Frankfurt University of Applied Sciences

Fachbereich 2: Informatik und Ingenieurwissenschaften

Bachelorarbeit

im Studiengang Informatik B. Sc.

zur Erlangung des akademischen Grades

Bachelor of Science

Thema:	Entwicklung und Implementierung eines Spiels für eine RGB-Matrix, deren Ansteuerung mit Einplatinencomputern erfolgt
Autor:	Dimitri Gubermann geb. am 02.02.1986 in Tiraspol/Moldawien d.gubermann@gmail.com Matrikelnummer: 980065
Eingereicht am:	14. Dezember 2017
1. Betreuer:	Prof. Dr. Christian Baun
2. Korreferent:	Prof. Dr. Thomas Gabel

Eidesstattliche Erklärung

Hiermit versichere ich, Dimitri Gubermann an Eides statt, dass ich die vorliegende Arbeit, selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter der Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form an irgendeiner Stelle als Prüfungsleistung vorgelegt worden.

Ort, Datum: Frankfurt am Main, den 14.12.2017 Unterschrift:

Danksagung

An dieser Stelle würde ich mich bei alle denen bedanken, die mich bei meinen gesetzten Zielen während der Anfertigung meiner Abschlussarbeit unterstützt haben.

Mein größter Dank geht an meine Frau Inna, die mich von Anfang meines Studiums motiviert und unterstützt hat.

Auch Herrn Maic Fischer danke ich sehr, der mich immer mit elektrotechnischem Wissen versorgt hat und seiner Frau Barbara Kinon für das Korrekturlesen meiner Abschlussarbeit.

Ich danke auch Herrn Prof. Dr. Christian Baun für das vorgeschlagene Thema und seine Unterstützung während der Anfertigung dieser Abschlussarbeit. Und natürlich Herrn Prof. Dr. Thomas Gabel, der sich bereit erklärt hat, als Korreferent bei dieser Abschlussarbeit mitzuarbeiten.

Kurzfassung

Die vorliegende Abschlussarbeit beschäftigt sich mit der Entwicklung und Implementierung eines Spiels auf einem Einplatinencomputer namens Raspberry Pi 3. Dabei erfolgt die Ausgabe des Spiels auf einer oder mehreren RGB-Matrixen. Das Pong ähnliche Retrospiel aus den 1970er-Jahren wird in der Programmiersprache Python implementiert. Zusätzlich werden ein Prototyp einer Anschlussplatine für den Anschluss von RGB-Matrixen und für die Steuerung des Spiels zwei Prototypen von Spielcontrollern entwickelt.

Abstract

This thesis describes the development and implementation of a game on a single-board computer called Raspberry Pi 3. The game output follows on one or more RGB- Matrixes. The retro game like Pong from 1970's is implemented in the Python programming language. In addition, a prototype of connection board for RGB- Matrixes and two prototypes of game controller to controll the game are being developed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

SoC	System-on-Chip
ARM	Advanced RISC Machines
GPU	Graphics Processing Unit
RAM	Random Access Memory
MHz	Megahertz
GHz	Gigahertz
MB	Megabyte
GB	Gigabyte
MB/s	Megabyte pro Sekunde
GPIO	General Purpose Input Output
LPDDR	Low Power Double Data Rate Memory
HDMI	High Definition Multimedia Interface
SD-Karte	Secure Digital Memory Card
CSI	Camera Serial Interface
DSI	Display Serial Interface
USB	Universal Serial Bus
LED	Light-Emitting Diode
RGB	Red, Green, Blue (Rot, Grün, Blau)
CPU	Central Processing Unit
SSH	Secure Shell
RDP	Remote Desktop Protocol
CLK	Taktsignal
CLI	Command-Line Interface
VGA	Video Graphics Array
DVI	Digital Visual Interface

PWM	Pulsweitenmodulation
IDLE	Integrated Developement Environment
GND	Chassis Ground (Masse)
VCC	Voltage at the common collector
V	Volt
A	Ampere
mA	Milliampere
mm	Millimeter
ms	Millisekunde

1 Einleitung

Die Einplatinencomputer haben in den letzten Jahren einen festen Platz in der IT-Branche belegt. Dabei handelt es sich um Kleinrechner, die in der Lage sind, völlig autonom zu agieren. Die für den Betrieb benötigten Komponenten finden auf einer Leiterplatte Platz. Dabei wird die Fläche der Platinen von 100 Quadratzentimeter in Regel nicht überschritten. Je nach Art der Verwendung oder Einsatzort kann das System mit unterschiedlichsten Peripheriegeräten ausgestattet sein. So kann zum Beispiel für die Ausgabe ein Monitor und für die Eingabe eine Tastatur und Maus angeschlossen werden. Der Speicher kann mit einer SD-Karte oder einem anderen externen Speichermedium aufgerüstet werden. Der Einsatz der Einplatinencomputern ist sehr vielfältig, so sind diese in der Mess- und Automatisierungstechnik, Multimedia, Webtechnologien, Unterhaltungsindustrie sowie Luft und Raumfahrt zu finden. Dabei können die Systeme auf verschiedene Arten und Weisen betreiben werden, in denen man das System als alleinstehendes oder in Verbindung mit zusätzlichen Erweiterungsplatinen im Automatisierungsbereich einsetzt. [1]

Aus zahlreichen Einplatinenherstellern konnte der Raspberry Pi von der *Raspberry Pi Foundation* eine hohe Popularität erzielen und eine große Gemeinschaft mit enthusiastischen Bastlern aufbauen. Am Anfang hatte die Foundation vor, etwa zwischen 10.000 und 20.000 Stück zu verkaufen [2] mit dem Ziel einen kostengünstigen Rechner mit frei zugänglicher Software, um die Begeisterung bei Jugendlichen und Erwachsenen für das Programmieren und Experimentieren zu wecken [3] und den Zugang zu Computern und der digitalen Welt zu ermöglichen [4]. Mittlerweile wurde der Kleinerechner im Kreditkartenformat, über 12.5 Millionen Mal verkauft (Stand Februar 2017) und belegt somit den dritten Platz in der Rangliste der meistverkauften Computer [5]. Der Kleinrechner besitzt viele Anschlussschnittstellen. Eine von denen ist die frei programmierbare Schnittstelle. An diese kann man nicht nur verschiedene Sensoren, Motoren oder Erweiterungsplatinen, sondern auch Ausgabegeräte wie LCD-Bildschirme anschließen.

1.1 Ziel

Das Ziel dieser Arbeit ist, ein Spiel auf einem Einplatinencomputer zu entwickeln, das dem Spiel Pong gleicht und die Ausgabe auf einer oder mehreren RGB-Matrizen erfolgt. Zusätzliche erfolgt eine Entwicklung von diversen Komponenten die für das Spiel und die RGB-Matrizen relevant sind.

1.2 Motivation

Die Technologie, der RGB-Matrix kommt ursprünglich aus der Werbebranche in Form einer Werbetafel. Diese Werbetafeln sind digitale Anzeigen, die als Aushängeschilder oder großflächige Bedeckungen an Hausfasenden zu finden sind. Unter anderem finden diese ihren Einsatz im Straßenverkehr, als Verkehrsschilder, die per Fernsteuerung an die Verkehrsbedürfnisse angepasst werden. Für den privaten- und experimentellen Einsatz stehen kleinflächige RGB-Matrixen zur Verfügung. Diese Matrixen lassen sich mit unterschiedlichen Mikrocontrollern ansteuern, unter anderem auch mit dem Raspberry Pi. Dafür stehen zahlreiche Dokumentationen im Internet bereit. Unter anderem präsentieren die Entwickler kurze Demonstrationen ihrer Arbeiten auf unterschiedlichen Videos, Fotos und Blogportalen. Dabei werden nur kurze Videoausschnitte oder einzelne Fotos von Spielen, Laufschriften und zahlreichen Animationen veröffentlicht um zu demonstrieren, wie weit man die Ideen umsetzten kann. Die fehlende Beschreibung der Entwicklungsprozesse ist der Grund für die Motivation und die Entscheidung für das Thema dieser Abschlussarbeit gewesen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit besteht aus sieben Kapiteln. Im ersten Kapitel ist eine kurze Einleitung. Im zweiten Kapitel ist das Entwicklungssystem beschrieben. Das dritte Kapitel, beschreibt den Installationsvorgang und die Einrichtung des Betriebssystems Raspbian. Im vierten Kapitel sind die Grundlagen in Bezug auf die Erweiterungsanschlüsse erläutert. Der Hauptteil dieser Arbeit beginnt im fünften Kapitel mit der Beschreibung der Anforderungen an die Komponenten und des Spiels. Das sechste Kapitel beschreibt die Implementierung der Komponenten und des Spiels, sowie eine Untersuchung der diversen Ausgabe Varianten auf der RGB-Matrix. Im letzten Kapitel befindet sich die Auswertung der verfassten Arbeit.

2 Stand der Technik

In diesem Kapitel wird das Entwicklungssystem beschrieben, auf der die Entwicklung des Spiels und Ansteuerung der RGB-Matrix im Verlauf dieser Arbeit erfolgt. Unter anderem wird auf die Anschlüsse und Funktion des Standard Ein- und Ausgangs, kurz GPIO genannt, sowie das zu verwendende Betriebssystem eingegangen.

2.1 Entwicklungssystem

Seit dem Verkaufsstart 2012 ist mittlerweile die dritte Generation des Raspberry Pi auf dem Markt, der Raspberry Pi 3. Neben der Generationsbezeichnung eins bis drei existieren noch unterschiedliche Modelle wie A, B oder B+ und zwei Mini Modelle mit der Bezeichnung Zero beziehungsweise Zero W (Stand 15.10.2017). Diese Modellbezeichnung gibt eine Information über die Anzahl der USB Anschlüsse, Anzahl der Standard Ausgabe- und Eingabeanschlüsse (GPIO) und die Größe des Arbeitsspeichers, der je nach Modell von 256 MB bis 1 GB reicht. [6]

- Erste Generation
  - Raspberry Pi 1 Model A
  - Raspberry Pi 1 Model B
  - Raspberry Pi 1 Model B+
  - Raspberry Pi 1 Model A+
- Zweite Generation
  - Raspberry Pi 2 Model B
  - Raspberry Pi 2 Model B 1.2
- Dritte Generation
  - Raspberry Pi 3 Model B
- Zero Modelle
  - Raspberry Pi Zero
  - Raspberry Pi Zero W

Das in dieser Arbeit verwendete Entwicklungssystem Raspberry Pi 3 ist mit einem SoC BCM2837 von der Firma Broadcom ausgestattet. Auf diesem Chip ist ein 64bit ARMv8 mit vier Kernen Cortex A53 Prozessor, der mit 1.2 GHz taktet, eine Grafikeinheit mit zwei Kernen VideoCore IV GPU und 1 GB DDR2 Arbeitsspeicher verbaut. Die Grafikeinheit hat einen Takt von 400 MHz und verfügt über eine OpenGL ES 2.0 Unterstützung, einen Hardwarebeschleuniger OpenVG sowie einen Decoder 1080p30 H.264

für hochauflösende Video Darstellung. Für die Kommunikation steht ein Kabel gebundener Ethernet 10/100 Anschluss zur Verfügung und für die Drahtlose Verbindung ein BCM43438 Chip von der Firma Broadcom, der über einen 2.4GHz 802.11n WLAN Standard und einen Bluetooth Chip, der über Klasse 4.1 verfügt. Als nichtflüchtiger Speicher dient eine Micro SD Karte, auf der sich das Betriebssystem befindet. Zum Anschließen von Erweiterungsplatinen oder Sensoren stehen 40 Stiftleistenanschlüsse bereit. Für den Anschluss von Ausgabegeräten ist ein HDMI Anschluss und eine Kombibuchse 3.5 mm mit einer analogen Audio-Video Ausgabe verbaut. Darüber hinaus stehen zwei serielle Anschlüsse DSI und CSI zum Anschluss von einem Monitor und einer Kamera zu Verfügung. Für den Anschluss von USB Geräten, wie Maus und Tastatur sind vier USB Anschlüsse vorhanden sowie ein Micro USB Anschluss für eine Stromquelle, die eine Spannung von 5 V und Stromstärke von 2.5 A entgegennimmt. [7] In der Tabelle 1, ist die komplette Spezifikation des Systems aufgelistet, um einen Überblick der verbauten Komponenten, Abmessungen und Anschlüssen zu verdeutlichen.

Aktuell liegen die Anschaffungskosten durchschnittlich bei 34 Euro (Stand 15.10.2017), dabei können die Preise von Händler zu Händler variieren. Je nach Betrieb können weitere Investitionen entstehen, in dem man zusätzliche Komponente benötigt, wie einen Monitor, HDMI Kabel, Netzteil, Eingabegeräte und eine Speicherkarte. [8]

Tabelle 1: Raspberry Pi 3 Model B Spezifikation[1]

Bezeichnung	**Typ**
SoC	Broadcom BCM2837
CPU	4x ARM Cortex-A53, 1.2 GHz
GPU	Broadcom VideoCore IV
RAM	1 GB LPDDR2 (900 MHz)
Netzwerk	10/100 Ethernet, 2.4 GHz 802.11n WLAN
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Speicher	microSD
Erweiterung	40-pol, männlich
Anschlüsse	3.5 mm analoge Audio-Video Buchse,
	4x USB 2.0, Ethernet, CSI, DSI
Abmessungen	8.50/5.6/1.7 cm
Stromversorgung	5 V bis zu 2.5 A über microUSB Anschluss

[1] Vgl.: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

2.1.1 Erweiterungsanschlüsse

Der Raspberry Pi 3 verfügt über einen doppelreihigen Stiftleistenanschluss mit 40 Kontakten für den Anschluss von elektronischen Geräten. Die Stiftanschlüsse haben unterschiedliche Funktionen. Ein Teil der Anschlüsse hat eine feste Belegung und ist nicht veränderbar. Dazu gehören vier Anschlüsse die für Stromversorgung von elektronischen Geräten, die eine Spannung von 3.3 Volt beziehungsweise 5 Volt aufweisen, sowie acht Anschlüsse mit 0 Volt Masse und zwei Anschlüsse für das Ansprechen der EEPROMs, die auf einer Erweiterungsplatine verbaut sind. Die anderen sind je nach Vorhaben frei programmierbar. [9]

Für die Bezeichnung der Erweiterungsanschlüsse existieren drei verschiedene Nummerierungsarten. Die eine ist die physische Bezeichnung der Kontakte. Diese werden von 1 bis 40 durchnummeriert. Dabei fängt man in der oberen linken Ecke mit der 1 und beendet die Nummerierung in der unteren rechten Ecke mit 40 (siehe Abbildung 1).

Abbildung 1: Nummerierung der Anschlüsse

Die linke Reihe beinhaltet die ungeraden Zahlen und die rechte Reihe die geraden. Die zweite Nummerierungsart ist die BCM Bezeichnung. Diese Bezeichnungen beziehen sich auf die Broadcom Anschlussnummern, die man in der offiziellen Dokumentation des BCM2837-Chips vorfindet. Die dritte Nummerierungsart ist der Anschlussname. Dieser Name (GPIO) mit einer Nummer und mit der Bezeichnung in den Klammern weist auf die Funktion des Anschlusses hin. In der Tabelle 2 sind alle Erweiterungsanschlüsse des Raspberry Pi 3 aufgelistet. [10]

Tabelle 2: Erweiterungsanschlüsse[2]

Bezeichnung	Pol	Pol	Bezeichnung
+3.3 V	1	2	+ 5 V
(SDA) GPIO 2	3	4	+5 V
(SCL) GPIO 3	5	6	GND
(GPCLK0) GPIO 4	7	8	GPIO 14 (TXD)
GND	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GND
GPIO 22	15	16	GPIO 23
+ 3.3 V	17	18	GPIO 24
(MOSI) GPIO 10	19	20	GND
(MISO) GPIO 9	21	22	GPIO 25
(SCLK) GPIO 11	23	24	GPIO 8 (CE0)
GND	25	26	GPIO 7 (CE1)
ID_SD	27	28	ID_SC
GPIO 5	29	30	GND
GPIO 6	31	32	GPIO 12
GPIO 13	33	34	GND
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
GND	39	40	GPIO 21

2.2 Speicher

Das System verfügt über keinen verbauten Speicher, auf dem man ein Betriebssystem oder Daten permanent speichern kann. Als Speicher dient eine microSD Karte, für die ein extra Schacht auf der Unterseite vorgesehen ist. Vor der Verwendung muss diese microSD Speicherkarte Mindestkriterien zu Kapazität und Geschwindigkeit aufweisen. Die Kapazität der verwendeten Speicherkarte ist vom Vorhaben abhängig. Die minimale Speicherkapazität soll nicht unter 4 GB sein. Empfehlenswert ist eine Karte zu verwenden, die eine Kapazität von 8 GB aufweist. [11] Die Klasse der Schreibrate ist das zweite Kriterium. Die Speicherklasse soll mindestens 4 betragen. Diese Zahl gibt an, welche Schreibrate die Karte beim Schreiben der Daten auf die Karte unterstützt. In diesem Fall soll eine Übertragungsrate von 4 MB pro Sekunde nicht unterschreiten. Diese Zahl ist auf der Karte in einem Kreis zu finden. Die Zahlen können 2, 4, 6 und 10 sein. Es existieren noch zusätzliche Bezeichnungen für die Geschwindigkeit, die sogenannte UHS Klasse,

[2] Vgl.: https://de.pinout.xyz

die bei Hochgeschwindigkeitskarten vorzufinden sind (siehe Abbildung 2). Eine Bezeichnung ist mit einer römischen Zahl (I bis III) neben dem SD Karten Logo dargestellt und gibt die Übertragungsgeschwindigkeit an. Diese können Geschwindigkeiten zwischen 50 MB/s und 300 MB/s erreichen. Die zweite Bezeichnung, die Zahl in einem U, gibt die mindestens verfügbare Schreibrate an. So steht zum Beispiel eine 1 für eine mindestens verfügbare Schreibrate von 10 MB/s und eine 3 für 30 MB/s. [12]

Abbildung 2: microSD Karte der UHS-Klasse

2.3 Betriebssystem

Für den Raspberry Pi existieren viele unterschiedliche Betriebssysteme. Die Wahl des Betriebssystems ist vom Vorhaben abhängig. Je nach Einsatzzweck ist ein dafür vorgesehenes Betriebssystem vorhanden, zum Beispiel für den Betrieb als Multimediacenter, Web-Server, Datencenter oder als Spielkonsole. Das offiziell von der Raspbery Foundation unterstützte Betriebssystem ist das *Raspbian*. Raspbian ist eine Portierung von *Debian 9 Stretch* für den Raspberry Pi. Neben dem Raspbian existieren auch verschiedene Betriebssysteme wie zum Beispiel OpenSuSe, Ubuntu, ArchLinux, sowie Betriebssysteme, die nicht auf Linux basieren. Dazu zählen OpenELEC, Risc OS, BSD Variante und Windows 10 IOT. Die Zahl dieser Betriebssysteme ist mittlerweile auf 40 angestiegen. [13]

2.4 RGB-Matrix

Eins der relevantesten Komponenten, neben dem Raspberry Pi, in dieser Arbeit ist die RGB-Matrix. Die RGB-Matrix dient als ein Ausgabegerät für grafische Inhalte. Der Hersteller ist *Adafruit Industries*. Der Anschaffungspreis einer RGB-Matrix, in der Größe 16x32 der Marke Adafruit, liegt aktuell bei 24,95 US Dollar (umgerechnet 24,23 Euro mit dem Stand vom 22.11.2017). Neben Adafruit Industries existieren noch andere Hersteller, die baugleiche oder ähnliche Fertigungen aufweisen. Es existieren unterschiedliche Größen, die von 16x32 Bildpunkte bis zu 64x64 Bildpunkte reichen. Die Ansteuerung der Matrix erfolgt über den Standard Ein- und Ausgang des Raspberry Pi. Zusätzlich lassen sich diese parallel oder seriell kaskadieren. Als Stromquelle dient ein

externes Netzteil. Dieses versorgt die Matrix mit einer Spannung von 5 V und einem Strom von 2 A. [14]

2.4.1 Anschlüsse

Eine RGB Matrix besitzt drei Anschlüsse. Zwei von diesen Anschlüssen sind 16-polige Wannenstecker. Einer ist für den Signaleingang und der andere ist für den Signalausgang zuständig. Der dritte ist für den Anschluss an eine Stromquelle. Dieser verfügt über einen Ein- und Ausgang, mit dem eine weitere Matrix mit Strom versorgt werden kann. Je nach Hersteller, existieren unterschiedliche Bezeichnungen der Kontakte der Wannenstecker, die aber dieselbe Funktionalität haben. So kann die Zählung der Kontakte, die für RGB Datenleitungen zuständig sind, mit der 0 oder mit einer 1 anfangen. Auch der Kontakt der das Strobe Signal entgegennimmt, kann *strobe*, *latch* oder *lat* heißen. Es kommen auch Matrixen vor, die über keine Bezeichnung der Kontakte verfügen. Nur der Netzteilanschluss hat die Bezeichnung VCC und GND (siehe Abbildung 3).

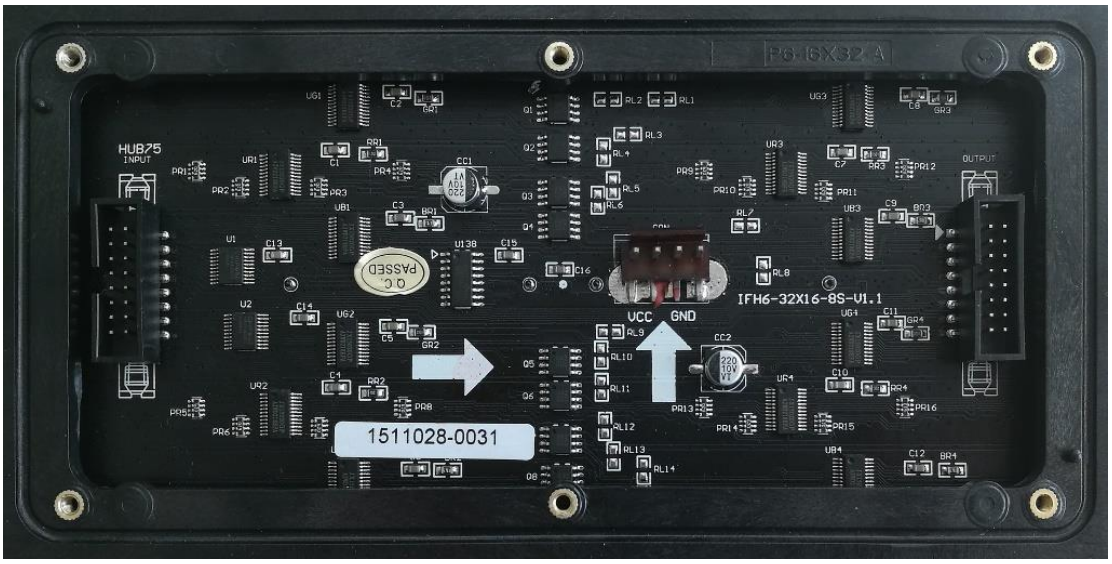


Abbildung 3: RGB-Matrix Rückseite

2.4.2 Funktionalität der Anschlüsse

In diesem Unterabschnitt ist die Funktionalität der einzelnen Anschlüsse der RGB-Matrix beschrieben.

Adressleitungen

Über die Adressleitungen lassen sich immer zwei Zeilen parallel ansteuern. Das bedeutet, dass die Leitung A, B und C eine 0 oder eine 1 liefert. Wenn alle Adressleitungen eine 1 liefern und somit eine Bitfolge von 111 ergibt was als Dezimalzahl 7 bedeutet.

Bei einer 16x32 Matrix lassen sich insgesamt 8x2 Zeilen ansteuern. Bei einer 32x32 Matrix ist eine zusätzliche Leitung D vorhanden. Mit vier Datenleitungen lassen sich 16x2 Zeilen ansteuern. [15]

RGB Datenleitungen

Die RGB Datenleitungen sind doppelt vorhanden. Ein Teil der Anschlüsse R1, G1, B1 sind für die obere Hälfte und die Leitungen R2, G2, B2 sind für die untere Hälfte zuständig. [15]

CLK Taktleitung

Die positive Taktflanke bewirkt die Verschiebung der Daten im Schieberegister um eine Position weiter. [15]

Strobe

Mit dieser negativen Flanke, die kurzen HIGH oder LOW liefert, lassen sich Bitmuster des Schieberegisters in einem Ausgangsdatenregister (Strobe) zwischenspeichern. [15]

OE/Output Enable

Das ist die negative Flanke, verfügt über ein HIGH- und LOW-Pegel und ist für den Ausgangsregister der anstehenden Daten zuständig. [15]

3 Vorbereitung und Installation

Dieses Kapitel beschreibt die Installation und Einrichtung des Betriebssystems Raspbian auf dem Raspberry Pi 3.

3.1 Benötigte Komponenten

Für die Einrichtung der Entwicklungsumgebung sollten folgende Geräte und Komponente bereitstehen:

- ein Rechner mit mindestens Windows 7 als Betriebssystem
- eine microSD-Speicherkarte mit Adapter
- ein Speicherkartenlese- und Schreibgerät
- Peripheriegeräte, wie Tastatur, Maus und Monitor
- verschiedene Kabel wie das HDMI-Kabel (gegebenenfalls einen Adapter HDMI auf VGA- oder DVI-Anschluss), Netzwerkkabel, Micro USB-Kabel
- Netzteil für den Raspberry Pi mit 5 V und 2 A

3.2 Vorbereitung

Als erstes lädt man ein Abbild des Betriebssystems herunter und entpackt dieses. Ein aktuelles Abbild des Betriebssystems kann man auf der Internetseite der Raspberry Pi Foundation[3] herunterladen. Dort stehen zwei Varianten des Betriebssystems zu Verfügung. Die eine Variante hat die Bezeichnung *Raspbian Stretch with Desktop* und ist mit einer grafischen Arbeitsoberfläche ausgestattet. Die zweite Variante ist die *Raspbian Stretch Lite*. Hierbei ist das Betriebssystem nur auf das notwendigste beschränkt und kommt standardmäßig ohne eine grafische Arbeitsoberfläche. Beide Varianten sind in der Version *September 2017* vorhanden (Stand 20. Oktober 2017). Im weiterem Verlauf dieser Arbeit kommt die *Raspbian Stretch with Desktop* Variante zum Einsatz. Zusätzlich sind zwei Anwendungen notwendig. Eine Anwendung, mit der man ein Packet entpacken kann, zum Beispiel die Anwendung *7-Zip*[4] und eine Anwendung mit der man ein Abbild auf eine microSD-Speicherkarte schreiben kann. Zum Schreiben eignet sich die Anwendung *Win32 Disk Imager*[5]. Beide Anwendungen kann man auf der Internetseite der Hersteller herunterladen. [16]

[3] Raspberry Pi Foundation: www.raspberrypi.org

[4] 7-Zip Herstellerseite: www.7-zip.de

[5] Win32 Disk Imager Herstellerseite: https://sourceforge.net/projects/win32diskimager

3.3 Beschreibung der microSD-Karte

Um ein Abbild auf die microSD-Karte zu übertragen, öffnet man die Anwendung Win32 Disk Imager. Nachdem die Anwendung gestartet ist, wählt man die Abbilddatei aus. Als nächstes ist der Datenträgerbuchstabe von der microSD-Karte auszuwählen. Um den Schreibvorgang zu beginnen, betätigt man die Schaltfläche *Schreiben*. Nach der Betätigung der Schaltfläche *Schreiben*, erscheint eine Warnung. Diese Warnung weist auf die Löschung der vorhandenen Daten auf dem Datenträger hin. Ist man mit der Löschung einverstanden, ist diese mit *Ja* zu bestätigen und der Schreibvorgang beginnt. Dabei wird die Speicherkarte in zwei Partitionen unterteilt. Die eine Partition ist für den Ladevorgang zuständig und auf der anderen Partition ist das System selbst abgelegt. In der Abbildung 4 ist eine Zusammensetzung der Bildschirmaufnahmen zu sehen, in denen die wichtigsten Schritte in Bezug auf den Schreibvorgang dargestellt sind. [16]

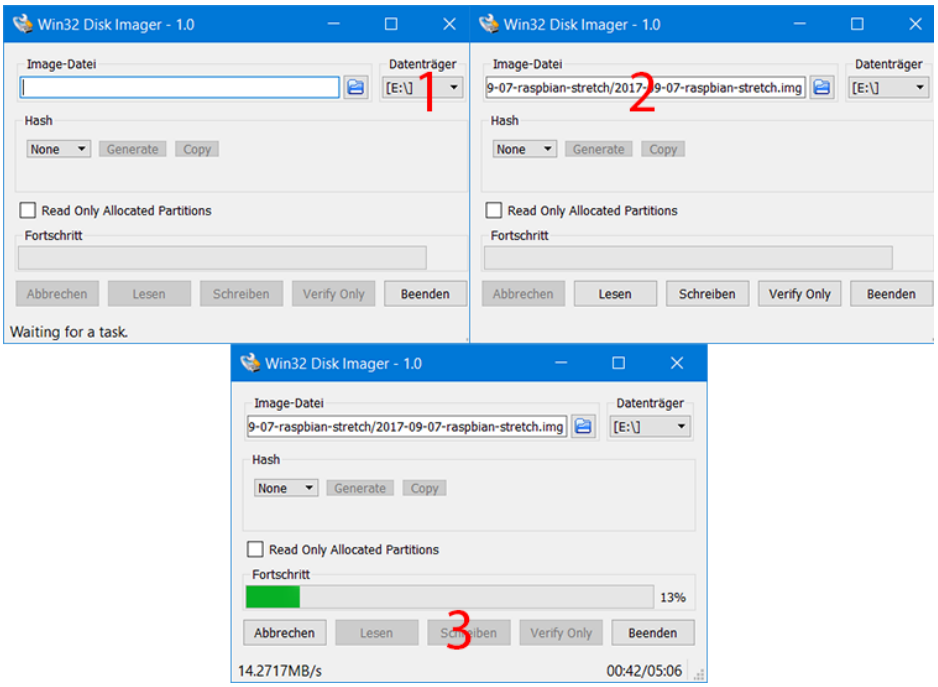


Abbildung 4: Bildschirmaufnahmen: Win32 Disk Imager

3.4 Konfiguration des Betriebssystems Raspbian

Bevor man den Raspberry Pi in Betrieb nimmt, müssen die gewünschten Peripheriegeräte wie Monitor, Eingabegeräte und die microSD-Karte angeschlossen sein. Als nächstes schließt man das Netzteil an. Das System schaltet sich ein und fährt das Betriebssystem hoch. Nachdem das Betriebssystem hochgefahren ist, erscheint die grafische Arbeitsoberfläche auf dem Bildschirm. Es ist empfehlenswert, als erstes das System einzurichten

und das Standardpasswort zu ändern. Der Standard Benutzername lautet *pi* und das Passwort *raspberry*. Um in die Konfigurationsanwendung aufzurufen, betätigt man die Himbeere in der oberen linken Ecke, dann auf *Preferences* und anschließend auf *Raspberry Pi Configuration* (siehe Abbildung 5). In der Konfigurationsanwendung hat man die Möglichkeit, das System zu konfigurieren, Schnittstellen einzustellen, Leistung der Grafikkarte zu verändern sowie Spracheinstellungen durchzuführen.

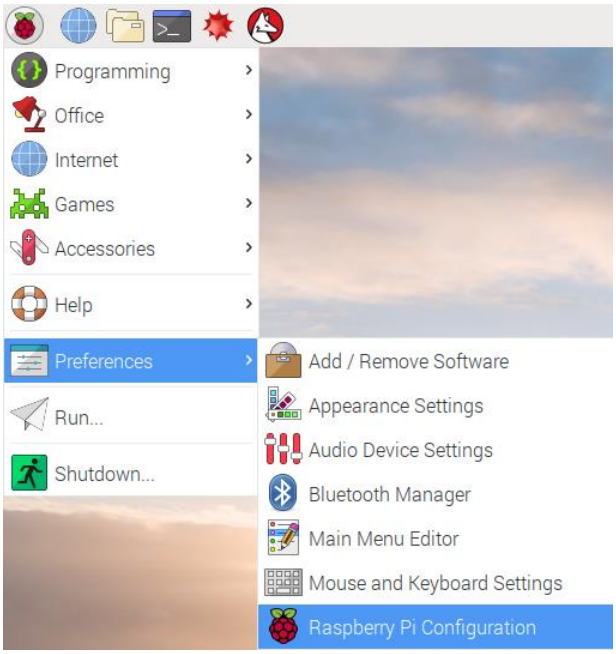


Abbildung 5: Bildschirmaufnahme: Raspberry Pi Konfiguration

3.4.1 System Aktualisieren

Um das System zu aktualisieren, öffnet man das Terminalfenster und führt den Befehl *sudo apt-get update* aus. Mit diesem Befehl aktualisiert man eine Liste, mit zur Verfügung stehenden Paketen. Anschließend kann man die installierten Pakete mit dem *upgrade* Befehl aktualisieren, in dem man *sudo apt-get upgrade* im Terminalfenster eingibt und ausführt.

3.5 Fernzugriff

3.5.1 SSH einrichten

Standardmäßig ist das SSH Protokoll aus Sicherheitsgründen deaktiviert. Dieses Protokoll kann man in der Konfigurationsansicht und im Reiter *Schnittstellen* aktivieren oder man schaltet den Raspberry Pi aus und entnimmt die microSD-Karte. Anschließend schließt man diese an den Rechner an. Auf dem Laufwerk mit der Bezeichnung *Boot* ist

eine leere Datei ohne Erweiterung namens *SSH*, zu erstellen. Für die Erstellung der Datei kann man jeden beliebigen Texteditor verwenden. Beim nächsten Systemstart aktiviert das System den SSH Protokoll automatisch. [16]

3.5.2 SSH unter Windows verwenden

Das Betriebssystem Windows verfügt nicht über eine native SSH Anwendung. Für den Zugriff auf den Raspberry Pi über den SSH Protokoll eignet sich die Anwendung *Putty*[6]. Für den Aufbau der Verbindung soll die IP Adresse des Rasberry Pis bekannt sein. Ist die IP-Adresse nicht bekannt, kann man diese mit Hilfe eines Netzwerk-Scanners ermitteln. Für diesen Zweck kann die Anwendung *Angry IP Scanner*[7] behilflich sein. Die IP Adresse erkennt man an dem Hostnamen des Raspberry Pi. Standardmäßig ist der Hostname *raspberrypi* (siehe Abbildung 6). Als erstes öffnet man die Anwendung Putty und gibt die Zugangsdaten in die vorgesehenen Felder ein. Die Standardzugangsdaten lauten, wie im Abschnitt 3.4 erwähnt, *pi* für den Benutzernamen und *raspberry* als Passwort.

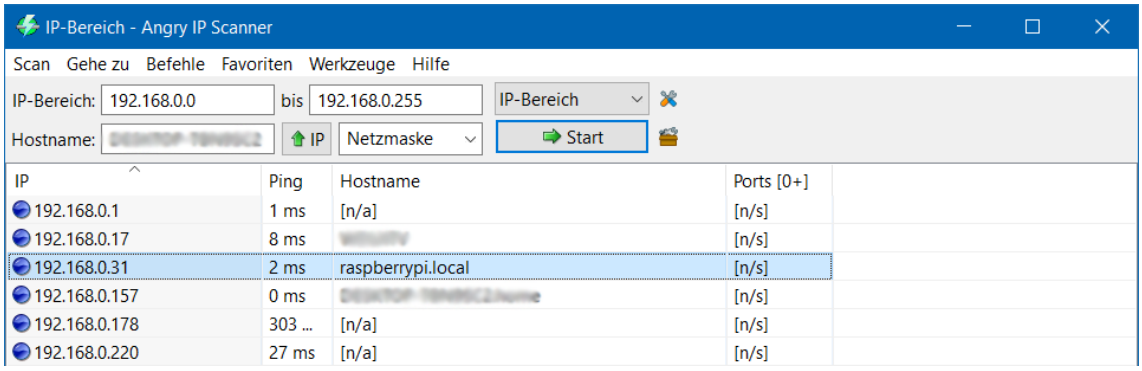


Abbildung 6: Bildschirmaufnahme: Angry IP Scanner

3.5.3 XRDP Einrichten

Für diesen Abschnitt ist eine erfolgreiche Verbindung über einen SSH Protokoll vorausgesetzt, da die Ausführung der Befehle und die Konfiguration des Betriebssystems über die Konsole erfolgt.

Eine weitere Möglichkeit das System per Fernzugriff zu bedienen, ist das RDP Protokoll. Um den Zugriff über dieses Protokoll zu ermöglichen, ist die Installation eines *XRDP Server* notwendig. Vor der Installation sind ein paar Änderungen am System durchzuführen.

[6] Putty Herstellerseite: http://www.putty.org/

[7] Herstellerseite http://angryip.org

Als erstes muss das System so konfiguriert sein, dass die grafische Arbeitsoberfläche automatisch beim Hochfahren startet. Das Starten kann man in der Konfigurationsanwendung einstellen. Dazu führt man den Befehl *sudo raspi-config* aus. Nach dem die Anwendung gestartet ist (siehe Abbildung 7), wählt man den Punkt *3 Boot Option* aus und anschließend den Punkt *B1 Desktop/CLI*. Im nächsten Schritt stehen zwei Auswahlpunkte bereit. Der Punkt *B3 Desktop* ermöglicht das automatische Starten von der Arbeitsoberfläche. Der Punkt *B4 Desktop Autologin* bietet zusätzlich ein automatisches Anmelden. Das automatische Anmelden geschieht nur unter der Bedingung, dass der Benutzername *pi* und das Passwort *raspberry* ist.

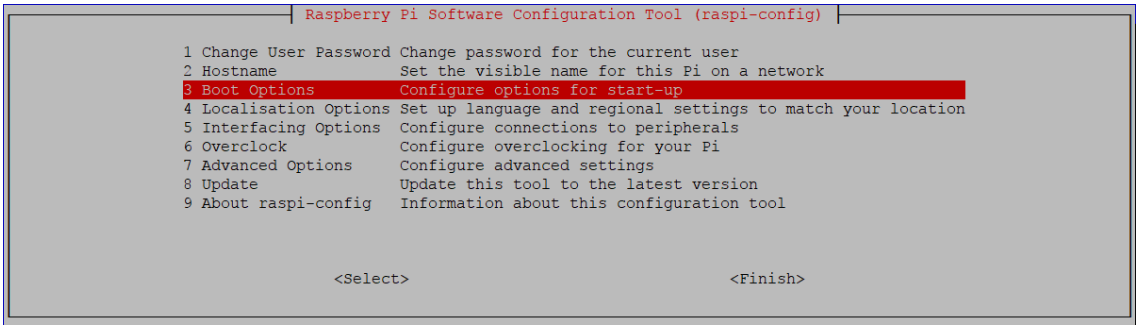


Abbildung 7: Bildschirmaufnahme: Raspberry Pi Software Configuration Tool

Im nächsten Schritt ist der *RealVNC-Server* zu deinstallieren. Das Deinstallieren erfolgt mit dem Befehl *sudo apt-get purge realvnc-vnc-server*.

Nach dem Deinstallationsprozess, kann man den XRDP Server mit dem Befehlt *sudo apt-get install xrdp* installieren. Anschließend ist das System neu zu starten.

Unter Windows geschieht der Fernzugriff mit Hilfe der Anwendung *Remotedesktopverbindung* (siehe Abbildung 8). [17] Als Zugangsdaten kann man die gleichen verwenden (falls nicht geändert), die im Abschnitt 3.4 erwähnt sind.

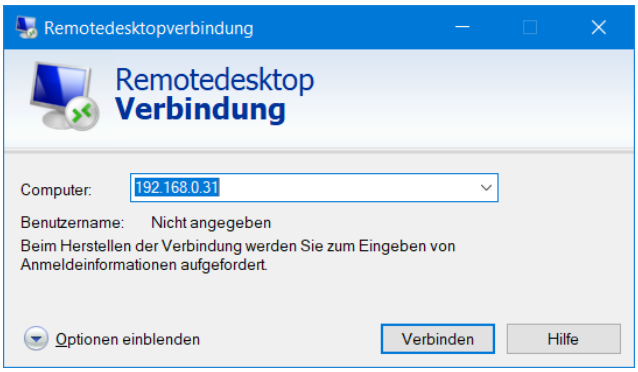


Abbildung 8: Bildschirmaufnahme: Remotedesktop Verbindung

4 Grundlagen

4.1 Ansteuerung der Standard Ein- und Ausgabe Anschlüsse

Bevor man sich an die Programmierung der Erweiterungsanschlüsse wendet, ist es empfehlenswert, über ein wenig Hintergrundwissen zu verfügen, um Schäden am Raspberry Pi und an elektronischen Geräten, die angeschlossen sind, zu vermeiden. Es ist davon abzusehen, externe Geräte über die Erweiterungsanschnlüsse mit Strom zu betreiben, da diese nur geringe Stromstärke als Ausgang vertragen. Ein Kontakt, der auf Ausgang geschaltet ist liefert eine Spannung von 3.3 V, dabei darf die Stromstärke von 16 mA und maximal 50 mA bei allen Anschlüssen inklusive Anschluss 1 und 17 nicht überschritten werden. [18] Für die Versorgung von elektrischen Geräten mit Strom ist es empfehlenswert, zu einer externen Stromquelle zu greifen, wie in Abbildung 9 zu sehen ist. Diese kann wahlweise mit einem Stromadapter über einen Stecker- oder über einen USB-Anschluss betrieben werden.

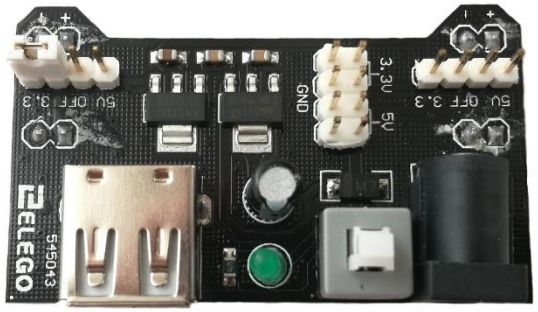


Abbildung 9: Externe Stromversorgung

4.2 Ansteuerung mit einem Taster

Ein Taster lässt sich auf zwei Wegen Implementieren, in dem man den Grundzustand des GPIO Anschlusses definiert. Dafür aktiviert man die Pull-Wiederstände der Erweiterungsanschlüsse. Der Pull-Widerstand kann entweder auf Pullup oder auf Pulldown geschaltet sein. Dabei wird der Spannungswert bei einem Pullup (HIGH) hochgezogen und bei Pulldown (LOW) runtergezogen. Einen hohen (HIGH) Spannungswert, das Hinaufziehen erreicht man, durch Verbinden eines GPIO Kontaktes über einen Widerstand mit einem Kontakt, der dauerhaft über einen hohen Spannungswert verfügt. In der Regel verwendet man einen Anschluss der Betriebsspannung (+VCC). Für den niedrigen Spannungswert, das Runterziehen, erreicht man, in dem ein GPIO Anschluss über einen Widerstand mit der Masse verbindet.

Die Wahl zwischen diesen beiden Methoden ist von der Aufstellung der Schaltung abhängig. Für eine einfache Implementierung eines Tasters, der beim Auslösen den Zustand von 0 auf eine 1 beziehungsweise von 1 auf 0 wechseln soll, ist es empfehlenswert die Pullup Methode zu wählen. [19]

4.2.1 Implementierung

Für die Implementierung benötigt man den Raspberry Pi, ein Steckbrett, einige Verbindungskabel und einen Taster. Als erstes baut man eine Schaltung auf. Den Schaltplan kann man aus der Abbildung 10 entnehmen.

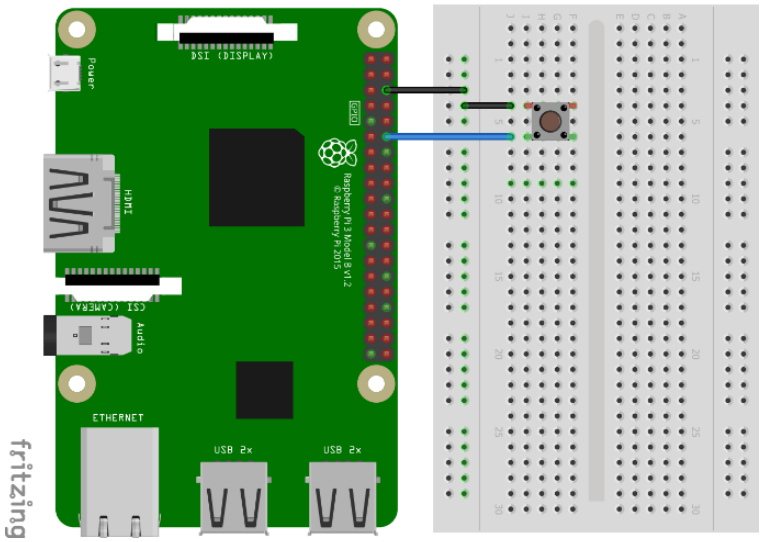


Abbildung 10: Aufbau der Schaltung mit Taster[8]

Es ist darauf zu achten, dass man den Taster richtig anschließt. Ein Taster sieht auf den ersten Blick durch seine quadratische Form von allen Seiten gleich aus. Dabei sind zwei gegenüberliegenden Seiten glatt und zwei Seiten, von denen je zwei Kontakte herausragen. Kontakte die eine längere Entfernung zu einander aufweisen, sind miteinander verbunden. Die anderen, die nebeneinander sind, haben einen unterbrochenen Stromkreis, der sich beim Betätigen des Tasters schließt. Für ein besseres Verständnis ist die Abbildung 11 zu betrachten.

[8] Erstellt mit der Anwendung Fritzing (http://fritzing.org)

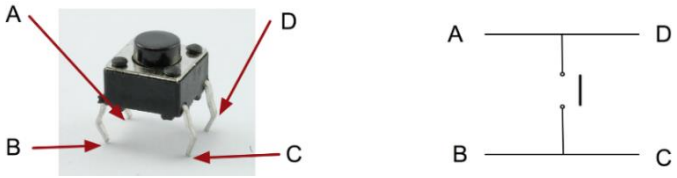


Abbildung 11: Taster und Schaltplan eines Tasters[9]

Nach dem Aufbau der Schaltung, folgt die Implementierung. Das Betriebssystem Raspbian verfügt standardmäßig über zwei vorinstallierte Versionen der Programmiersprache Python die Version 2.7 und die Version 3.5. Im weiterem Verlauf dieser Arbeit erfolgt die Implementierung mit der Python Version 3.5. Als Entwicklungsumgebung kommt die Python IDLE in der Version 3 zum Einsatz. Für die Implementierung benötigt man den RPi.GPIO Paket für Python, um die GPIO Anschlüsse anzusteuern. Um zu testen, ob dieses Paket auf dem System installiert ist, gibt man dazu im Terminalfenster folgendes ein:

```
$ sudo python3
>>> import RPi.GPIO as GPIO
```

Erscheint nach der Eingabe einer Fehlermeldung ist das Paket nicht vorhanden. Das Paket kann man über das Terminalfenster mit folgenden Befehlen installieren [20]:

```
$ sudo apt−get update
$ sudo apt-get install python3-rpi.gpio
```

Im Listing 1 ist ein Python-Quellcode abgebildet. Um diesen Quellcode auszuführen, ruft man die Entwicklungsumgebung *Python IDLE 3* auf und gibt den Quellcode ein. Anschließend speichert man den Quellcode unter dem Namen *taster.py* ab. Um den gespeicherten Code auszuführen, navigiert man im Terminalfenster an den Ort, in dem die Datei gespeichert ist und gibt anschließend *sudo python3 taster.py* ein. Kommt es zu keinem Kompilierungsfehler, kann man den Taster betätigen. Nach dem Betätigen des Tasters erscheint eine Zeile im Terminalfenster mit dem Inhalt *Taster gedrueckt*. [21]

```
1 # Titel: taster.py
2 # Autor: Dimitri Gubermann
3
4 import RPi.GPIO as GPIO
5 import time
6 # Board Nummerierung der GPIO-Pins
```

[9] Vgl.: http://razzpisampler.oreilly.com/images/rpck_1102.png

```
GPIO.setmode(GPIO. BOARD)
# Aktivierung des internen Pullup-Widerstandes
GPIO.setup(12, GPIO.IN, pull_up_down = GPIO.PUD_UP)
# In der While-Schleife erfolgt die Überprüfung des Zustandes
while True:
# Beim Drücken des Tasters kommt es zur Schließung des Stromkreises
# und der Eingangswert ändert sich von 1 auf 0
# In der If-Anweisung kommt es zu dieser Überprüfung
   if(GPIO.input(12) ==0):
      print("Taster gedrueckt")
   time.sleep(0.1) # Entprellzeit
# cleanup() für das Zurücksetzen der GPIO-Anschlüsse
GPIO.cleanup()
```

Listing 1: Quellcode taster.py

4.3 Ansteuerung einer mehrfarbigen Leuchtdiode

Eine Leuchtdiode ist grundsätzlich nicht direkt an eine Stromquelle anzuschließen. Der Grund dafür ist, dass die Leuchtdiode eine feste Betriebsspannung benötigt und diese unter der des Raspberry Pis von 3.3 V liegt. Aus diesem Grund ist es notwendig, die Spannung zu begrenzen. Das Eingrenzen der Spannung und Stromstärke kann man durch einen Vorwiderstand bewirken. Die Wahl des Vorwiderstandes kann man errechnen indem man das ohmsche Gesetz anwendet (siehe Formel 1). Das R steht für den Widerstand, U - für die elektrische Spannung und das I - für den elektrischen Strom. Für die Berechnung benötigt man die Werte von der Vorwärtsspannung (U) und dem Vorwärtsstrom (I). Diese Werte kann man aus dem Datenblatt der Leuchtdiode entnehmen. Die verwendete Leuchtdiode hat eine Vorwärtsspannung von 1,95 V für Farbe Rot, 3,3 V für Farbe Grün beziehungsweise für Farbe Blau und Vorwärtsstrom von 20 mA.

Da 20 mA der Maximalwert ist, sollte man zu einem niedrigeren Wert greifen. 10 mA reichen aus, um die Leuchtdiode zum Leuchten zu bringen. Im weiteren Verlauf dieses Abschnittes erfolgt die Berechnung mit den Werten für die rote Leuchtdiode.

$$R = \frac{U}{I} \tag{1}$$

Nachdem alle Werte ermittelt sind, kann man die Berechnung durchführen (siehe Formel 2). Das Ergebnis beträgt laut der Berechnung 135 Ohm und somit sollte der Vorwiderstand diesen Wert haben. In der Regel verwendet man einen nächst kleineren oder nächst größeren Vorwiderstand, da es, laut der E-Reihe E12, keinen Vorwiderstand mit dem Wert 135 Ohm gibt. In diesem Fall kann man entweder einen 120 Ohm oder 150 Ohm

Vorwiderstand verwenden. Um zu kontrollieren, ob der gewählter Vorwiderstand nicht zu klein oder zu groß ist, kann man mit Hilfe der Umformung der Formel 1 nach I (Vorwärtsstrom) berechnen (siehe Formel 3). Gemäß den Ergebnissen aus den Formeln 4 und 5 schafft der Vorwiderstand mit 120 Ohm einen Vorwärtsstrom von 11,3 mA und der mit 150 Ohm 9 mA. Mit 9 mA kann es vorkommen, dass die Stromstärke nicht ausreicht, um die Leuchtdiode zum Leuchten zu bringen. Aus diesem Grund sollte man den 120 Ohm Vorwiderstand verwenden. [22] Beim Versuchsaufbau kam ein 220 Ohm Vorwiderstand zum Einsatz. Dabei leuchtete die Leuchtdiode mit einem Vorwärtsstrom von 6,1 mA auf.

$$R = \frac{3{,}3 - 1{,}95}{0{,}01} = 135\ Ohm \tag{2}$$

$$I = \frac{U}{R} \tag{3}$$

$$I = \frac{3{,}3 - 1{,}95}{120} = 11{,}3\ mA \tag{4}$$

$$I = \frac{3{,}3 - 1{,}95}{150} = 9\ mA \tag{5}$$

Eine mehrfarbige Leuchtdiode hat vier Kontakte. Der längere Kontakt heißt Kathode und die kürzeren Kontakte heißen Anoden. In der Abbildung 12 ist der Aufbau einer mehrfarbigen Leuchtdiode dargestellt.

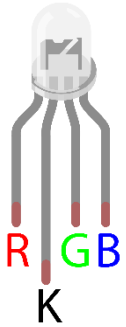


Abbildung 12: Mehrfarbige Leuchtdiode[10]

[10] Erstellt mit der Anwendung Fritzing (http://fritzing.org)

Die Kontakte R, G, B sind Anoden und die Beschriftungen weisen auf die Leuchtfarbe hin. Der Kontakt mit der Beschriftung K weist auf die Kathode hin. Nachdem die Einzelheiten geklärt sind, kann man die Schaltung aufbauen. Den Aufbau der Schaltung kann man aus der Abbildung 13 entnehmen. Um diese Leuchtdiode zum Leuchten zu bringen, kann man den Quellcode aus dem Anhang A verwenden und nach demselben Vorgehen, wie im Abschnitt 4.2.1 beschrieben, ausführen. Beim Ausführen diesen Codes leuchtet die Leuchtdiode in verschiedenen Farben auf.

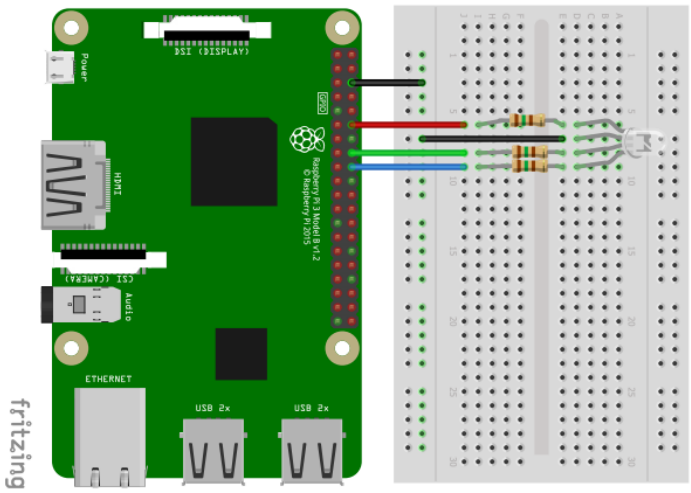


Abbildung 13: Schaltplan mit einer mehrfarbigen Leuchtdiode[11]

[11] Erstellt mit der Anwendung Fritzing (http://fritzing.org)

5 Design

Um das Ziel dieser Arbeit zu erreichen werden gewisse Anforderungen, die erfüllt sein müssen, vorausgesetzt. Diese Anforderungen werden im Verlauf dieses Kapitels anhand der Spielregeln, Spielaufbau und einzusetzenden Peripheriegeräte beschrieben.

5.1 Spielanforderungen

Das Spiel *Pong auf einer RGB-Matrix* soll eine annähernde Nachbildung des Spiels Pong aus den 70er Jahren sein. Die Bedienung des Spiels muss durch zwei Spieler mit Hilfe von zwei zusätzlichen Peripheriegeräten erfolgen. Das Spielfeld muss sich dynamisch an die Größe des Ausgabegerätes anpassen können. Dabei muss die Begrenzung des Spielfeldes visuell gekennzeichnet sein. Die Spielerschläger müssen sich proportional zum Spielfeld in der Größe ändern können. Für die Mitverfolgung des Spielstandes muss eine visuelle Darstellung auf dem Spielfeld vorhanden sein. Die Ausgabe für den Punktestand ist möglichst mittig zu platzieren. Das Spiel ist beendet, indem ein Spieler eine festgelegte Anzahl der Punkte erreicht hat. Nachdem das Spiel beendet ist, muss ein visueller Hinweis die Spieler auf das Spielende aufmerksam machen.

5.2 Spielregeln

Nach dem das Spiel gestartet ist, erscheint ein Ball an einer zufälligen vertikalen Position, in der horizontalen Mitte des Spielfeldes. Der Ball setzt sich diagonal in Richtung eines Spielers auf dem Spielfeld in Bewegung. Der Spieler muss diesen Ball mit seinem Schläger abschlagen. Nachdem, der Ball abgeschlagen ist, bewegt sich dieser in Richtung des anderen Spielers. Falls ein Spieler den Ball mit dem Schläger verfehlt, so bekommt der gegnerische Spieler einen Punkt. Verfehlt ein Spieler den Ball, so erfolgt der Abschlag vom gegnerischem Spielfeld.

5.3 Spieleinstellungen

Den Spielern muss eine Möglichkeit geboten sein das Spiel anzupassen. Das Anpassen des Spiels bezieht sich auf die Größe der RGB-Matrix und auf die Anzahl der angeschlossenen RGB-Matrixen. Zusätzlich können weitere Einstellmöglichkeiten definiert sein.

5.4 Ausgabe Methoden

Es ist zu untersuchen mit welchen Methoden man den grafischen Inhalt auf der RGB-Matrix ausgeben kann.

5.5 Audioeffekte

Das Spiel muss ein akustisches Ausgabegerät ansteuern können. Dabei müssen die Spieler während des Spiels beim Aufprallen des Balles an die Spielfeldbegrenzung, Schläger oder beim Erzielen eines Punktes durch erklingen von Tönen informiert werden.

5.6 Peripheriegeräte

5.6.1 Spielcontroller

Jeder Spieler muss über einen Spielcontroller verfügen. Die Spieler sind in der Lage mit einem Spielcontroller das Spiel zu bedienen. Die Bedienung bezieht sich auf die Steuerung der Spielschläger. Jeder Spielcontroller muss über einen Anschluss und zwei Knöpfe verfügen. Mit diesen Knöpfen kann der Spielschläger hoch und runter bewegt werden. Für ein besseres Verständnis ist in der Abbildung 14 ein Entwurf eines Spielcontrollers dargestellt.

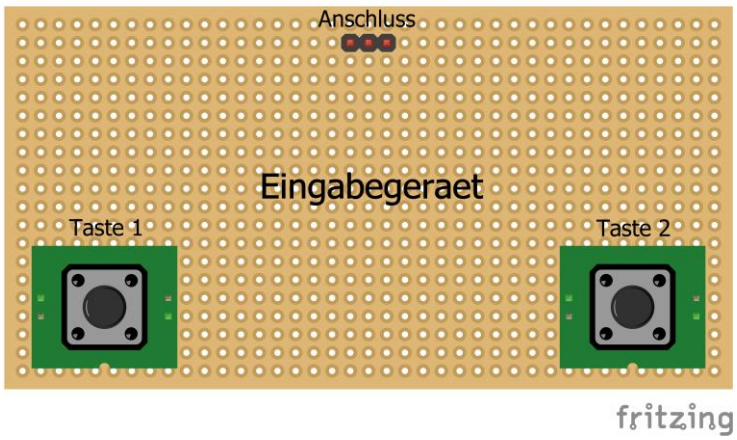


Abbildung 14: Entwurf eines Spielcontrollers[12]

[12] Erstellt mit der Anwendung Fritzing (http://fritzing.org)

5.6.2 Anschlussplatine

Für den Anschluss von RGB-Matrixen und Spielcontrollern muss eine Anschlussplatine entwickelt werden. Die Platine muss über fünf Anschlüsse verfügen. Zwei Anschlüsse sind zum Anschließen von RGB-Matrixen zuständig, zwei Anschlüsse für die beiden Spielcontroller und ein Anschluss, um die Platine mit dem Raspberry Pi zu verbinden. Eine Entwurfsansicht von der Anschlussplatine ist in der Abbildung 15 zu sehen.

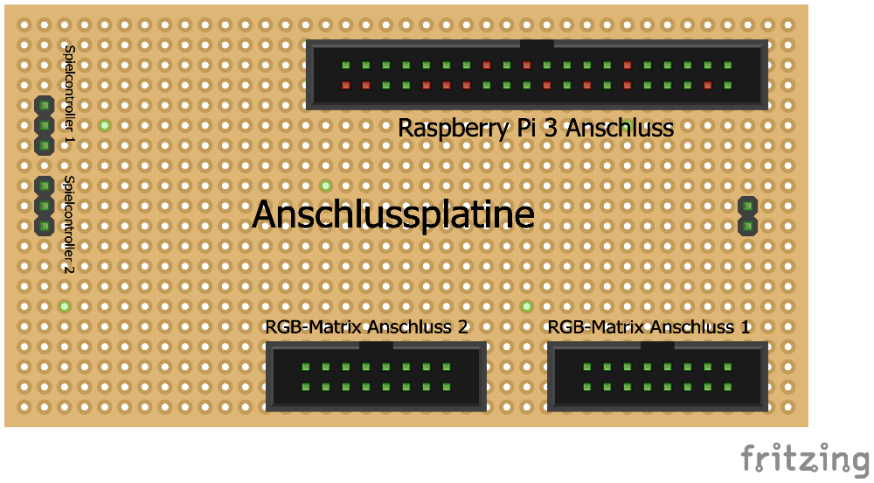


Abbildung 15: Entwurf einer Anschlussplatine[13]

[13] Quelle: Erstellt mit der Anwendung Fritzing (http://fritzing.org)

6 Implementierung

Dieses Kapitel beschreibt die Implementierung von Peripheriegeräten und des Spiels sowie das Anschließen der RGB-Matrix an den Raspberry Pi 3.

6.1 RGB Matrix anschließen

6.1.1 Verkabelung

An den Raspberry Pi 3 ist es möglich bis zu drei Matrixen parallel anzuschließen. Diese parallel angeschlossenen Matrixen lassen sich noch zusätzlich kaskadieren, indem man diese mit einem Flachbandkabel miteinander verbindet. Durch den parallelen Anschluss von drei RGB-Matrixen sind alle freiprogrammierbaren Kontakte belegt und es ist nicht mehr möglich, weitere Geräte oder Sensoren anzuschließen. Im weiteren Verlauf dieser Arbeit kommen maximal zwei parallel geschaltete Matrixen zum Einsatz.

In der Tabelle 3 sind die Signalbelegungen eingetragen. Kontakte, die in den Spalten *Pol* sind, geben die Polnummerierung der Stiftleistenanschlüsse (GPIO) an. Die in den Spalten *Matrix Eingang* sind für die Bezeichnungen der Anschlüsskontakte der Matrix. Die Signaleingänge Strobe, Clock, OE, A, B, C und GND jeder Matrix sind an dieselben Signalquellen angeschlossen. Bei jeder RGB Datenleitungen ist eine Zahl in eckigen Klammern. Diese Zahl gibt die Anzahl der Matrizen an, die parallel angeschlossen sind. [23]

Tabelle 3: Verkabelung der RGB-Matrix[14]

Matrix Eingang	Pol	Pol	Matrix Eingang
	1	2	
	3	4	
	5	6	GND
Strobe	7	8	
	9	10	
Clock	11	12	OE
[1] G1	13	14	
A	15	16	B
	17	18	C
[1] B2	19	20	
[1] G2	21	22	D*
[1] R1	23	24	[1] R2
	25	26	[1] B1
	27	28	
[2] G1	29	30	
[2] B1	31	32	[2] R1
[2] G2	33	34	
[2] R2	35	36	
	37	38	[2] B2
	39	40	

*Ist nur bei Matrixen ab einer Größe von 32x32 vorhanden.

Arten der Verkabelung

Für den Anschluss einer oder mehreren RGB-Matrixen benötigt man unterschiedliche Komponenten in Abhängigkeit nach welcher Art man die Matrix anschließt. Hat man vor die Verkabelung nur mit Verbindungskabeln durchzuführen, benötigt man gegebenenfalls ein 16 poliges Flachbandkabel und ein Steckbrett. Ein Streckbrett wird benötigt, wenn man mehr als eine RGB-Matrix anschließt. Die eine Art ist, dass man die Kontakte der RGB-Matrix und die Kontakte des Raspberry Pis mit Verbindungkabeln verbindet. Dabei kann ein 16 poliges Flachbandkabel als Verlängerung dienen und die andere Art ist, dass man die RGB-Matrix über eine Anschlussplatine (auch Erweiterungsplatine genannt) anschließt. In der Abbildung 16 ist die Art der Verkabelung mit Verbindungskabeln dargestellt.

[14] Vgl.: https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/wiring.md

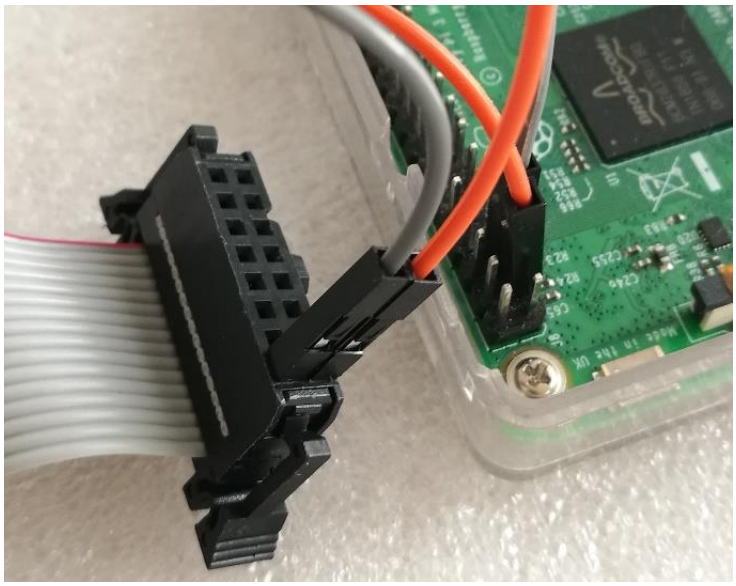


Abbildung 16: Beispiel einer Verkabelung, mit Verbindungskabeln

6.1.2 Anschlussplatine

Für die Implementierung einer Anschlussplatine, benötigt man außer einer Lochrasterplatine noch zusätzliche Komponenten. Diese Komponenten sind in der nachfolgenden Auflistung beschrieben.

- **Lochrasterplatine**

  Die Lochrasterplatine besteht aus Hartpapier mit einer Abmessung von 50 mm in der Länge und 100 mm in der Breite und einem Rastermaß von 2,54 mm. Der Bohrdurchmesser beträgt 1 mm.

- **Wannenstecker**

  Für den Anschluss an den Raspberry Pi benötigt man einen 40 poligen Wannenstecker und zwei 16 polige Wannenstecker für den Anschluss von zwei RGB-Matrixen. Alle Wannenstecker haben einen geraden Aufbau sowie einen Rastermaß von 2,54 mm.

- **Stiftleiste**

  Für den Anschluss von zwei Eingabegeräte benötigt man zwei Stück von dreipoligen Stiftleisten. Zusätzlich, um RGB-Matrixen in verschiedenen Größen anzusteuern, benötigt man zwei zweipolige Stiftleisten. Alle Stiftleisten haben einen geraden Aufbau und einen Rastermaß von 2,54 mm.

- **Kupferlitze**

  Um die Kontakte der einzelnen Komponenten miteinander zu verbinden, benötigt man Kupferlitze. Diese Kupferlitze soll einadrig, isoliert, flexibel und in verschiedenen Farben sein.

- **Kurzschlussbrücke**

  Mit einer Kurzschlussbrücke lassen sich zwei Stiftleisten mit einander verbinden (überbrücken). Die Kurzschlussbrücke hat einen Rastermaß 2,54 mm.

- **Flachbandkabel**

  Mit einem 40 poligen Flachbandkabel verbindet man die Erweiterungsanschlüsse des Raspberry Pi mit dem 40 poligen Wannenstecker auf der Anschlussplatine. Das Flachbandkabel hat einen Rastermaß von 2,54 mm

Die Platzierung und der Aufbau der Komponenten sind in der Abbildung 17 dargestellt und den Schaltplan für die Verbindung der Kontakte, kann man aus dem Anhang B: *Schaltplan einer Anschlussplatine* entnehmen. Beim Zusammenbau der Komponenten ist auf den ausreichenden Abstand zwischen den Anschlüssen *RGB-Matrix Anschluss 1* und *RGB-Matrix Anschluss 2* zu achten. Der Grund für den Abstand ist, dass es Ausführungen von den Flachbandkabeln gibt, die an den Außenseiten des Wannensteckers einrasten. Sind die Wannenstecker zu nah aneinander positioniert, kann es vorkommen, dass man nur eine Matrix anschließen kann, im ungünstigsten Fall keine.

Nach dem Zusammenbau der Anschlussplatine ist es empfehlenswert, alle Kontakte mit einem Multimeter auf die Durchgangsfunktionalität und Kurzschluss zu überprüfen, um Schäden an den Geräten zu vermeiden. Es kann unter Umständen vorkommen, dass ein Drahthaar an einen benachbarten Kontakt drankommt oder durch unpräzises Arbeiten zwei Kontakte miteinander verlötet sind.

Wurde die Überprüfung durchgeführt, so kann man mit einem 40 poligem Flachbandkabel die Anschlussplatine mit dem Raspberry Pi 3 verbinden. Der Umschalter lässt sich im Zusammenhang mit einer Kurzschlussbrücke verwenden, in dem man die benachbarten Stiftleisten überbrückt.

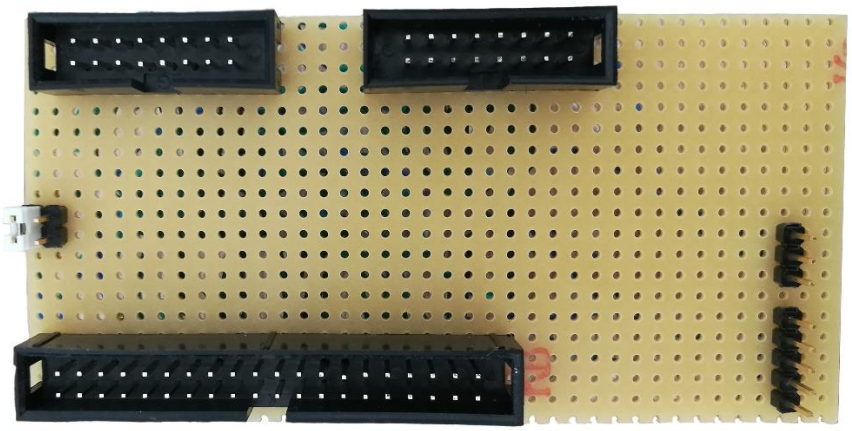


Abbildung 17: Eine fertige Anschlussplatine

6.1.3 Stromquelle

Die RGB-Matrix wird nicht über die Erweiterungsanschlüsse des Raspberry Pis mit Strom versorgt. Für die Versorgung der RGB-Matrix mit Strom benötigt man eine externe Stromquelle. Als Stromquelle eignet sich ein Stromadapter, der am Ausgang eine Spannung von 5 V und einen Strom von mindestens 2 A liefert. Für den Anschluss benötigt man passende Kabel und gegebenenfalls noch einen zusätzlichen Adapter, um den Stromadapter mit dem Stromkabel der Matrix zu verbinden. Das Stromkabel, der an die RGB-Matrix angeschlossen wird, kann über eine Spaten- oder über eine Rohrkabelschuhverbinder verfügen. Dieses Verbindungkabel kann man an einen passenden Stromadapter anschließen, der über Schraubklemmen verfügt.

Bei dieser Arbeit kommt ein Stromadapter mit der Produktbezeichnung YU-50-5 zum Einsatz. Dieser Adapter hat einen Niedervolt-Stecker als Anschluss. Um diesen mit dem Stromkabel der RGB-Matrix zu verbinden, benötigt man eine Niedervolt-Buchse (Kupplung) mit Schraubenanschluss und mit einem Durchmesser von 5,5 mm außen und 2,1 mm innen. In der Abbildung 18 ist die Kombination der einzelnen Komponente abgebildet. Das rote und das schwarze Kabel sind die Stromkabel der RGB-Matrix. Diese zwei Kabel sind mit der Niedervolt-Buchse verbunden. Das rechte Stück ist der Niedervolt-Stecker des Stromadapters.

Abbildung 18: Kombination der einzelnen Komponente

6.2 RGB Matrix ansteuern

6.2.1 Programmierschnittstelle

Das Betriebssystem Raspbian verfügt über keine native Ansteuerung der RGB-Matrix. Um diese anzusteuern, braucht man eine Programmierschnittstelle. Diese Programmierschnittstelle kann man von der Projektseite als Quellcode herunterladen. Dieses Projekt *rpi-rgb-led-matri* steht auf der Entwicklerplattform *GitHub* zu Verfügung. Mit Hilfe der Anwendung *Git* kann man eine Kopie des Projektes auf dem System erstellen. Standardmäßig ist diese Anwendung nicht auf dem System installiert. Um die Anwendung *Git* zu installieren, gibt man im Terminalfenster *sudo apt-get install git* ein. Um eine Kopie des Projektes zu erstellen, gibt man im Terminalfenster folgenden Befehl ein:

```
$ git clone https://github.com/hzeller/rpi-rgb-led-matrix
```

Nachdem der Kopiervorgang abgeschlossen ist, navigiert man in dem erstellten Verzeichnis *rpi-rgb-led-matrix*. In diesem Verzeichnis führt man den Befehl *make* aus, um den Quellcode zu kompilieren. Zusätzlich kann man den *make* Befehl mit dem Parameter *-C examples-api-use* ausführen. Durch diesen Parameter wird auch im Verzeichnis mit Beispielanwendungen der Quellcode kompiliert.

6.2.2 Audiogerät ausschalten

Bevor man mit der Ansteuerung beginnt, ist es notwendig die interne Audiokarte zu deaktivieren. Lässt man die Audiokarte aktiviert, verursacht diese eine Störung auf dem zwölften Kontaktanschluss des Raspberry Pis. Standardmäßig erfolgt die Verwendung des zwölften Kontaktes durch den LIRC-Kerneltreiber. An diesen Kontakt kann man den Infrarot-Empfänger anschließen und als Signaleingang verwenden. Aber wenn man eine Audio Karte verwendet, um über diese Audiosignale auszugeben, kommt es vor, dass sich der Audio-Kanal als PWM-Signal über den zwölften Pol ausgibt. [24] Die Audiokarte lässt sich über die Konfigurationsdatei *config.txt*, die sich in der Partition *Boot* befindet, ausschalten. Dazu fügt man an das Ende der Konfigurationsdatei eine Parameterzeile *dtparam=audio=off* ein. Anschließend muss man das System Neustarten, um die Änderungen zu aktivieren.

6.2.3 Erste Ausgabe

Um zu testen, ob man die Matrix richtig angeschlossen hat, soll man zuerst überprüfen, ob die Verkabelung richtig sitzt. Im nächsten Schritt, öffnet man ein Terminalfenster und navigiert in dem erstellten Verzeichnis *rpi-rgb-led-matrix* und anschließend in das Verzeichnis mit Beispielanwendungen *examples-api-use*. In diesem Verzeichnis befindet sich die Anwendung *demo*. Diese Anwendung demonstriert einen Teil der Möglichkeiten, wie weit die Umsetzung der Ideen auf der RGB-Matrix reichen kann. Die Anwendung lässt sich nur mit Administratorrechten ausführen. Standardmäßig sind die Einstellungen bei der Ausführung der Anwendung auf eine Größe von 32x32 Bildpunkten eingestellt.

Die Ausführung lässt sich mit Argumenten erweitern, mit denen man die Anzahl der Reihen, Anzahl der seriell angeschlossenen und die Anzahl der parallel angeschlossen RGB-Matrizen angibt. [23]

- --led-rows=<rows>
  - Mit diesem Argument gibt man an die Anzahl (8, 16, 32 oder 64) der Reihen einer Matrix an. Standardmäßig ist die Anzahl der Reihen auf 32 gesetzt.
- --led-chain=<chained>
  - Mit diesem Argument kann man die Anzahl der seriell angeschlossenen RGB-Matrizen angeben. Standardmäßig ist der Wert auf 1 gesetzt. Bei einer RGB-Matrix mit einer Größe von 64x32 muss man den Wert auf 2 setzen.
- --led-parallel=<parallel>
  - Mit diesem Argument kann man die Anzahl der parallel Angeschlossenen Matrizen angeben. Standardmäßig ist der Wert auf 1 gesetzt. Man kann, bei den Raspberry Pi Modellen A+ und B+ bis zu drei Matrizen parallel anschließen.

Beispiel

Man will die Anwendung *demo* ausführen, dabei sind zwei RGB-Matrizen parallel angeschlossen und jede Matrix hat eine Größe von 16x32 Bildpunkten. Dafür führt man die Anwendung mit Argument aus. Die Befehlszeile sieht wie folgt aus:

```
$ sudo examples-api-use /demo -D 0 --led-rows=16 --led-parallel=2
```

Nach der Ausführung der Anwendung, soll, wie in der Abbildung 19 abgebildet, ein rotierendes Viereck erscheinen.

Abbildung 19: Ausgabe auf der RGB-Matrix

6.3 Ausgabemethoden

Den grafischen Inhalt kann man auf verschiedenen Wegen auf der RGB-Matrix ausgeben. Die eine Methode ist im Abschnitt 6.2 beschrieben. Dabei erfolgt die Ansteuerung mit Hilfe von Anwendungen, die in der Programmiersprache C++ implementiert sind. Diese Anwendungen steuern direkt die RGB-Matrix an, in dem der grafische Inhalt im Code eingebettet ist. Mit einer anderen Methode, ist es möglich, eine oder mehrere RGB-Matrizen als Bildschirm zu verwenden. Dabei wird der Inhalt des Bildspeichers auf einer oder mehreren RGB-Matrizen ausgegeben. Die andere Methode ist, dass die C++ Bibliotheken sich in anderen Programmiersprachen einbinden lassen.

6.3.1 Umleitung des Bildspeichers

Um den Bildspeicher auf der RGB-Matrix auszugeben, ist es notwendig, das *rpi-fb-matrix* Projekt vom Entwickler *Adafruit Industries* herunterzuladen und dieses zu kompilieren. Das *rpi-fb-matrix* Projekt verwendet die Bibliotheken aus dem Projekt *rpi-rgb-led-matrix* vom Henner Zeller. Als Erstes, kopiert man sich den *rpi-fb-matrix* Projekt mit Hilfe der Anwendung *Git* und navigiert in das Verzeichnis *rpi-fb-matrix*. Als nächstes löscht man das *rpi-rgb-led-matrix* Verzeichnis, da die heruntergeladene Version der Bibliotheken nicht mit der Zuordnung der Verkabelung aus dem Abschnitt 6.1.1 übereinstimmt. Dafür kopiert man das komplette Verzeichnis *rpi-rgb-led-matrix* aus dem Abschnitt 6.2.1 in das *rpi-fb-matrix* Verzeichnis. Beim kompilieren des Projektes sind unter Umständen noch zusätzliche Pakete nötig, falls auf dem System diese Pakete (build-essential und libconfig++-dev) nicht vorhanden sind. [25] Die notwendigen Befehle zum Kompilieren des Projektes sehen wie folg aus:

```
$ git clone --recursive https://github.com/adafruit/rpi-fb-matrix.git
$ sudo apt-get install -y build-essential libconfig++-dev
$ cd rpi-fb-matrix
$ make
```

Nach dem Kompiliervorgang muss man noch die *config.txt* Datei anpassen. Diese Anpassung ist notwendig, um Komplikationen zu vermeiden. Dadurch erzwingt man automatisch beim Hochfahren des Systems eine feste Bildschirmauflösung und das automatische Einschalten des HDMI-Ausgangs. [25] Im Listing 2 sind die notwendigen Einträge aufgelistet. Anschließend ist das System neu zu starten.

```
1 # Standardmäßig ist der HDMI-Ausgang eingeschaltet
2 hdmi_force_hotplug=1
3 # Erzwingung einer festgelegten Auflösung
4 hdmi_group=2
5 hdmi_mode=1
6 hdmi_mode=87
7 hdmi_cvt 1024 768 60 6 0 0 0
```

Listing 2: Konfiguration der Bildschirmauflösung

Die Anwendung *rpi-fb-matrix* lässt sich mit den gleichen Argumenten, wie im Abschnitt 6.2.3 ausführen.

Beispiel: sudo ./rpi-fb-matrix --led-rows=16 --led-parallel=2

Als Ausgabe bekommt man den Inhalt des Bildspeichers auf der RGB-Matrix angezeigt. In diesem Fall ist es die grafische Arbeitsoberfläche des Raspbian, wie in der Abbildung 20 zu sehen ist. Dabei wird die tatsächliche Auflösung von 1024x768 auf die Matrixgröße von 32x32 herunter skaliert. Bei einem Sichtabstand ab einem Meter zu der RGB-Matrix lässt sich der Bildschirminhalt erkennen.

Diese Methode eignet sich um die RGB-Matrizen als Bildschirm einzusetzen.

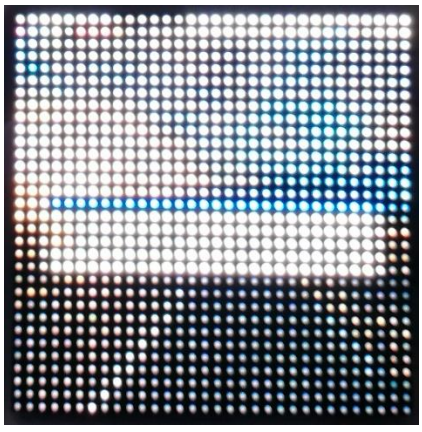


Abbildung 20: Bildspeicher auf der RGB-Matrix

6.3.2 Ansteuerung mit Python

Für diesen Abschnitt ist der Abschnitt 6.2 vorausgesetzt.

Neben der Programmiersprache C++, lassen sich die RGB-Matrizen auch mit Implementierungen in anderen Sprachen ansteuern, wie zum Beispiel mit der Sprache Python. Im Verzeichnis *rpi-rgb-led-matrix/bindings/python* sind einige Beispiele enthalten, wie man die RGB-Matrix mit Python ansteuert. Bevor man die Beispiele ausführt, ist es notwendig, zusätzliche Bibliotheken und Pakete für Python zu installieren. Als erstes, installiert man die Bibliothek *python3-dev* für die Erstellung von Paketen für Python und das Paket *pillow* für die Bearbeitung vom grafischem Inhalt mit Python. Im nächsten Schritt sind die Pakete aus dem Quellcode zu erstellen und anschließend sind diese zu installieren. Die notwendigen Befehle sehen wie folg aus [23]:

```
$ sudo apt-get update && sudo apt-get install python3-dev python3-pillow
$ make build-python PYTHON=$(which python3)
$ sudo make install-python PYTHON=$(which python3)
```

6.3.3 Geschwindigkeit

Mit einer passenden Implementierung lässt sich ermitteln, wie viele Bildpunkte pro Sekunde man mit einem Raspberry Pi 3 ansteuern kann. Damit lassen sich auch die Ausführungszeiten der Implementierungen in unterschiedlichen Programmiersprachen messen.

Das Ergebnis nach zehn Durchgängen hat ergeben, dass eine Implementierung mit Python im Schnitt 2823,44 Millisekunden benötig für einen Durchgang und 0,38 Megapixel pro Sekunde ansteuern kann.

Auch mit zehn Durchgängen, aber mit einer Implementierung in der Programmiersprache C++ hat ein besseres Ergebnis ergeben. Die durchschnittliche Ausführungszeit der Durchgänge beträgt 339 Millisekunden und mit einer Ansteuerung von 3,1 Megapixel pro Sekunde.

Diese Ergebnisse haben keine Auswirkung auf die Geschwindigkeit der Implementierung des Spiels, da die Ausgabe auf höchstens zwei RGB-Matrixen mit einer Größe von 2x16x32 Bildpunkten erfolgt. Demzufolge werden bei einer Ausgabefläche von 1024 Bildpunkten und bei 25 Bildern pro Sekunde höchstens 0,02 Megapixel pro Sekunde angesteuert und somit weit entfernt von 0,38 Megapixel.

Die Implementierungen der beiden Anwendungen befinden sich im Anhang D und E als Quellcode.

6.4 Spielcontroller

Für die Erstellung von zwei Spielcontrollern benötigt man zwei Lochrasterplatinen, ausreichend Kupferlitze, zwei Stück von dreipoligen Stiftleisten mit einem geraden Aufbau und vier Tasten. Die Lochrasterplatinen und die Kupferlitze haben dieselben Eckdaten, wie im Abschnitt 6.1.2 beschrieben. Alle Komponenten haben einen Rastermaß von 2,54 mm und einen Bohrdurchmesser von 1,0 mm.

Die Komponenten sind, wie in der Abbildung 21 abgebildet, zu positionieren. Anschließend sind die einzelnen Komponenten, gemäß dem Schaltplan aus Anhang C, miteinander zu verbinden. Auch hier ist es empfehlenswert, die Kontakte mit einem Voltmeter auf die Durchgangsfunktionalität oder Kurzschluss zu überprüfen. Mit passenden Kabeln lassen sich die Eingabegeräte mit der Anschlussplatine verbinden. Dafür eignen sich auch Verbindungskabel. Mit Hilfe der beiden Tasten kann der Spieler während des Spiels seinen Schläger vertikal hoch und runter bewegen, um den sich annähernden Ball abzuschlagen. Mit der linken Taste lässt sich der Schläger runter und mit der rechten Taste hoch bewegen.

Abbildung 21: Ein fertiger Spielcontroller

6.5 Externe Audiokarte

Laut Abschnitt 6.2.2, ist durch die Ansteuerung von RGB-Matrixen nicht mehr möglich, die interne Audiokarte zu verwenden. Als Abhilfe kann man eine externe Audiokarte über einen freien USB Anschluss anschließen.

Für die Umsetzung dieses Vorhabens kommt eine USB Audiokarte der Marke *LogiLink* zum Einsatz (siehe Abbildung 22). Um die benachbarten USB Anschlüsse durch die Audiokarte nicht zu verdecken, ist es vom Vorteil ein USB-Verlängerungskabel für die Verbindung der Audiokarte mit dem Raspberry Pi zu verwenden. Die Karte verfügt über zwei 3,5 mm Buchsenanschlüsse. Ein Anschluss ist für ein Mikrofon (rot) und der andere für den Anschluss von Lautsprechern oder einem Kopfhörer (grün) vorgesehen.

Abbildung 22: USB-Audiokarte

Bevor man die Audiokarte für die Tonausgabe verwendet, muss diese als Standard Audioausgabegerät im System definiert sein. Dafür konfiguriert man die *asound.conf* Datei. Diese befindet sich im Verzeichnis *etc*. Mit einem Texteditor, zum Beispiel mit *nano*, kann man die *asound.conf* Datei öffnen und den Inhalt aus dem Listing 3 eintragen. [26]

```
1 pcm.!default{ # Name
2    type hw      # Kernel PCM
3    card 1       # Nummer oder Name der Karte
4    device 0     # Gerätenummer
5    }
```

Listing 3: Eintrag asound.conf

6.5.1 Lautsprecher

Als Lautsprecher kommt ein Mini-Canon Bluetooth-Lautsprecher der Firma *EasyAcc* zum Einsatz. Dieser Lautsprecher wird über den Audioausgang der Audiokarte und den Audioeingang des Lautsprechers mit Hilfe eines Kabels verbunden.

Abbildung 23: Lautsprecher

6.6 Spiel-Implementierung

In diesem Abschnitt ist die Implementierung des Spiels *Pong auf einer RGB-Matrix* beschrieben. Die Implementierung erfolgt in der Programmiersprache Python in der Version 3. Als Entwicklungsumgebung kommt die schon im Kapitel 4 erwähnte *Python IDLE 3* zum Einsatz. In folgenden Abschnitten ist die grundlegende Implementierung des Spiels beschrieben. Der vollständige Quellcode befindet sich im Anhang F.

6.6.1 Python Pakete und Module

Für die Implementierung des Spiels ist es notwendig, zusätzliche Pakete und Module zu importieren und zu installieren.

Paket rgbmatrix

Mit dem Paket *rgbmatrix* und mit beinhaltenden Modulen *RGBMatrix, RGBMatrix-Options* und *graphics* kann man die RGB-Matrix ansteuern, konfigurieren und grafische Inhalte erstellen und anschließend auf einer oder mehreren RGB-Matrix ausgeben.

Modul argparse

Für die Definition und Verwendung von Argumenten ist das Modul *argparse* zuständig. Damit ist es möglich, eine Python Anwendung mit diversen Argumenten auszuführen.

Modul time

Modul *time*, bietet diverse Funktionen in Bezug auf die Zeit.

Modul random

Mit Modul *random* kann man mit integrierten Funktionen Pseudozufallszahlen erzeugen.

Modul RPi.GPIO

Der *RPi.GPIO* Modul ermöglicht aus einer Python Anwendung heraus die Konfiguration und den Zugriff auf die GPIOs des Raspberry Pis.

Modul threading

Mit dem Modul *threading* kann man einzelne Prozesse parallelisieren.

Paket PyGame

Das Paket *PyGame* beinhaltet eine diverse Sammlung von Modulen für die Implementierung von Spielen. Mit diesen Modulen lassen sich Grafische- und Audioinhalte verarbeiten sowie eine Kommunikation mit Eingabegeräten herstellen. Um PyGame Paket zu

installieren, öffnet man ein Terminalfenster und installiert zusätzliche Pakete auf dem System. Die zusätzlichen Pakete sind für den Kompilierprozess des PyGame Paketes notwendig. [27] Der notwendige Ausführungsbefehl für die Installation der Pakete sieht wie folgt aus:

```
$ sudo apt-get install mercurial python3-dev python3-numpy libav-tools libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev libsdl1.2-dev libportmidi-dev libswscale-dev libavformat-dev libavcodec-dev
```

Als nächstes lädt man mit der Anwendung *Mercurial* den Quellcode von der PyGame Projektseite herunter. Nachdem das Herunterladen abgeschlossen ist, navigiert man in das Verzeichnis *pygame* und stoßt den Kompilierprozess an. Nach dem der Quellcode kompiliert ist, installiert man den PyGame Paket. [27] Die Befehle für das Herunterladen, Kompilieren und Installieren des Paketes sehen wie folgt aus:

```
$ hg clone https://bitbucket.org/pygame/pygame
$ cd pygame
$ python3 setup.py build
$ sudo python3 setup.py install
```

6.6.2 Argumente

Die Entgegennahme der Argumente lässt sich mit der Funktion *ArgumentParser()*, des Moduls *argparse* realisieren und mit der Funktion *add_argument()* kann man die Argumente definieren und deklarieren. Für das Spiel sind sieben Argumente notwendig. Mit den Argumenten, wie in Listing 4 dargestellt ist, lassen sich diese definieren und initialisieren.

Notwendige Argumente

- Anzahl der Reihen einer RGB-Matrix
- Anzahl der seriell angeschlossenen RGB-Matrixen.
- Anzahl der parallel angeschlossenen RGB-Matrixen.
- Helligkeit der Bildpunkte
- Pulsweitenmodulation
- Die Spiel Geschwindigkeit
- Beim welchem Punktestand ist das Spiel zu Ende

```
parser = ArgumentParser()
parser.add_argument("-r", " --led-rows ", dest="ledrows",
                                    help="Anzahl der Reihen einer RGB-Matrix 16
                                    oder 32", type=int, default=16)
args = parser.parse_args()
options.rows = args.ledrows
```

Listing 4: Implementierung der Argumente

6.6.3 Spiel Einstellungen

Die RGB-Matrix wird standardmäßig mit vordefinierten Einstellungen angesteuert. Diese lassen sich manuell oder mit Hilfe von Argumenten ändern. Dafür steht die Funktion *options* im Modul *RGB-MatrixOptions* zu Verfügung. Mit dieser Funktion lassen sich die einzelnen Attribute, wie die Anzahl der Reihen, Anzahl der parallelen und seriellen Kaskadierungen, Helligkeit und die Pulsweitenmodulation, steuern. Eine Ausgewählte Konfigurationsübersicht ist im Listing 5 dargestellt.

```
options = RGBMatrixOptions()
options.rows = args.ledrows
options.chain_length = args.seriell
options.parallel = args.parallel
options.pwm_bits = 11
options.brightness = args.brightness
options.pwm_lsb_nanoseconds = args.pwmlsb
```

Listing 5: Einstellungen der RGB-Matrix

6.6.4 Initialisierung der Matrix und der Einstellungen

Um die Wirksamkeit der geänderten Einstellungen zu erreichen, ist es notwendig, diese zu initialisieren. Das Initialisieren geschieht mit der Funktion *RGBMatrix()*. Der grafische Inhalt wird nach dem Doppelpufferungsverfahren auf der RGB-Matrix dargestellt. Dadurch verringert man das vorkommende Flimmern, bei der Ausgabe vom grafischen Inhalten. Die Initialisierung der Matrix und der Einstellungen sind im Listing 6 dargestellt.

```
# Definition der RGB-Matrix
matrix = RGBMatrix(options = options)
double_buffer = matrix.CreateFrameCanvas()
```

Listing 6: Initialisierung der Matrix und Einstellungen

6.6.5 Positionierung der Objekte

Bevor man mit der Implementierung der grafischen Objekte beginnt, ist es empfehlenswert zu erfahren, wie die Positionierung der Objekte auf der RGB-Matrix erfolgt. Die Positionierung erfolgt in einem Koordinatensystem. Der Koordinatenursprung ist in der oberen linken Ecke. Der Koordinatenursprung hat die Koordinaten (X|Y)=(0|0). Die Länge der X-Achse berechnet man, in dem die Anzahl der Bildpunkte einer RGB-Matrix minus eins subtrahiert und die Differenz mit der Anzahl der seriell angeschlossenen Matrizen multipliziert. Die Höhe auf der Y-Achse berechnet man, indem man die Anzahl der Reihen einer RGB-Matrix minus eins subtrahiert und die Differenz mit der Anzahl der parallel angeschlossenen Matrixen multipliziert. [23] In der Abbildung 24, ist eine Skizze eines Koordinatensystems, in der die relevanten Komponente skizziert sind.

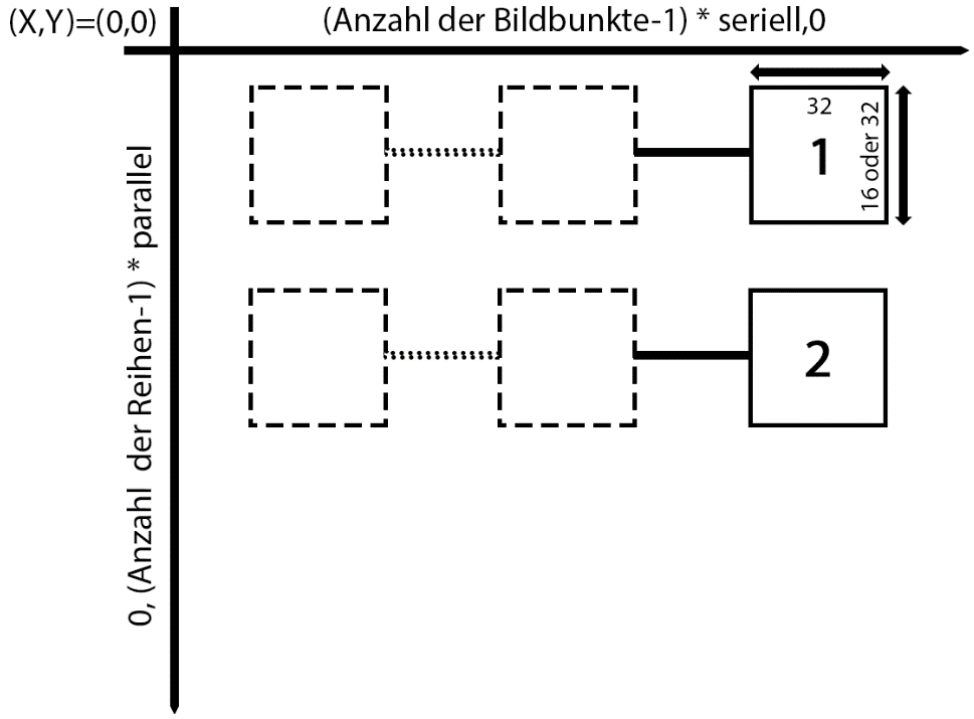


Abbildung 24: Positionierung im Koordinatensystem[15]

6.6.6 Variablen

An unterschiedlichen Stellen im Quellcode, zum Beispiel in den Funktionen, ist es notwendig, mit Werten zu arbeiten, die global definiert sind. Damit beseitigt man das wiederholte Berechnen des gleichen Wertes. Folgende Variablen sind für die Implementierung notwendig:

[15] Eigene Darstellung, angelehnt an https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/wiring.md

Vertikale- und Horizontale Anzahl der Bildpunkte

Es sind zwei Integer Variablen zu definieren und zu initialisieren. Diese liefern je einen Wert für die Horizontale- und die Vertikaleanzahl der Bildpunkte. Als Ausgangswerte kann man die Werte aus den Argumenten oder aus den *options* Werten verwenden.

Startpunkt der zwei Schläger auf der Y-Achse

Für die Ermittlung der Ausgangspunkte der beiden Schläger sind zwei Variablen vom Typ Integer zu definieren und zu initialisieren. Diese liefern den mittigen Punkt des Spielfeldes auf der Y-Achse.

Ballposition auf X- und Y-Achse

Für die mittige Ausgangsposition des Balles sind zwei Variablen vom Typ Integer zu definieren und zu initialisieren. Diese Variablen liefern die Werte der Positionen auf der X- und Y-Achse. Der Ausgangswert der Y-Achse soll per Zufallszahl (siehe Listing 7) definiert sein und der Wert auf der X-Achse ist die mittige Position auf der horizontalen Linie.

```
1 def randomnum(yAxis):
2     ypos=0
3     for i in range(yAxis):
4         ypos= random.randrange(2, yAxis-5, 1)
5     return ypos
```

Listing 7: Quellcode für eine Zufallszahl

Definition der Ballbewegung

Für die Bewegung des Balles sind zwei Variablen vom Typ Integer zu definieren und zu initialisieren. Die eine Variable gibt den Wert, für die Position auf der X-Achse und die andere Variable den Wert auf der Y-Achse. Dabei hat eine Variable den Wert +1 und die andere den Wert -1. Oder man generiert diese Werte per Zufallsgenerator.

Punktezähler

Für den Punktestand sind zwei Variablen vom Typ Integer zu definieren und zu initialisieren. Jede Variable hat einen Ausgangswert von 0.

Definition der Farben

Die vorherige Definition der einzelnen Farben erleichtert bei der späten Implementierung die Zuweisung der Farben an unterschiedliche Grafikelemente. Die Definition geschieht mit Hilfe des Moduls *graphics* und der Funktion *Color(R, G ,B)*. Die Funktion *Color()*

nimmt drei Integer Werte entgegen. Jeder Wert steht für eine Farbe (Rot, Grün, Blau) die jeweils im Bereich von 0 bis 255 sind. Damit lassen sich drei Grundfarben erstellen. Ein Beispiel für die Definition der drei Grundfarben ist im Listing 8 zusehen.

```
1 # Definition der Farben
2 red = graphics.Color(255, 0, 0)
3 green = graphics.Color(0, 255, 0)
4 blue = graphics.Color(0, 0, 255)
```

Listing 8: Definition der Farben

Definition und Auswahl der Schriftarten

Im Ordner *rpi-rgb-led-matri/fonts* befinden sich unterschiedliche monochrome Schriftarten. Diese Schriften eignen sich hervorragend für die Verwendung im Zusammenhang mit der RGB-Matrix. Bevor man eine Zeichenkette auf der RGB-Matrix ausgibt sind die Schriftarten, wie im Listing 9 dargestellt zu laden.

```
1 font = graphics.Font()
2 font.LoadFont("fonts/5x7.bdf")
```

Listing 9: Definition und Laden der Schriftarten

6.6.7 Spielfeld

Für die Definition der visuellen Spielfeldgröße erstellt man eine Funktion, die zwei integer Argumente entgegennimmt. Das eine Argument nimmt die horizontale Länge und das zweite Argument die vertikale Länge des Spielfeldes entgegen. Mit der Funktion *DrawLine(Speicher, x1, y1, x2, y2, Farbe)* aus dem Modul *graphics* kann man eine Linie auf der RGB-Matrix darstellen. Im Listing 10 ist der Quellcode der Funktion *matchfield()* dargestellt.

```
1 def matchfield(horizontal, vertical):
2     # Linie oben
3     graphics.DrawLine(double_buffer, 0, 0, horizontal, 0, blue)
4     # Linie unten
5     graphics.DrawLine(double_buffer, 0, vertical, horizontal, vertical, blue)
```

Listing 10: Quellcode der Funktion matchfield()

6.6.8 Spielball

Für die Definition eines Balles erstellt man eine Funktion, die zwei integer Argumente entgegennimmt. Das eine Argument stellt die Position auf der X-Achse und das zweite Argument die Position auf der Y-Achse dar. Für die Argumente sind, die im Abschnitt

6.6.6 beschriebenen globalen Variablen, zu verwenden. Der Ball ist durch einen Bildpunkt dargestellt. Dieser Bildpunkt lässt sich mit der Funktion *SetPixel(x, y, Farbe)* aus dem Paket *rgbmatrix* erstellen. Die Funktion *gameball()* ist im Listing 11 dargestellt.

```
1 def gameball(xAxis, yAxis):
2     double_buffer.SetPixel(xAxis, yAxis, 255, 0, 0)
```

Listing 11: Quellcodeausschnitt der Funktion gameball()

6.6.9 Ballbewegung

Die Implementierung der Ballbewegung erfolgt in dem man die Werte der Variablen für die Ballbewegung aus dem Abschnitt 6.6.6 zu den Variablen für die Ballposition addiert. Das Addieren kann in der Funktion *gameball()* erfolgen, die beim Durchlaufen einer Schleife aufgerufen wird, zum Beispiel in der Funktion für den Bildaufbau (siehe Listing 16). Im Listing 12 ist ein Implementierungsbeispiel für die Bewegung dargestellt.

```
1 ballposx=ballposx+ballmovementx
2 ballposy=ballposy+ballmovementy
```

Listing 12: Quellcodeausschnitt für die Ballbewegung

6.6.10 Spielschläger

Für die Definition der Schläger erstellt man je eine Funktionen pro Spieler. Für den linken Schläger nimmt die Funktion zwei Argumente und die Funktion für den rechten Schläger nimmt drei Argumente entgegen. Die Funktion mit zwei Argumenten bekommt den Wert der vertikalen Länge des Spielfeldes und einen Wert für die Startposition des Schlägers auf der Y-Achse. Der Wert für die Startposition ist im Abschnitt 6.6.6 beschrieben. Die Funktion für den zweiten Schläger nimmt zusätzlich ein drittes Argument mit dem Wert der horizontalen Länge des Spielfeldes entgegen. Bei einer Spielfeldgröße von 16 mal 32 Bildpunkten hat jeder Schläger eine Länge von zwei Bildpunkten. Die Länge verdoppelt sich proportional zu der Höhe des Spielfeldes.

6.6.11 Steuerung der Spielschläger

Für die Steuerung der Schläger aktiviert man vier interne Pullup-Widerstände der GPIOs. Die Aktivierung der Widerstände ist im Kapitel 4.2.1 beschrieben. Zusätzlich ist je Funktion pro Schläger zu implementieren. Jede Funktion enthält die Anweisung für die Aufwärts- und für die Abwärtsbewegung. Die Bewegung entsteht durch Betätigen der Tasten am Spielcontroller. Dadurch wird die Startposition der Schläger inkrementiert beziehungsweise dekrementiert. Die Entprellzeit darf bei dieser Implementierung nicht zu lang

oder zu kurz sein, damit das System ausreichend Zeit hat, um das empfange Signal bei Betätigen einer Taste zu verarbeiten. Ist die Entprellzeit zu lang gesetzt, kann es vorkommen, dass die Bewegung der Schläger teilweise oder komplett aussetzt. Ist die Entprellzeit zu kurz gesetzt, bewegt sich der Schläger zu schnell. Die optimale Entprellzeit für diese Funktion liegt bei 100 ms. Unter anderem ist auf die Spielfeldbegrenzung zu achten, damit sich die Schläger nicht über den Spielfeldrand bewegen. Ein Beispiel für die Implementierung dieser Funktion mit der Erkennung der oberen und unteren Spielfeldbegrenzung ist im Listing 13 dargestellt.

```
1  def gamecontroller1():
2      global paddle1
3
4      bordercollision=int((verticalled+1)/8)
5
6      while True:
7          input_state01 = GPIO.input(5)
8          input_state02 = GPIO.input(3)
9
10         if input_state01 == False:
11             if paddle1==verticalled-bordercollision:
12                 paddle1=verticalled-bordercollision
13             else:
14                 paddle1=paddle1+1
15
16         if input_state02 == False:
17             if paddle1==1:
18                 paddle1=1
19             else:
20                 paddle1=paddle1-1
21         time.sleep(0.1)
```

Listing 13: Implementierung der Funktion gamecontroller1()

6.6.12 Kollisionserkennung

Für die Kollisionserkennung erstellt man eine Funktion, die das Zusammenstoßen des Balles mit der Spielfeldbegrenzung, den Schlägern und das Verlassen des Spielfeldes erkennt. Das Erkennen der Kollision erreicht man, indem die Positionswerte auf der X und der Y-Achse der einzelnen Objekte miteinander vergleicht.

Obere Begrenzungslinie

Die Kollision mit der oberen Begrenzungslinie erkennt man, indem man überprüft, ob die Position des Balles auf der Y-Achse den Wert 1 hat.

Untere Begrenzungslinie

Die Kollision mit der unteren Begrenzungslinie erkennt man, indem man überprüft, ob die Position des Balles auf der Y-Achse den Wert von Anzahl der vertikalen Bildpunkten minus eins hat.

Linke Seite

Das Verlassen des Spielfeldes auf der linken Seite erkennt man, indem man überprüft, ob die Position des Balles auf der X-Achse den Wert kleiner 0 hat.

Rechte Seite

Das Verlassen des Spielfeldes auf der rechten Seite erkennt man, indem man überprüft, ob die Position des Balles auf der X-Achse größer ist als die Anzahl der horizontalen Bildpunkte.

Schläger

Die Kollision mit einem Schläger erkennt man, indem die Position des Balles auf der X- und Y-Achse der Position des Schlägers entspricht.

Auf- und Abprallwinkel

Bei einer erfolgreichen Erkennung eine Kollision ist den Ball abzuprallen. Das Abprallen erreicht man, indem ein Wert auf der X- oder Y-Achse, der für die Bewegung des Balles zuständig ist, invertiert. Dabei ist zu beachten, dass der Abprallwinkel dem Aufprallwinkel gleicht. Für ein besseres Verständnis ist ein Ablauf der Kollision in der Abbildung 25 dargestellt.

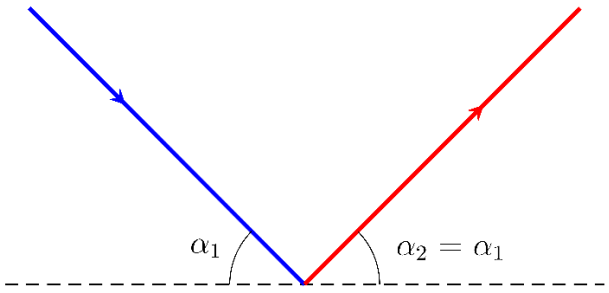


Abbildung 25: Auf- und Abrallwinkel

6.6.13 Punkt

Nachdem der Ball das Spielfeld auf der linken oder rechten Seite verlassen hat, ist der Wert einer der Variablen für den Punktestand zu inkrementieren. Die entsprechenden Variablen sind im Abschnitt 6.6.6 beschrieben.

6.6.14 Spielstand

Für die Definition einer Anzeige für den Spielstand während des Spiels ist eine Funktion zu implementieren, die vier Argumente entgegennimmt. Die ersten zwei Argumente nehmen die Anzahl der erreichten Punkte der beiden Spieler entgegen. Das dritte und vierte Argument geben die Größe des Spielfeldes an. Für die Darstellung einer Zeichenkette mit dem Punktestand verwendet man die Funktion *DrawText()* aus dem Modul *graphics*. Bei der Verwendung der Funktion *DrawText()*, ist auf die Verschiebung des Zeichenkette nach rechts zu achten. Die Verschiebung geschieht, indem eine Zehnerstelle oder eine Hunderterstelle auf der linken Seite der Anzeige dazukommt. Für ein besseres Verständnis, ist im Listing 14 ein Ausschnitt der Funktion, die den Punktestand anzeigt, dargestellt.

```
1 def showpoints(points1, points2, xAxis, yAxis):
2     pointsresult=str(points1)+":"+str(points2)
3     if xAxis == 31 and points1<=9:
4         xpos= xAxis -22
5     graphics.DrawText(double_buffer, font, xpos, 8, blue, pointsresult)
```

Listing 14: Quellcodeausschnitt der Funktion showpoints()

6.6.15 Audioeffekte

Die Audioeffekte lassen sich mit Hilfe von Audiodateien erstellen, indem bei einem bestimmten Ereignis eine bestimmte Audiodatei abgespielt wird. Das Abspielen der Audiodateien kann man mit dem Modul *mixer* aus das Paket PyGame realisieren. Als erstes ist den PyGame Paket zu initialisieren. Im nächsten Schritt lädt man die Audiodatei und anschließend spielt diese ab. Im Listing 15 ist ein Beispiel dargestellt, wie man eine Audiodatei abspielt. In dieser Arbeit kommen 8 Bit Klänge zum Einsatz. Die 8 Bit Klänge, kann man von der *opengameart.org* Internetseite herunterladen.

```
1 pygame.init()
2
3 def playsoundeffects():
4     pygame.mixer.music.load('ping_pong_8bit_plop.ogg')
5     pygame.mixer.music.play(0)
```

Listing 15: Quelcodeausschnitt zum Abspielen der Klänge

6.6.16 Spielende

Das Ende des Spiels definiert man, indem man eine Funktion implementiert, die überprüft, ob ein Spieler den vordefinierten Punktestand erreicht hat. Diese Funktion nimmt drei Argumente entgegen. Als Vergleichswert für die Punkte der einzelnen Spieler kann

man den Wert aus dem Abschnitt 6.6.6 verwenden. Zusätzlich erstellt man, wie im Abschnitt 6.6.14 beschrieben, mit der Funktion *DrawText()* einen Hinweis auf das Spielende und die Ausgabe des Spielstandes. Diese Anzeige bleibt für eine gewisse Zeit eingeblendet. Anschließend startet das Spiel von vorne.

6.6.17 Ausgangsposition

Nach dem Spielende sind alle Objekte und Werte auf die Ausgangsposition zu setzen. Die Ausgangspositionen definiert man mit einer zusätzlichen Funktion. Diese Funktion wird im Anschluss der Anzeige für das Spielende aufgerufen. Die Werte für die Ballposition, Punktestand und die Position der Schläger sind aus dem Abschnitt 6.6.6 zu entnehmen.

6.6.18 Bildaufbau

Mit der Funktion *SwapOnVSync()* aus dem Modul *RGBMatrix* lassen sich die einzelnen grafischen Objekte aus dem Bildspeicher zusammenfügen und auf der RGB-Matrix ausgeben. Dafür erstellt man eine Funktion mit einer Endlosschleife. Beim jedem Durchlauf der Schleife ist die grafische Darstellung auf der RGB-Matrix zu bereinigen. Das Bereinigen erreicht man mit der Funktion *Clear()* aus dem Modul *RGBMatrix*. Als nächstes sind die Anweisungen für den Aufruf der Funktionen zu erstellen, die für Erstellung der grafischen Inhalte im Bildspeicher zuständig sind. Nach dem die Anweisungen erstellt sind, fasst man diese zu einer gemeinsamen Bildsequenz mit der Funktion *SwapOnVSync()* zusammen. Für die Festlegung einer Bildwechselfrequenz nimmt man die Funktion *sleep()* aus dem Modul *time*. Damit lässt sich die Anzahl der anzuzeigenden Bilder pro Sekunde bestimmen. Eine Bildwechselfrequenz von 25 Hz entspricht der Darstellung von 25 Bildern pro Sekunde. Diese Frequenz erreicht man, indem Zeitwert der *sleep()* Funktion 40ms beträgt. Für ein besseres Verständnis, ist ein Ausschnitt der Endlosschleife mit dem Aufruf der *SwapOnVSync()* Funktion im Listing 16 abgebildet.

```
1 while True:
2     matrix.Clear()
3     matchfield (horizontalleds, verticalled)
4     showpoints(pointsplayer1, pointsplayer2, horizontalleds, verticalled)
5     gamepaddle1 (verticalled, paddle1)
6     gamepaddle2 (horizontalleds, verticalled, paddle2)
7     collision()
8     gameball(ballposx, ballposy)
```

```
9     double_buffer = matrix.SwapOnVSync(double_buffer)
10    time.sleep(0.04)
```

Listing 16: Quellcodeausschnitt der Funktion buildframe()

6.6.19 Parallele Prozesse

Das Spiel benötigt vier Threads. Die einzelnen Threads sind für die Aufrufe der Funktionen, die für den Aufbau des Bildes, Steuerung der Schläger und Ausgabe der Klänge zuständig sind. Mit den Threads lassen sich Prozesse parallel ausführen und unabhängig voneinander steuern. Der Grund für die Parallelisierung der Prozesse ist, dass die einzelnen Funktionen eine längere oder kürzere Ausführungszeit, beziehungsweise Reaktionszeit benötigen. Ruft man alle Funktionen hintereinander auf, ohne die Threads einzusetzen, verläuft die Abarbeitung der Funktionen linear. Erst wenn ein Prozess beendet ist, startet ein neuer. Die Threads kann man mit den Funktionen aus dem Module *threading* erstellen. Die Erstellung der Threads ist im Listing 17 dargestellt.

```
1  def funkthread():
2      threads=[]
3      threads.append(threading.Thread(target=buildframe, args=()))
4      threads.append(threading.Thread(target=gamecontroller1, args=()))
5      threads.append(threading.Thread(target=gamecontroller2, args=()))
6      threads.append(threading.Thread(target=playsoundeffects, args=()))
7
8      for t in threads:
9          t.daemon =True
10         t.start()
11     for t in threads:
12         t.join()
```

Listing 17: Quellcode der Funktion funkthread()

6.6.20 Spiel ausführen

Bei einer erfolgreichen Umsetzung der Anforderungen aus dem Kapitel 5 durch Implementierungsbeschreibung aus diesem Kapitel kann man das Spiel über ein Terminalfenster ausführen. Das Spiel wird mit Administratorrechten mit Hilfe von vordefinierten Argumenten ausgeführt.

Beispiel

Das Spielfeld bilden zwei parallel angeschlossene RGB-Matrixen. Jede Matrix besteht aus 16 Reihen und die Helligkeit der einzelnen Leuchtpunkte beträgt 80 Prozent.

Das Spiel ist zu Ende, wenn ein Spieler einen Punktestand von zehn erreicht hat. Der Befehl zum Ausführen lautet:

```
$ sudo python3 spielpong.py --led-rows=16 --led-parallel =2 --led-brightness =80
  --end=10
```

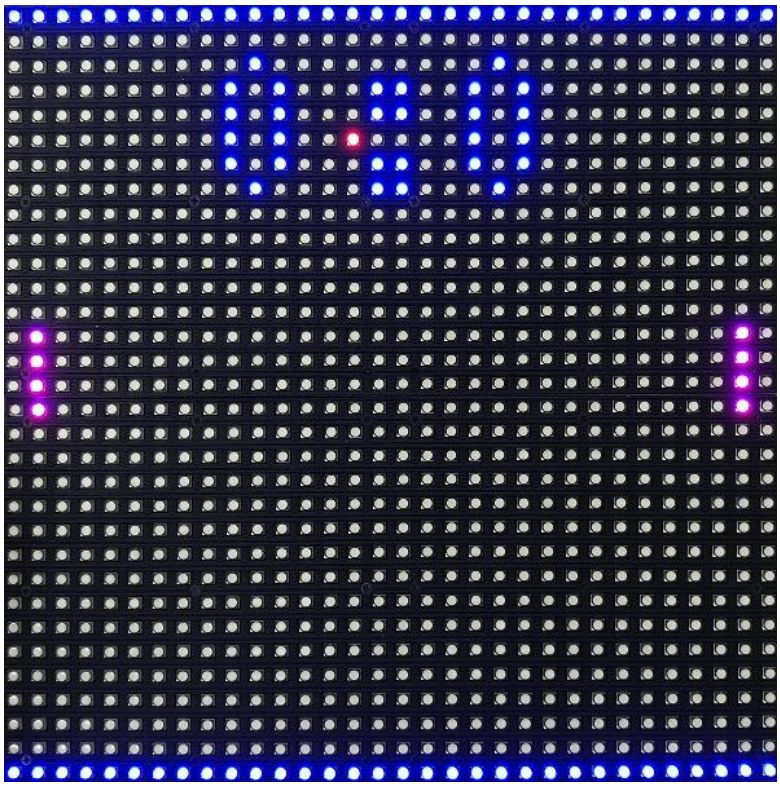


Abbildung 26: Spiel Pong auf einer RGB-Matrix

7 Fazit

Die Zielsetzung der vorliegenden Arbeit war ein Spiel vollständig auf einem Einplatinencomputer zu entwickeln und zu implementieren. Dabei erfolgte die Ausgabe auf mehreren RGB-Matrixen. Neben der Ansteuerung RGB-Matrixen wurden unterschiedliche Methoden für die Ansteuerung der Matrixen untersucht. Zusätzlich wurden zwei Prototypen der Geräte entwickelt, die für den Anschluss von RGB-Matrixen und die Steuerung des Spiels zuständig sind. Die Implementierung des Spiels hat gezeigt, dass auch mit einer Programmiersprache, die eine langsamere Ausführungszeit als C++ hat, ein gutes Ergebnis erzielt werden kann, solange man sich bewusst ist, welche Anforderungen gesetzt sind und mit welchen Mitteln diese umgesetzt werden. Eine ausgiebige Untersuchung ist vorausgesetzt, um ein gewünschtes Ergebnis zu erzielen.

Eine der größten Hürden während der Ausarbeitung war sich für eine passende Lösung aus zahlreichen Beschreibungen in Blogs und Internetforen für die Ansteuerung der RGB-Matrixen zu entscheiden. Die meisten Lösungen sind teilweise unvollständig, veraltet oder befanden sich noch in der Entwicklungsphase.

Ein weiteres Hindernis war die Verkabelung der RGB-Matrixen. Bei einer einfachen Verkabelung verliert man schnell den Überblick und ein unkomplizierter Auf- und Abbau der Komponenten ermöglichte keinen sicheren Transport. Auch die Beauftragung für die Erstellung von Anschlussplatinen aus den Entwürfen von Henner Zeller kamen aus zeitlichen Gründen nicht in Frage. Aus diesem Grund fiel die Entscheidung auf die Entwicklung von einer eigenen Anschlussplatine.

Als Ideen für zukünftige Arbeiten wären zum Beispiel der Einsatz von RGB-Matrixen als Bildschirmersatz für Ausgabe von Videomaterial und Übertragung von Inhalten und Information mittels mobiler Geräten wie Smartphones oder Tablets, zu untersuchen.

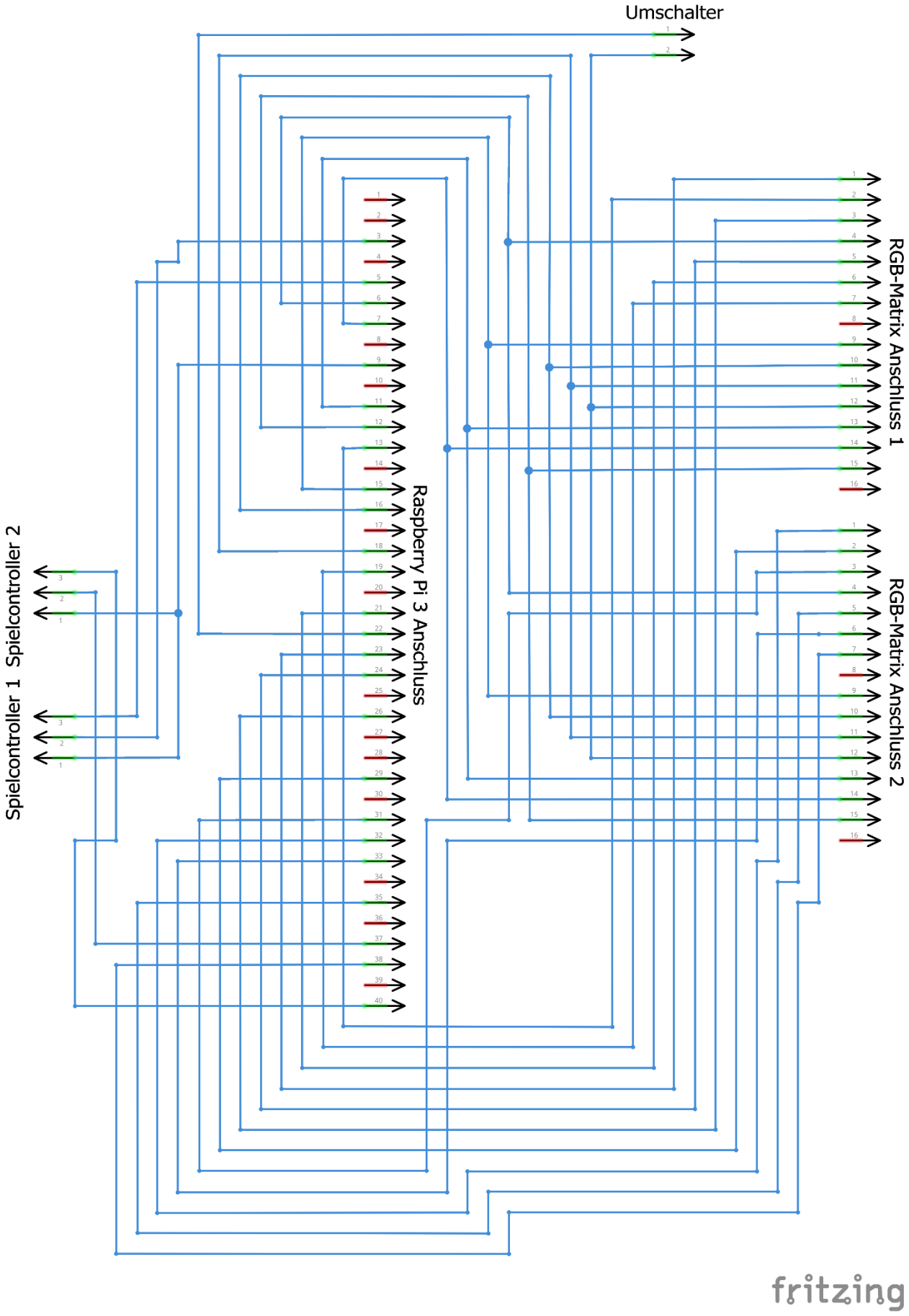
Anschließend lässt sich sagen, dass die Realisierung der gesetzten Ziele zu voller Zufriedenheit erfolgt sind und es zeigt, dass ein Raspberry Pi sich nicht nur zum Erlernen von Programmiersprachen und Grundlagen von Elektrotechnik eignet, sondern auch für die Umsetzung von unterschiedlichen Projekten.

Anhang A: Ansteuerung einer RGB-LED

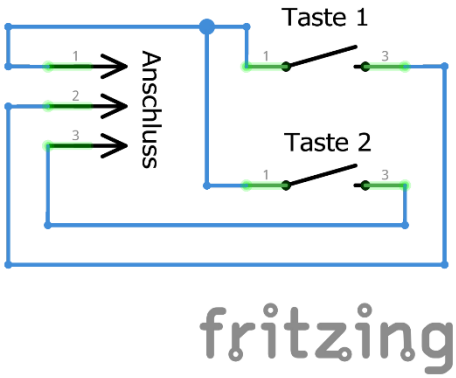
```
# Titel: rgbled.py
# Autor Dimitri Gubermann

import RPi.GPIO as GPIO
import time
import threading

# Initialisierung der GPIO Anschlüsse
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT) # rot
GPIO.setup(16, GPIO.OUT) # grün
GPIO.setup(18, GPIO.OUT) # blau

# Funktion für die Farbe rot
def redcolor():
        while True:
                # Farbe rot leuchtet
                GPIO.output(12,GPIO.HIGH)
                time.sleep(0.4)
                # Farbe rot leuchtet nicht
                GPIO.output(12,GPIO.LOW)
                time.sleep(0.8)

# Funktion für die Farbe grün
def greencolor():
        while True:
                GPIO.output(16,GPIO.HIGH)
                time.sleep(2.0)
                GPIO.output(16,GPIO.LOW)
                time.sleep(1.0)

# Funktion für die Farbe blau
def bluecolor():
        while True:
                GPIO.output(18,GPIO.HIGH)
                time.sleep(1.0)
                GPIO.output(18,GPIO.LOW)
                time.sleep(1.5)

def shining_colors():
    threads = []
    threads.append(threading.Thread(target=redcolor, args=()))
    threads.append(threading.Thread(target=greencolor, args=()))
    threads.append(threading.Thread(target=bluecolor, args=()))
    for t in threads:
        t.daemon = True
        t.start()
    for t in threads:
        t.join()
```

```
50
51 def main():
52     try:
53         shining_colors()
54     except KeyboardInterrupt:
55         pass
56     finally:
57         GPIO.cleanup()
58
59 if __name__ == '__main__':
60     main()
```

Anhang B: Schaltplan einer Anschlussplatine



Anhang C: Schaltplan für den Spielcontroller



Anhang D: Geschwindigkeitsmessung mit Python

```
1  # Quelle: https://github.com/hzeller/rpi-rgb-led-matrix
2  # Autor: Henner Zeller
3
4  from rgbmatrix import RGBMatrix
5  import time
6
7  canvas = RGBMatrix(32, 1, 1)
8
9  w = canvas.width
10 h = canvas.height
11 loops = 1024
12
13 start = time.time();
14
15 for i in range(0, loops):
16     col = i % 256;
17     for y in range(0, h):
18         for x in range(0, w):
19             canvas.SetPixel(x, y, col, 0, 0);
20
21 duration = time.time() - start
22
23 pixels = w * h * loops;
24 pixels_per_sec = pixels / duration;
25 print("%d pixels, %dms; %.1f Megapixels/s; %.1fHz frame update rate"
26       % (pixels, 1000.0 * duration, pixels_per_sec / 1e6,
27          loops / duration))
```

Anhang E: Geschwindigkeitsmessung mit C++

```
1  // Quelle: https://github.com/hzeller/rpi-rgb-led-matrix
2  // Autor: Henner Zeller
3  // g++ -Iinclude -o speedtest speedtest.cc -Llib -lrgbmatrix -lpthread
4
5  #include "led-matrix.h"
6  #include <unistd.h>
7  #include <stdio.h>
8  #include <sys/time.h>
9
10 using rgb_matrix::RGBMatrix;
11
12 typedef int64_t tmillis_t;
13 static tmillis_t GetTimeInMillis() {
14    struct timeval tp;
15    gettimeofday(&tp, NULL);
16    return tp.tv_sec * 1000 + tp.tv_usec / 1000;
17 }
18
19 int main(int argc, char *argv[]) {
20   RGBMatrix *canvas = rgb_matrix::CreateMatrixFromFlags(&argc, &argv, NULL);
21   if (canvas == NULL)
22     return 1;
23
24   const int w = canvas->width();
25   const int h = canvas->height();
26   const int loops = 1024;
27
28   const tmillis_t start = GetTimeInMillis();
29
30   for (int i = 0; i < loops; ++i) {
31     const uint8_t col = i & 0xff;
32     for (int y = 0; y < h; ++y) {
33       for (int x = 0; x < w; ++x) {
34         canvas->SetPixel(x, y, col, 0, 0);
35       }
36     }
37   }
38   const tmillis_t duration = GetTimeInMillis() - start;
39   const int pixels = w * h * loops;
40   const float pixels_per_sec = 1000.0 * pixels / duration;
41   printf("%d pixels, %lldms; %.1f Megapixels/s; %.1fHz frame update rate\n",
42          pixels, duration, pixels_per_sec / 1e6,
43          1000.0 * loops / duration);
44
45   canvas->Clear();
```

```
46   delete canvas;
47
48   return 0;
49 }
```

Anhang F: Pong auf einer RGB-Matrix

```
1  # Datei: spielpong.py
2  # Titel: Pong auf einer RGB-Matrix
3  # Autor: Dimitri Gubermann
4
5  from rgbmatrix import RGBMatrix, RGBMatrixOptions
6  from rgbmatrix import graphics
7  from argparse import ArgumentParser
8  import time
9  import os,sys
10 import random
11 import RPi.GPIO as GPIO
12 import threading
13 import pygame
14
15 # Paket PyGame initialisieren
16 pygame.init()
17
18 # Initialisieren der GPIO Anschlüsse
19 GPIO.setmode(GPIO.BOARD)
20 GPIO.setup(5, GPIO.IN, pull_up_down = GPIO.PUD_UP)
21 GPIO.setup(3, GPIO.IN, pull_up_down = GPIO.PUD_UP)
22 GPIO.setup(40, GPIO.IN, pull_up_down = GPIO.PUD_UP)
23 GPIO.setup(37, GPIO.IN, pull_up_down = GPIO.PUD_UP)
24
25 # Definition und Deklaration der Argumente
26 # Anzahl der Reihen 16, 32 ist eine natürliche Zahl (integer)
27 # Anzahl der seriell angeschlossenen RGB-Matrizen ist eine natürliche Zahl (integer)
   # Anzahl der parallel angeschlossenen RGB-Matrizen (1-2) ist iene natürliche Zahl
28 (integer)
29 # Helligkeit der RGB-Matrix 0-100, ist eine natürliche Zahl (integer)
30 # PWM ist eine natürliche Zahl (integer)
31 # Geschwindigkeit ist eine Fließkommazahl(float)
32 # Spielende ist eine natürliche Zahl (integer)
33
34 parser = ArgumentParser()
35
36 parser.add_argument("-r", "--led-rows", dest="ledrows",
                help="Anzahl der Reihen einer RGB-Matrix 16,32 oder 64", type=int,
37 default=16)
38 parser.add_argument("-s", "--led-chain", dest="seriell",
                help="Anzahl der seriell angeschlossenen RGB-Matrizen", type=int,
39 default=1)
40 parser.add_argument("-p", "--led-parallel", dest="parallel",
                help="Anzahl der parallel angeschlossenen RGB-Matrizen 1-2", type=int,
41 default=1)
42 parser.add_argument("-b", "--led-brightness", dest="brightness",
```

```
43                  help="Helligkeit der RGB-Matrix 0-100", type=int, default=100)
44  parser.add_argument("-pw", "--led-pwm-lsb-nanoseconds", dest="pwmlsb",
45                  help="PWM standard 130", type=int, default=130)
46  parser.add_argument("-sp", "--speed", dest="speed",
47                  help="Geschwindigkeit des Spiels", type=float, default=0.04)
48  parser.add_argument("-e", "--end", dest="end",
49                  help="Spielen bis x Punkte, Standard ist 10", type=int, default=10)
50
51  args = parser.parse_args()
52
53  # Hier ist die Konfiguration der RGB-Matrix definiert und deklariert.
54  options = RGBMatrixOptions()
55  options.rows = args.ledrows
56  options.chain_length = args.seriell
57  options.parallel = args.parallel
58  options.hardware_mapping = 'regular'
59  options.pwm_bits = 11
60  options.brightness = args.brightness
61  # Wenn es zu stark flimmert, hoch oder runtersetzen
62  options.pwm_lsb_nanoseconds = args.pwmlsb
63
64  #Definition der RGB-Matrix
65  matrix = RGBMatrix(options = options)
66  double_buffer = matrix.CreateFrameCanvas()
67
68  # Definition der Farben
69  red = graphics.Color(255, 0, 0)
70  green = graphics.Color(0, 255, 0)
71  blue = graphics.Color(0, 0, 255)
72
73  # Variablen für die vertikale und horizontale Anzahl der LEDs
74  horizontalleds = options.chain_length*32-1
75  verticalled = options.parallel*options.rows-1
76
77  # Berechnung der Position auf der y-Achse der beiden Schläger
78  def paddleposition(yAxis):
79
80      ypos=0
81      ypos = ((yAxis-1-(16*options.parallel)/8))/2+1
82
83      return ypos
84
85  # Variable für die Position und Bewegung der Schläger
86  paddle1=int(paddleposition(verticalled))
87  paddle2=int(paddleposition(verticalled))
88
```

```
89  # Wert für die Zufallsposition auf der vertikalen Achse
90  def randomnum(yAxis):
91
92      ypos=0
93
94      for i in range(yAxis):
95          ypos= random.randrange(2, yAxis-5, 1)
96
97      return ypos
98
99  # Zufälliger Anstoß nach links oder rechts
100 def kickoffleftright():
101
102     leftright=0
103     tempnum=0
104
105     for i in range(10):
106         tempnum=random.randrange(-10, 10, 1)
107
108     if(tempnum>-1):
109         leftright=+1
110
111     elif(tempnum<1):
112         leftright=-1
113
114     return leftright
115
116 # Ballposition
117 ballposx=int(options.chain_length*32/2)
118 ballposy=randomnum(verticalled)
119
120 # Ballbewegung
121 ballmovementx = kickoffleftright()
122 ballmovementy = kickoffleftright()
123
124 # Variablen für den Punktestand
125 pointsplayer1 = 0
126 pointsplayer2 = 0
127
128 # Variable für Ton
129 soundnum=0
130
131 # Variable für die Zeit zum zählen der Zeit beim Erzielen von Punkt
132 timecounter=0
133
134 # Laden der Schriftarten
```

```
135 font = graphics.Font()
136 font.LoadFont("fonts/5x7.bdf")
137
138 # Funktion für das Inkrementieren der Punkte
139 def gamescore(gamescore):
140
141     gamescore=gamescore+1
142     return gamescore
143
144 # Funktion für die Anzeigen der Punkte
145 def showpoints(points1,points2,xAxis,yAxis):
146
147     pointsresult=str(points1)+":"+str(points2)
148     xpos=0
149     ypos=0
150
151     if xAxis == 31 and points1<=9:
152         xpos=xAxis-22
153
154     elif xAxis == 31 and points1>=10:
155         xpos=xAxis-27
156
157     elif xAxis == 31*options.chain_length+1 and points1<=9:
158         xpos=xAxis-38
159
160     elif xAxis == 31*options.chain_length+1 and points1>=10:
161         xpos=xAxis-43
162
163     graphics.DrawText(double_buffer, font, xpos, 8, blue, pointsresult)
164
165 # Funktion für den Aufbau des Spielfeldes
166 def matchfield(horizontal, vertical):
167     #DrawLine(Startpunkt(x,y), Endpunkt(x1,y2))
168
169     # Linie oben
170     graphics.DrawLine(double_buffer, 0, 0, horizontal, 0, blue)
171
172     # Linie unten
173     graphics.DrawLine(double_buffer, 0, vertical, horizontal, vertical, blue)
174
175 # Diese Funktion berechnet die Länge des linken Schlägers
176 # und positioniert diese auf dem Spielfeld
177 def gamepaddle1(yAxis,ypos):
178
179     paddlelength=int((yAxis+1))/8
180
```

```
181     double_buffer.SetPixel(1, ypos, 255, 0, 255)
182 
183     for i in range(1, int(paddlelength)):
184         double_buffer.SetPixel(1, ypos+i, 255, 0, 255)
185 
186 # Diese Funktion berechnet die Länge des rechten Schlägers
187 # und positioniert diese auf dem Spielfeld
188 def gamepaddle2(xAxis,yAxis,ypos):
189 
190     paddlelength=int((yAxis+1))/8
191 
192     double_buffer.SetPixel(xAxis-1, ypos,   255, 0, 255)
193 
194     for i in range(1, int(paddlelength)):
195         double_buffer.SetPixel(xAxis-1, ypos+i, 255, 0, 255)
196 
197 # Diese Funktion ist für die vertikale Bewegung des linken
198 # Schlägers zuständig
199 def gamecontroller1():
200     global paddle1
201 
202     bordercollision=int((verticalled+1)/8)
203 
204     while True:
205         input_state01 = GPIO.input(5)
206         input_state02 = GPIO.input(3)
207 
208         if input_state01 == False:
209             if paddle1==verticalled-bordercollision:
210                 paddle1=verticalled-bordercollision
211 
212             else:
213                 paddle1=paddle1+1
214 
215         if input_state02 == False:
216             if paddle1==1:
217                 paddle1=1
218 
219             else:
220                 paddle1=paddle1-1
221 
222         time.sleep(0.1)
223 
224 # Diese Funktion ist für die vertikale Bewegung des rechten
225 # Schlägers zuständig
226 def gamecontroller2():
```

```
227     global paddle2
228
229     bordercollision=int((verticalled+1)/8)
230
231     while True:
232         input_state01 = GPIO.input(40)
233         input_state02 = GPIO.input(37)
234         if input_state01 == False:
235             if paddle2==verticalled-bordercollision:
236                 paddle2=verticalled-bordercollision
237
238             else:
239                 paddle2=paddle2+1
240
241         if input_state02 == False:
242
243             if paddle2==1:
244                 paddle2=1
245
246             else:
247                 paddle2=paddle2-1
248
249         time.sleep(0.1)
250
251 # Diese Funktion ist für die Positionierung und Bewegung des Balles
252 # auf dem Spielfeld zuständig
253 def gameball(xAxis,yAxis,timex):
254     global timecounter
255     global ballposx
256     global ballposy
257
258     if timex==int(3/args.speed):
259         # Ballposition
260         ballposx=int(options.chain_length*32/2)
261         ballposy=randomnum(verticalled)
262         timecounter=0
263
264     elif timex==0:
265         ballposx=ballposx+ballmovementx
266         ballposy=ballposy+ballmovementy
267         double_buffer.SetPixel(xAxis, yAxis, 255, 0, 0)
268
269 # Diese Funktion ist für die Kollisionserkennung zuständig
270 def collision():
271     global paddle1
272     global paddle2
```

```
273     global ballmovementx
274     global ballmovementy
275     global ballposx
276     global ballposy
277     global pointsplayer1
278     global pointsplayer2
279     global soundnum
280     global timecounter
281
282     soundnum=0
283     above=1
284     below=options.rows*options.parallel-2
285     leftside=0
286     rightside=32*options.chain_length-1
287     paddlesize=(verticalled+1)/8+1
288
289     ##Kollision mit der rechten Seite##
290     if ballposx==rightside:
291        pointsplayer1=gamescore(pointsplayer1)
292        soundnum=2
293
294     elif ballposx > 32*options.chain_length-1:
295        timecounter=timecounter+1
296        ballmovementx = +1
297        ballmovementy = +1
298
299     ##Kollision mit der oberen Linie##
300     if ballposy==above:
301        ballmovementy = +1
302        soundnum=1
303
304     ##Kollision mit der linken Seite##
305     if ballposx==leftside:
306        pointsplayer2=gamescore(pointsplayer2)
307        soundnum=2
308
309     elif ballposx < 0:
310        timecounter=timecounter+1
311        ballmovementx = -1
312        ballmovementy = -1
313
314     ##Kollision mit der unteren Seite##
315     if ballposy==below:
316        ballmovementy = -1
317        soundnum=1
318
```

```
319     ##Kollision mit dem Schläger1##
320     if ((paddle1-1 <= ballposy <= paddle1+paddlesize) and (ballposx==2)):
321         ballmovementx = +1
322         soundnum=1
323 
324     ##Kollision mit dem Schläger2##
        if (paddle2-1 <= ballposy <= paddle2+paddlesize) and (ballposx==horizontalleds-
325 2):
326         ballmovementx = -1
327         soundnum=1
328 
329 # Diese Funktion ist für die Ausgabe von Punkten am Ende des Spiels zuständig
330 def gameover(points1,points2,xAxis):
331     global soundnum
332 
333     xpos=0
334     ypos=14
335     soundnum=0
336     pointsresult=str(points1)+":"+str(points2)
337 
338     if xAxis == 31 and points1<=9:
339         xpos=xAxis-22
340 
341     elif xAxis == 31 and points1>=10:
342         xpos=xAxis-27
343 
344     elif xAxis == 31*options.chain_length+1 and points1<=9:
345         xpos=xAxis-38
346 
347     elif xAxis == 31*options.chain_length+1 and points1>=10:
348         xpos=xAxis-43
349 
350     if(xAxis>31):
351         graphics.DrawText(double_buffer, font, 1, 7, blue, "Spielendende")
352 
353     elif(verticalled>15):
354         graphics.DrawText(double_buffer, font, 1, 10, blue, "Spiel-")
355         graphics.DrawText(double_buffer, font, 1, 17, blue, "ende")
356         ypos=25
357 
358     elif(verticalled<16):
359         graphics.DrawText(double_buffer, font, 1, 7, blue, "Ende")
360 
361     graphics.DrawText(double_buffer, font, xpos, ypos, blue, pointsresult)
362 
363 # Diese Funktion setzt alle Komponente auf die
```

```
364 # Ausgangspositionen zurück.
365 def startingposition():
366
367     global ballposx
368     global ballposy
369     global ballmovementx
370     global ballmovementy
371     global pointsplayer1
372     global pointsplayer2
373     global paddle1
374     global paddle2
375
376     # Ballposition zurücksetzen
377     ballposx=int(options.chain_length*32/2)
378     ballposy=randomnum(verticalled)
379     ballmovementx = kickoffleftright()
380     ballmovementy = kickoffleftright()
381
382     # Punktestand auf 0 setzen
383     pointsplayer1 = 0
384     pointsplayer2 = 0
385
386     # Spielschläger auf Ausgangsposition setzen
387     paddle1=int(paddleposition(verticalled))
388     paddle2=int(paddleposition(verticalled))
389
390 # Funktion zum Abspielen von Audioklängen
391 def playsoundeffects():
392     global soundnum
393
394     while True:
395         if( soundnum==1 and timecounter<1):
396             pygame.mixer.music.load('ping_pong_8bit_plop.ogg')
397             pygame.mixer.music.play(0)
398
399         elif(soundnum==2 and timecounter<1):
400             pygame.mixer.music.load('ping_pong_8bit_beeep.ogg')
401             pygame.mixer.music.play(0)
402
403             soundnum=0
404         time.sleep(args.speed)
405
406 # Mit dieser Funktion werden einzelne grafische Objekte zu einem Bild
407 # zusammengefügt und auf der RGB-Matrix ausgegeben
408 def buildframe():
409
```

```
410     global double_buffer
411     seccounter=0
412 
413     while True:
414         matrix.Clear()
415         # Spielstand überprüfen und ggf Spiel beenden
416         matchfield(horizontalleds, verticalled)
417         if pointsplayer1==args.end or pointsplayer2==args.end:
418             gameover(pointsplayer1,pointsplayer2,horizontalleds)
419             seccounter=seccounter+1
420             if seccounter==int(5/args.speed):# Pause von 5 Sek
421                 seccounter=0
422                 startingposition()
423         else:
424             showpoints(pointsplayer1,pointsplayer2,horizontalleds,verticalled)
425             gamepaddle1(verticalled,paddle1)
426             gamepaddle2(horizontalleds,verticalled,paddle2)
427             collision()
428             gameball(ballposx,ballposy, timecounter)
429 
430         double_buffer = matrix.SwapOnVSync(double_buffer)
431         time.sleep(args.speed)
432 
433 # Mit dieser Funktion werden die einzelnen Prozesse parallelisiert.
434 def funkthread():
435     threads=[]
436     threads.append(threading.Thread(target=buildframe, args=()))
437     threads.append(threading.Thread(target=gamecontroller1, args=()))
438     threads.append(threading.Thread(target=gamecontroller2, args=()))
439     threads.append(threading.Thread(target=playsoundeffects, args=()))
440 
441     for t in threads:
442         t.daemon =True
443         t.start()
444     for t in threads:
445         t.join()
446 
447 def main():
448     try:
449         funkthread()
450     except KeyboardInterrupt:
451         pass
452     finally:
453         GPIO.cleanup()
454 
455 # Aufruf der Hauptfunktion
```

```
456 if __name__ == "__main__":
457     if(args.parallel<3):
458         main()
459     else:
460         print("Es werden maximal 2 parallel angeschlossene RGB-Matrixen unterstützt.")
```

Literaturverzeichnis

[1] **Lerch, Reinhard.** Elektrische Messtechnik : Analoge, digitale und computergestützte Verfahren. 7. Erlangen : Springer Vieweg, 2016, vgl. S. 606

[2] **t3n** [Online abgerufen am 12.10.2017] URL: http://t3n.de/news/computer-raspberry-pi-c64-806761/

[3] **Michael Kofler,** Charly Kühnast, Christoph Scherbeck. *Raspberry Pi : das umfassende Handbuch.* 2., aktualisierte und erw. Aufl. s.l. : Rheinwerk Verlag GmbH, 2015, vgl. S. 15

[4] **Raspberry Pi Foundation** [Online abgerufen am 12.10.2017] URL: www.raspberrypi.org/about

[5] **Raspberry Pi Foundation** [Online abgerufen am 13. 10 2017] URL: https://www.raspberrypi.org/magpi/raspberry-pi-sales/

[6] **elektronik-kompendium .de** [Online abgerufen am 15.10.2017] URL: www.elektronik-kompendium.de/sites/com/1904221.htm

[7] **Raspberry Pi Foundation** [Online abgerufen am 13. 10 2017] URL: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[8] **Raspberry Pi Foundation** [Online abgerufen am 15.10.2017] URL: https://www.raspberrypi.org/products

[9] **pinout.xyz** [Online abgerufen am 13. 10 2017] URL: https://de.pinout.xyz/

[10] **Michael Kofler,** Charly Kühnast, Christoph Scherbeck. *Raspberry Pi : das umfassende Handbuch.* 2., aktualisierte und erw. Aufl. s.l. : Rheinwerk Verlag GmbH, 2015, vgl. S. 357-358

[11] **Raspberry Pi Foundation** [Online abgerufen am 13.10.2017] URL: https://www.raspberrypi.org/documentation/installation/sd-cards.md

[12] **raspberry.tips** [Online abgerufen am 14.10.2017] URL: https://raspberry.tips/raspberrypi-einsteiger/die-richtige-sd-karte-fuer-den-raspberry-pi/

[13] **elektronikpraxis.vogel.de** [Online abgerufen am 14.10.2017] URL: https://www.elektronikpraxis.vogel.de/40-betriebssysteme-fuer-den-raspberry-pi-a-488934/

[14] **Adafruit Industries** [Online abgerufen am 22.11.2017] URL: https://www.adafruit.com

[15] **electronicfreakblog.wordpress.com** [Online abgerufen am 22.10.2017] URL: https://electronicfreakblog.wordpress.com/2015/11/29/hack-einer-1632-rgb-led-matrix-teil-i-die-vorueberlegungen/comment-page-1/

[16] **Raspberry Pi Foundation** [Online abgerufen am 20.10.2017] URL: https://www.raspberrypi.org/documentation/

[17] **datenreise.de** [Online abgerufen am 15.10.2017]
URL: https://www.datenreise.de/raspberry-pi-remote-desktop-xrdp-installieren/

[18] **Michael Kofler,** Charly Kühnast, Christoph Scherbeck. *Raspberry Pi : das umfassende Handbuch.* 2., aktualisierte und erw. Aufl. s.l. : Rheinwerk Verlag GmbH, 2015, vgl. S. 358

[19] **elektronik-kompendium.de** [Online abgerufen am 26.10.2017]
URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2102181.htm

[20] **SourceForge** [Online abgerufen am 26.10.2017]
URL: https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

[21] **oreilly.com** [Online abgerufen am 26.10.2017]
URL: http://razzpisampler.oreilly.com/ch07.html

[22] **elektronik-kompendium.de** [Online abgerufen am 26.10.2017]
URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2006051.htm

[23] **github.com** [Online abgerufen am 23.10.2017]
URL: https://github.com/hzeller/rpi-rgb-led-matrix

[24] **Michael Kofler,** Charly Kühnast, Christoph Scherbeck. *Raspberry Pi : das umfassende Handbuch.* 2., aktualisierte und erw. Aufl. s.l. : Rheinwerk Verlag GmbH, 2015, vgl. S. 359

[25] **Adafruit Industries** [Online abgerufen am 10.11.2017]
URL: https://learn.adafruit.com/raspberry-pi-led-matrix-display/overview

[26] **www.alsa-project.org** [Online abgerufen am 15.11.2017]
URL: https://www.alsa-project.org/main/index.php/Asoundrc

[27] **pygame.org** [Online abgerufen am 10.11.2017] URL: http://www.pygame.org