BBS COCHEM
Berufsbildende Schule

Abb. 1

Thema der Praktischen Prüfung:

Planung, Aufbau und Programmierung des Geschicklichkeitsspiels "Heißer Draht" in optimierter Form auf Basis des Mikrocontrollers ESP32

Abb. 2

Update

Bearbeitung beginnt am: 04. Februar 2021

Abgabefrist: 12. März 2021

Projekt Betreuer: [redacted]

Von Leon Hürter [redacted]

Abb. 3

Abb. 4

Inhaltsverzeichnis

Bearbeitet von Leon Hürter = L.H.

Bearbeitet von [redacted]

Zusammen bearbeitet = Z

Abbildungsverzeichnis

+ = aus dem Internet / nicht von uns selbst

Einleitung

Unser Thema und unsere Aufgabe in der praktischen Prüfung ist die Planung, der Aufbau und die Programmierung eines Geschicklichkeitsspiels namens „Der Heiße Draht“. Programmiert wird das Ganze mit einem ESP32 Microcontroller und einer Arduino IDE. Der praktische Aufbau findet auf einem Holzbrett mit Kupferdraht, und auf einem Board mit einigen Platinen für den ESP32. Das Spiel läuft so ab das die Stoppuhr gestartet wird und dann die Kupfer-Öse über den Kupfer-Draht-Parkour geführt werden muss.

Vorgehensweise/Problemstellung

Zunächst ist es wichtig das man sich alle benötigten Komponenten möglichst schnell besorgt, damit man in erster Linie die Sachen, die man gemacht hat, auch austesten kann, aber auch damit man am Ende des Projekts nicht noch auf nicht vorhandene Komponenten warten muss und so in Zeitverzug kommt. Es ist aber auch möglich sich nur die Komponenten für einen Teil des Projekts zu besorgen. So haben wir zum Beispiel damit angefangen die Stoppuhr auf dem OLED-Display zu programmieren, da hierzu nur der ESP und das Display gebraucht wurden. Diese mussten noch mit ein paar Jumper Kabeln verbunden werden und so konnte man anfangen zu testen. Als die Stoppuhr fertig war haben wir uns also mit den nächsten Komponenten and das nächste Thema begeben (die Fehleranzeige zu programmieren). Wenn man nach dieser Methode vorgeht kommt es weniger schnell zu Problemen und man kann strukturiert sein Projekt fertigstellen.

Zielsetzung

Als Ziel unseres Projekts soll der heiße Draht an sich fertig aufgebaut und verkabelt sein. Die Optimierungen sollten erfolgreich erstellt und eingebaut sein. Das bedeutet: Die Stoppuhr kann mit einem Taster gestartet werden mit einem anderen gestoppt und wieder mit einem anderen Resettet werden. Wenn nun der Draht berührt wird, verliert man ein Leben, das bedeutet das eine der Drei LEDs erlischt. Um den Fehler nicht nur optisch, sondern auch audiovisuell darzustellen, ertönt hier auch der Piezo Buzzer. Das Spiel endet, wenn entweder alle drei Leben verloren wurden oder der Draht-Parkour erfolgreich absolviert wurde und die Stoppuhr gestoppt wurde.

Komponentenliste für Projekt „heißer Draht“

- ✓ -Holzbrett
- ✓ -Isolierband (für Stick)
- ✓ -Jumper Kabel
- ✓ -Breadboard
- ✓ -3 Taster
- ✓ -3x 10k Ohm Widerstand (für Taster)
- ✓ -3 LEDs (Rot, Grün, Gelb)
- ✓ -3 Vorwiederstände (270 Ohm)
- ✓ -Kupferdraht
- ✓ -OLED Display
- ✓ -ESP32
- ✓ -Arduino DIE
- ✓ -Piezo Buzzer

Abb. 5

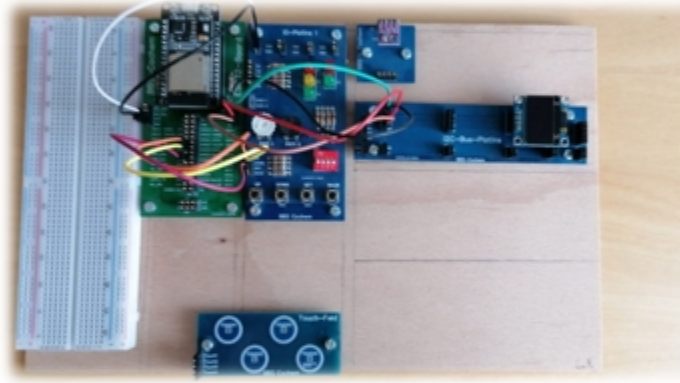


Abb. 6

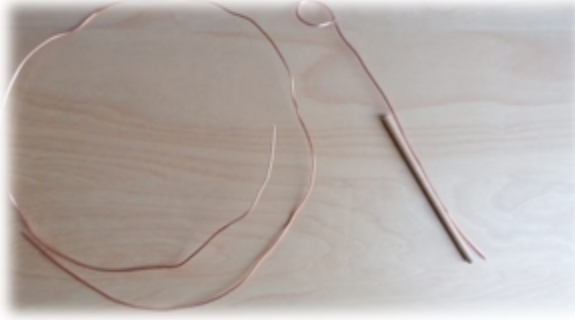


Abb. 7

Software

Display Stoppuhr

Ein Teil unseres Projekts ist die Stoppuhr, welche auf einem OLED Display ausgegeben wird und insgesamt über drei Taster gesteuert wird. Einen Start-Taster, einen Stopp-Taster und einen Reset-Taster.

Stoppuhr_Display

```
//////////////////Stoppuhr_Display//////////////////////////////

//Taster werden als die Pins 16,17 und 18 deklariert.
#define Taster_Start 16
#define Taster_Stopp 17
#define Taster_Reset 18
```

Abb. 8

Danach werden noch die Bibliotheken eingebunden und die Variablen deklariert, bevor die Setup() anfängt.

```
//Bibliotheken für die Displaynutzung werden eingebunden.
#include <Wire.h>
#include <SSD1306Wire.h>

//Variablendeklaration zur Zeit-Berechnung.
int Sekunden;
int Minuten;
int Stunden;
unsigned long StartZeit;
unsigned long AktuelleZeit;
boolean Merker=0;

SSD1306Wire anzeige(0x3C, 21, 22);      //OLED-Display wird eingebunden. (mit "anzeige" Variable)
```

Abb. 9

In der Setup selbst wird nur das Display initialisiert (mit „anzeige.init();“) und formatiert (z.B. Schriftart geändert mit „setFont“ oder Schwarz und Weiß getauscht mit „invertDisplay“). Außerdem wird die Reset-Funktion hier aufgerufen um das Display bei Programmstart nicht leer zu lassen.

```
void setup() {
  anzeige.init();                     //Display wird intitialisiert.
  anzeige.clear();
  anzeige.setFont(ArialMT_Plain_16);  //Display wird formatiert(festlegen von Schriftart und Farben)
  anzeige.invertDisplay();
  Reset();                            //Reset-Funktion wird aufgerufen um Display zu beschreiben.
}
```

Abb. 10

Jetzt kommt die Funktion Stoppuhr, welche die Zeit-Werte berechnen und ausgeben soll.

```
//Stoppuhr-Funktion:
void Stoppuhr()
{
  //Die Zeiten werden berechnet.
  Sekunden = (60 + AktuelleZeit / 1000)% 60;
  Minuten = (60 + AktuelleZeit / 60000)% 60;
  Stunden = (25 + AktuelleZeit / 3600000)% 25;
  //Stringumwandlung, da mit Integer nicht geschrieben werden kann.
  String Stunden_zahl = String(Stunden,DEC);
  String Minuten_zahl = String(Minuten,DEC);
  String Sekunden_zahl = String(Sekunden,DEC);
  //Display Ausgabe:
  anzeige.clear();
  anzeige.drawString(0,0,"Stoppuhr");
  anzeige.drawVerticalLine(0,10,100);
  anzeige.drawString(5,20,Stunden_zahl);
  anzeige.drawString(15,20,"h");
  anzeige.drawString(30,20,Minuten_zahl);
  anzeige.drawString(40,20,"m");
  anzeige.drawString(55,20,Sekunden_zahl);
  anzeige.drawString(75,20,"s");
  anzeige.display();
}
```

Abb. 11

Wie man sieht, werden als erstes die Werte berechnet. Hier ist die Formel immer gleich: **Zeit= Anzahl bis zur nächsten Einheit + Aktuelle Zeit (wird später berechnet) / die Millisekunden der Zeit % Anzahl bis zur nächsten Einheit.** Hierbei gibt das Prozent-Zeichen den Rest aus. Hierbei kann anstatt der Aktuellen Zeit auch millis() verwendet werden, dann wird aber immer von der Systemzeit ausgegangen und die Uhr läuft im Hintergrund weiter. Die berechneten Werte sind aber alles noch Integer Werte und da diese nicht ausgegeben werden können folgt die Stringumwandlung. Hier werden die jeweiligen Integer-Werte in die String-Variable mit „Zeit_zahl" geschrieben. Da jetzt alle Werte berechnet wurden und ausgebbar sind folgt logischer Weise die Ausgabe auf dem Display. Mit dem ersten Befehl wird das Display zunächst bereinigt, die folgenden Befehle sind alle zur Ausgabe auf dem Display und haben eine ähnliche Struktur. Mit dem letzten Befehl wird dann alles ausgegeben und die Funktion ist abgeschlossen.

Nach der Stoppuhr-Funktion, folgt jetzt die Reset-Funktion. Diese soll alle Werte wieder auf Null setzten und das Programm in den Ausgangszustand setzen.

Zunächst werden hier alle Variablen auf Null, bzw. false gesetzt. Danach wird das Display bererinigt und wieder so beschrieben, dass der Ausgangszustand erreicht wird. (Diese Funktion wird auch in der setup() verwendet, um den Bildschirm zu erzeugen.)

```
//Reset-Funktion:
void Reset()
{
  //Mit Reset wird alles auf "0" gesetzt.
  Sekunden=0;
  Minuten=0;
  Stunden=0;
  Merker=0;
  anzeige.clear();
  anzeige.drawString(0,0,"Stoppuhr");
  anzeige.drawVerticalLine(0,10,100);
  anzeige.drawString(5,20,"0");
  anzeige.drawString(15,20,"h");
  anzeige.drawString(30,20,"0");
  anzeige.drawString(40,20,"m");
  anzeige.drawString(55,20,"00");
  anzeige.drawString(75,20,"s");
  anzeige.display();
}
```

Abb. 12

Nun kommen wir zur letzten Funktion der loop(). Hier werden die Funktionen bei Tasterdruck ausgeführt. In der ersten If-Verzweigung wird der Start-Taster abgefragt.

```
void loop() {

//Wenn der Start-Taster gedrückt wird, nimmt die Variable "StartZeit" millis() an.
//Außerdem wird die Boolsche Variable "Merker" auf 1 also True gesetzt.
if (digitalRead(Taster_Start)==HIGH)
{
  StartZeit=millis();
  Merker=1;
}
//Wird der Taster nicht gedrückt, wird die aktuelle Zeit berechnet: "millis" - "StartZeit"
else
{
  AktuelleZeit=millis() - StartZeit;
}
```

Abb. 13

Wenn er gedrückt wird bekommt die Variable „StartZeit" den Wert der Zeit in der das Programm schon läuft, also millis(). Das sind zum Beispiel 3 Sekunden. Außerdem wird die Boolsche Variable „Merker" auf 1 also true gesetzt. Dazu später mehr.

Wichtig: Die Zeit-Variable muss „unsigned long" sein, da millis() nur das zurück gibt. Ansonsten kann es zu einem semantischen Fehler kommen.

Wird der Start-Taster aber gerade nicht betätigt, dann wird die Variable „Aktuelle Zeit" berechnet. Dazu benötigen wir die die vorhin deklarierte StartZeit-Variable und wieder den millis-Befehl. Die AktuelleZeit benötigen wir in der Stoppuhr Funktion. (Diese Variable muss auch „unsigned long" sein.) Der Reset-Taster ist relativ Simpel: Wird er gedrückt, wird die Reset-Funktion ausgeführt und alles wird zurückgesetzt.

```
//Wird der Reset Taster Gedrückt, wird auch die Reset-Funktion ausgeführt.
if (digitalRead(Taster_Reset)==HIGH)
{
  Reset();
}
```

Abb. 14

Nun die letzten Zeilen Programm-Code der Display Stoppuhr. Mit einer „While-Schleife" wird hier der Merker abgefragt. Solange wie der Merker auf 1, also true steht, läuft die Stoppuhr. !Der Merker wurde bei betätigung des Start-Tasters auf 1 gesetzt. In der While-Schleife wird aber nun auch der Stopp-Taster abgefragt.

```
//Während der Merker auf 1 steht wird die Stoppuhr ausgeführt.
while (Merker==1)
{
  Stoppuhr();
  //Wird aber währenddessen der Stopp-Taster gedrückt, so wird die Schleife durch "break" abgebrochen
  //und der Merker wird auf 0 also False gesetzt.
  if (digitalRead(Taster_Stopp)==HIGH)
  {
    Merker=0;
    break;
  }
  break;
}
}
```

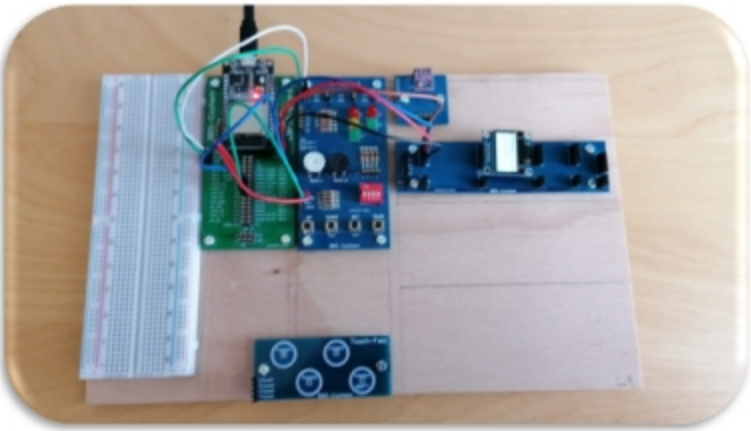
Abb. 15

Wenn dieser nämlich gedrückt wird, dann wird der Merker sofort auf 0 gesetzt und mit dem Befehl „break“ wird aus der Schleife ausgebrochen. Auf dem Display sieht das dann so aus, dass die Zeit stehen bleibt und man sie einfach ablesen kann. Das war es auch schon mit dem Stoppuhr-Programm. **Der Quellqode dazu ist hier zu finden:**

Stoppuhr_Display.ino

Auf dem ESP32-Board aufgebaut sieht das Ganze dann so aus:

Abb. 16



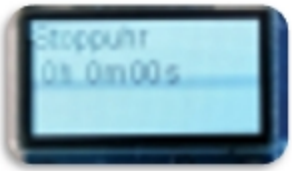


Abb. 18

Die Pinbelegung zum Display:

ESP32 3v3 - IO-Platine 3v3
ESP32 GND - IO-Platine GND
ESP32 SCL - I2C-Bus SCL
ESP32 3v3 - I2C-Bus 3v3
ESP32 GND - I2C-Bus GND
ESP32 Pin16 – IO-Platine UP
ESP32 Pin17 – IO-Platine DOWN
ESP32 Pin18 – IO-Platine SET

Abb. 17

Fehleranzeige programmieren

Der andere Programmier-Teil unseres Projektes war die Fehleranzeige. Das heißt, bei Berührung des Drahts soll eine unserer drei LEDs ausgehen. Das symbolisiert, dass ein Leben verloren wurde. Außerdem wird bei einem Fehler zusätzlich zur LED noch ein Ton von einem Piezo Buzzer abgegeben. Nun aber zur Programmbeschreibung.

Fehleranzeige1

```
//////////Fehleranzeige/////////////////

#define Draht 27
#define Buzzer 4
#define LED_Rot 19
#define LED_Gelb 21
#define LED_Gruen 22

int z;
```

Abb. 19

Zunächst werden wieder alle Pins definiert damit Sie später im Programm leichter auszumachen sind. Als Variablen haben wir hier nur eine Integer-Variable namens „z". Das ist unser Zähler.

In der setup() wird Input für den Taster und Output für die LEDs und den Buzzer deklariert. Außerdem wird der Zähler auf 0 gesetzt. Diesmal haben wir zwei eigene Funktionen geschrieben. Die erste davon soll einen Ton aus dem Buzzer ausgeben.

```
void setup() {
  pinMode(Draht, INPUT);
  pinMode(Buzzer, OUTPUT);
  pinMode(LED_Rot, OUTPUT);
  pinMode(LED_Gelb, OUTPUT);
  pinMode(LED_Gruen, OUTPUT);
  z = 0;
}
```

Abb. 20

```
void Ton()
{
  {
    digitalWrite(Buzzer, HIGH);
    delay(250);
    digitalWrite(Buzzer, LOW);
    delay(250);
  }
}
```

Abb. 21

Dazu wird hier einfach nur der Buzzer einmal auf HIGH gesetzt dann eine Verzögerung von 250ms und dann wieder auf LOW gesetzt und wieder eine Verzögerung. Damit macht er einfach einen kurzen Ton.

Die zweite Funktion, die wir haben ist die Alarm Funktion. Hier sollen alle LEDs blinken in dem Sie in einer for-Schleife immer wieder auf HIGH und LOW gesetzt werden. Zwischen dem Blinken ertönt außerdem immer wieder der Buzzer, indem hie die Ton-Funktion eingesetzt wird. Die Schleife soll 3-mal durchlaufen.

```
void Alarm()
{
  int i;
  for (i = 1; i <= 4; i++)
  {
    digitalWrite(LED_Gruen, HIGH);
    digitalWrite(LED_Gelb, HIGH);
    digitalWrite(LED_Rot, HIGH);
    Ton();
    delay(500);
    digitalWrite(LED_Gruen, LOW);
    digitalWrite(LED_Gelb, LOW);
    digitalWrite(LED_Rot, LOW);
    Ton();
    delay(500);
  }
}
```

Abb. 22

Nun wieder zur System-Funktion loop(). Hier wird wieder alles ausgeführt und hier benötigen wir auch zum ersten Mal den Zähler. Zunächst sollen ja alle LEDs an sein, um die drei Leben visuell darzustellen. Diese werden angeschaltet, wenn der Zähler 0 ist, da wir den Zähler in der Setup ja auf 0 gesetzt haben werden zu Programmstart alle LEDs angeschaltet. Nun da unser Zähler ja nicht um sonst seinen Namen trägt wird dieser, wenn der Draht berührt wird immer um eins hochgezählt. Da aber wenn der Draht berührt wird ja auch ein Fehler des Spielers begangen wurde, wird hier ein Ton erzeugt, um den Fehler zu verdeutlichen.

```
void loop() {
  //Alle LEDs zu Beginn anschalten:
  if (z == 0)
  {
    digitalWrite(LED_Gruen, HIGH);
    digitalWrite(LED_Gelb, HIGH);
    digitalWrite(LED_Rot, HIGH);
  }

  //Wenn der Draht berührt wird, erhöht sich der Zähler um 1.
  if (digitalRead(Draht) == HIGH)
  {
    z = z + 1;
    Ton();
  }
```

Abb. 23

Je nachdem wo der Zähler nun steht, beziehungsweise wie oft der Draht berührt wurde, gehen sie entsprechenden LEDs aus. Wenn der Draht 3-mal berührt wurde sind alle LEDs aus und das Spiel ist verloren. Da auch klar dargestellt wird das man Verloren hat kommt hier unsere Alarm-Funktion zum Einsatz.

```
//Grüne LED bei Berührung aus. (erstes Leben verloren)
if (z == 1)
{
  digitalWrite(LED_Gruen, LOW);
}

//Gelbe LED bei Berührung aus. (zweites Leben verloren)
if (z == 2)
{
  digitalWrite(LED_Gelb, LOW);
  digitalWrite(LED_Gruen, LOW);
}

//Rote LED bei Berührung aus. (drittes Leben verloren)(Spiel aus).
if (z == 3)
{
  digitalWrite(LED_Rot, LOW);
  digitalWrite(LED_Gelb, LOW);
  digitalWrite(LED_Gruen, LOW);
  delay(500);
  z=4;
  Alarm();
}
```

Abb. 24

```
//Reset in Ausgangsstellung
if (z > 3)
{
  z = 0;
  digitalWrite(LED_Gruen, HIGH);
  digitalWrite(LED_Gelb, HIGH);
  digitalWrite(LED_Rot, HIGH);
}
}
```

Abb. 25

Zuvor wurde der Zähler nach dem Alarm noch auf 4 gesetzt. Wenn der Zähler größer als 3 ist wird er hier zurückgesetzt und alle LEDs gehen wieder an. (Der Ausgangszustand ist erreicht.) So muss das Programm nicht andauernd neu starten.

Das war es auch schon wieder mit der Fehleranzeige.

Der Quellcode dazu ist hier zu finden:

Fehleranzeige1.ino

Auf dem ESP32-Board aufgebaut sieht das Ganze dann so aus:

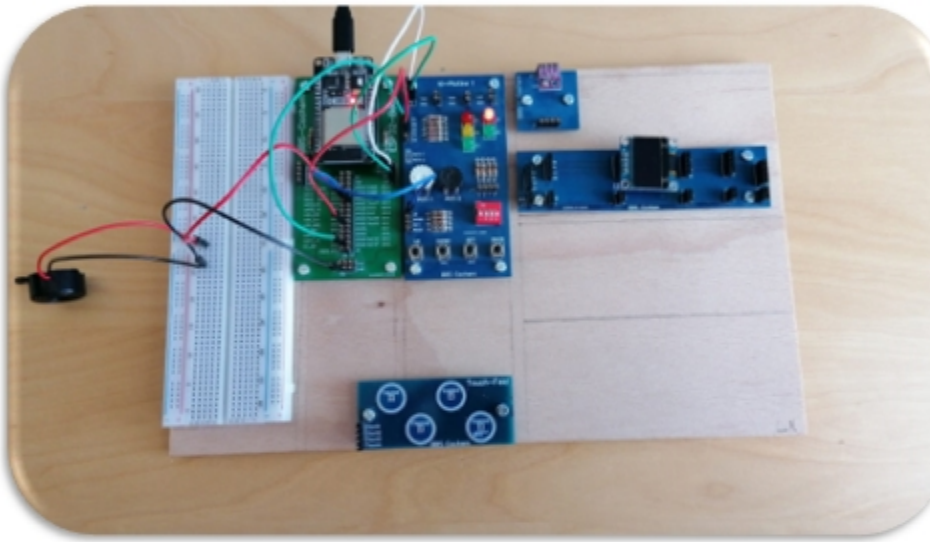


Abb. 26

Abb. 27 Die Pinbelegung für die Fehleranzeige:

ESP32 3v3 - IO-Platine 3v3
ESP32 GND - IO-Platine GND

ESP32 GND – Buzzer GND
ESP32 Pin4 – Buzzer Plus

ESP32 Pin19 – IO-Platine D1
ESP32 Pin21 – IO-Platine D2
ESP32 Pin22 – IO-Platine D3

ESP32 GND – Stick
ESP32 Pin 23 – Draht

Hardware

Aufbau/Komponentenbeschreibung

Die zwei Hauptteile unseres Projektes in der Hardware sind der ESP32 und „Der heiße Draht". Der ESP-32 ist ein 32-Bit-Mikrocontroller, welcher am 6. September 2016 veröffentlicht wurde. Zwei Eigenschaften des ESP-32 sind Wifi und Bluetooth. Zudem verfügt er über 48 Pins. Auf dem ESP-32 befindet sich das Programm für die Lebensanzeige des Spiels für die drei LEDs benötigt werden da man drei Leben besitzt. Diese LEDs (Leuchtdioden) sind sogenannte Halbleiterdioden, welche Licht, Infrarotstrahlung oder Ultraviolettstrahlung aussenden können. Es ist wichtig diese Leuchtdioden in Durchlassrichtung zu schalten damit sie funktionieren. LEDs benötigen nur wenig Energie für eine große Leuchtkraft. Bei berühren des Drahtes mit dem Stick verliert man eines der drei Leben und eine LED geht aus. Nachdem man alle drei seiner Leben verloren hat ertönt ein geräusch durch den Buzzer und man muss durch einen Taster reseten. Nach dem betätigen des Tasters für eines resets gehen alle drei LEDs wieder an und das Spiel begint von neuem. Um für die LEDs eine passende Stromversorgung bereitzustellen sind vor ihnen 270 Ohm Wiederstände welche die Farbreihenfolge Rot, Lila, Braun, Gold haben eingebaut um die 3.3V des Esps zu verringern. Diese drei Leds werden über die Pins D1,D2 und D3 angesteuert. Diese drei Pins werden mit drei pins an den ESP-32 angeschlossen. Als Alternative zu den LEDs können auch LED-Strips genutzt werden, da bei LED-Strips verschiedene Farben eingestellt werden können.

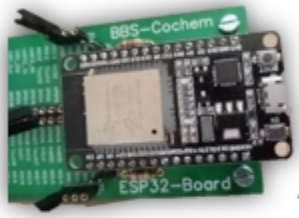


Abb. 28

Abb. 29

Jegliche Verbindungen zwischen den Pins und den Bauteilen bestehen aus Jumperkabeln mit einer länge von 14cm. Diese Jumperkabel sind kleine Steckbrücken, die als eine Form von Kurzschlusssteckern auf die Kontakte von Stiftleisten gesteckt werden. Dadurch werden die Pins, welche auf diese Jumper gesteckt werden, elektrisch miteinander verbunden (gebrückt). Die LEDs sowie die Wiederstände befinden sich auf einer IO Platine welche über Jumperkabel von den Pins VDD zu 3v3 und GND(Ground) zu GND verbunden ist um mit Strom versorgt zu werden und in Kontakt mit dem ESP-32 zu sein. Auf dieser Platine sind zudem Taster, Schalter und auch Regler mit dazugehörigen Wiederständen enthalten.

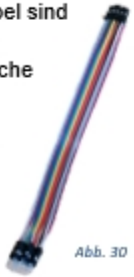


Abb. 30

Die Eigentlich wichtigste verbindung in diesem projekt ist jedoch die zwichen dem ESP- 32 und dem Stick des Heißen Drahtes sowie der Führungsschiene. Zu beginn waren wir uns nicht sicher wie wir diese Sachen anschließen da wir nur wussten, dass ein Stromkreis geschlossen wird wenn man mit dem Stick die Führungsschiene berührt. Uns wurde dann klar, dass die berührung zwischen Stick und Draht im Prinzip die gleiche Funktion wie ein Taster aufweißt bei dem ein Signal gesendet wird sobald er gedrückt wird. Ein Taster ist ein Wippschalter, in dem eine Feder enthalten ist, wodurch er nach betätigen in die Ausgangsstellung zurückspringt. Beim Drücken des Tasters gibt er einen Impuls an das Relais. Die Spule des Relais zieht an und schließt bzw. öffnet den Kontakt des Relais (Ein Relais kann öffnen und schließen). Da bei einem Taster und bei unserem Projekt ein Stromkreis geöffnet und geschlossen wird haben wir die zwei Komponenten wie einen Taster angeschlossen. Der Stick wird an GND angeschlossen und die Schiene an einen Pin des ESPs wodurch bei Berühren der Schiene mit dem Stick ein Signal an den Pin des ESPs gesendet wird.

Abb. 31

Die Führungsschiene wird auf einem 40cm mal 34cm großem Holzbrett befestigt. Und an einem der zwei Enden des Drahtes wird ein Jumperkabel befestigt und in einen Pin gesteckt. Die Führungsschiene besteht aus einem ungefähr 60cm langem Kupferdraht welcher 5 Biegungen besteht. Diese Biegungen sind dafür da, um es zu erschweren der Schiene mit dem Stick zu folgen, ohne sie zu berühren. Der Stick ist 30cm lang und hat eine kleine Schlaufe am Ende an der ein Kabel festgelötet ist, welches an den GND Pin angeschlossen ist. Auf der anderen Seite des Sticks befindet sich eine etwas größere Öse, welche dafür da ist, um an der Schiene entlang geführt zu werden. An der Öse befindet sich eine kleine Öffnung, um auf die Schiene geführt zu werden. Durch das Signal welches bei Kontakt zwischen stick uns Schiene entsteht verliert man nicht nur ein Leben und eine LED geht aus, sondern es ertönt auch ein Geräusch.

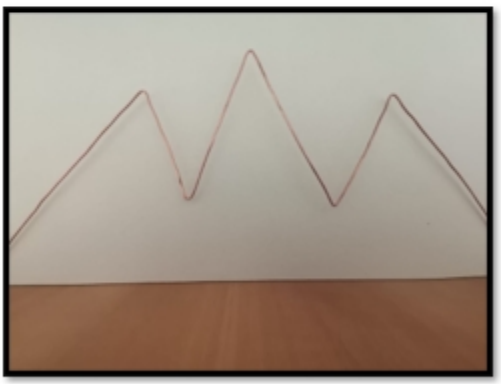


Abb. 32

Dieses Geräusch kommt von einem Piezo Buzzer welcher mit zwei Kabeln, welche an ihm enthalten sind an das Breadboard (Steckplatine) angeschlossen wird. Eine Steckplatine ist für die mechanischen Befestigung und elektrischen Verbindung von elektronischen Bauteilen für Versuchsschaltungen und Experimente da. Diese Kabel sind wiederrum an zwei Jumperkabel angeschlossen, welche zum GND Pin und einem freien Pin des ESPs führen, um ihn ansteuern zu können. Der Piezo Buzzer ist ein sogenannter Summer (Buzzer), Ein Buzzer ist ein Gerät, welches eine Art summenden Ton erzeugen kann.

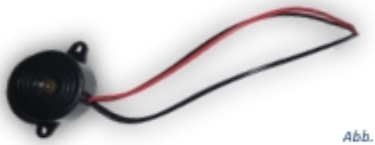
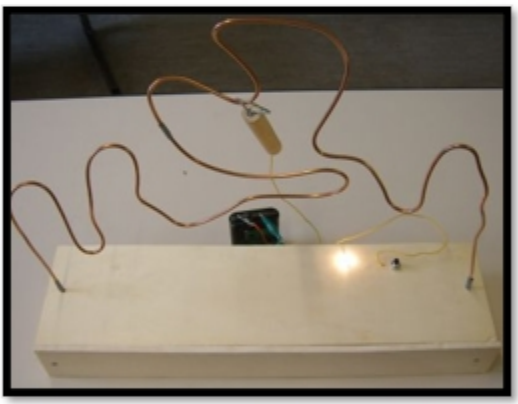


Abb. 33

Optimierungen am Spiel „heißer Draht“

Das Prinzip des Spieles Heißer Draht ist es Geschicklichkeit zu beweisen. Ziel ist es, eine Draht-Öse so schnell wie möglich über einen gebogenen Draht zu führen, ohne diesen mit der Öse zu berühren. Öse und Draht sind dabei an eine Spannungsquelle angeschlossen und bilden einen unterbrochenen Stromkreis. Bei einer Berührung des Drahtes wird der Stromkreis geschlossen. Bei Schließen des Stromkreises ertönt meist ein Ton oder eine Led leuchtet auf. Diese Schaltung kann meist mit einem Schalter aktiviert oder deaktiviert werden. Dieses Prinzip ist im Grunde identisch zu dem unseres Projektes.

Abb. 34



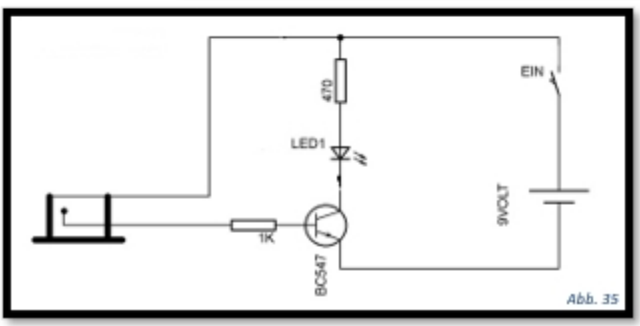
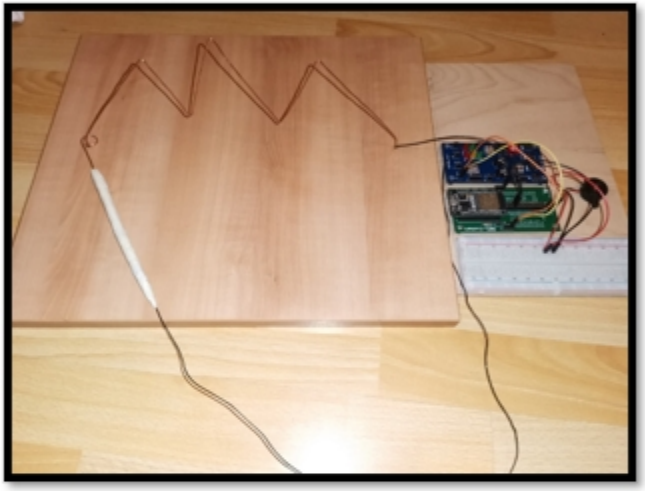


Abb. 35

Jedoch haben wir noch Verbesserungen vorgenommen. Diese Verbesserungen werden ermöglicht durch die Anbindung unseres ESP-32. Durch ein Programm welches noch drei zusätzliche Leds anbindet welche drei Leben darstellen sollen. Bei einer Berührung des Drahtes geht eine Led aus und man verliert ein Leben. Zusätzlich ertönt ein Summen, nachdem alle drei Leben verloren sind und alle Leds aus sind. Eine Version unseres Programmes enthält ein Display, mit dem man seine eigene Zeit stoppen kann und auch wieder resetten kann. Dadurch wird unsere Version des Spieles spannender und herausfordernder.

Abb. 36



Kritische Reflexion

Als abschließendes, ehrliches und kritisches Fazit werden wir hier noch ein paar Worte zu dem Thema schreiben. Bei unserem Projekt handelte es sich um eine verbesserte Version des Spiels „heißer Draht". Dies beinhaltet zusammengefasst das Hinzufügen von einer Lebensanzeige (3 LEDs), einem Fehlerton durch einen Buzzer, eines ESP32-Mikrocontrollers zum Programmieren und eines OLED-Displays, um die Zeit zu stoppen. Im Großen und Ganzen sind wir mit unserem Arbeitsergebnis zufrieden, sowie mit unserer Vorgehensweise bei der Bearbeitung des Projektes. Zwar hatten wir am Anfang Startschwierigkeiten mit der Anbindung des heißen Drahts an den ESP32, welche jedoch nach ausreichender Recherche geklärt wurden. *Der Weg war das Ziel.* Eine große Hilfe bei der Bearbeitung des Projektes war unsere Betreuende Lehrkraft Herr Franzen, welcher uns bei Fragen immer zur Verfügung stand und so gut es ging weitergeholfen hat. Außerdem hat uns geholfen, dass wir als Ersatzleistung für das verpflichtende Schulpraktikum zum Erreichen der Fachhochschulreife, bei der wir auch ein Projekt gemacht haben über das Programmieren eines ATtinys. Dadurch hatten wir ein Gefühl dafür, wie und in welcher Geschwindigkeit wir arbeiten müssen. Durch den verwendeten ESP32 gibt es natürlich nahezu unendlich viele Möglichkeiten das Spiel zu erweitern und zu verbessern. Zum Beispiel hätte man die gestoppte Zeit speichern und wieder ausgeben können, oder dass wenn man das Ziel erreicht auch in irgendeiner Weise eine Meldung bekommt. Jedoch bestand nicht genügend Zeit, um viele Verbesserungen vorzunehmen, auch aus dem Grund da wir zeitgleich noch Unterricht hatten. Alles in allem war es eine sehr lehrreiche Erfahrung, bei der wir viel über Projektabläufe und Teamarbeit lernen konnten. Diese Erfahrung gesammelt zu haben wird uns in unserem späteren Berufsleben eine große Hilfe sein.

Leon Hürter und [redacted]

Abb. 37

Literaturverzeichnis/Quellen:

https://sites.google.com/site/bastelnelektroelektrik/home/grundlagen-heiser-draht-bauen

https://draeger-it.blog/arduino-spiel-1-der-heisse-draht/

https://michaelsarduino.blogspot.com/2015/11/heier-draht-mit-dem-arduino.html

https://www.kreativekiste.de/heisser-draht-spiel-arduino-projekt

https://forum.arduino.cc/index.php?topic=554257.0

https://michaelsarduino.blogspot.com/2015/11/heier-draht-mit-dem-arduino.html

https://iotspace.dev/5-einfache-arduino-projekte-maerz-2020/

Erklärung zur praktischen Prüfung

Erklärung zu Plagiaten

Ich erkläre hiermit, dass ich die vorgelegte Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe.

Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quelle kenntlich gemacht.

Mir ist Folgendes bekannt: Die ungekennzeichnete oder nicht angemessen gekennzeichnete Übernahme von fremden Texten (= Plagiat) gilt als schwerer Verstoß gegen das Urheberrecht sowie gegen die Ethik wissenschaftlichen Arbeitens (Respekt vor der Leistung anderer). Dies gilt nicht zuletzt für Quellen aus dem Internet, die mindestens mit dem Autor (soweit recherchierbar), Titel (sofern vorhanden), Adresse und Recherchedatum auszuweisen sind.
Ich bin mir dessen bewusst, dass die Aufdeckung eines Plagiatsfalles mit einer Abwertung der inhaltlichen Bewältigung geahndet werden kann.

Anhang

Alle Bilder, welche nicht von uns selbst gemacht wurden, sind im jeweiligen Bild selbst verlinkt. (siehe Abbildungsverzeichnis)

Zum praktischen Aufbau: Die zwei geschriebenen Programme in dieser Dokumentation werden separiert auf zwei verschiedenen ESP-32 laufen, da der ESP nicht überlastet werden soll und so alles flüssiger läuft.