Band ermittelt

Abschließend wird ein Boolean mit dem Wert „wahr“ oder „falsch“ zurückgegeben, welches indiziert, ob der Code dem geforderten Aufbau entspricht oder nicht.

4.Ergebnis

Schlussendlich sind wir sehr zufrieden, da wir fast alles Geplante umgesetzt haben und unser G.A.T.E System funktioniert. Der einzige, nicht- funktionierende vorgenommene Aspekt, ist die Zeiteintragung in die Datenbank bei dem Betreten und Verlassen des Restaurants. Die Bearbeitung dieses Features war uns auf Grund fehlender Zeit schlussendlich nicht mehr möglich.

Abb. 16: Gesamtes G.A.T.E. System

Abb. 17: seitliche Ansicht

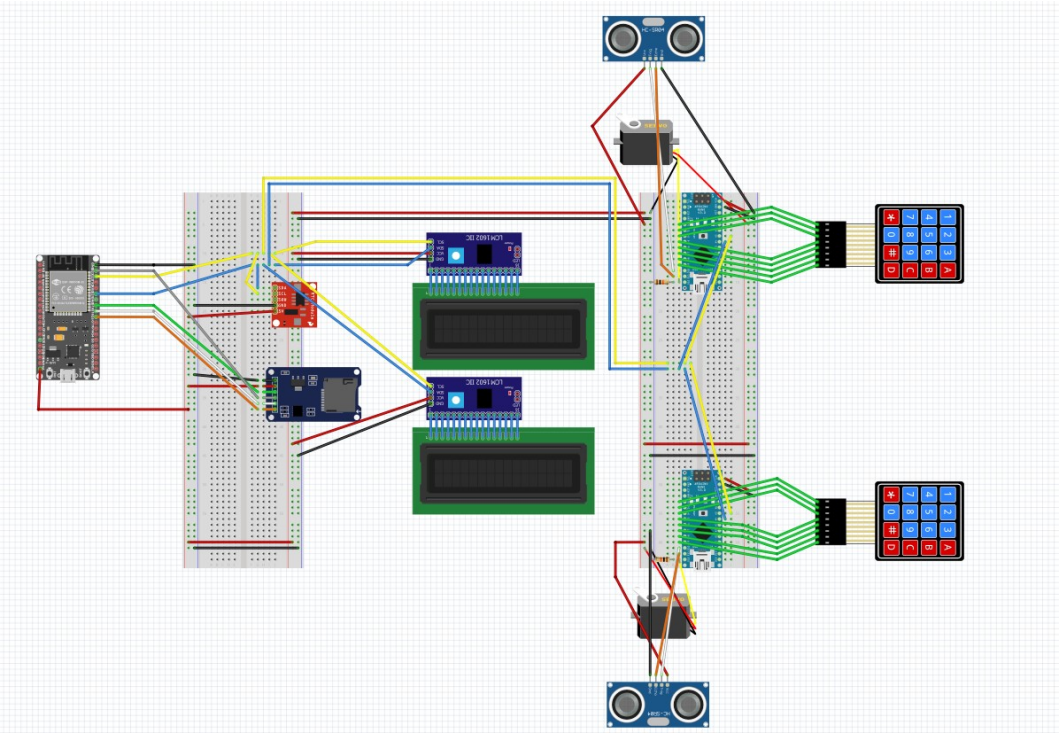

Abb. 18: Der fertige Schaltplan

Anleitung zu Benutzung des G.A.T.E.-Systems:

Scannen sie den QR-Code und befolgen sie die im YouTube-Tutorial gezeigten Schritte (https://www.youtube.com/watch?v=OrmeNNKRrV4).

Sollte dies nicht möglich sein befolgen Sie bitte die folgende Anleitung:

1. **Verbinden sie sich mit dem Netzwerk Esp32-Access-Point.**
2. **Gehen sie in ihren Browser und rufen sie die Website 192.168.4.1 auf.**
3. **Geben Sie nun ihre Daten ein und drücken sie auf "Abschicken".**
4. **Sie sollten nun ihren Code angezeigt bekommen, bitte merken Sie sich diesen gut.**
5. **Um nun das Restaurant betreten zu können, geben Sie ihren Code auf dem Tastenfeld ein und drücken Sie auf die #-Taste.**
6. **Sie können nach dem Öffnen der Schranke nun das Restaurant betreten.**
7. **Um das Restaurant zu verlassen, müssen Sie dann nur noch ihren Code erneut eingeben und können hinaus gehen.**

SCAN MICH

Abb. 19: Anleitung auf der Außenseite

Jedoch gab es auch Schwierigkeiten, die wir gerne noch aufzählen würden. Wir hatten als Extra geplant, einen RFID Sensor einzubauen, welcher uns helfen sollte, die Schranke zu öffnen, da man nur so die Seite des Systems öffnen kann und diese danach wieder schließt. Leider haben wir den Sensor nur außerhalb des gesamten Systems zum Laufen bekommen. Als wir dann allerdings alles zusammengefügt hatten, ist die Information, dass die Karte an den Sensor gehalten wurde, verloren gegangen, weshalb das System nur selten funktionierte. Aufgrund eines größeren Problems mit der Kommunikation zwischen dem Esp32 und dem einem Arduino, trafen wir die Entscheidung, uns vor Allem auf die Nutzbarkeit des Hauptprojektes zu konzentrieren und die Bearbeitung der Erweiterungen erst einmal auf Eis zu legen.

Anschließend würden wir als allgemeine Verbesserung noch vorschlagen, eine überarbeitete Datenbank für die Zeit zu erstellen. In dieser würde dann detailliert jeder Zeitpunkt vermerkt sein, zu dem eine Person rein- oder rausgegangen ist und nicht nur ihr letzter Besuch. So wäre es möglich, um einiges genauer nachzuvollziehen, wann eine infizierte Person, mit anderen Kontakt hatte.

5.Fazit

Zuerst einmal haben wir gelernt, dass ein Arduino Projekt nicht gerade leicht zu debuggen ist und dass die Hardware einen echt verzweifeln lassen kann. Zudem haben wir auch gelernt, dass man solche Probleme häufig mit sehr abstraktem Denken lösen kann. Auf Grund dessen haben wir uns vorgenommen, in der Zukunft mehr schöpferische Pausen einzulegen, da man bei einer anderen Beschäftigung häufig auf neue Ideen kommt, die uns dann auch oft weitergeholfen haben. Was jedoch kein Problem war, war die allgemeine Ausführung des Projekts, da wir uns zuvor einen Plan gemacht haben, wie wir das Projekt angehen wollen und diesen auch so gut wie möglich eingehalten haben. Dadurch mussten wir nichts nachbestellen und kamen in dieser Hinsicht nicht in Verzug. Trotz des Homeschooling Stress‘ und dem Faktor das wir uns nicht wirklich treffen konnten, haben wir versucht, das Beste daraus zu machen, auch wenn wir uns insgesamt wahrscheinlich etwas zu viel vorgenommen haben, was auch zur unvollständigen Einbindung der RTC geführt hat. Des Weiteren haben wir uns über verschiedene Möglichkeiten im Internet vernetzt gehalten und konnten dadurch uns austauschen, gegenseitig Sachen beibringen und hatten dazu noch eine Menge Spaß. Rückblickend auf die Ziele, die wir uns zu Beginn der Facharbeit gesetzt haben, können wir mit Zuversicht behaupten, dass unser G.A.T.E. System dazu in der Lage wäre, den Alltag von gastronomischen Betrieben und deren Kunden zu Coronazeiten zu vereinfachen (vor allem wenn noch bestimmte Erweiterungen umgesetzt werden würden) und dabei den Infektionsschutz in den Vordergrund zu stellen.

6.Literaturverzeichnis

Bibliotheken:

https://github.com/johnrickman/LiquidCrystal_I2C

https://github.com/me-no-dev/ESPAsyncWebServer

https://github.com/me-no-dev/AsyncTCP

Alle anderen Bibliotheken sind schon vorinstalliert oder können über den internen „Bibliothekverwalter“ installiert werden.

Tutorials:

https://randomnerdtutorials.com/esp32-async-web-server-espasyncwebserver-library/

https://learn.adafruit.com/adafruit-data-logger-shield/using-the-real-time-clock

https://randomnerdtutorials.com/esp32-data-logging-temperature-to-microsd-card/

Zitat 1:

https://de.wikipedia.org/wiki/ESP32

7.Abbildungsverzeichnis

Abbildung 16, 4 (Quelle: Nelly Schrader – eigene Darstellung)**:**
Gesamtes G.A.T.E. System

Abbildung 17, 4 (Quelle: Nelly Schrader – eigene Darstellung)**:**
seitliche Ansicht

Abbildung 18, 4 (Quelle: Daniel Gluch – Fritzing)**:**
Der fertige Schaltplan

Abbildung 19, 4 (Quelle: Kilian Arendt – GIMP 2)**:**
Anleitung auf der Außenseite

8.Anhang

Code vom ESP32:

```
#include <LiquidCrystal_I2C.h>
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include "RTClib.h"

// Variablen für die Displays
LiquidCrystal_I2C lcd1(0x26, 16, 2);
LiquidCrystal_I2C lcd2(0x27, 16, 2);
RTC_DS1307 rtc;

// Pins für die Kommunikation mit den Nanos (für die Servos)
const byte digComAusgang = 26;
const byte digComEingang = 27;

// Hier wird der empfangene Code von den Nanos drin gespeichert
String number_to_compare;
String received_number = "#####n";

// Name des Wlan-Hotspots
const char* ssid     = "ESP32-Access-Point";

// Initialisierung des WebServers
AsyncWebServer server(80);

// Hier wird die http Anfrage drin gespeichert
String header;

// Anhand dieser Konstanten werden später die einzelnen Eingaben
der Nutzer voneinander unterschieden
const char* PARAM_INPUT_1 = "name";
const char* PARAM_INPUT_2 = "nname";
const char* PARAM_INPUT_3 = "email";
```

```
// In diesen Variablen werden die Daten bei der Registrierung dann
gespeichert
String inputvName;
String inputnName;
String inputEmail;

String current_msg;

// Hier werden die aktuellen Personen gezählt.
int person_counter = 0;

// Das ist die "Stellschraube für die maximale Personenanzahl"
const byte MAX_PERSONS = 20;

// Variablen für die Turingmaschine
char band[] = {
        '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
        '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'
        };
short position = 1;

int flag = 0;
boolean result;

// HTML-Code der Website
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
  <title>ESP Input Form</title>
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  </head><body>
  <form action="/get">
    Vorame: <input type="text" name="name"><br>
    Nachname: <input type="text" name="nname"><br>
    EMail: <input type="text" name="email"><br>
    <input type="submit" value="Submit">
  </form><br>
</body></html>)rawliteral";


// Was soll gesendet werden, wenn eine falsche URL eingegeben wird
void notFound(AsyncWebServerRequest *request) {
  request->send(404, "text/plain", "Not found");
}


void setup() {
  // I2C Bus als Master beitreten
  Wire.begin();

  // Initiieren der LCDs
  lcd1.init();
  delay(20);
  lcd2.init();
  delay(20);
  lcd1.backlight();
  delay(20);
  lcd2.backlight();

  // Starten des Accesspoints
  WiFi.softAP(ssid);

  // Initialisiern der RTC
  rtc.begin();
```

```
  // Initialisierung der SD-Karte
  SD.begin(5);

  // Die Pins für die Nano Signale für die Servos
  pinMode(digComAusgang, OUTPUT);
  pinMode(digComEingang, OUTPUT);

  // Erster Code (für die nächste Registrierung) wird schonmal
generiert
  randomSeed(analogRead(0));
  do {
     number_to_compare = (String)genCode();
  } while(code_already_exists(number_to_compare));
  current_msg = "" + number_to_compare + "|";

  // Bereitstellen der Website
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });


  // Was passiert wenn Daten vom ESP ankommen
  server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request)
{
    String inputParam;
    // Eingegebene Daten werden ausgelesen
    if (request->hasParam(PARAM_INPUT_1) && request-
>hasParam(PARAM_INPUT_2) && request->hasParam(PARAM_INPUT_3)) {
      inputvName = request->getParam(PARAM_INPUT_1)->value();
      inputnName = request->getParam(PARAM_INPUT_2)->value();
      inputEmail = request->getParam(PARAM_INPUT_3)->value();
      inputParam = PARAM_INPUT_1;
    }
    else {
      inputvName = "No message sent";
      inputParam = "none";
    }

    long zeit = get_unixtime();
    // Kompletten Datenbank-String zusammenbauen
    current_msg += inputvName;
    current_msg += '|';
    current_msg += inputnName;
    current_msg += '|';
    current_msg += inputEmail;
    current_msg += '|';
    current_msg += (String)zeit;
    current_msg += '|';
    current_msg += (String)zeit;
    current_msg += "|0";
    // Auf SD-Karte schreiben
    write_string_to_EOF(current_msg);
    // Bestätigung an Nutzer senden
    request->send(200, "text/html", "HTTP GET request sent to your
ESP on input field ("
                                    + inputParam + ") with value:
" + inputvName + ", " + inputnName + ", " + inputEmail +
                                    "<br>Dein Code fuer das
G.A.T.E. lautet " + number_to_compare +
                                    "<br><a href=\"/\">Return to
Home Page</a>");
    // Zurücksetzen und auf nächsten Nutzer vorbereiten
    do {
       number_to_compare = (String)genCode();
    } while(code_already_exists(number_to_compare));
    current_msg = "" + number_to_compare + "|";
```

```
  });
  server.onNotFound(notFound);

  server.begin();
}

void loop(){
  // Nano 1 nach neuem Code fragen
  received_number = "";
  Wire.requestFrom(1, 6);    //7 Bytes vom "Eingangs"-Arduino
abfragen
  while (Wire.available()) {
    byte temp = Wire.read();
    received_number += (char)temp; // Direkt als Zeichen casten
  }

  // Auf Eingangs-Display anzeigen
  lcd1.clear();
  lcd1.setCursor(0, 0);
  lcd1.print("Code eingeben:");
  lcd1.setCursor(0, 1);
  lcd1.print(received_number.substring(0, 5));

  // Wenn jemand eine Eingabe bestätigt hat
  if (received_number.substring(5) == "y") {
    char temp[5];
    // Aufsplitten der Nachricht
    for (int i = 0; i<5; i++){
      temp[i] = received_number.charAt(i);
    }
    // Prüfung durch Turingmaschine
    if (tm(temp)) {
      // Prüfung ob der Nutzer existiert
      if (code_already_exists(received_number.substring(0, 5))) {
        // Prüfung ob der Nutzer schon im Gebäude ist
        if (!is_inside(received_number.substring(0, 5))) {
          // Prüfung ob noch Platz im Gebäude ist
          if (person_counter < MAX_PERSONS) {
            // Bestätigung anzeigen
            lcd1.clear();
            lcd1.setCursor(0, 0);
            lcd1.print("***Akzeptiert***");
            lcd1.setCursor(0, 1);
            lcd1.print("Oeffne Tuer");
            //Schranken öffnen
            digitalWrite(digComEingang, HIGH);
            delay(1000);
            digitalWrite(digComEingang, LOW);
            delay(1000);

            // Display aktualisieren
            lcd1.clear();
            lcd1.setCursor(0, 0);
            lcd1.print("***Drinnen***");
            lcd1.setCursor(0, 1);
            lcd1.print("Schliesse Tuer");
            // Datenbank aktualisieren
            toggle_user(received_number.substring(0, 5));
            // Zähler aktualisieren
            person_counter++;
          }
          // Fehlermeldung wenn kein Platz im Gebäude ist
          else {
            lcd1.clear();
            lcd1.setCursor(0, 0);
            lcd1.print("Restaurant ist voll");
```

```
          }
        }
        // Fehlermeldung wenn der Nutzer eigentlich schon drin ist
        else {
          lcd1.clear();
          lcd1.setCursor(0, 0);
          lcd1.print("Code abgelehnt");
        }
      }
      // Fehlermeldung wenn der Nutzer nicht im System existiert
      else {
        lcd1.clear();
        lcd1.setCursor(0, 0);
        lcd1.print("Code falsch");
      }
    }
    // Fehlermeldung wenn die Turingmaschine nicht bestätigt
    else {
      lcd1.clear();
      lcd1.setCursor(0, 0);
      lcd1.print("Unmoeglich");
    }
  }
  delay(500);

  // Nano 2 nach neuem Code fragen
  received_number = "";
  Wire.requestFrom(2, 6);    //7 Bytes vom "Ausgangs"-Arduino
abfragen
  while (Wire.available()) { // slave may send less than requested
    byte temp = Wire.read();
    received_number += (char)temp; // Direkt als Zeichen casten
  }

  // Auf Ausgangs-Display anzeigen
  lcd2.clear();
  lcd2.setCursor(0, 0);
  lcd2.print("Code eingeben:");
  lcd2.setCursor(0, 1);
  lcd2.print(received_number.substring(0, 5));

  // Wenn jemand eine Eingabe bestätigt hat
  if (received_number.substring(5) == "y") {
    char temp[5];
    // Aufsplitten der Nachricht
    for (int i = 0; i<5; i++){
      temp[i] = received_number.charAt(i);
    }
    // Prüfung durch Turingmaschine
    if (tm(temp)) {
      // Prüfung ob der Nutzer existiert
      if (code_already_exists(received_number.substring(0, 5))) {
        // Prüfung ob der Nutzer tatsächlich im Gebäude ist
        if (is_inside(received_number.substring(0, 5))) {
          // Bestätigung anzeigen
          lcd2.clear();
          lcd2.setCursor(0, 0);
          lcd2.print("***Akzeptiert***");
          lcd2.setCursor(0, 1);
          lcd2.print("Oeffne Tuer");
          //Schranken öffnen
          digitalWrite(digComAusgang, HIGH);
          delay(1000);
          digitalWrite(digComAusgang, LOW);
          delay(1000);
          // Wait for person to enter
```

```
          // Display aktualisieren
          lcd2.clear();
          lcd2.setCursor(0, 0);
          lcd2.print("***Draußen***");
          lcd2.setCursor(0, 1);
          lcd2.print("Schliesse Tuer");
          // Datenbank aktualisieren
          toggle_user(received_number.substring(0, 5));
          // Zähler aktualisieren
          person_counter--;
        }
        // Fehlermeldung wenn der Nutzer nicht drin ist
        else {
          lcd2.clear();
          lcd2.setCursor(0, 0);
          lcd2.print("Code abgelehnt");
        }
      }
      // Fehlermeldung wenn der Nutzer nicht im System existiert
      else {
        lcd2.clear();
        lcd2.setCursor(0, 0);
        lcd2.print("Code falsch");
      }
    }
    // Fehlermeldung wenn die Turingmaschine nicht bestätigt
    else {
      lcd2.clear();
      lcd2.setCursor(0, 0);
      lcd2.print("Unmoeglich");
    }
  }
  delay(500);
}

// Diese Funktion nimmt eine Zeichenkette als Parameter und
schreibt diese
// ans Ende der aktuellen Datenbankdatei
void write_string_to_EOF(String to_write) {
  File myFile;
  // Welche der Dateien existiert gerade
  if (SD.exists("/data.txt")) {
    myFile = SD.open("/data.txt", FILE_WRITE);
  }
  else if (SD.exists("/data2.txt")) {
    myFile = SD.open("/data2.txt", FILE_WRITE);
  }
  else {
    myFile = SD.open("/data.txt", FILE_WRITE);
  }
  // Schreiben
  if (myFile) {
    myFile.seek(myFile.size());
    myFile.println(to_write);
    myFile.close();
  }
}

// Diese Funktion nimmt eine Zeichenkette als Paramter und
überprüft die aktuelle
// Datenbankdatei und liefert wahr als Rückgabewert wenn diese
Zeichenkette als Code existiert
boolean code_already_exists(String to_compare) {
  byte next_zeichen;
  boolean skip_flag = false;
```

```
  String current_code;
  File myFile;
  // Welche der Dateien existiert gerade
  if (SD.exists("/data.txt")) {
    myFile = SD.open("/data.txt");
  }
  else if (SD.exists("/data2.txt")) {
    myFile = SD.open("/data2.txt");
  }

  while (myFile.available()) {
    next_zeichen = (byte)myFile.read();
    // Wenn ein Trennzeichen gelesen wurde
    if (next_zeichen == 124) {
      // Wenn wir nicht gerade mittendrin sind
      if (!skip_flag) {
        // Wenn das gelesene mit dem Parameter übereinstimmt
        if (to_compare == current_code) {
          myFile.close();
          return true;
        }
        current_code = "";
      }
      skip_flag = true;
    }
    // Wenn wir tatsächlich gerade am Anfang einer Zeile sind (Wo
der Code steht)
    if (!skip_flag) {
      // Wenn da eine Zahl steht
      if (isDigit((char)next_zeichen)) {
        // Eine Ziffer des Codes der Zeile
        current_code += (char)next_zeichen;
      }
    }
    // Wenn wir das Ende einer Zeile erreicht haben
    if (next_zeichen == 13) {
      // Zurücksetzen der flag, dass jetzt theoretisch wieder ein
Code ausgelesen werden kann
      skip_flag = false;
    }
  }
  // Standardmäßig falsch
  myFile.close();
  return false;
}


// Diese Funktion nimmt eine Zeichenkette als Paramter und
überprüft die aktuelle
// Datenbankdatei und liefert wahr als Rückgabewert wenn diese
Zeichenkette als Code existiert
// und wenn der Besitzer dieser Zeichenkette sich gerade im
Gebäude befindet
boolean is_inside(String user_string) {
  byte next_zeichen;
  bool skip_flag = false;
  String current_code;
  bool result;
  File myFile;
  // Welche der Dateien existiert gerade
  if (SD.exists("/data.txt")) {
    myFile = SD.open("/data.txt");
  }
  else if (SD.exists("/data2.txt")) {
    myFile = SD.open("/data2.txt");
  }
```

```
  while (myFile.available()) {
    next_zeichen = (byte)myFile.read();
    // Wenn ein Trennzeichen gelesen wurde
    if (next_zeichen == 124) {
        // Wenn wir nicht gerade mittendrin sind
      if (!skip_flag) {
        // Wenn das gelesene mit dem Parameter übereinstimmt
        if (user_string == current_code) {
          short counter = 0;
          // Weiterlesen, bis 5 weitere Trennzeichen gelesen
wurden
          do {
            next_zeichen = (byte)myFile.read();
            if (next_zeichen == 124) {
              counter++;
            }
          } while (counter != 5);
          char temp = (char)myFile.read();
          // Überprüfung ob da eine 0 oder 1 steht
          if (temp == '0') {
            result = false;
          }
          else if (temp == '1') {
            result = true;
          }
          myFile.close();
          return result;
        }
        current_code = "";
      }
      skip_flag = true;
    }
    if (!skip_flag) {
      if (isDigit((char)next_zeichen)) {
        current_code += (char)next_zeichen;
      }
    }
    if (next_zeichen == 13) {
      skip_flag = false;
    }
  }
  myFile.close();
  return false;
}


// Diese Funktion nimmt eine Zeichenkette als Paramter und
überprüft die aktuelle
// Datenbankdatei ob diese Zeichenkette als Code existiert und
ändert den "Status"
// Des Nutzers -> ob ersich im Gebäude befindet oder nicht
void toggle_user(String user_string) {
  byte next_zeichen;
  boolean skip_flag = false;
  String current_code;
  File oldFile;
  File newFile;
  // Überprüfung welche der Dateien existiert und Erstellung der
anderen
  if (SD.exists("/data.txt")) {
    oldFile = SD.open("/data.txt");
    newFile = SD.open("/data2.txt", FILE_WRITE);
  }
  else if (SD.exists("/data2.txt")) {
    oldFile = SD.open("/data2.txt");
```

```
    newFile = SD.open("/data.txt", FILE_WRITE);
  }

  while (oldFile.available()) {
    next_zeichen = (byte)oldFile.read();
    newFile.print((char)next_zeichen);
    if (next_zeichen == 124) {
      if (!skip_flag) {
        if (user_string == current_code) {
          short counter = 0;
          do {
            next_zeichen = (byte)oldFile.read();
            // Alles erstmal übernehmen
            newFile.print((char)next_zeichen);
            if (next_zeichen == 124) {
              counter++;
            }
          } while (counter != 5);
          // Ersetzen der "Status" Variable
          char old = (char)oldFile.read();
          if (old == '1') {
            newFile.print(0);
          }
          else {
            newFile.print(1);
          }
        }
        current_code = "";
      }
      skip_flag = true;
    }
    if (!skip_flag) {
      if (isDigit((char)next_zeichen)) {
        current_code += (char)next_zeichen;
      }
    }
    if (next_zeichen == 13) {
      skip_flag = false;
    }
  }

  // Alte Datei löschen, neue speichern
  String oldFileName = oldFile.name();
  oldFile.close();
  newFile.close();
  if (oldFileName == "/data.txt") {
    SD.remove("/data.txt");
  }
  else if(oldFileName == "/data2.txt") {
    SD.remove("/data2.txt");
  }
}

// Diese Funktion generiert einen theoretisch möglichen Code der
eine korrekte Quersumme hat
long genCode() {
  long tenthousand = 10000;
  long random1 = random(1, 10);
  long random2 = random(1, 10);
  long random3 = random(1, 10);
  long random4 = random(1, 10);
  long finalnumber = random1 + random2 + random3 + random4;
  while(finalnumber > 9) {
    int finalhilfe = finalnumber /10;
    finalnumber = finalnumber - (finalhilfe*10);
    finalnumber = finalnumber + finalhilfe;
```

```
  }

  long codehilf = random1*tenthousand + random2*1000 + random3*100
+ random4*10 + finalnumber;
  return codehilf;
}

// Diese Funktion gibt den aktuellen UnixZeitstempel von der RTC
wieder
long get_unixtime() {
  rtc.isrunning();
  DateTime now = rtc.now();
  return now.unixtime();
}

// Implementierung der Turingmaschine
bool tm(char input[]) {
    reset_tm();
    const char NONE = 'N';
    const char LEFT = 'L';
    const char RIGHT = 'R';

    short state = 0;
    write_input_to_band(input);
    short final_state = 42;

    char read_value;
    char write_value;
    int next_state;
    char next_move;
    while (true) {
       read_value = band[position];
       write_value = get_write_value(state, read_value);
       if (write_value == '-1') {
           return false;
       }
       next_state = get_next_state(state, read_value);
       if (next_state == final_state) {
           return true;
       }
       else if (next_state == -1) {
           return false;
       }
        next_move = get_next_move(state, read_value);
        if (next_move == '-1') {
            return false;
        }
        band[position] = write_value;
        state = next_state;
        if (next_move == NONE) {
            position = position;
        }
        else if (next_move == RIGHT) {
            position++;
        }
        else if (next_move == LEFT) {
            position--;
        }
    }
}


void reset_tm() {
  for (short i = 0; i < 20; i++) {
    band[i] = '#';
  }
```

```
  position = 1;
}


void write_input_to_band(char input[]) {
    short backup = position;
    for (short i = 0; i < 5; i++) {
        band[position] = input[i];
        position++;
    }
    position = backup;
}

char get_write_value(int state, char read){
    switch (state)
    {
    case 0:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 1:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
```

```
            break;
        }
    case 2:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 3:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 4:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
```

```
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 5:
        switch (read)
        {
        case '#':
            return '|';
        default:
            break;
        }
    case 6:
        switch (read)
        {
        case '#':
            return '0';
        default:
            break;
        }
    case 7:
        switch (read)
        {
        case '#':
            return '0';
        default:
            break;
        }
    case 8:
        switch (read)
        {
        case '#':
            return '0';
        default:
            break;
        }
    case 9:
        switch (read)
        {
        case '#':
            return '0';
        default:
            break;
        }
    case 10:
        switch (read)
        {
        case '#':
            return '0';
        default:
            break;
        }
    case 11:
        switch (read)
        {
        case '#':
```

```
            return '0';
        default:
            break;
        }
    case 12:
        switch (read)
        {
        case '#':
            return '|';
        default:
            break;
        }
    case 13:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        case '1':
            return '1';
        default:
            break;
        }
    case 14:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 15:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '0';
        case '2':
            return '1';
        case '3':
            return '2';
        case '4':
            return '3';
        case '5':
            return '4';
        case '6':
```

```
            return '5';
        case '7':
            return '6';
        case '8':
            return '7';
        case '9':
            return '8';
        default:
            break;
        }
    case 16:
        switch (read)
        {
        case '#':
            return '#';
        case '0':
            return '0';
        case '1':
            return '0';
        case '2':
            return '1';
        case '3':
            return '2';
        case '4':
            return '3';
        case '5':
            return '4';
        case '6':
            return '5';
        case '7':
            return '6';
        case '8':
            return '7';
        case '9':
            return '8';
        default:
            break;
        }
    case 17:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
```

```
case 18:
    switch (read)
    {
    case '0':
        return '1';
    case '1':
        return '0';
    default:
        break;
    }
case 19:
    switch (read)
    {
    case '#':
        return '0';
    case '|':
        return '|';
    case '0':
        return '0';
    case '1':
        return '1';
    case '2':
        return '2';
    case '3':
        return '3';
    case '4':
        return '4';
    case '5':
        return '5';
    case '6':
        return '6';
    case '7':
        return '7';
    case '8':
        return '8';
    case '9':
        return '9';
    default:
        break;
    }
case 20:
    switch (read)
    {
    case '#':
        return '0';
    default:
        break;
    }
case 21:
    switch (read)
    {
    case '#':
        return '|';
    default:
        break;
    }
case 22:
    switch (read)
    {
    case '|':
        return '|';
    case '0':
        return '0';
    case '1':
        return '1';
    case '2':
```

```
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 23:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        case '1':
            return '0';
        default:
            break;
        }
    case 24:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '1';
        default:
            break;
        }
    case 25:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
```

```
case 26:
    switch (read)
    {
    case '0':
        return '1';
    case '1':
        return '2';
    case '2':
        return '3';
    case '3':
        return '4';
    case '4':
        return '5';
    case '5':
        return '6';
    case '6':
        return '7';
    case '7':
        return '8';
    case '8':
        return '9';
    case '9':
        return '0';
    default:
        break;
    }
case 27:
    switch (read)
    {
    case '0':
        return '1';
    case '1':
        return '2';
    case '2':
        return '3';
    case '3':
        return '4';
    default:
        break;
    }
case 28:
    switch (read)
    {
    case '|':
        return '|';
    case '0':
        return '0';
    default:
        break;
    }
case 29:
    switch (read)
    {
    case '0':
        return '0';
    case '1':
        return '0';
    case '2':
        return '1';
    case '3':
        return '2';
    case '4':
        return '3';
    case '5':
        return '4';
    case '6':
```

```
            return '5';
        case '7':
            return '6';
        case '8':
            return '7';
        case '9':
            return '8';
        default:
            break;
        }
    case 30:
        switch (read)
        {
        case '0':
            return '0';
        case '1':
            return '0';
        case '2':
            return '1';
        case '3':
            return '2';
        case '4':
            return '3';
        case '5':
            return '4';
        case '6':
            return '5';
        case '7':
            return '6';
        case '8':
            return '7';
        case '9':
            return '8';
        default:
            break;
        }
    case 31:
        switch (read)
        {
        case '0':
            return '1';
        case '1':
            return '2';
        case '2':
            return '3';
        case '3':
            return '4';
        case '4':
            return '5';
        case '5':
            return '6';
        case '6':
            return '7';
        case '7':
            return '8';
        case '8':
            return '9';
        case '9':
            return '0';
        default:
            break;
        }
    case 32:
        switch (read)
        {
        case '0':
```

```
            return '1';
        case '1':
            return '2';
        case '2':
            return '3';
        case '3':
            return '4';
        case '4':
            return '5';
        case '5':
            return '6';
        case '6':
            return '7';
        case '7':
            return '8';
        case '8':
            return '9';
        default:
            break;
        }
    case 33:
        switch (read)
        {
        case '0':
            return '0';
        default:
            break;
        }
    case 34:
        switch (read)
        {
        case '0':
            return '0';
        default:
            break;
        }
    case 35:
        switch (read)
        {
        case '|':
            return '|';
        default:
            break;
        }
    case 36:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        default:
            break;
        }
    case 37:
        switch (read)
        {
        case '1':
            return '0';
        case '2':
            return '1';
        case '3':
            return '2';
        case '4':
            return '3';
        case '5':
```

```
            return '4';
        case '6':
            return '5';
        case '7':
            return '6';
        case '8':
            return '7';
        case '9':
            return '8';
        default:
            break;
        }
    case 38:
        switch (read)
        {
        case '#':
            return '#';
        case '|':
            return '|';
        case '0':
            return '0';
        case '1':
            return '1';
        case '2':
            return '2';
        case '3':
            return '3';
        case '4':
            return '4';
        case '5':
            return '5';
        case '6':
            return '6';
        case '7':
            return '7';
        case '8':
            return '8';
        case '9':
            return '9';
        default:
            break;
        }
    case 39:
        switch (read)
        {
        case '|':
            return '|';
        default:
            break;
        }
    case 40:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        default:
            break;
        }
    case 41:
        switch (read)
        {
        case '0':
            return '0';
        default:
```

```
            break;
        }
    case 42:
        break;
    case 43:
        switch (read)
        {
        case '|':
            return '|';
        default:
            break;
        }
    case 44:
        switch (read)
        {
        case '|':
            return '|';
        case '0':
            return '0';
        case '1':
            return '1';
        default:
            break;
        }

    default:
        return '-1';
    }
    return '-1';
}

int get_next_state(int state, char read){
    switch (state)
    {
    case 0:
        switch (read)
        {
        case '0':
            return 1;
        case '1':
            return 1;
        case '2':
            return 1;
        case '3':
            return 1;
        case '4':
            return 1;
        case '5':
            return 1;
        case '6':
            return 1;
        case '7':
            return 1;
        case '8':
            return 1;
        case '9':
            return 1;
        default:
            break;
        }
    case 1:
        switch (read)
        {
        case '0':
            return 2;
        case '1':
```

```
            return 2;
        case '2':
            return 2;
        case '3':
            return 2;
        case '4':
            return 2;
        case '5':
            return 2;
        case '6':
            return 2;
        case '7':
            return 2;
        case '8':
            return 2;
        case '9':
            return 2;
        default:
            break;
        }
    case 2:
        switch (read)
        {
        case '0':
            return 3;
        case '1':
            return 3;
        case '2':
            return 3;
        case '3':
            return 3;
        case '4':
            return 3;
        case '5':
            return 3;
        case '6':
            return 3;
        case '7':
            return 3;
        case '8':
            return 3;
        case '9':
            return 3;
        default:
            break;
        }
    case 3:
        switch (read)
        {
        case '0':
            return 4;
        case '1':
            return 4;
        case '2':
            return 4;
        case '3':
            return 4;
        case '4':
            return 4;
        case '5':
            return 4;
        case '6':
            return 4;
        case '7':
            return 4;
        case '8':
```

```
            return 4;
        case '9':
            return 4;
        default:
            break;
        }
    case 4:
        switch (read)
        {
        case '0':
            return 5;
        case '1':
            return 5;
        case '2':
            return 5;
        case '3':
            return 5;
        case '4':
            return 5;
        case '5':
            return 5;
        case '6':
            return 5;
        case '7':
            return 5;
        case '8':
            return 5;
        case '9':
            return 5;
        default:
            break;
        }
    case 5:
        switch (read)
        {
        case '#':
            return 6;
        default:
            break;
        }
    case 6:
        switch (read)
        {
        case '#':
            return 7;
        default:
            break;
        }
    case 7:
        switch (read)
        {
        case '#':
            return 8;
        default:
            break;
        }
    case 8:
        switch (read)
        {
        case '#':
            return 9;
        default:
            break;
        }
    case 9:
        switch (read)
```

```
        {
        case '#':
            return 10;
        default:
            break;
        }
    case 10:
        switch (read)
        {
        case '#':
            return 11;
        default:
            break;
        }
    case 11:
        switch (read)
        {
        case '#':
            return 12;
        default:
            break;
        }
    case 12:
        switch (read)
        {
        case '#':
            return 13;
        default:
            break;
        }
    case 13:
        switch (read)
        {
        case '|':
            return 14;
        case '0':
            return 13;
        case '1':
            return 13;
        default:
            break;
        }
    case 14:
        switch (read)
        {
        case '0':
            return 15;
        case '1':
            return 15;
        case '2':
            return 15;
        case '3':
            return 15;
        case '4':
            return 15;
        case '5':
            return 15;
        case '6':
            return 15;
        case '7':
            return 15;
        case '8':
            return 15;
        case '9':
            return 15;
        default:
```

```
            break;
        }
    case 15:
        switch (read)
        {
        case '0':
            return 16;
        case '1':
            return 17;
        case '2':
            return 17;
        case '3':
            return 17;
        case '4':
            return 17;
        case '5':
            return 17;
        case '6':
            return 17;
        case '7':
            return 17;
        case '8':
            return 17;
        case '9':
            return 17;
        default:
            break;
        }
    case 16:
        switch (read)
        {
        case '#':
            return 19;
        case '0':
            return 16;
        case '1':
            return 17;
        case '2':
            return 17;
        case '3':
            return 17;
        case '4':
            return 17;
        case '5':
            return 17;
        case '6':
            return 17;
        case '7':
            return 17;
        case '8':
            return 17;
        case '9':
            return 17;
        default:
            break;
        }
    case 17:
        switch (read)
        {
        case '|':
            return 44;
        case '0':
            return 17;
        case '1':
            return 17;
        case '2':
```

```
            return 17;
        case '3':
            return 17;
        case '4':
            return 17;
        case '5':
            return 17;
        case '6':
            return 17;
        case '7':
            return 17;
        case '8':
            return 17;
        case '9':
            return 17;
        default:
            break;
        }
    case 18:
        switch (read)
        {
        case '0':
            return 13;
        case '1':
            return 18;
        default:
            break;
        }
    case 19:
        switch (read)
        {
        case '#':
            return 20;
        case '|':
            return 19;
        case '0':
            return 19;
        case '1':
            return 19;
        case '2':
            return 19;
        case '3':
            return 19;
        case '4':
            return 19;
        case '5':
            return 19;
        case '6':
            return 19;
        case '7':
            return 19;
        case '8':
            return 19;
        case '9':
            return 19;
        default:
            break;
        }
    case 20:
        switch (read)
        {
        case '#':
            return 21;
        default:
            break;
        }
```

```
case 21:
    switch (read)
    {
    case '#':
        return 22;
    default:
        break;
    }
case 22:
    switch (read)
    {
    case '|':
        return 23;
    case '0':
        return 22;
    case '1':
        return 22;
    case '2':
        return 22;
    case '3':
        return 22;
    case '4':
        return 22;
    case '5':
        return 22;
    case '6':
        return 22;
    case '7':
        return 22;
    case '8':
        return 22;
    case '9':
        return 22;
    default:
        break;
    }
case 23:
    switch (read)
    {
    case '|':
        return 28;
    case '0':
        return 23;
    case '1':
        return 24;
    default:
        break;
    }
case 24:
    switch (read)
    {
    case '|':
        return 25;
    case '0':
        return 24;
    default:
        break;
    }
case 25:
    switch (read)
    {
    case '0':
        return 26;
    case '1':
        return 26;
    case '2':
```

```
            return 26;
        case '3':
            return 26;
        case '4':
            return 26;
        case '5':
            return 26;
        case '6':
            return 26;
        case '7':
            return 26;
        case '8':
            return 26;
        case '9':
            return 26;
        default:
            break;
        }
    case 26:
        switch (read)
        {
        case '0':
            return 22;
        case '1':
            return 22;
        case '2':
            return 22;
        case '3':
            return 22;
        case '4':
            return 22;
        case '5':
            return 22;
        case '6':
            return 22;
        case '7':
            return 22;
        case '8':
            return 22;
        case '9':
            return 27;
        default:
            break;
        }
    case 27:
        switch (read)
        {
        case '0':
            return 22;
        case '1':
            return 22;
        case '2':
            return 22;
        case '3':
            return 22;
        default:
            break;
        }
    case 28:
        switch (read)
        {
        case '|':
            return 29;
        case '0':
            return 28;
        default:
```

```
            break;
        }
    case 29:
        switch (read)
        {
        case '0':
            return 30;
        case '1':
            return 31;
        case '2':
            return 31;
        case '3':
            return 31;
        case '4':
            return 31;
        case '5':
            return 31;
        case '6':
            return 31;
        case '7':
            return 31;
        case '8':
            return 31;
        case '9':
            return 31;
        default:
            break;
        }
    case 30:
        switch (read)
        {
        case '0':
            return 34;
        case '1':
            return 33;
        case '2':
            return 33;
        case '3':
            return 33;
        case '4':
            return 33;
        case '5':
            return 33;
        case '6':
            return 33;
        case '7':
            return 33;
        case '8':
            return 33;
        case '9':
            return 33;
        default:
            break;
        }
    case 31:
        switch (read)
        {
        case '0':
            return 29;
        case '1':
            return 29;
        case '2':
            return 29;
        case '3':
            return 29;
        case '4':
```

```
            return 29;
        case '5':
            return 29;
        case '6':
            return 29;
        case '7':
            return 29;
        case '8':
            return 29;
        case '9':
            return 32;
        default:
            break;
        }
    case 32:
        switch (read)
        {
        case '0':
            return 29;
        case '1':
            return 29;
        case '2':
            return 29;
        case '3':
            return 29;
        case '4':
            return 29;
        case '5':
            return 29;
        case '6':
            return 29;
        case '7':
            return 29;
        case '8':
            return 29;
        default:
            break;
        }
    case 33:
        switch (read)
        {
        case '0':
            return 35;
        default:
            break;
        }
    case 34:
        switch (read)
        {
        case '0':
            return 39;
        default:
            break;
        }
    case 35:
        switch (read)
        {
        case '|':
            return 36;
        default:
            break;
        }
    case 36:
        switch (read)
        {
        case '|':
```

```
            return 37;
        case '0':
            return 36;
        default:
            break;
        }
    case 37:
        switch (read)
        {
        case '1':
            return 38;
        case '2':
            return 38;
        case '3':
            return 38;
        case '4':
            return 38;
        case '5':
            return 38;
        case '6':
            return 38;
        case '7':
            return 38;
        case '8':
            return 38;
        case '9':
            return 38;
        default:
            return -1;
        }
    case 38:
        switch (read)
        {
        case '#':
            return 43;
        case '|':
            return 38;
        case '0':
            return 38;
        case '1':
            return 38;
        case '2':
            return 38;
        case '3':
            return 38;
        case '4':
            return 38;
        case '5':
            return 38;
        case '6':
            return 38;
        case '7':
            return 38;
        case '8':
            return 38;
        case '9':
            return 38;
        default:
            break;
        }
    case 39:
        switch (read)
        {
        case '|':
            return 40;
        default:
```

```
            break;
        }
    case 40:
        switch (read)
        {
        case '|':
            return 41;
        case '0':
            return 40;
        default:
            break;
        }
    case 41:
        switch (read)
        {
        case '0':
            return 42;
        default:
            break;
        }
    case 42:
        break;
    case 43:
        switch (read)
        {
        case '|':
            return 30;
        default:
            break;
        }
    case 44:
        switch (read)
        {
        case '|':
            return 18;
        case '0':
            return 44;
        case '1':
            return 44;
        default:
            break;
        }

    default:
        return -1;
    }
    return -1;
}

char get_next_move(int state, char read){
    switch (state)
    {
    case 0:
        switch (read)
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
```

```
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 1:
        switch (read)
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 2:
        switch (read)
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 3:
        switch (read)
```

```
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 4:
        switch (read)
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 5:
        switch (read)
        {
        case '#':
            return 'R';
        default:
            break;
        }
    case 6:
        switch (read)
        {
        case '#':
            return 'R';
        default:
            break;
        }
```

```
case 7:
    switch (read)
    {
    case '#':
        return 'R';
    default:
        break;
    }
case 8:
    switch (read)
    {
    case '#':
        return 'R';
    default:
        break;
    }
case 9:
    switch (read)
    {
    case '#':
        return 'R';
    default:
        break;
    }
case 10:
    switch (read)
    {
    case '#':
        return 'R';
    default:
        break;
    }
case 11:
    switch (read)
    {
    case '#':
        return 'R';
    default:
        break;
    }
case 12:
    switch (read)
    {
    case '#':
        return 'L';
    default:
        break;
    }
case 13:
    switch (read)
    {
    case '|':
        return 'L';
    case '0':
        return 'L';
    case '1':
        return 'L';
    default:
        break;
    }
case 14:
    switch (read)
    {
    case '0':
        return 'L';
    case '1':
```

```
            return 'L';
        case '2':
            return 'L';
        case '3':
            return 'L';
        case '4':
            return 'L';
        case '5':
            return 'L';
        case '6':
            return 'L';
        case '7':
            return 'L';
        case '8':
            return 'L';
        case '9':
            return 'L';
        default:
            break;
        }
    case 15:
        switch (read)
        {
        case '0':
            return 'L';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 16:
        switch (read)
        {
        case '#':
            return 'R';
        case '0':
            return 'L';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
```

```
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 17:
        switch (read)
        {
        case '|':
            return 'R';
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 18:
        switch (read)
        {
        case '0':
            return 'L';
        case '1':
            return 'L';
        default:
            break;
        }
    case 19:
        switch (read)
        {
        case '#':
            return 'R';
        case '|':
            return 'R';
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
```

```
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 20:
        switch (read)
        {
        case '#':
            return 'R';
        default:
            break;
        }
    case 21:
        switch (read)
        {
        case '#':
            return 'L';
        default:
            break;
        }
    case 22:
        switch (read)
        {
        case '|':
            return 'L';
        case '0':
            return 'L';
        case '1':
            return 'L';
        case '2':
            return 'L';
        case '3':
            return 'L';
        case '4':
            return 'L';
        case '5':
            return 'L';
        case '6':
            return 'L';
        case '7':
            return 'L';
        case '8':
            return 'L';
        case '9':
            return 'L';
        default:
            break;
        }
    case 23:
        switch (read)
        {
        case '|':
            return 'R';
        case '0':
            return 'L';
        case '1':
            return 'R';
        default:
            break;
        }
```

```
case 24:
    switch (read)
    {
    case '|':
        return 'R';
    case '0':
        return 'R';
    default:
        break;
    }
case 25:
    switch (read)
    {
    case '0':
        return 'R';
    case '1':
        return 'R';
    case '2':
        return 'R';
    case '3':
        return 'R';
    case '4':
        return 'R';
    case '5':
        return 'R';
    case '6':
        return 'R';
    case '7':
        return 'R';
    case '8':
        return 'R';
    case '9':
        return 'R';
    default:
        break;
    }
case 26:
    switch (read)
    {
    case '0':
        return 'L';
    case '1':
        return 'L';
    case '2':
        return 'L';
    case '3':
        return 'L';
    case '4':
        return 'L';
    case '5':
        return 'L';
    case '6':
        return 'L';
    case '7':
        return 'L';
    case '8':
        return 'L';
    case '9':
        return 'L';
    default:
        break;
    }
case 27:
    switch (read)
    {
    case '0':
```

```
            return 'L';
        case '1':
            return 'L';
        case '2':
            return 'L';
        case '3':
            return 'L';
        default:
            break;
        }
    case 28:
        switch (read)
        {
        case '|':
            return 'R';
        case '0':
            return 'R';
        default:
            break;
        }
    case 29:
        switch (read)
        {
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 30:
        switch (read)
        {
        case '0':
            return 'L';
        case '1':
            return 'L';
        case '2':
            return 'L';
        case '3':
            return 'L';
        case '4':
            return 'L';
        case '5':
            return 'L';
        case '6':
            return 'L';
        case '7':
            return 'L';
        case '8':
```

```
            return 'L';
        case '9':
            return 'L';
        default:
            break;
        }
    case 31:
        switch (read)
        {
        case '0':
            return 'L';
        case '1':
            return 'L';
        case '2':
            return 'L';
        case '3':
            return 'L';
        case '4':
            return 'L';
        case '5':
            return 'L';
        case '6':
            return 'L';
        case '7':
            return 'L';
        case '8':
            return 'L';
        case '9':
            return 'L';
        default:
            break;
        }
    case 32:
        switch (read)
        {
        case '0':
            return 'N';
        case '1':
            return 'N';
        case '2':
            return 'N';
        case '3':
            return 'N';
        case '4':
            return 'N';
        case '5':
            return 'N';
        case '6':
            return 'N';
        case '7':
            return 'N';
        case '8':
            return 'N';
        default:
            break;
        }
    case 33:
        switch (read)
        {
        case '0':
            return 'L';
        default:
            break;
        }
    case 34:
        switch (read)
```

```
        {
        case '0':
            return 'L';
        default:
            break;
        }
    case 35:
        switch (read)
        {
        case '|':
            return 'L';
        default:
            break;
        }
    case 36:
        switch (read)
        {
        case '|':
            return 'L';
        case '0':
            return 'L';
        default:
            break;
        }
    case 37:
        switch (read)
        {
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
            return 'R';
        case '6':
            return 'R';
        case '7':
            return 'R';
        case '8':
            return 'R';
        case '9':
            return 'R';
        default:
            break;
        }
    case 38:
        switch (read)
        {
        case '#':
            return 'L';
        case '|':
            return 'R';
        case '0':
            return 'R';
        case '1':
            return 'R';
        case '2':
            return 'R';
        case '3':
            return 'R';
        case '4':
            return 'R';
        case '5':
```

```
                return 'R';
            case '6':
                return 'R';
            case '7':
                return 'R';
            case '8':
                return 'R';
            case '9':
                return 'R';
            default:
                break;
            }
        case 39:
            switch (read)
            {
            case '|':
                return 'L';
            default:
                break;
            }
        case 40:
            switch (read)
            {
            case '|':
                return 'L';
            case '0':
                return 'L';
            default:
                break;
            }
        case 41:
            switch (read)
            {
            case '0':
                return 'N';
            default:
                break;
            }
        case 42:
            break;
        case 43:
            switch (read)
            {
            case '|':
                return 'L';
            default:
                break;
            }
        case 44:
            switch (read)
            {
            case '|':
                return 'L';
            case '0':
                return 'R';
            case '1':
                return 'R';
            default:
                break;
            }

        default:
            return '-1';
        }
        return '-1';
}
```

Code vom „Eingangs“-Arduino:

```
#include <Wire.h>
#include <Keypad.h>
#include <Servo.h>

// Servos
Servo servo1;
Servo servo2;

// Abstandssensoren
int trigPin = 10;
int echoPin = 11;
// Variablen für die Ultraschallsensoren
long duration;
int distance;

//Hier kommt das Signal vom ESP für die Servos an
int comPin = 12;
// Servo Variablen
boolean turauf = false;
boolean turzu = false;
boolean checkSchranke = false;
long myTimer = 0;
long myTimeout = 2000;

// Array für
char lastCode[6] = {'#', '#', '#', '#', '#', 'n'};

// Variablen für das Zahlenfeld
const byte ROWS = 4;              //4x4-Keypad
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );

char eingabe[5] = {'#', '#', '#', '#', '#'};  //array für Code
eingabe (# = unbelegt)
boolean arrayvoll = false;                    //boolean für check,
ob array voll ist

char validation_char = 'n';

void setup() {
  // I2C Bus als Slave mit Adresse 1 beitreten
  Wire.begin(1);
  Wire.onRequest(requestEvent);
  pinMode(trigPin, OUTPUT);  //Abstandssensor
  pinMode(echoPin, INPUT);   //Abstandssensor
  pinMode(comPin, INPUT);   // Verbindung zum ESP
  // Noch beide Servos im Code, nur einer wird benutzt
  servo1.attach(A2);
  servo2.attach(A3);
  // Sicherstellen, dass die zu sind
  servo1.write(0);
  servo2.write(140);
}
```

```
void loop() {
  // Letzte Tastenfeld-Eingabe auslesen
  char key = keypad.getKey();
  if (key) {
    // Wenn Buchstabentasten, ignorieren
    if (key == 'A' || key == 'B' || key == 'C' || key == 'D') {
//Unbelegte Keys
      ;
    } else if (key == '#') {                  //wenn Eingabe key(#)
gedrückt
      //Eingabe wird überprüft:
      lastCode[5] = 'y';
    } else if (key == '*') {         //wenn löschen-key(*) gedrückt
wird
      //löscht letzte Eingabe
      for (int i = 0; i < 5; i++) {
        if (eingabe[i] == '#' && i != 0) { //löscht die letzte
eingegebene Zahl
          eingabe[i - 1] = '#';
          break;
        }
        if (i == 4) {                //wenn alles belegt ist,
löscht es die letzte zahl
          eingabe[i] = '#';
          arrayvoll = false;
        }
      }

    } else {                         //wenn Zahl eingegeben wurde:
      if (arrayvoll) {               //löscht das komplette array
wenn es voll war
        cleararray();
      }

      for (int i = 0; i < 5; i++) { //setzt die zahl an nächste
freie Stelle
        arrayvoll = true;
        if (eingabe[i] == '#') {
          eingabe[i] = key;
          if (i != 4) {
            arrayvoll = false; // array ist noch nicht komplett
gefüllt
          } else {
            arrayvoll = true;  //array ist nun komplett gefüllt
          }
          break;
        }
      }
    }
  }
  for (int i = 0; i < 5; i++) {
    lastCode[i] = eingabe[i];
  }
  delay(50);
  // Gucken ob der ESP will dass die Schrank aufgemacht wird
  if (digitalRead(comPin)) {
    turauf = true;
  }
  if (turauf) {
    opendoor();
  }
}

// Diese Methode öffnet die Schranke mit dem Servo
// und lässt sie so lange offen wie die globale
// Boolean dazu true ist
```

```
void opendoor() {
  servo1.write(110);
  servo2.write(30);
  if (millis() > myTimeout + myTimer && !checkSchranke) {
    myTimer = millis();
    checkSchranke = true;
  }
  // US-Sensor ablesen
  if (checkSchranke) {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration * 0.034 / 2;
    if (distance < 30) {
      turzu = true;
      myTimer = millis();
    }
  }
  // Durchgehen detected und kurz gewartet
  if (millis() > myTimeout + myTimer && turzu) {
    checkSchranke = false;
    turauf = false;
    turzu = false;
    servo1.write(0);
    servo2.write(140);
  }
}

// Event wenn der ESP nach Code fragt
void requestEvent() {
  for (int i = 0; i < 6; i++) {
    Serial.println(lastCode[i]);
    Wire.write(lastCode[i]);
  }
  if (lastCode[5] == 'y') {
    cleararray();
    lastCode[5] = 'n';
  }
}

// Diese Funktion resettet das Eingaben-Array
void cleararray() {        //löscht alle eingegebenen Werte
  for (int i = 0; i < 5; i++) {
    eingabe[i] = '#';
  }
}
```

Code vom „Ausgangs“-Arduino:

```
#include <Wire.h>
#include <Keypad.h>
#include <Servo.h>

// Servos
Servo servo1;
Servo servo2;

// Abstandssensoren
int trigPin = 10;
int echoPin = 11;
// Variablen für die Ultraschallsensoren
long duration;
int distance;
```

```
//Hier kommt das Signal vom ESP für die Servos an
int comPin = 12;
// Servo Variablen
boolean turauf = false;
boolean turzu = false;
boolean checkSchranke = false;
long myTimer = 0;
long myTimeout = 2000;

// Speicherung der Eingaben
char lastCode[6] = {'#', '#', '#', '#', '#', 'n'};

// Variablen für das Zahlenfeld
const byte ROWS = 4;               //4x4-Keypad
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6,7,8,9};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );

char eingabe[5] = {'#', '#', '#', '#', '#'};  //array für Code
eingabe (# = unbelegt)
boolean arrayvoll = false;                    //boolean für check,
ob array voll ist

char validation_char = 'n';

void setup() {
    // I2C Bus als Slave mit Adresse 2 beitreten
  Wire.begin(2);
  Wire.onRequest(requestEvent);
  pinMode(trigPin, OUTPUT);  //Abstandssensor
  pinMode(echoPin, INPUT);   //Abstandssensor
  pinMode(comPin, INPUT);   // Verbindung zum ESP
    // Noch beide Servos im Code, nur einer wird benutzt
  servo1.attach(A2);
  servo2.attach(A3);
  // Sicherstellen, dass die zu sind
  servo1.write(170);
  servo2.write(20);
}

void loop() {
  // Letzte Tastenfeld-Eingabe auslesen
  char key = keypad.getKey();
  if (key){
    // Wenn Buchstabentasten, ignorieren
    if (key =='A' || key =='B' || key =='C' || key=='D'){
//Unbelegte Keys
      ;
    } else if(key =='#') {                    //wenn Eingabe key(#)
gedrückt
      //Eingabe wird überprüft:
      lastCode[5] = 'y';
    } else if(key =='*'){            //wenn löschen-key(*) gedrückt
wird
      //löscht letzte Eingabe
      for(int i = 0; i<5; i++) {
```

```
        if(eingabe[i] == '#' && i!=0) {  //löscht die letzte
eingegebene Zahl
          eingabe[i-1] = '#';
          break;
        }
        if(i == 4){                  //wenn alles belegt ist,
löscht es die letzte zahl
          eingabe[i] = '#';
          arrayvoll =false;
        }
      }

    } else {                        //wenn Zahl eingegeben wurde:
      if(arrayvoll){                //löscht das komplette array
wenn es voll war
        cleararray();
      }

      for(int i = 0; i<5; i++) {    //setzt die zahl an nächste
freie Stelle
        arrayvoll = true;
        if(eingabe[i] =='#'){
          eingabe[i] = key;
          if(i != 4){
            arrayvoll = false; // array ist noch nicht komplett
gefüllt
          } else{
            arrayvoll = true;  //array ist nun komplett gefüllt
          }
          break;
        }
      }
    }
  }
  for(int i =0; i<5; i++) {
    lastCode[i] = eingabe[i];
  }
  delay(50);
  // Gucken ob der ESP will dass die Schrank aufgemacht wird
  if(digitalRead(comPin)){
    turauf =true;
  }
  if(turauf){
    opendoor();
  }
}

// Diese Methode öffnet die Schranke mit dem Servo
// und lässt sie so lange offen wie die globale
// Boolean dazu true ist
void opendoor(){
  servo1.write(50);
  servo2.write(120);
  if(millis() > myTimeout + myTimer && !checkSchranke){
    myTimer = millis();
    checkSchranke = true;
  }
  // US-Sensor ablesen
  if(checkSchranke){
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration*0.034/2;
```

```
    if(distance < 30){
      turzu = true;
      myTimer = millis();
    }
  }
  // Durchgehen detected und kurz gewartet
  if(millis() > myTimeout + myTimer && turzu){
    checkSchranke = false;
    turauf = false;
    turzu = false;
    servo1.write(170);
    servo2.write(20);
  }
}

// Event wenn der ESP nach Code fragt
void requestEvent(){
  for (int i = 0; i<6; i++){
    Serial.println(lastCode[i]);
    Wire.write(lastCode[i]);
  }
  if (lastCode[5] == 'y') {
    cleararray();
    lastCode[5] = 'n';
  }
}

// Diese Funktion resettet das Eingaben-Array
void cleararray() {         //löscht alle eingegebenen Werte
  for(int i = 0; i<5; i++) {
    eingabe[i] = '#';
  }
}
```