



powered by



FRANZIS

ADVENTSKALENDER PROGRAMMIEREN MIT MINECRAFT

TM



HANDBUCH

FRANZIS





Adventskalender 2018 für Raspberry Pi

Minecraft ist eines der beliebtesten Computerspiele weltweit – und das, obwohl es kein wirkliches Spielziel hat. Gerade die offene Spielwelt, in der jeder für sich Landschaften bauen und eigene Träume verwirklichen kann, fasziniert über 100 Millionen Spieler weltweit. Der Spieltitel setzt sich aus den beiden englischen Begriffen *mine* (= Bergbau, Rohstoffabbau) und *craft* (= verarbeiten) zusammen. Und genau darum geht es in Minecraft: Es gilt, Rohstoffe aus der Landschaft abzubauen und sie zu neuen Landschaften oder Objekten zu verarbeiten.

Neben den bekannten Versionen für PC und verschiedene Spielkonsolen gibt es Minecraft auch auf dem Raspberry Pi – und sogar kostenlos vorinstalliert. Die Raspberry-Pi-Version bietet einen großen Vorteil, denn sie ist über

eine eigens entwickelte Schnittstelle in der Programmiersprache Python programmierbar. Python ist ebenfalls auf dem Raspberry Pi vorinstalliert und ermöglicht unter anderem die Ansteuerung externer Hardware. So ist es auf dem Raspberry Pi möglich, mit Minecraft LEDs zum Blinken zu bringen oder mit Hardwaretasten und Sensorkontakten die Spielfigur „Steve“ in Minecraft zu steuern.

Dieser Adventskalender enthält jeden Tag ein kleines Hardwareexperiment für den Raspberry Pi, dazu das passende Python-Programm und Elemente der Minecraft-Welt. Alle Experimente funktionieren sowohl mit dem Raspberry Pi 3 als auch dem Pi 3 B+ und der aktuellen Version des Betriebssystems Raspbian (getestet ab NOOBS 2.7.0).



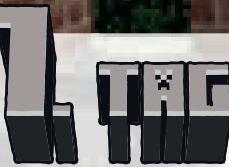
Ausschnitt aus der Minecraft-Welt des Adventskalenders.

Minecraft

Minecraft, das beliebte Weltenbauer-Spiel, ist in der aktuellen Raspbian-Version bereits vorinstalliert. In Minecraft erkundet man eine schier endlose Welt, die aus einfachen Würfeln erbaut ist. Die Blöcke bestehen aus verschiedenen Materialien und können abgebaut werden, um sie als Rohstoffe zu verarbeiten und daraus andere Dinge zu bauen. Der Spieler übernimmt die Rolle der Spielfigur Steve und steuert diese. Dabei kann man die Szene aus Steves Perspektive sehen oder aus Sicht einer Kamera von außen darauf blicken.

Wer Minecraft kennt, wird sich auch in der Pi Edition schnell zurechtfinden. Die Steuerung des Spiels und die Bewegungen in der Spielwelt laufen sehr ähnlich ab. Wer Minecraft noch nicht kennt, braucht sich nur wenige Tasten zu merken:

- Mit der Maus dreht man sich, ohne eine Maustaste zu drücken, um die eigene Achse und neigt den Blick nach oben oder unten. Das Spiel reagiert sehr schnell, man muss also aufpassen, dass man sich beim Drehen nicht überschlägt.
- Mit vier auch aus anderen Spielen bekannten Buchstabentasten bewegt man sich: mit **[W]** nach vorne, mit **[S]** nach hinten, mit **[A]** nach links und mit **[D]** nach rechts. Bei Stufen im Gelände steigt man während der Bewegung automatisch nach oben oder unten.
- Mit der **[Leertaste]** kann man in die Höhe springen. Drückt man die **[Leertaste]** zweimal kurz hintereinander, wird auf den Flugmodus umgeschaltet. In diesem Modus schwebt man und ist nicht mehr an den Boden gebunden. Im Flugmodus steigt man durch längeres Drücken der **[Leertaste]** weiter nach oben.
- Umgekehrt duckt man sich mit der linken **[Umschalt]-Taste** etwas nach unten. Im Flugmodus verringert man mit dieser Taste die Flughöhe.
- Die Taste **[E]** öffnet das Inventar, in dem jede Menge unterschiedlicher Blöcke zum Bau zur Verfügung stehen. Acht verschiedene Blöcke oder Werkzeuge sind in der Inventarliste am unteren Bildschirmrand jederzeit verfügbar. Hier wählt man mit den Tasten **[1]** bis **[8]** oder mit dem Mausrad das gewünschte Objekt aus.
- Ein Klick mit der linken Maustaste entfernt den angeklickten Block, ein Klick mit der rechten Maustaste platziert einen Block des gewählten Typs an der angeklickten Position.
- Die **[Esc]**-Taste blendet ein Menü ein, über das man das Spiel verlassen oder über ein Symbol links oben auf die Sicht eines außenstehenden Betrachters wechseln kann.
- Die **[Tab]**-Taste befreit die Maus aus dem Minecraft-Fenster, wenn man zwischendurch in ein anderes Programm wechseln möchte.



Heute im Adventskalender

- 1 Steckbrett (SYB 46)
- 1 LED grün mit Vorwiderstand
- 2 GPIO-Verbindungskabel

Raspberry Pi vorbereiten

Um den Raspberry Pi in Betrieb zu nehmen, braucht man:

- USB-Tastatur und Maus
- HDMI-Kabel für Monitor
- Netzwerkkabel oder WLAN
- MicroSD-Karte mit Betriebssystem Raspbian Stretch
- Micro-USB-Handyladegerät als Netzteil (mindestens 2000 mA, besser 2500 mA)

Das Netzteil muss als Letztes angeschlossen werden. Damit schaltet sich der Raspberry Pi automatisch ein. Es gibt keinen Ein/Aus-Schalter.

Betriebssysteminstallation in Kürze

Für alle, die ihren Raspberry Pi noch nicht mit der aktuellen Raspbian-Version betriebsbereit haben, folgt hier die Systeminstallation in zehn Schritten:

1. NOOBS (mindestens Version 2.7.0) von www.raspberrypi.org/downloads auf den PC herunterladen und Zip-Archiv auf die Festplatte entpacken.
2. Wurde die SD-Karte bereits benutzt, mit SD-Formatter im PC neu formatieren: www.sdcard.org/downloads/formatter_4. Dabei **Format Size Adjustment** einschalten (die SD-Karte muss mindestens 8 GB groß sein).
3. Alle Dateien und Unterverzeichnisse von NOOBS auf die SD-Karte kopieren.
4. SD-Karte aus dem PC nehmen, in den Raspberry Pi stecken und booten.
5. Ganz unten **Deutsch** als Installationssprache wählen. Damit wird automatisch auch die deutsche Tastatur ausgewählt.
6. Das Häkchen beim vorausgewählten Raspbian-Betriebssystem setzen und oben links auf **Install** klicken. Nach Bestätigung einer Sicherheitsabfrage, dass die Speicherkarte überschrieben wird, startet die Installation, die einige Minuten dauert. Nach abgeschlossener Installation bootet der Raspberry Pi neu.
7. Im Menü unter **Einstellungen** das Tool **Raspberry Pi Konfiguration** starten. Auf der Registerkarte **Lokalisierung** im Feld **Zeitzone festlegen** die Optionen **Europe** und **Berlin** auswählen. **Sprachumgebung** und **Tastatur** sollten automatisch auf Deutsch gesetzt sein. Sollte dies nicht der Fall sein, deutsche Lokalisierung und Tastatur auswählen.

8. Um auf einem Raspberry Pi 3 B+ WLAN zu nutzen, muss über den Button **WiFi Land festlegen** die Option **DE - Germany** ausgewählt werden. Das WLAN auf dem Raspberry Pi 3 funktioniert noch unabhängig von dieser Einstellung.
9. Auf der Registerkarte **Schnittstellen** den Schalter **SSH** auf **Aktiviert** setzen, wenn Sie vom PC über das Netzwerk Daten auf den Raspberry Pi übertragen wollen.
10. Auf **OK** klicken und den Raspberry Pi über den Menüpunkt **Shutdown** neu booten. Eine eventuell auftauchende Warnung wegen eines unsicheren Passworts kann ignoriert werden.

Num-Lock-Taste

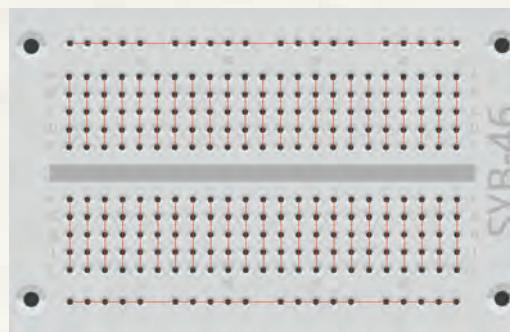
Wie bei fast allen Linux-Systemen ist auch beim Start von Raspbian der Ziffernblock standardmäßig ausgeschaltet. Drücken Sie die Num-Lock-Taste auf der Tastatur, um ihn einzuschalten.

Die wichtigsten Bauteile kurz erklärt

Steckbrett

Um elektronische Schaltungen schnell aufbauen zu können, ohne löten zu müssen, enthält der Adventskalender ein Steckbrett. Hier können elektronische Bauteile direkt in ein Lochraster eingesteckt werden.

Bei diesem Steckbrett sind alle äußeren Längsreihen über Kontakte (X und Y) miteinander verbunden. Diese Kontaktreihen werden oft als Plus- und Minuspol zur Stromversorgung der Schaltungen genutzt. In den anderen Kontaktreihen sind jeweils fünf Kontakte (A bis E und F bis J) quer miteinander verbunden, wobei in der Mitte der Platine eine Lücke ist. So können hier größere Bauelemente eingesteckt und nach außen hin verdrahtet werden.

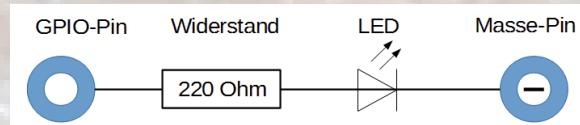


Die Verbindungen auf dem Steckbrett.

LEDs

LEDs (zu Deutsch: Leuchtdioden) leuchten, wenn Strom in Durchflussrichtung durch sie fließt. LEDs werden in Schaltungen mit einem pfeilförmigen

Dreieckssymbol dargestellt, das die Flussrichtung vom Pluspol zum Minuspol oder zur Masseleitung angibt. Eine LED lässt in Durchflussrichtung nahezu beliebig viel Strom durch und hat dabei nur einen sehr geringen Widerstand. Um den Durchflussstrom zu begrenzen und damit ein Durchbrennen der LED zu verhindern, wird üblicherweise zwischen dem verwendeten GPIO-Pin und der Anode der LED oder zwischen Kathode und Massepin ein 220-Ohm-Vorwiderstand eingebaut. Dieser Vorwiderstand schützt auch den GPIO-Ausgang des Raspberry Pi vor zu hohen Stromstärken. Bei den LEDs im Adventskalender ist der Vorwiderstand bereits eingebaut, weshalb sie direkt an die GPIO-Pins angeschlossen werden können.



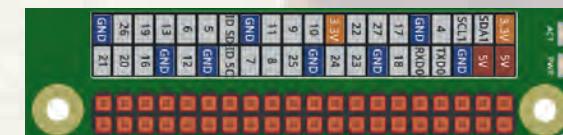
Schaltplan einer LED mit Vorwiderstand.

LED in welcher Richtung anschließen?

Die beiden Anschlussdrähte einer LED sind unterschiedlich lang. Der längere ist der Pluspol oder die Anode, der kürzere die Kathode. Einfach zu merken: Das Pluszeichen hat einen Strich mehr als das Minuszeichen, und deshalb ist auch der Draht etwas länger. Außerdem sind die meisten LEDs auf der Minusseite abgeflacht, vergleichbar mit einem Minuszeichen. Auch leicht zu merken: Kathode = kurz = Kante.

GPIO-Verbindungskabel

Die farbigen Verbindungskabel haben alle auf einer Seite einen Stecker, auf der anderen Seite eine Buchse, die auf einen GPIO-Pin des Raspberry Pi passt. Die Stecker werden in das Steckbrett gesteckt. Die programmierbaren GPIO-Pins auf dem Raspberry Pi haben Nummern, die Masse-Pins sind in der Abbildung mit GND gekennzeichnet.



Pinbelegung der GPIO-Pins.

Tips

Im Download zum Adventskalender ist diese Abbildung als Grafik **gpio.png** enthalten. Kopieren Sie diese auf den Raspberry Pi, um sie immer leicht zur Hand zu haben, oder verwenden Sie sie einfach als Desktophintergrund.

Sollten Sie die Grafik mal nicht zur Verfügung haben, liefert der Befehl **pinout** in einem Kommandozeilenfenster ein einfaches Pinbelegungsschema im Textmodus.



Vorsichtsmaßnahmen

Auf keinen Fall sollte man irgendwelche GPIO-Pins miteinander verbinden und abwarten, was passiert.

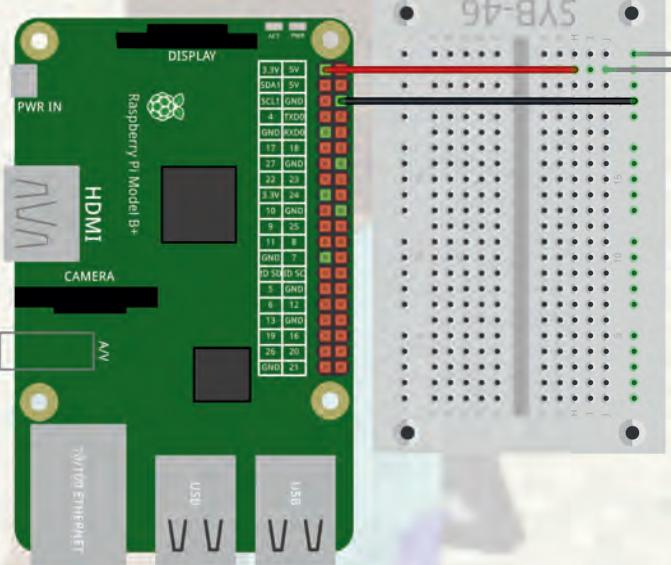
Nicht alle GPIO-Pins lassen sich frei programmieren. Ein paar sind für die Stromversorgung und andere Zwecke fest eingerichtet.

Einige GPIO-Pins sind direkt mit Anschlüssen des Prozessors verbunden, weshalb ein Kurzschluss den Raspberry Pi komplett zerstören kann. Verbindet man über einen Schalter oder eine LED zwei Pins miteinander, muss immer ein Schutzwiderstand zwischengeschaltet werden. Eine Ausnahme bilden die LEDs mit eingebautem Vorwiderstand.

Für Logiksignale muss immer der +3,3-V-Pin verwendet werden, der bis 50 mA belastet werden kann. Die GND-Pins sind Masseleitungen für Logiksignale.

Die beiden mit 5 V bezeichneten Pins in der Ecke liefern +5 V zur Stromversorgung externer Hardware. Hier kann so viel Strom entnommen werden, wie das USB-Netzteil des Raspberry Pi liefert. Diese Pins dürfen aber nicht mit einem GPIO-Eingang verbunden werden.

LED leuchtet



Die erste LED leuchtet am Raspberry Pi.

Für das erste Experiment wird kein Programm benötigt. Der Raspberry Pi dient hier nur als Stromversorgung für die LED. Das Experiment zeigt, wie LEDs angeschlossen werden. Achten Sie darauf, dass die LED richtig herum eingebaut ist. Die flache Seite ist in der Abbildung rechts.

Bei den meisten Schaltungen dient die Kontaktleiste an der einen Längsseite des Steckbretts als Massekontakt. Hier werden die Kathoden aller LEDs

eingelegt und mit einem Kabel mit einem GND-Pin auf dem Raspberry Pi verbunden.

Bauteile

- 1 Steckbrett SYB-46
- 1 LED grün mit Vorwiderstand
- 2 GPIO-Verbindungskabel

Dateien zum Download

Die im Adventskalender verwendeten Programme und die Minecraft-Welt gibt es hier zum Download: bit.ly/c-adventskalender-minecraft-18.

Das Downloadarchiv zum Adventskalender enthält dieses Handbuch als PDF in Farbe, damit Sie auf den Schaltplänen die einzelnen Leitungen besser erkennen können.

Öffnen Sie die Webseite direkt im vorinstallierten Browser auf dem Raspberry Pi und laden Sie die Zip-Datei in das Benutzerverzeichnis </home/pi> herunter.



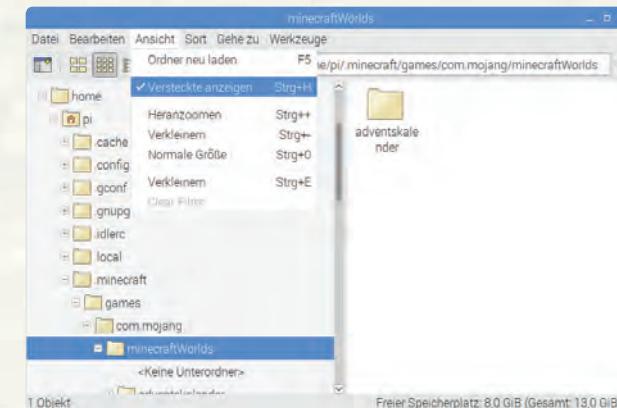
Starten Sie den Dateimanager auf dem Raspberry Pi. Er zeigt beim Start automatisch das Home-Verzeichnis an. Klicken Sie mit der rechten Maustaste auf die heruntergeladene Zip-Datei und wählen Sie im Kontextmenü [Hier entpacken](#).



Kopieren Sie mit dem Dateimanager den Ordner [adventskalender](#) der Minecraft-Welt in das Verzeichnis </home/pi/.minecraft/games/com.mojang/minecraftWorlds> auf dem Raspberry Pi.

Hinweis

Dieses Verzeichnis ist direkt nach der Installation noch nicht vorhanden. Minecraft legt die Ordnerstruktur beim ersten Start automatisch an. Danach kann die Welt in das Verzeichnis kopiert werden.



Um das Verzeichnis der Minecraft-Welten zu sehen, muss im Dateimanager-Menü unter [Ansicht](#) die Option [Versteckte anzeigen](#) eingeschaltet sein.

Für Ihre Notizen



2. TAG

Heute im Adventskalender

- 1 LED rot mit Vorwiderstand
- 2 GPIO-Verbindungskabel

LED leuchtet beim Betreten einer bestimmten Fläche

Das Experiment des 2. Tages lässt die grüne LED leuchten, wenn man mit der Figur eine Fläche betritt, die sich in der vorgegebenen Minecraft-Welt direkt vor dem ersten Gebäude links neben dem Eingang zum Weihnachtsmarkt befindet. Gesteuert wird das Ganze über ein Programm in Python.

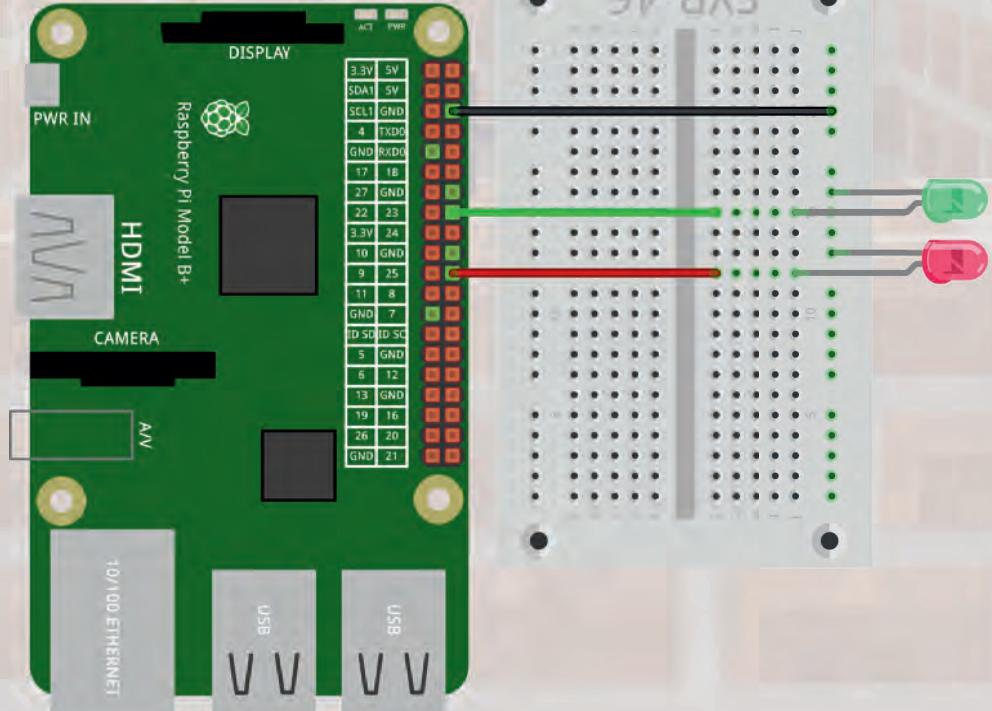


Beim Betreten der grauen Fläche vor dem Gebäude leuchtet die LED.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 3 GPIO-Verbindungskabel

Die Programmiersprache Python ist auf dem Raspberry Pi vorinstalliert, sogar in zwei verschiedenen Versionen. Python 3 ist nicht, wie der Name vermuten lässt, einfach eine neuere Version von Python 2. Die Sprachen verwenden teilweise eine andere Syntax. Programme sind also nicht eins zu eins kompatibel. Viele externe Bibliotheken sind nur für eine der beiden Versionen erhältlich. Entwickler von Python-Programmen müssen ihren Nutzern also immer mitteilen, mit welcher Version ein Programm funktioniert. Wir verwenden in diesem Adventskalender immer das moderne Python 3.



Zwei LEDs am Raspberry Pi.

Starten Sie im Menü unter *Entwicklung* das Programm Python 3. *IDLE* ist eine komplette Python-Shell und Entwicklungsumgebung. Für den Start in die Programmierung sind keine zusätzlichen Komponenten nötig.

Öffnen Sie über *File/Open* das Programm *02mc.py* aus dem Download oder gehen Sie in der Python-Shell über *File/New* in ein neues Fenster und tippen Sie das Programm ein.

Das Programm

Das Programm zeigt wichtige Techniken der Python-Programmierung, die Sie für die weiteren Programme brauchen werden. Um ein Python-Programm für Minecraft nutzen zu können, muss Minecraft bereits laufen. Verlassen Sie das Minecraft-Fenster mit der Taste **Tab**, wechseln Sie dann in die Python-Shell und starten Sie dort das Programm mit der Taste **F5**. Danach können Sie mit einem einfachen Klick in das Minecraft-Fenster ins Spiel zurückkehren.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(23, GPIO.OUT)
GPIO.setup(25, GPIO.OUT)

while True:
    p = mc.player.getTilePos()
    if (p.x==8 and p.z>=3 and p.z <=4):
        GPIO.output(23, True)
        GPIO.output(25, False)
    else:
        GPIO.output(25, True)
        GPIO.output(23, False)
```

So funktioniert das Programm

```
#!/usr/bin/python
```

Python-Programme, die über die Kommandozeile gestartet werden, müssen am Anfang immer obige Zeile enthalten. Bei Programmen, die nur über die Python-Shell gestartet werden, ist das nicht nötig. Aus Gründen der Kompatibilität sollten Sie sich aber angewöhnen, diese Zeile am Anfang jedes Python-Programms einzutragen.

```
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO
```

Ein großer Vorteil von Python ist die einfache Erweiterbarkeit um neue Funktionen aus Funktionsbibliotheken. Für nahezu jede Aufgabe gibt es bereits fertige Bibliotheken, sodass Sie viele Standardaufgaben nicht mehr selbst zu lösen brauchen. Die Bibliothek `mcpi.minecraft` wird für die Verbindung zu Minecraft benötigt. Hier sind alle Steuerungsfunktionen enthalten. Die Bibliothek `RPi.GPIO` wird für die Unterstützung der GPIO-Pins importiert.

```
mc = minecraft.Minecraft.create()
```

Diese Zeile ist in jedem Programm enthalten, das Minecraft nutzt. Sie generiert das Python-Objekt `mc` für das laufende Minecraft-Spiel. Objekte sind weit mehr als nur einfache Variablen. Sie können verschiedene Eigenschaften haben und über Methoden beeinflusst werden. Methoden werden durch einen Punkt getrennt direkt hinter dem Objektnamen angegeben. Verschiedene Methoden dieses Objekts können dann ins Spiel eingreifen oder Daten aus dem Spiel auslesen.

```
GPIO.setmode(GPIO.BCM)
```

Die Bibliothek `RPi.GPIO` unterstützt zwei Bezeichnungsverfahren für die Pins. Im Modus `BCM` werden die bekannten GPIO-Portnummern verwendet, die auch in Scratch und anderen Programmiersprachen genutzt werden. Im Modus `BOARD` entsprechen die Bezeichnungen den Pin-Nummern auf der Raspberry-Pi-Platine. Üblicherweise wird `BCM` verwendet. In den Aufbauzeichnungen sind die Pin-Nummern im Modus `BCM` abgebildet.

```
GPIO.setup(23, GPIO.OUT)
GPIO.setup(25, GPIO.OUT)
```

Die Funktion `GPIO.setup` initialisiert einen GPIO-Pin als Ausgang oder als Eingang. Der erste Parameter bezeichnet den Pin je nach gewähltem Modus (`BCM` oder `BOARD`) mit seiner GPIO- oder Pin-Nummer. Der zweite Parameter ist entweder `GPIO.OUT` für einen Ausgang oder `GPIO.IN` für einen Eingang.

```
while True:
```

Schleifen mit `while` laufen so lange, wie die dahinter angegebene Bedingung wahr ist. Da die Bedingung hier einfach `True` lautet, also immer wahr ist, läuft die Schleife endlos.

Einrückungen sind in Python wichtig

In den meisten Programmiersprachen werden Programmschleifen oder Entscheidungen eingerückt, um den Programmcode übersichtlicher zu machen. In Python dienen diese Einrückungen nicht nur der Übersichtlichkeit, sondern sind für die Programmlogik zwingend nötig. Dafür braucht man in dieser Sprache keine besonderen Zeichen, um Schleifen oder Entscheidungen zu beenden.

```
p = mc.player.getTilePos()
```

Die eingerückten Zeilen werden in jedem Schleifendurchlauf einmal ausgeführt. Diese Zeile liest die Koordinaten des Minecraft-Klotzes aus, auf dem die Spielfigur gerade steht, und schreibt sie in die Variable `p`.

Das Minecraft-Koordinatensystem

Koordinaten in Minecraft werden in Rastereinheiten in Größe der Klotze gezählt. Im Gegensatz zu vielen bekannten 3D-Programmen hat die horizontale Ebene in Minecraft x- und z-Koordinaten. Die y-Achse zeigt nach oben in die Luft.

Die Spielfigur soll die kleine graue Fläche vor der ersten Weihnachtsmarkthütte links hinter dem Eingang betreten. Diese Fläche liegt in der x-Achse bei der Koordinate 8, in der z-Achse zwischen 3 und 4. Ob die Figur dort steht, wird mit einer `if`-Abfrage geprüft:

```
if (p.x==8 and p.z>=3 and p.z <=4):
```

Innerhalb der `if`-Abfrage sind drei Bedingungen mit `and` verknüpft. Sie müssen alle gleichzeitig wahr sein, damit die Abfrage im Ganzen wahr ergibt. `p.x` liefert die x-Koordinate des in `p` gespeicherten Punkts, `p.z` liefert die z-Koordinate.

```
    GPIO.output(23, True)
    GPIO.output(25, False)
```

Die eingerückten Zeilen werden immer dann ausgeführt, wenn die `if`-Abfrage wahr liefert.

Die Funktion `GPIO.output` setzt den Status eines GPIO-Pins. Jeder Pin kann auf zwei verschiedene Zustände gesetzt werden. `True` schaltet den Pin ein, `False` schaltet ihn wieder aus. Diese Zeilen schalten die LED an GPIO-Pin 23 ein und die LED an GPIO-Pin 25 aus.

Ergibt die `if`-Abfrage nicht wahr, weil die Spielfigur außerhalb der grauen Fläche steht, wird der zweite Teil der Abfrage hinter `else` ausgeführt.

```
else:
    GPIO.output(25, True)
    GPIO.output(23, False)
```

Diese Zeilen schalten die LED an GPIO-Pin 25 ein und die LED an GPIO-Pin 23 aus.

Starten Sie das Programm mit der Taste `[F5]`. Eventuell auftauchende Warnungen wegen verwendeter GPIO-Ports können Sie ignorieren. Die rote LED leuchtet. Bewegen Sie die Spielfigur auf die graue Fläche, leuchtet die grüne LED. Verlässt die Spielfigur diese Fläche wieder, leuchtet erneut die rote LED.

Für Ihre Notizen



3. TAG

Heute im Adventskalender

- 1 LED gelb mit Vorwiderstand

Ampel am Eingang zum Weihnachtsmarkt

Das Experiment des 3. Tages stellt eine Ampel dar. Wenn Sie sich mit der Spielfigur auf die graue Steinfläche unter dem Eingangstor stellen, schaltet die Ampel auf Grün.



Am Eingang zum Weihnachtsmarkt leuchtet eine Ampel aus drei LEDs.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 4 GPIO-Verbindungskabel

Das Programm

Das Programm [03mc.py](#) schaltet die drei LEDs durch einen Ampelzyklus, wenn die Spielfigur auf dem Wartebalken auf der Straße steht.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO
```

Drei LEDs simulieren eine Ampel am Raspberry Pi.

```
import time

mc = minecraft.Minecraft.create()
rot = 0
gelb = 1
gruen = 2
Ampel = [18,23,25]

GPIO.setmode(GPIO.BCM)
GPIO.setup(Ampel[rot], GPIO.OUT, initial=True)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=False)

try:
    while True:
        p = mc.player.getTilePos()
        mat = mc.getBlock(p.x, p.y-1, p.z)
        if mat==98:
            GPIO.output(Ampel[gelb],True)
            time.sleep(0.6)
            GPIO.output(Ampel[rot],False)
            GPIO.output(Ampel[gelb],False)
            GPIO.output(Ampel[gruen],True)
```

```
time.sleep(2)
GPIO.output(Ampel[gruen],False)
GPIO.output(Ampel[gelb],True)
time.sleep(0.6)
GPIO.output(Ampel[gelb],False)
GPIO.output(Ampel[rot],True)
time.sleep(2)
```

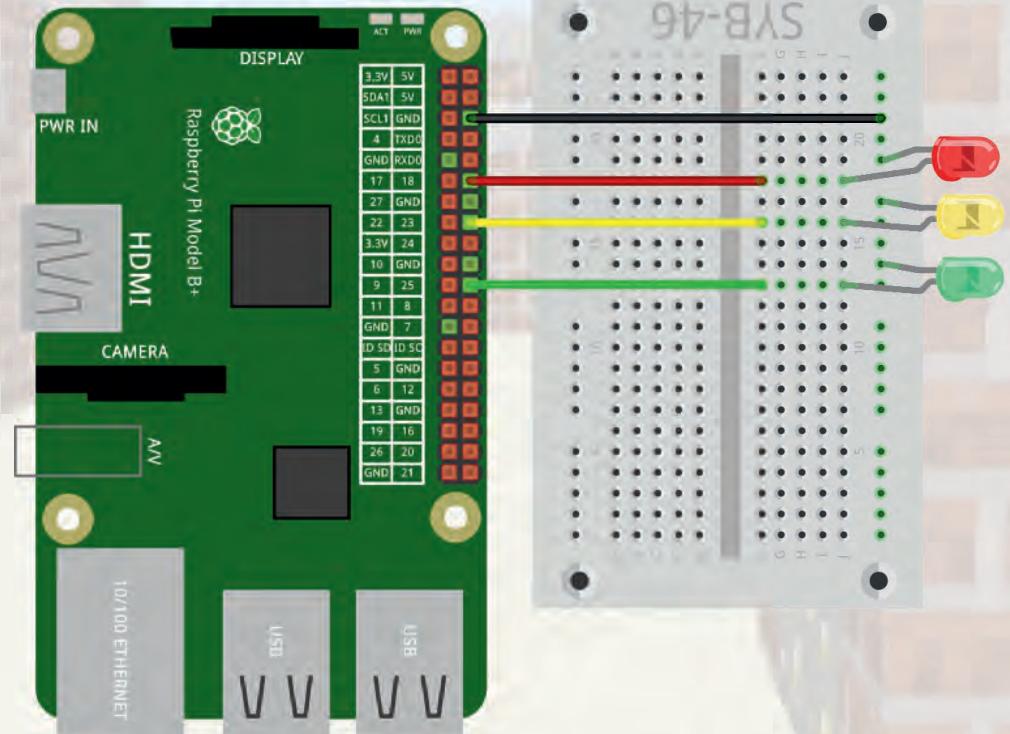
```
except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

In den folgenden Programmbeschreibungen werden immer nur neue Funktionen erwähnt. Sollten Sie also einmal einen Tag übersprungen haben, empfiehlt es sich, zumindest die Programmbeschreibung zu lesen, da dieses Wissen bei späteren Tagen vorausgesetzt wird.

```
import time
```

Zusätzlich zu den bereits bekannten Bibliotheken wird die Bibliothek `time` importiert, die verschiedene Funktionen zur Berechnung von Zeiten enthält. Damit werden die Wartezeiten zwischen den Ampelphasen erstellt.



```

rot    = 0
gelb   = 1
gruen  = 2
Ampel=[18,23,25]

```

Um die GPIO-Pins im Programm leichter unterscheiden und das Programm auch einfacher an einen anderen Schaltungsaufbau anpassen zu können, werden die Pin-Nummern am Anfang in der Liste `Ampel[]` gespeichert. Die Indizes der drei Listenelemente werden in den drei Variablen `rot`, `gelb` und `gruen` gespeichert und können so sehr einfach zugeordnet werden.

```

GPIO.setup(Ampel[rot], GPIO.OUT, initial=True)
GPIO.setup(Ampel[gelb], GPIO.OUT, initial=False)
GPIO.setup(Ampel[gruen], GPIO.OUT, initial=False)

```

Die drei GPIO-Pins werden über die Liste als Ausgänge eingerichtet. Der zusätzliche Parameter `initial` schaltet die rote LED ein und die beiden anderen aus.

```

try:
    while True:

```

Um zu vermeiden, dass Python beim Programmstart GPIO-Warnungen ausgibt, wird die Endlosschleife in eine `try...except`-Konstruktion eingebettet, die am Ende die GPIO-Ports wieder schließt.

```
p = mc.player.getTilePos()
```

Am Anfang speichert die Endlosschleife wieder in jedem Durchlauf die Position der Spielfigur in der Variablen `p`.

```
mat = mc.getBlock(p.x, p.y-1, p.z)
```

Die Methode `mc.getBlock()` liest die Material-ID des Blocks unterhalb der Spielfigur aus. Dazu wird von der y-Koordinate 1 subtrahiert.

```
if mat==98:
```

Jeder Block hat ein typisches Material. Diese Materialien haben jeweils eigene Nummern. Die Material-ID der Blöcke für den Wartebalken unter dem Eingangsportal ist `98`.

Hinweis

Am 6. Tag finden Sie eine Liste aller Materialien in Minecraft mit den zugehörigen IDs.

== ist nicht gleich =

Das doppelte Gleichheitszeichen `==` steht für eine Gleichheitsabfrage, das einfache Gleichheitszeichen `=` wird dagegen für Variablenzuweisungen verwendet.

```

GPIO.output(Ampel[gelb],True)
time.sleep(0.6)

```

Steht die Spielfigur auf einem Block mit dieser Material-ID, beginnt der Ampelzyklus, indem die gelbe LED zusätzlich zu der bereits leuchtenden roten eingeschaltet wird. Danach wartet das Programm 0,6 Sekunden.

```

GPIO.output(Ampel[rot],False)
GPIO.output(Ampel[gelb],False)
GPIO.output(Ampel[gruen],True)
time.sleep(2)

```

Anschließend werden die rote und die gelbe LED aus- und die grüne LED eingeschaltet. In diesem Zustand wartet die Ampel zwei Sekunden. Auch bei realen Verkehrsampeln dauert die Grünphase deutlich länger als die Zwischenphasen Rot/Gelb und Gelb.

```

GPIO.output(Ampel[gruen],False)
GPIO.output(Ampel[gelb],True)
time.sleep(0.6)
GPIO.output(Ampel[gelb],False)
GPIO.output(Ampel[rot],True)
time.sleep(2)

```

Anschließend schaltet die Ampel über Gelb auf Rot und wartet wieder zwei Sekunden, bevor die Endlosschleife einen weiteren Durchlauf startet.

```

except KeyboardInterrupt:
    GPIO.cleanup()

```

Die letzten beiden Zeilen sind nicht mehr eingerückt. Sie werden erst dann ausgeführt, wenn der Benutzer die Endlosschleife mit der Tastenkombination `Strg+C` abbricht.

Dazu muss das Python-Fenster im Vordergrund sein. `Strg+C` im Minecraft-Fenster wird von Python ignoriert.

Am Ende eines Programms sollten alle verwendeten GPIO-Pins wieder zurückgesetzt werden, um Warnungen beim nächsten Programmstart zu vermeiden. Die letzte Zeile dieses Programms erledigt das für alle vom Programm initialisierten GPIO-Pins auf einmal. Pins, die von anderen Programmen initialisiert wurden, bleiben davon unberührt. So wird der Ablauf dieser anderen möglicherweise parallel laufenden Programme nicht gestört.

Der Ampelzyklus

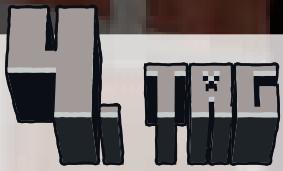
Farbe(n)	Zeit
Rot/Gelb	0.6 sek
Grün	2.0 sek
Gelb	0.6 sek
Rot	2.0 sek

Schon gewusst...?

- Der Begriff „Ampel“ kommt von dem mittelhochdeutschen Wort „ampulla“ und bezeichnet ein kleines Gefäß für Öl oder andere Flüssigkeiten. Im Mittelalter wurde damit das Ewige Licht in katholischen Kirchen bezeichnet. Die ersten Verkehrsampeln hingen an Drahtseilen über Kreuzungen und sahen daher ähnlich aus.
- Die offizielle Bezeichnung von Verkehrsampeln ist nach § 43 der deutschen Straßenverkehrsordnung (StVO) „Lichtzeichenanlage“, abgekürzt: LZA. In der österreichischen Straßenverkehrsordnung und in der Signalisationsverordnung (SSV) zum Schweizer Straßenverkehrsgesetz werden Verkehrsampeln als „Lichtsignalanlage“, abgekürzt: LSA, bezeichnet. Eine „Lichtzeichenanlage“ ist in Österreich dagegen eine Anlage an Bahnübergängen.
- In Deutschland dürfen Radfahrer bei roter Ampel rechts von einem Radweg auf einen anderen abbiegen, wenn dabei keine Fahrbahn überquert wird. In den Niederlanden ist das nur erlaubt, wenn ein entsprechendes Zusatzschild an der Ampel angebracht ist.
- Ampeln in der DDR hatten teilweise eine Grün/Gelb-Phase zwischen den Phasen Grün und Gelb. Dieses Signal wurde mit der Wiedervereinigung abgeschafft, da es nicht den Richtlinien für Lichtsignalanlagen (RiLSA) entspricht.
- Das ostdeutsche Ampelmännchen entsprach ebenfalls nicht den RiLSA und wurde an vielen Stellen umgebaut. Nach Protesten aus der Bevölkerung wurden die Richtlinien angepasst und das Ost-Ampelmännchen nachträglich wieder für zulässig erklärt.
- Die Dauer der roten und grünen Ampelphasen wird je nach Situation unterschiedlich eingestellt. Die Rot/Gelb-Phase dauert eine Sekunde. Für Gelb gibt es Vorgaben in den RiLSA, die von der zulässigen Geschwindigkeit auf der Straße abhängen:

Geschwindigkeit	Gelbphase
50 km/h	3 sek.
60 km/h	4 sek.
70 km/h	5 sek.

- In Düsseldorf und Wismar gibt es Fußgängerampeln mit Gelbsignal.
- Alle Ampeln weltweit haben die gleiche Anordnung der Lichter: Rot oben, Grün unten - mit einer Ausnahme: Das rote Licht der Tipperary-Hill-Ampel in Syracuse, USA ist unten, das grüne oben. Anhänger der irischstämmigen Bevölkerung empfanden es als Diskriminierung, dass das grüne Licht unter dem roten Licht hängt, und haben die Ampel mehrfach beschädigt. Die Zerstörungswut endete erst, nachdem die Anordnung der Lichter umgedreht wurde.



Heute im Adventskalender

- 2 GPIO-Verbindungskabel

Ein Schlag mit dem Schwert lässt die LEDs blinken

Das Programm des 4. Tages schaltet eine kurze Lauflichtsequenz ein, wenn die Spielfigur auf einen bestimmten Klotz schlägt, der vor der ersten Weihnachtshütte in der Minecraft-Welt steht. Hat die Spielfigur das Schwert in der Hand, kann der Spieler mit der linken Maustaste Blöcke zerschlagen. Ein Klick mit der rechten Maustaste löst in Python ein Event aus, worauf das Programm beliebig reagieren kann.



Beim Schlag auf den goldenen Block leuchtet ein Lauflicht.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 4 GPIO-Verbindungskabel

Hinweis

In diesem und vielen weiteren Programmen finden Sie das Zeichen ↵. An dieser Stelle passt eine Programmzeile nicht in eine Druckzeile.

Versuchen Sie nicht, dieses Zeichen einzugeben, sondern schreiben Sie diese und die folgende Programmzeile einfach in eine Zeile, auch wenn diese bei späteren Programmen sehr lang wird.

In Python können Programmzeilen nicht beliebig umgebrochen werden, da ein Zeilenende eine logische Funktion im Programm hat.

Das Programm

Das Programm `04mc.py` liest die Minecraft-Ereignisse, die durch Schläge auf Blöcke ausgelöst werden. Schlägt die Spielfigur auf einen Block aus dem Material `GLOWSTONE_BLOCK`, wird eine Lauflichtsequenz ausgelöst.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
LED = [18,23,25]

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)

try:
    while True:
        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit. pos.y, hit.pos.z)
            if bl.id == block.GLOWSTONE_BLOCK.id:
                for i in LED:
                    GPIO.output(i,True)
                    time.sleep(0.05)
                    GPIO.output(i,False)

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

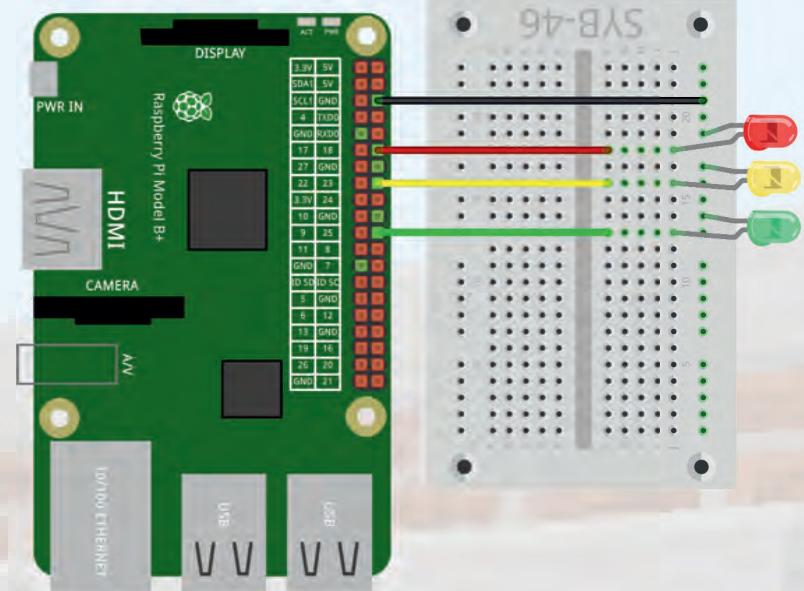
`LED = [18,23,25]`

Sobald die bereits bekannten Bibliotheken importiert sind und die Verbindung zu Minecraft hergestellt ist, wird eine Liste mit den drei für die LEDs verwendeten GPIO-Pins angelegt.

```
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
```

Diese GPIO-Pins werden als Ausgänge initialisiert und ausgeschaltet.

```
try:
    while True:
        for hit in mc.events.pollBlockHits():
```



Lauflicht aus drei LEDs am Raspberry Pi.

Die Hauptschleife des Programms liest jetzt in jedem Schleifendurchlauf die Liste der Events, die durch Schwertschläge auf Blöcke ausgelöst werden. Die Variable `hit` enthält unter anderem die Koordinaten des getroffenen Blocks.

```
bl = mc.getBlockWithData(hit.pos.x, hit. pos.y, hit.pos.z)
```

Diese Zeile speichert den getroffenen Block in der Variablen `bl`.

```
if bl.id == block.GLOWSTONE_BLOCK.id:
```

Wenn das Material dieses Blocks `GLOWSTONE_BLOCK` ist, wird der Lichteffekt ausgelöst. In diesem Programm funktioniert das auch, wenn Sie einen anderen Block aus dem gleichen Material mit dem Schwert zerschlagen. Die Koordinaten des Blocks werden nicht geprüft.

```
for i in LED:
    GPIO.output(i,True)
    time.sleep(0.05)
    GPIO.output(i,False)
```

Eine Schleife lässt die LEDs kurz hintereinander für jeweils 0,05 Sekunden aufblinken. Diese Schleife läuft genau einmal. Danach wartet die Hauptschleife wieder, bis auf einen Block geschlagen wird.

Das Programm läuft, bis der Benutzer in der Python-Shell die Tastenkombination `[Strg]+[C]` drückt. Zum Schluss werden die GPIO-Pins zurückgesetzt.

5. TAG

Heute im Adventskalender

- 1 Taster

Sandlandschaft bauen

Das Programm des 5. Tages baut Minecraft-Blöcke, wenn man auf einen Taster drückt. Bei jedem Druck auf den Taster werden neun Blöcke in einer quadratischen Anordnung (3 x 3) gebaut, und die Spielfigur wird anschließend mittig darauf gestellt. Durch mehrfaches Drücken der Taste lassen sich so Türme bauen.

Das Programm verwendet das Material Sand, das eine besondere Eigenschaft hat. Die meisten Minecraft-Blöcke lassen sich frei in der Luft an jeder beliebigen y-Koordinate bauen. Sandblöcke dagegen fallen immer bis zum Boden, auch im Wasser. Dadurch kann man mit dem Programm und kleinen Bewegungen Hügellandschaften erzeugen.



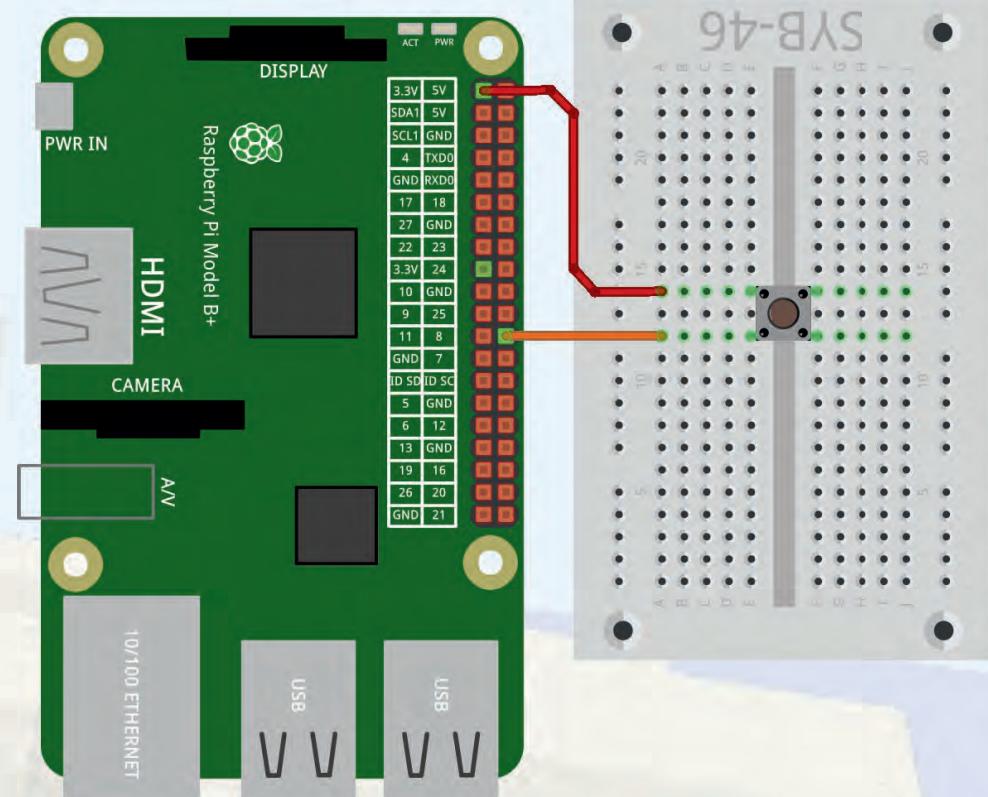
Beim Bauen von Landschaften lässt sich der Effekt am besten erkennen, wenn man die Landschaft von außen sieht und nicht mit den Augen der Spielfigur.

Bauteile

- 1 Steckbrett
- 1 Taster
- 2 GPIO-Verbindungskabel

GPIO-Ports können nicht nur Daten ausgeben, zum Beispiel über LEDs, sondern auch zur Dateneingabe dienen. Dazu müssen sie im Programm als Eingang definiert werden.

Liegt auf einem als Eingang definierten GPIO-Port ein +3,3-V-Signal an, wird dieses als logisches `True` bzw. `1` ausgewertet.



Ein Taster auf dem Steckbrett.

Taster

Zur Eingabe verwenden wir im folgenden Projekt einen Taster, der direkt auf die Steckplatine gesteckt wird. Der Taster hat vier Anschlusspins, wobei je zwei gegenüberliegende (großer Abstand) miteinander verbunden sind. Solange die Taste gedrückt ist, sind alle vier Anschlüsse miteinander verbunden. Im Gegensatz zu einem Schalter rastet ein Taster nicht ein. Die Verbindung wird beim Loslassen sofort wieder getrennt.

Ein Druck auf den Taster verbindet in unserer Schaltung den GPIO-Pin 8 mit +3,3 V. Lässt man den Taster wieder los, bekommt der Eingang einen undefinierten Zustand, was in der Digitalelektronik nicht passieren darf. Für solche Fälle verfügen alle GPIO-Pins über sogenannte Pulldown-Widerstände, die einen Eingang, an dem kein Signal anliegt, automatisch auf `False` herunterziehen.

Achtung

Verwenden Sie nie die +5-V-Pins des Raspberry Pi für Logiksignale in Schaltungen. 5 V würden die GPIO-Eingänge überlasten und den Raspberry Pi beschädigen.

Das Programm

Hinter dem Weihnachtsmarkt befindet sich eine unregelmäßig geformte Sandlandschaft. Hier können Sie aus Sand weitere Klötze bauen. Das Programm `05mc.py` baut bei jedem Druck auf den Taster neun Blöcke in einer quadratischen Anordnung (3 x 3) rund um die aktuelle Position der Spielfigur und stellt die Spielfigur anschließend mittig darauf.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
t1 = 8

GPIO.setmode(GPIO.BCM)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
```

```

p = mc.player.getTilePos()
mc.setBlocks(p.x-1, p.y, p.z-1, p.x+1, p.y, ←
p.z+1, block.SAND)
mc.player.setPos(p.x, p.y+1, p.z)
time.sleep(0.2)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert das Programm

t1 = 8

Sobald die bereits bekannten Bibliotheken importiert sind und die Verbindung zu Minecraft hergestellt ist, wird der für den Taster verwendete GPIO-Pin in der Variablen t1 gespeichert.

```
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
```

Dieser GPIO-Pin wird als Eingang initialisiert, und der auf dem Raspberry Pi eingebaute Pulldown-Widerstand wird eingeschaltet.

```

try:
    while True:
        if GPIO.input(t1)==True:

```

Die Endlosschleife wartet darauf, dass der Taster gedrückt wird. Dabei bekommt der GPIO-Pin den Wert True, da er über den Taster mit +3,3 V verbunden wird.

```
p = mc.player.getTilePos()
```

Jetzt wird die Position der Spielfigur in der Variablen p gespeichert.

```
mc.setBlocks(p.x-1, p.y, p.z-1, p.x+1, p.y, ←
p.z+1, block.SAND.id)
```

Die Funktion setBlocks() setzt mehrere Blöcke auf einmal in einem quadratischen Raster. Dazu brauchen nur zwei gegenüberliegende Ecken dieses Rasters angegeben zu werden. Hier ist das Raster 3 × 3 Einheiten in der Fläche groß und eine Einheit dick.

```
mc.player.setPos(p.x, p.y+1, p.z)
```

Jetzt wird die Spielfigur eine Einheit nach oben versetzt, sodass sie sich auf dem mittleren der neu angelegten Blöcke befindet.

```
time.sleep(0.2)
```

Das Programm wartet 0,2 Sekunden, um Tastenprellen zu vermeiden. So wird verhindert, dass ein etwas längeres Drücken des Tasters gleich mehrere Schichten von Blöcken anlegt. Danach startet die Endlosschleife neu.

Einige Taster sitzen ziemlich locker im Steckbrett. Hier sollte die time.sleep-Zeit verlängert werden, um versehentliches Mehrfachbauen zu vermeiden.

Das Programm läuft, bis der Benutzer in der Python-Shell die Tastenkombination **Strg**+**C** drückt. Zum Schluss werden die GPIO-Pins zurückgesetzt.

Wahr oder nicht ...?

Rund um Minecraft™ ranken sich diverse Mythen und angebliche Fakten.

- Ein Minecraft™-Block hat einen Meter Kantenlänge in der realen Welt.
- Die Minecraft™-Welt hat eine quadratische Fläche von maximal 64.000.000 Blöcken Kantenlänge, das entspricht einer Fläche von 4.096.000.000 km². Das entspricht etwa dem Achtfachen der Erdoberfläche.
- Ein Tag dauert in Minecraft™ 20 Minuten (nicht Pi Edition).
- Ein goldener Apfel in Minecraft™ würde 154 Tonnen in der realen Welt wiegen.
- Hört man die Laute des „Enderman“ (nicht Pi Edition) rückwärts, hört man unter anderem „Hi“, „Hello“ und „What's up“.
- Der Creeper in Minecraft™ (nicht Pi Edition) sollte ursprünglich ein Schwein werden und entstand versehentlich durch Vertauschen der Koordinatenachsen.
- Der Codename während der Entwicklung von Minecraft™ war „Cave Game“.
- Bewirft man eine Spinne mit einem Unsichtbarkeitstrank, bleiben nur noch die leuchtenden Augen zu sehen.
- Ein TNT-Block fällt 27 Blöcke, bevor er explodiert.
- Man braucht 3.000.000.000 TNT-Blöcke, um einen Block Bedrock zu zerschlagen.
- Ein Schaf mit Namen „jeb_“ verändert ständig seine Farbe (nicht Pi Edition).
- Skelette und Zombies (nicht Pi Edition) verbrennen am Tag nicht, wenn sie auf Seelensand stehen.
- Als Gegenstück zum Nether, der unterirdischen Höllenwelt, sollte eine Himmelsdimension ins Spiel kommen, was aber in einer der Beta-Versionen verworfen wurde.
- Angeblich wächst Zuckerrohr auf Sand schneller als auf anderen Landschaften.
- Es gibt weniger Kürbisse als Diamanten in der Minecraft™-Welt.

Für Ihre Notizen



6. TAG

Heute im Adventskalender

- 1 m Schaltdraht

Schaltdraht

Heute ist Schaltdraht im Adventskalender enthalten. Der Schaltdraht wird für verschiedene Experimente benötigt, um Brücken zwischen verschiedenen Kontaktreihen auf dem Steckbrett herzustellen. Schneiden Sie den Draht mit einem kleinen Seitenschneider je nach Experiment auf die passenden Längen zu. Um die Drähte besser in das Steckbrett stecken zu können, empfiehlt es sich, sie leicht schräg abzuschneiden, sodass eine Art Keil entsteht. Entfernen Sie an beiden Enden auf einer Länge von etwa einem halben Zentimeter die Isolierung.

LEDs dimmen

LEDs können zwei verschiedene Zustände annehmen, ein und aus. Das Gleiche gilt für die als digitale Ausgänge definierten GPIO-Ports. Demnach wäre es theoretisch nicht möglich, eine LED zu dimmen.

Mit einem Trick erreicht man es dennoch, die Helligkeit einer LED an einem digitalen GPIO-Port zu regeln. Lässt man eine LED schnell genug blinken, nimmt das menschliche Auge das nicht mehr als Blinken wahr. Die als Pulsweitenmodulation (PWM) bezeichnete Technik erzeugt ein pulsierendes Signal, das in sehr kurzen Abständen ein- und ausgeschaltet wird. Die Spannung des Signals bleibt immer gleich, nur das Verhältnis zwischen Level *False* (0 V) und Level *True* (+3,3 V) wird verändert. Das Tastverhältnis gibt das Verhältnis der Länge des eingeschalteten Zustands zur Gesamtdauer eines Schaltzyklus an. Je kleiner das Tastverhältnis, desto kürzer ist die Leuchtzeit der LED innerhalb eines Schaltzyklus. Dadurch wirkt die LED dunkler als eine permanent eingeschaltete LED.



Links: Tastverhältnis 50 % – rechts: Tastverhältnis 20 %.

Warum 50 Hertz die ideale Frequenz für PWM ist

Das menschliche Auge nimmt Lichtwechsel schneller als 20 Hertz nicht mehr wahr. Da das Wechselstromnetz in Europa eine Frequenz von 50 Hertz nutzt, blinken viele Beleuchtungskörper mit dieser Frequenz, die vom Auge nicht wahrgenommen wird. Würde eine LED mit mehr

als 20 Hertz, aber weniger als 50 Hertz blinken, kann es zu Interferenzen mit anderen Lichtquellen kommen, wodurch der Dimmefekt nicht mehr gleichmäßig erscheint.



Bei Annäherung an den GLOWSTONE-Block leuchten die LEDs heller.

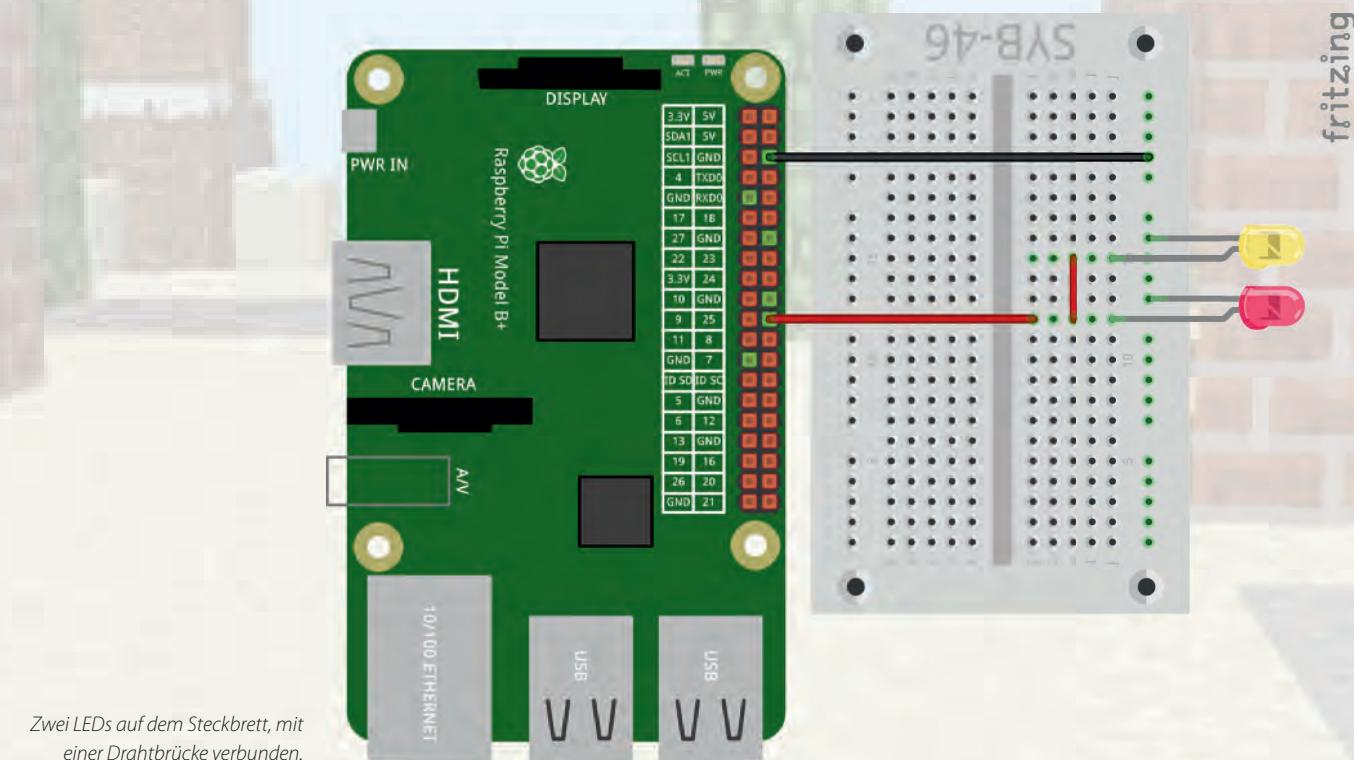
Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 2 GPIO-Verbindungsleitung
- 1 Drahtbrücke

Die Drahtbrücke wird dazu verwendet, zwei Kontaktreihen des Steckbretts zu verbinden, um so zwei LEDs an einem GPIO-Pin des Raspberry Pi anzuschließen. Das ist möglich, um beide LEDs genau gleich zu schalten. Schließen Sie auf diese Weise nicht zu viele LEDs an einen GPIO-Pin an, da er sonst überlastet werden kann. Zwei oder drei LEDs funktionieren aber problemlos.

Das Programm

Das Programm `06mc.py` zeigt, wie PWM-Signale mit Python ausgegeben werden. Wenn Sie sich mit der Figur wie in der Abbildung vor die zweite Weihnachtsmarkthütte stellen und sich dem Block aus GLOWSTONE-Material nähern, werden die beiden LEDs immer heller. An den zwei verschiedenfarbigen LEDs ist die Wirkung des PWM-Effektes bei unterschiedlichen Farben gut zu erkennen.



Zwei LEDs auf dem Steckbrett, mit einer Drahtbrücke verbunden.

```

#!/usr/bin/python
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
LED = 25

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT, initial=False)

pwm = 0
l = GPIO.PWM(LED, 50)
l.start(pwm)

try:
    while True:
        p = mc.player.getTilePos()
        if p.z>=5 and p.z<=15:
            pwm = 10*(15-p.z)
        l.ChangeDutyCycle(pwm)
        time.sleep(0.1)

except KeyboardInterrupt:
    l.stop()
    GPIO.cleanup()

```

So funktioniert das Programm

`LED = 25`

Sobald die bereits bekannten Bibliotheken importiert sind und die Verbindung zu Minecraft hergestellt ist, wird der für die LEDs verwendete GPIO-Pin in der Variablen `LED` gespeichert.

`GPIO.setup(LED, GPIO.OUT, initial=False)`

Dieser GPIO-Pin wird als Ausgang initialisiert und ausgeschaltet.

`pwm = 0`

Das Tastverhältnis des PWM-Signals wird in der Variablen `pwm` gespeichert und am Anfang auf `0` gesetzt.

`l = GPIO.PWM(LED, 50)`

Die Funktion `GPIO.PWM()` aus der GPIO-Bibliothek ist entscheidend für die Ausgabe von PWM-Signalen. Diese Funktion benötigt zwei Parameter: den GPIO-Pin und die Frequenz des PWM-Signals. In unserem Fall wird der GPIO-Pin über die Variable `LED` festgelegt, und die Frequenz beträgt `50` Hertz (Schwingungen pro Sekunde).

`GPIO.PWM()` erzeugt ein Objekt, das in der Variablen `l` gespeichert wird.

`l.start(pwm)`

Die Methode `start()` startet die Generierung des PWM-Signals. Dazu muss noch ein Tastverhältnis angegeben werden. In unserem Fall ist das Tast-

verhältnis `0`, was vorher in der Variablen `pwm` gespeichert wurde. Die LED ist also immer ausgeschaltet. Python verwendet für PWM Werte zwischen `0` und `100`, die direkt dem prozentualen Anteil der Zeit entsprechen, in dem der Pin innerhalb eines Frequenzzyklus eingeschaltet ist. Ein PWM-Wert von `25` schaltet die LED also ein Viertel der Zeit ein, die übrigen drei Viertel des Zyklus aus.

```

try:
    while True:
        p = mc.player.getTilePos()

        if p.z>=5 and p.z<=15:
            pwm = 10*(15-p.z)

```

Die Hauptschleife des Programms liest immer wieder die Position des Spielers aus und speichert diese im Objekt `p`.

```

        l.ChangeDutyCycle(pwm)
        time.sleep(0.1)

```

Befindet sich die Spielfigur an einer z-Koordinate zwischen `5` und `15`, wird daraus ein neuer PWM-Wert errechnet. Die verwendete Formel ergibt bei der z-Koordinate `15`, weit weg von dem Block aus GLOWSTONE-Material, einen PWM-Wert `0`. Steht die Spielfigur direkt vor diesem Block, ergibt sich ein PWM-Wert von `100`. Ein PWM-Signal von `100` entspricht einer voll eingeschalteten LED.

```

        time.sleep(0.1)

```

Unabhängig davon, was diese Abfragen ergeben haben, wird das PWM-Signal auf den neuen Wert gesetzt und danach eine Zehntelsekunde gewartet. Solche Wartezeiten werden in Schleifen eingefügt, damit der Raspberry Pi nicht vollständig damit beschäftigt ist, das Python-Programm auszuführen, denn sonst würde sich Minecraft deutlich trüger verhalten.

```

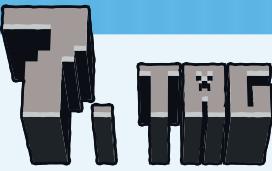
except KeyboardInterrupt:
    l.stop()
    GPIO.cleanup()

```

Beim Drücken der Tastenkombination `Strg + C` auf der Tastatur wird vor dem Zurücksetzen der GPIO-Pins noch das PWM-Signal beendet.

Die Minecraft™-Blocktypen

Material	ID	Material	ID
AIR	0	STONE_SLAB	44
STONE	1	BRICK_BLOCK	45
GRASS	2	TNT	46
DIRT	3	BOOKSHELF	47
COBBLESTONE	4	MOSS_STONE	48
WOOD_PLANKS	5	OBSIDIAN	49
SAPLING	6	TORCH	50
BEDROCK	7	FIRE	51
WATER_FLOWING	8	STAIRS_WOOD	53
WATER	8	CHEST	54
WATER_STATIONARY	9	DIAMOND_ORE	56
LAVA_FLOWING	10	DIAMOND_BLOCK	57
LAVA	10	CRAFTING_TABLE	58
LAVA_STATIONARY	11	FARMLAND	60
SAND	12	FURNACE_INACTIVE	61
GRAVEL	13	FURNACE_ACTIVE	62
GOLD_ORE	14	DOOR_WOOD	64
IRON_ORE	15	LADDER	65
COAL_ORE	16	STAIRS_COBBLESTONE	67
WOOD	17	DOOR_IRON	71
LEAVES	18	REDSTONE_ORE	73
GLASS	20	SNOW	78
LAPIS_LAZULI_ORE	21	ICE	79
LAPIS_LAZULI_BLOCK	22	SNOW_BLOCK	80
SANDSTONE	24	CACTUS	81
BED	26	CLAY	82
COBWEB	30	SUGAR_CANE	83
GRASS_TALL	31	FENCE	85
WOOL	35	GLOWSTONE_BLOCK	89
FLOWER_YELLOW	37	BEDROCK_INVISIBLE	95
FLOWER_CYAN	38	STONE_BRICK	98
MUSHROOM_BROWN	39	GLASS_PANE	102
MUSHROOM_RED	40	MELON	103
GOLD_BLOCK	41	FENCE_GATE	107
IRON_BLOCK	42	GLOWING_OBSIDIAN	246
STONE_SLAB_DOUBLE	43	NETHER_REACTOR_CORE	247



Heute im Adventskalender

- ## ■ 1 Taster

Bäume bauen

Normalerweise baut man mit Minecraft jeden Block einzeln. Das Programm des 7. Tages baut mit einem Druck auf einen Taster einen Baum aus insgesamt 19 Kötzen, drei Felder von der Spielfigur entfernt. Ein Druck auf die andere Taste löscht diesen Baum wieder.



Ein Baum, auf Tastendruck gebaut.

Bauteile

- 1 Steckbrett
 - 2 Taster
 - 2 Drahtbrücken
 - 3 GPIO-Verbindungskabel

Das Programm

Das Programm **07mc.py** baut auf Tastendruck einen Baum, drei Felder in x-Richtung von der Spielfigur entfernt. Beim Drücken der anderen Taste wird an der gleichen Stelle ein Kubus aus dem Material **AIR** gebaut, der den Baum löscht.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time
```

```
mc = minecraft.Minecraft.create()
t1 = 16
t2 = 8

GPIO.setmode(GPIO.BCM)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            mc.setBlock(p.x+3, p.y, p.z, block.WOOD)
            mc.setBlocks(p.x+2, p.y+1, p.z-1, p.x+4,
p.y+2, p.z+1, block.LEAVES)
            time.sleep(0.2)
        if GPIO.input(t2)==True:
            p = mc.player.getTilePos()
            mc.setBlocks(p.x+2, p.y, p.z-1, p.x+4,
p.y+2, p.z+1, block.AIR)
            time.sleep(0.2)

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

```
t1 = 16  
t2 = 8
```

Sobald die bereits bekannten Bibliotheken importiert sind und die Verbindung zu Minecraft hergestellt ist, werden die für die beiden Taster verwendeten GPIO-Pins in den Variablen `t1` und `t2` gespeichert.

```
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)  
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
```

Diese GPIO-Pins werden als Eingänge initialisiert, und die Pulldown-Widerstände werden eingeschaltet.

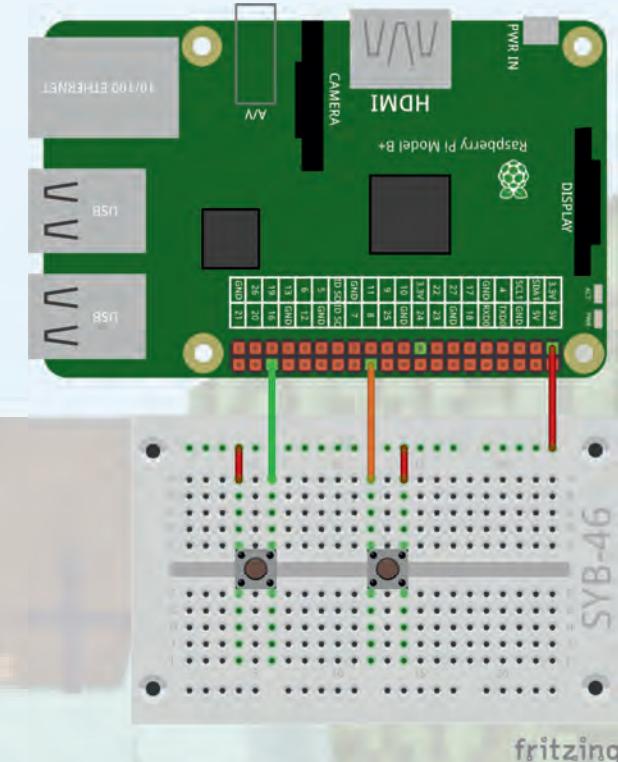
```
try:  
    while True:  
        if GPIO.input(t1)==True:  
            p = mc.player.getTilePos()
```

Die Endlosschleife des Programms wartet darauf, dass einer der Taster gedrückt wird. Wird der Taster **t1** gedrückt, speichert das Programm die Position des Klotzes, auf dem die Spielfigur steht, im Objekt **p**.

```
mc.setBlock(p.x+3, p.y, p.z, block.WOOD)
```

Anschließend wird drei Koordinateneinheiten in x-Richtung von der Spielfigur entfernt ein Block aus dem Material **WOOD** gebaut, der den Baumstamm darstellt.

```
mc.setBlocks(p.x+2, p.y+1, p.z-1, p.x+4,  
p.y+2, p.z+1, block.LEAVES)
```



Zwei Taster auf dem Steckbrett

Danach wird ein Quader aus 18 Blöcken gebaut – zwei Ebenen von je 3×3 Blöcken im Quadrat –, der die stilisierte Baumkrone darstellt, und mittig auf den Baumstamm gesetzt.

```
time.sleep(0.2)
```

Das Programm wartet 0,2 Sekunden, um Tastenprellen zu vermeiden und zu verhindern, dass der Raspberry Pi ständig mit der Abfrage der Taster beschäftigt ist.

```
if GPIO.input(t2)==True:  
    p = mc.player.getTilePos()  
    mc.setBlocks(p.x+2, p.y, p.z-1, p.x+4, ↴  
y+2, p.z+1, block.AIR)  
    time.sleep(0.2)
```

Wird der Taster **t2** gedrückt, speichert das Programm ebenfalls die Position des Klotzes, auf dem die Spielfigur steht, und baut dann einen Kubus aus dem Material **AIR**, der den Baum einschließlich Stamm ausfüllt und damit löscht. Um in Minecraft Klötzte zu löschen, baut man einfach an der gleichen Stelle sogenannte **Luftklötze**.

8. TAG

Heute im Adventskalender

- 2 GPIO-Verbindungskabel

Hecke bauen

Das Programm des 8. Tages baut Hecken auf dem Weihnachtsmarktgelände. Eine Taste baut Hecken in x-Richtung, die andere in z-Richtung.



Der Spieler baut mit zwei Tasten Hecken.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 2 Taster
- 2 Drahtbrücken
- 7 GPIO-Verbindungskabel

Das Programm

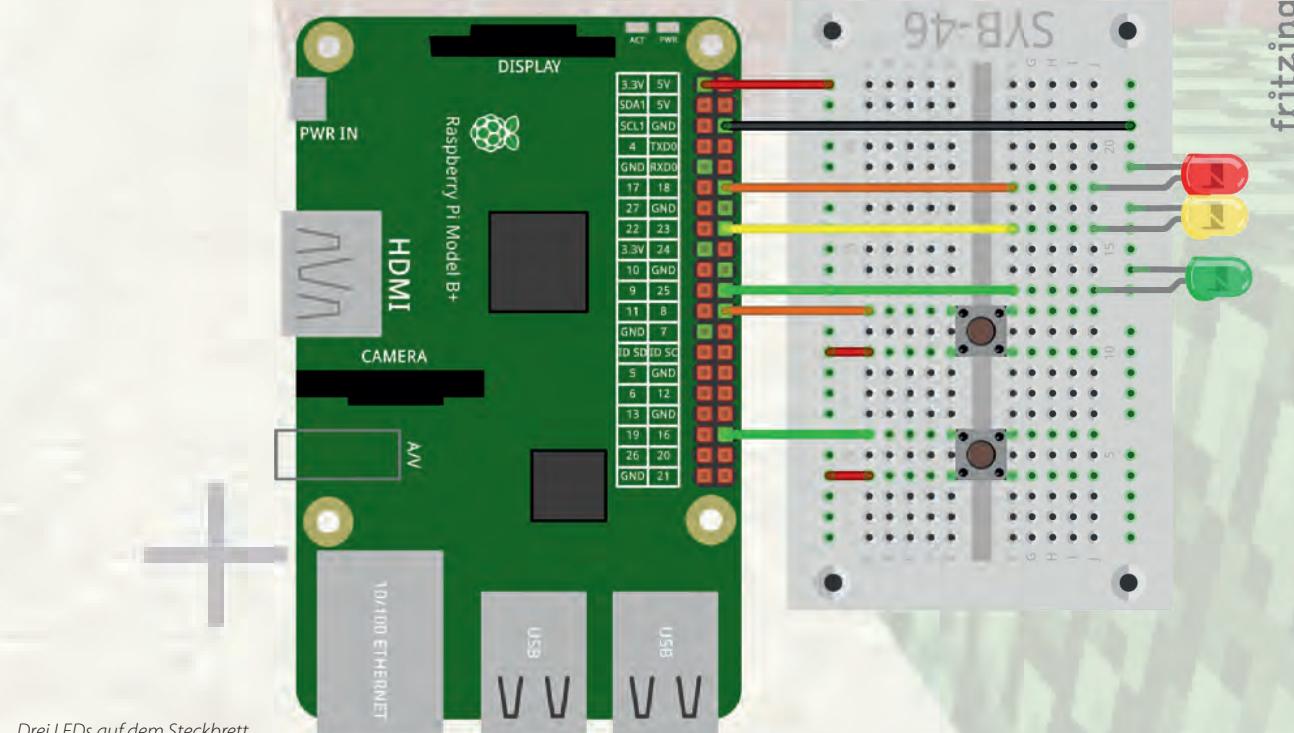
Das Programm [08mc.py](#) baut von der aktuellen Position der Spielfigur aus eine Hecke aus drei Klötzen in x- oder z-Richtung, je nachdem, welche Taste gedrückt wurde. Die Hecke wird in der y-Höhe der Spielfigur gebaut. Um Hecken einfach aneinanderzubauen, klettern Sie mit der Spielfigur auf eine Hecke. Steht die Spielfigur auf einer Hecke, wird die nächste Hecke in y-Richtung eine Einheit weiter unten, also wieder auf dem Boden, gebaut.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
t1 = 16
t2 = 8
LED = [18,23,25]

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            m = mc.getBlock(p.x, p.y-1, p.z)
            for i in range(3):
                GPIO.output(LED[i],True)
                if m == block.LEAVES.id:
                    mc.setBlock(p.x, p.y-1, p.z+1+i, block.LEAVES)
                else:
                    mc.setBlock(p.x, p.y, p.z+1+i, block.LEAVES)
                time.sleep(0.05)
                GPIO.output(LED[i],False)
                time.sleep(0.2)
        if GPIO.input(t2)==True:
            p = mc.player.getTilePos()
            m = mc.getBlock(p.x, p.y-1, p.z)
            for i in range(3):
                GPIO.output(LED[i],True)
                if m == block.LEAVES.id:
                    mc.setBlock(p.x+1+i, p.y-1, p.z, block.LEAVES)
                else:
                    mc.setBlock(p.x+1+i, p.y, p.z, block.LEAVES)
                time.sleep(0.05)
                GPIO.output(LED[i],False)
                time.sleep(0.2)
except KeyboardInterrupt:
    GPIO.cleanup()
```



fritzing

So funktioniert das Programm

```
LED = [18,23,25]
```

Auch diesmal wird eine Liste mit den drei GPIO-Pins der LEDs angelegt, die dann als Ausgänge initialisiert und ausgeschaltet werden. Diese LEDs leuchten beim Bau eines Segments der Hecke nacheinander kurz auf.

```
try:  
    while True:  
        if GPIO.input(t1)==True:  
            p = mc.player.getTilePos()
```

Die Endlosschleife fragt in jedem Durchlauf ab, ob einer der Taster gedrückt ist. Ist der Taster **t1** gedrückt, speichert das Programm die aktuelle Position der Spielfigur in dem Objekt **p**.

```
    m = mc.getBlock(p.x, p.y-1, p.z)
```

Das Material des Blocks, auf dem die Figur steht, wird in der Variablen **m** gespeichert, um daran später zu erkennen, ob die Hecke in der gleichen y-Ebene oder eine Ebene tiefer gebaut werden soll.

```
    for i in range(3):  
        GPIO.output(LED[i],True)
```

Jetzt startet eine Schleife, die dreimal durchlaufen wird, um drei Klötze für die Hecke zu bauen. Als Erstes wird die LED mit der Nummer des aktuellen Schleifendurchlaufs eingeschaltet.

```
        if m == block.LEAVES.id:  
            mc.setBlock(p.x+1+i, p.y-1, p.z, block.  
LEAVES) ←
```

Ist das gespeicherte Material **LEAVES**, steht die Spielfigur auf einer Hecke. In diesem Fall wird der nächste Block in y-Richtung eine Ebene unterhalb der Spielfigur und in x-Richtung eine Einheit vor der Spielfigur gebaut. Zusätzlich wird der Schleifenzähler – im ersten Durchlauf 0 – hochgesetzt. Auf diese Weise entstehen in den drei Schleifendurchläufen drei Blöcke der Hecke, in positiver x-Richtung von der Spielfigur wegführend.

```
        else:  
            mc.setBlock(p.x+1+i, p.y, p.z, block.  
LEAVES) ←
```

Steht die Spielfigur nicht auf einer Hecke, wird die Hecke in der gleichen y-Ebene gebaut, in der sich die Spielfigur befindet.

```
        time.sleep(0.05)  
        GPIO.output(LED[i],False)
```

Am Ende jedes Schleifendurchlaufs wartet das Programm 0,05 Sekunden, damit die LEDs kurz zu sehen sind. Danach werden die LEDs wieder ausgeschaltet, und der nächste Durchlauf der Schleife startet.

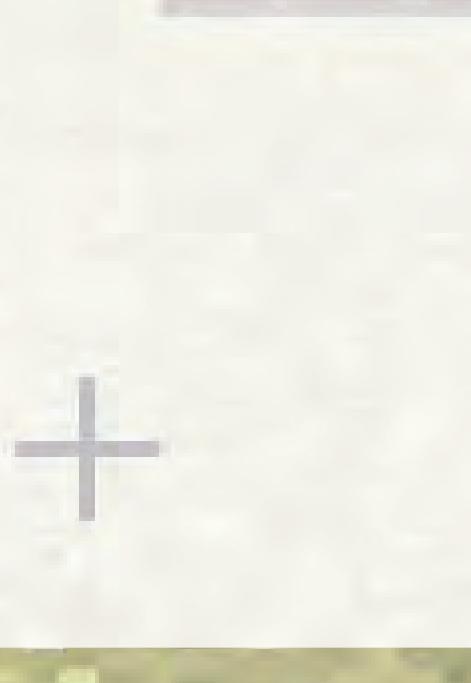
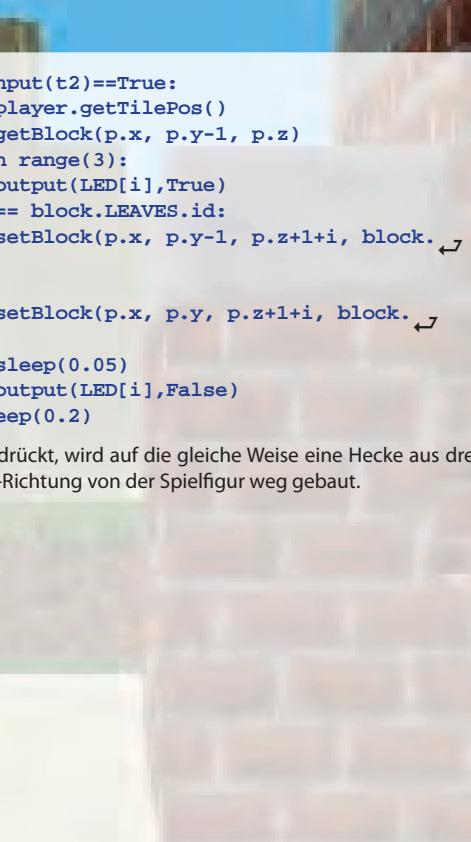
```
        time.sleep(0.2)
```

Nach dem letzten Durchlauf wartet das Programm 0,2 Sekunden, um Tastenprellen zu vermeiden.

```
if GPIO.input(t2)==True:  
    p = mc.player.getTilePos()  
    m = mc.getBlock(p.x, p.y-1, p.z)  
    for i in range(3):  
        GPIO.output(LED[i],True)  
        if m == block.LEAVES.id:  
            mc.setBlock(p.x, p.y-1, p.z+1+i, block.  
LEAVES) ←  
        else:  
            mc.setBlock(p.x, p.y, p.z+1+i, block.  
LEAVES) ←  
    time.sleep(0.05)  
    GPIO.output(LED[i],False)  
    time.sleep(0.2)
```

Wird die Taste **t2** gedrückt, wird auf die gleiche Weise eine Hecke aus drei Klötzen in positiver z-Richtung von der Spielfigur weg gebaut.

Für Ihre Notizen



Heute im Adventskalender

- 1 Stück Knete
- 1 20-MOhm-Widerstand (rot-schwarz-blau)

Widerstand

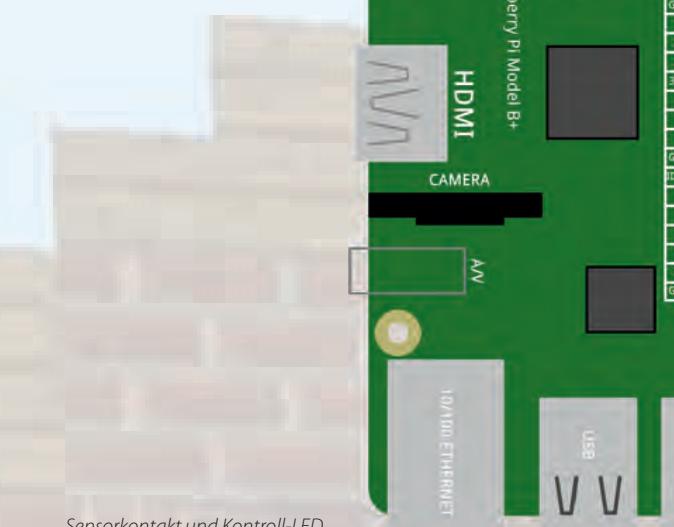
Widerstände werden zur Strombegrenzung an empfindlichen elektronischen Bauteilen sowie als Vorwiderstände für LEDs verwendet. Die Maßeinheit für den Widerstand ist das Ohm. 1000 Ohm werden als ein Kiloohm bezeichnet, abgekürzt kOhm, und 1000 kOhm als ein Megaohm, abgekürzt MOhm. Oft wird für die Einheit Ohm auch das Zeichen Ω (Omega) verwendet.

Die farbigen Ringe auf den Widerständen geben den Widerstandswert an. Sie sind deutlich leichter zu erkennen als winzig kleine Zahlen, die man nur noch auf ganz alten Widerständen findet, und mit etwas Übung lässt sich auch der Widerstandswert gut ablesen.

Die meisten Widerstände haben vier solcher Farbringe. Die ersten beiden stehen für Ziffern, der dritte bezeichnet einen Multiplikator und der vierte die Toleranz. Dieser Toleranzring ist meistens gold- oder silberfarben – Farben, die bei den ersten Ringen nicht vorkommen. Dadurch ist die Leserichtung immer eindeutig. Der Toleranzwert selbst spielt in der Digitalelektronik kaum eine Rolle. Die Tabelle zeigt die Bedeutung der farbigen Ringe auf Widerständen.

Farbe	Widerstandswert in Ohm			
	1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Multiplikator)	4. Ring (Toleranz)
Silber			$10^{-2} = 0,01$	$\pm 10\%$
Gold			$10^{-1} = 0,1$	$\pm 5\%$
Schwarz	0		$10^0 = 1$	
Braun	1	1	$10^1 = 10$	$\pm 1\%$
Rot	2	2	$10^2 = 100$	$\pm 2\%$
Orange	3	3	$10^3 = 1.000$	
Gelb	4	4	$10^4 = 10.000$	
Grün	5	5	$10^5 = 100.000$	$\pm 0,5\%$
Blau	6	6	$10^6 = 1.000.000$	$\pm 0,25\%$
Violett	7	7	$10^7 = 10.000.000$	$\pm 0,1\%$
Grau	8	8	$10^8 = 100.000.000$	$\pm 0,05\%$
Weiß	9	9	$10^9 = 1.000.000.000$	

In welcher Richtung ein Widerstand eingebaut wird, ist egal. Bei LEDs dagegen spielt die Einbaurichtung eine wichtige Rolle.

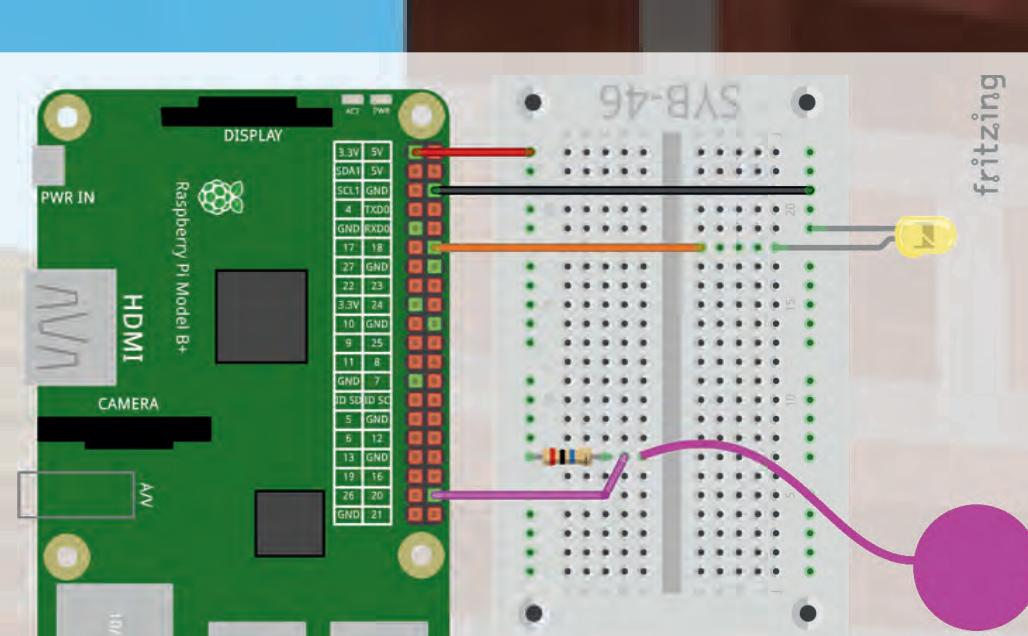


Sensorkontakt aus Knete

Ampeln, Türöffner, Lichtschalter und Automaten werden heute oft mit Sensorkontakten gesteuert, die man nur zu berühren braucht. Taster, die wirklich gedrückt werden müssen, werden immer seltener. Im Experiment des 9. Tages bringen Sie einen Steinblock über einen einfachen Sensorkontakt zum Leuchten.



Ein Sensorkontakt bringt Steine zum Leuchten.

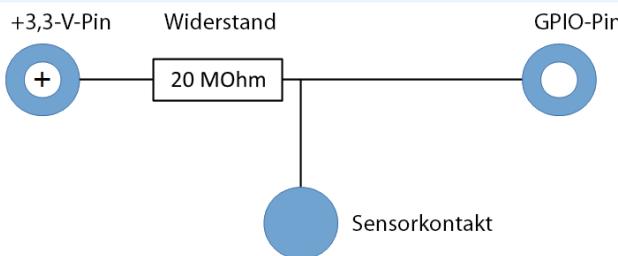


Bauteile

- 1 Steckbrett
- 1 LED gelb mit Vorwiderstand
- 1 20-MOhm-Widerstand (rot-schwarz-blau)
- 4 GPIO-Verbindungskabel
- 1 Knetekontakt

So funktionieren Sensorkontakte

Der als Eingang geschaltete GPIO-Pin ist über einen extrem hochohmigen Widerstand (20 MOhm) mit +3,3 V verbunden, sodass ein schwaches, aber eindeutig als High definiertes Signal anliegt. Ein Mensch, der nicht gerade frei in der Luft schwebt, ist immer geerdet und liefert über die elektrisch leitfähige Haut einen Low-Pegel. Berührt dieser Mensch jetzt einen Sensorkontakt, wird das schwache High-Signal von dem deutlich stärkeren Low-Pegel der Hand überlagert und zieht den GPIO-Pin auf Low.



Wie hoch allerdings der Widerstand zwischen Hand und Masse wirklich ist, hängt von vielen Dingen ab, unter anderem von Schuhen und Fußböden. Barfuß im nassen Gras ist die Verbindung zur Masse der Erde am besten, aber auch auf Steinfußböden funktioniert es meistens gut. Holzfußböden isolieren stärker, Kunststoffbodenbeläge sind oft sogar positiv aufgeladen. Sollte der Sensorkontakt nicht funktionieren, berühren Sie die LED auf der Masseschiene des Steckbretts und den eigentlichen Sensor gleichzeitig. Dann ist die Masseverbindung auf jeden Fall hergestellt.

Knete leitet den Strom etwa so gut wie menschliche Haut. Sie lässt sich leicht in jede beliebige Form bringen, und ein Knetekontakt fasst sich viel besser an als ein einfaches Stück Draht. Die Fläche, mit der die Hand den Kontakt berührt, ist deutlich größer. So kommt es nicht so leicht zu einem Wackelkontakt. Stecken Sie ein etwa 20 Zentimeter langes Stück Schaltdraht, von dem Sie an beiden Enden etwa einen Zentimeter weit die Isolierung entfernt haben, in ein Stück Knete und dessen anderes Ende in das Steckbrett. Sollten Sie den Massekontakt benötigen, können Sie, um einen besseren Kontakt zur Hand herzustellen, einen zweiten Knetekontakt an der Masseschiene anschließen.

Das Programm

Das Programm `09mc.py` prüft in einer Endlosschleife, ob der Knetekontakt berührt wird. Ist das der Fall, werden die beiden Blöcke, die den Tresen der Weihnachtsmarkthütte rechts vom Eingang darstellen, durch das Material `GOLD_ORE` ersetzt. Wird der Knetekontakt nicht berührt, erscheinen diese beiden Blöcke aus dem Material `COAL_ORE`. Beide Materialien ähneln sich stark, sodass die Veränderung wie ein Aufleuchten des Blocks aussieht. Zusätzlich leuchtet die LED immer dann, wenn der Knetekontakt berührt wird.

```

#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
LED = 18
k1 = 20

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT, initial=False)
GPIO.setup(k1, GPIO.IN)
    
```

```

try:
    while True:
        if GPIO.input(k1) == False:
            GPIO.output(LED, True)
            mc.setBlocks(3, 2, 4, 3, 2, 5, block.GOLD_ORE)
        else:
            GPIO.output(LED, False)
            mc.setBlocks(3, 2, 4, 3, 2, 5, block.COAL_ORE)
        time.sleep(0.05)

except KeyboardInterrupt:
    GPIO.cleanup()
    
```

So funktioniert das Programm

```

LED = 18
k1 = 20
    
```

Die Nummern der GPIO-Pins der LED und des Knetekontakts werden in Variablen gespeichert.

```

GPIO.setup(k1, GPIO.IN)
    
```

Damit die Sensorkontakte funktionieren, muss zuerst der interne Pulldown-Widerstand am GPIO-Pin 20 ausgeschaltet werden. Deshalb fehlt in der Zeile `GPIO.setup()` der dritte Parameter.

```

try:
    while True:
        if GPIO.input(k1) == False:
            GPIO.output(LED, True)
            mc.setBlocks(3, 2, 4, 3, 2, 5, block.GOLD_ORE)
        else:
            GPIO.output(LED, False)
            mc.setBlocks(3, 2, 4, 3, 2, 5, block.COAL_ORE)
        time.sleep(0.05)
    
```

Die Endlosschleife überprüft, ob der Knetekontakt berührt wurde. Da bei Berührung der GPIO-Pin mit Masse verbunden wird, ohne Berührung dagegen ein High-Signal anlegt, fragt die `if`-Abfrage, ob der Pin auf `False` steht.

```

GPIO.output(LED, True)
    
```

Ist das der Fall, wird die LED eingeschaltet.

```

mc.setBlocks(3, 2, 4, 3, 2, 5, block.GOLD_ORE)
    
```

Die Methode `.setBlocks()` baut einen $1 \times 1 \times 2$ Blöcke großen Quader aus dem leuchtenden Material `GOLD_ORE`.

```

else:
    GPIO.output(LED, False)
    mc.setBlocks(3, 2, 4, 3, 2, 5, block.COAL_ORE)
    
```

Wird der Knetekontakt dagegen nicht berührt, wird die LED ausgeschaltet und ein Quader aus dem dunkelgrauen Material `COAL_ORE` gebaut.

```

time.sleep(0.05)
    
```

Die kurze Wartezeit soll das Flackern zwischen den beiden Zuständen des Blocks verhindern. Je nachdem, wie gut die Masseverbindung über den Knetekontakt ist, müssen Sie diese Zeit eventuell etwas verlängern.

Für Ihre Notizen



10. TAG

Heute im Adventskalender

- 1 RGB-LED mit Vorwiderstand

RGB-LEDs

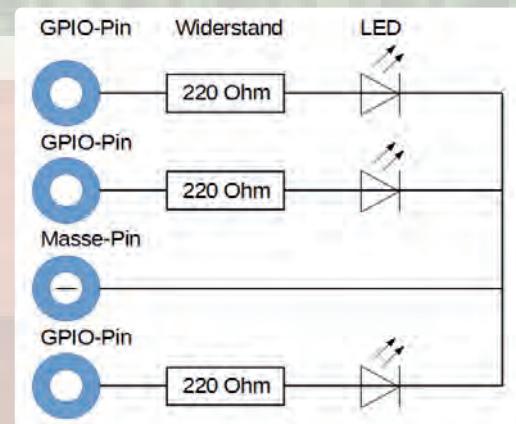
Eine normale LED leuchtet immer nur in einer Farbe. Die im Adventskalender enthaltenen RGB-LEDs können wahlweise in mehreren Farben leuchten. Hier sind drei LEDs verschiedener Farben in ein transparentes Gehäuse eingebaut. Jede dieser drei LEDs hat eine eigene Anode, über die sie mit einem GPIO-Pin verbunden wird. Die Kathode, die mit der Masseleitung verbunden wird, ist nur einmal vorhanden. Deshalb hat eine RGB-LED vier Anschlussdrähte.



Anschlusspins einer RGB-LED.

Die Anschlussdrähte der RGB-LEDs sind unterschiedlich lang, um sie eindeutig kenntlich zu machen. Anders als bei normalen LEDs ist die Kathode hier der längste Draht.

RGB-LEDs funktionieren wie drei einzelne LEDs und brauchen deshalb auch drei Vorwiderstände. In den RGB-LEDs in diesem Adventskalender sind sie ebenfalls bereits eingebaut.



Schaltplan für eine RGB-LED mit drei Vorwiderständen.

RGB-LEDs steuern

Zwischen den ersten beiden Weihnachtsmarkthütten auf der linken Seite befindet sich eine graue Fläche auf dem Boden. Das Programm baut hier drei farbige Blöcke, die die drei Farben der RGB-LED darstellen. Schlägt die Spielfigur mit dem Schwert auf eine Farbe, leuchtet diese in der Minecraft-Welt auf, und auch die entsprechende Farbkomponente der RGB-LED wird eingeschaltet. Ein weiterer Schlag schaltet die Farbe wieder aus. Durch Einschalten zweier Farbkomponenten lassen sich Farben mischen.



Die drei farbigen Blöcke für die RGB-LED.

Bauteile

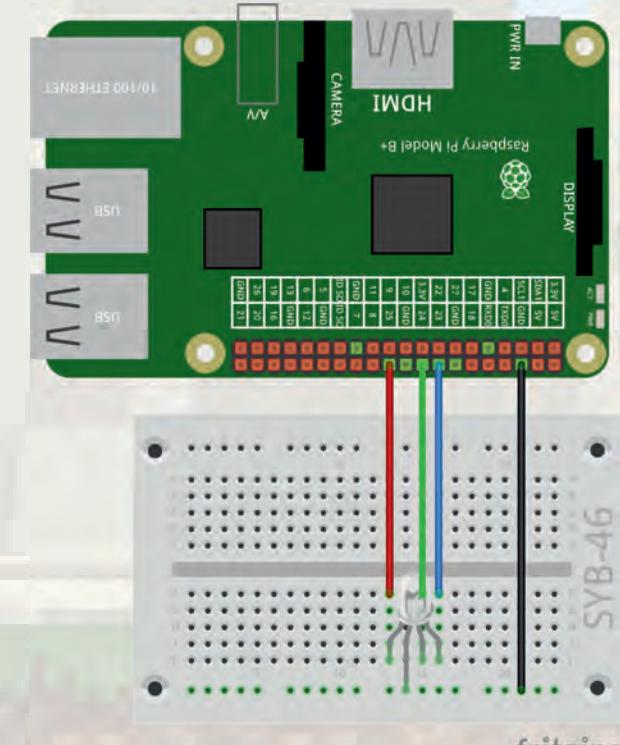
- 1 Steckbrett
- 1 RGB-LED mit Vorwiderstand
- 4 GPIO-Verbindungskabel

Das Programm

Das Programm `10mc.py` schaltet beim Schlag auf einen Block aus einem bestimmten Material Farben der RGB-LED ein oder aus und lässt auch die Blöcke auf dem Weihnachtsmarkt leuchten.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
r = 25
g = 24
b = 23
r0 = 14
g0 = 13
b0 = 11
r1 = 1
g1 = 5
b1 = 3
```



RGB-LED auf dem Steckbrett.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(r, GPIO.OUT, initial=False)
GPIO.setup(g, GPIO.OUT, initial=False)
GPIO.setup(b, GPIO.OUT, initial=False)

mc.setBlock(9, 2, 7, block.WOOL.id, r0)
mc.setBlock(9, 2, 8, block.WOOL.id, g0)
mc.setBlock(9, 2, 9, block.WOOL.id, b0)

try:
    while True:
        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit.
pos.y, hit.pos.z)
            if bl.id == block.WOOL.id and bl.data == r0:
                mc.setBlock(9, 2, 7, block.WOOL.id, r1)
                GPIO.output(r, True)
            if bl.id == block.WOOL.id and bl.data == r1:
                mc.setBlock(9, 2, 7, block.WOOL.id, r0)
                GPIO.output(r, False)
            if bl.id == block.WOOL.id and bl.data == g0:
```

```

mc.setBlock(9, 2, 8, block.WOOL.id, g1)
GPIO.output(g, True)
if bl.id == block.WOOL.id and bl.data == g1:
    mc.setBlock(9, 2, 8, block.WOOL.id, g0)
    GPIO.output(g, False)
if bl.id == block.WOOL.id and bl.data == b0:
    mc.setBlock(9, 2, 9, block.WOOL.id, b1)
    GPIO.output(b, True)
if bl.id == block.WOOL.id and bl.data == b1:
    mc.setBlock(9, 2, 9, block.WOOL.id, b0)
    GPIO.output(b, False)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert das Programm

```

r = 25
g = 24
b = 23

```

Nach dem Laden der Bibliotheken werden diverse Variablen angelegt. **r**, **g** und **b** bezeichnen die GPIO-Pins der drei Farben der RGB-LED.

```

r0 = 14
g0 = 13
b0 = 11

```

Die Blöcke, die die RGB-LED steuern, bestehen alle aus dem Material **WOOL**, das über einen zusätzlichen Parameter unterschiedliche Farben annehmen kann. Die drei Variablen **r0**, **b0** und **g0** bezeichnen die Farben im ausgeschalteten Zustand der RGB-LED.

```

r1 = 1
g1 = 5
b1 = 3

```

Die drei Variablen **r1**, **b1** und **g1** bezeichnen die Farben im eingeschalteten Zustand der RGB-LED.

```

GPIO.setmode(GPIO.BCM)
GPIO.setup(r, GPIO.OUT, initial=False)
GPIO.setup(g, GPIO.OUT, initial=False)
GPIO.setup(b, GPIO.OUT, initial=False)

```

Die drei GPIO-Pins werden als Ausgänge initialisiert und ausgeschaltet.

```

mc.setBlock(9, 2, 7, block.WOOL.id, r0)
mc.setBlock(9, 2, 8, block.WOOL.id, g0)
mc.setBlock(9, 2, 9, block.WOOL.id, b0)

```

Jetzt werden zwischen den ersten beiden Weihnachtsmarkthütten auf der linken Seite drei farbige Blöcke aus dem Material **WOOL** gebaut, die den drei ausgeschalteten Farbkomponenten der RGB-LED entsprechen.

```

try:
    while True:
        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit. pos.y, hit.pos.z)

```

Die Hauptschleife des Programms wartet darauf, dass die Spielfigur mit dem Schwert auf einen Block schlägt, und speichert diesen Block im Objekt **bl**.

```

if bl.id == block.WOOL.id and bl.data == r0:
    mc.setBlock(9, 2, 7, block.WOOL.id, r1)
    GPIO.output(r, True)

```

Besteht der Block aus dem Material **WOOL** und hat er die in **r0** gespeicherte Farbe (Rot ausgeschaltet), wird an der gleichen Position ein hellroter Block gleichen Materials mit der Farbe **r1** (Rot eingeschaltet) angelegt. An jeder Position kann immer nur ein Block existieren; neue ersetzen ältere automatisch. Anschließend wird die rote Farbe der RGB-LED eingeschaltet.

```

if bl.id == block.WOOL.id and bl.data == r1:
    mc.setBlock(9, 2, 7, block.WOOL.id, r0)
    GPIO.output(r, False)

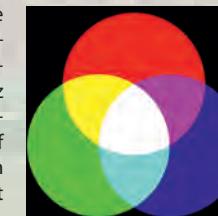
```

Schlägt die Spielfigur aber umgekehrt auf einen Block aus dem Material **WOOL** und der in **r1** gespeicherten Farbe (Rot eingeschaltet), wird an der gleichen Position ein dunkelroter Block gleichen Materials mit der Farbe **r0** (Rot ausgeschaltet) angelegt und die rote Farbe der RGB-LED ausgeschaltet.

Auf die gleiche Weise werden auch Schläge auf den grünen und den blauen Block ausgewertet und die entsprechenden Farben der RGB-LED ein- und ausgeschaltet.

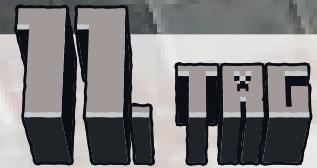
Additive Farbmischung

RGB-LEDs nutzen die sogenannte additive Farbmischung. Dabei werden die drei Lichtfarben Rot, Grün und Blau addiert und ergeben am Ende reines Weiß. Im Gegensatz dazu verwendet ein Farbdrucker die subtraktive Farbmischung. Jede Farbe wirkt auf einem weißen Blatt wie ein Filter, der einen Teil des weiß reflektierten Lichts wegnimmt (also subtrahiert). Drückt man alle drei Druckerfarben übereinander, ergibt es Schwarz, das gar kein Licht mehr reflektiert.



Für Ihre Notizen





Heute im Adventskalender

- 2 GPIO-Verbindungskabel

Hecken im Gelände bauen

Das Programm des 11. Tages baut eine Hecke aus zwei verschiedenen Pflanzenarten an der Stelle, wo die Figur gerade steht. Damit lässt sich eine Hecke durch das Gelände ziehen, indem man die Figur laufen lässt und die entsprechende Taste drückt.



Mit zwei Tasten lassen sich Hecken unterschiedlicher Art bauen.

Bauteile

- 1 Steckbrett
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 2 Taster
- 2 Drahtbrücken
- 6 GPIO-Verbindungskabel

Das Programm

Im Programm `11mc.py` bewegen Sie die Spielfigur wie gewohnt. Drücken Sie eine der Tasten, baut das Programm an der aktuellen Position der Spielfigur einen Block aus dem Material `LEAVES`, das über einen zusätzlichen Parameter verschiedene Farbschattierungen annehmen kann. Die beiden Tasten bauen Hecken unterschiedlicher Farben. Beim Tastendruck leuchtet eine der LEDs kurz auf. Nach dem Bau eines Blocks befindet sich die Spielfigur direkt in der Hecke. Auf diese Weise lassen sich Hecken unmittelbar aneinander anschließen, indem man mit der Spielfigur einfach weiterläuft.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
LED1 = 18
LED2 = 23
t1 = 8
t2 = 16

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED1, GPIO.OUT, initial=False)
GPIO.setup(LED2, GPIO.OUT, initial=False)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            mc.setBlock(p.x, p.y, p.z, block.LEAVES.id, 4)
            GPIO.output(LED1, True)
            time.sleep(0.1)
            GPIO.output(LED1, False)
        if GPIO.input(t2)==True:
            p = mc.player.getTilePos()
            mc.setBlock(p.x, p.y, p.z, block.LEAVES.id, 9)
            GPIO.output(LED2, True)
            time.sleep(0.1)
            GPIO.output(LED2, False)

except KeyboardInterrupt:
    GPIO.cleanup()
```

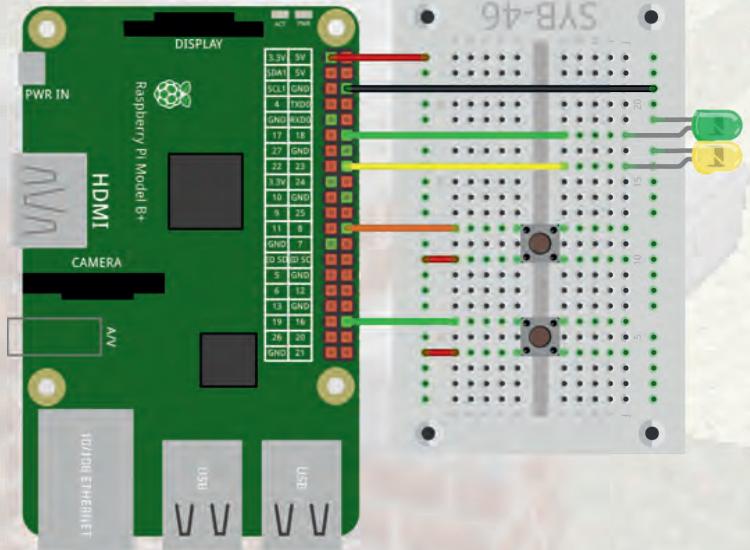
So funktioniert das Programm

Auch hier werden am Anfang Bibliotheken importiert und Variablen für die verwendeten GPIO-Pins angelegt.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED1, GPIO.OUT, initial=False)
GPIO.setup(LED2, GPIO.OUT, initial=False)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)
```

`LED1` und `LED2` enthalten die Nummern der verwendeten GPIO-Pins für die LEDs, `t1` und `t2` diejenigen der GPIO-Pins für die Taster. Die LEDs leuchten beim Bau eines Segments der Hecke kurz auf.

```
try:
    while True:
```



RGB-LED auf dem Steckbrett.

```
if GPIO.input(t1)==True:
    p = mc.player.getTilePos()
```

Die Endlosschleife fragt in jedem Durchlauf ab, ob einer der Taster gedrückt ist. Ist der Taster `t1` gedrückt, speichert das Programm die aktuelle Position der Spielfigur in dem Objekt `p`.

```
mc.setBlock(p.x, p.y, p.z, block.LEAVES.id, 4)
```

Jetzt wird ein Block aus dem Material `LEAVES` und der Farbe `4` an der Position gebaut, an der die Figur steht.

```
GPIO.output(LED1, True)
time.sleep(0.1)
GPIO.output(LED1, False)
```

Anschließend wird die LED1 für 0,1 Sekunden ein- und danach wieder ausgeschaltet. Diese Zeit wird gleichzeitig dafür verwendet, Tastenprellen zu vermeiden.

```
if GPIO.input(t2)==True:
    p = mc.player.getTilePos()
    mc.setBlock(p.x, p.y, p.z, block.LEAVES.id, 9)
    GPIO.output(LED2, True)
    time.sleep(0.1)
    GPIO.output(LED2, False)
```

Wird die Taste `t2` gedrückt, wird auf die gleiche Weise ein Block aus dem Material `LEAVES` und der Farbe `9` gebaut, wobei die LED2 kurz aufleuchtet.

12. TAG

Heute im Adventskalender

- 1 20-MOhm-Widerstand (rot-schwarz-blau)

Knetkontakte und Steve steuern die RGB-LED

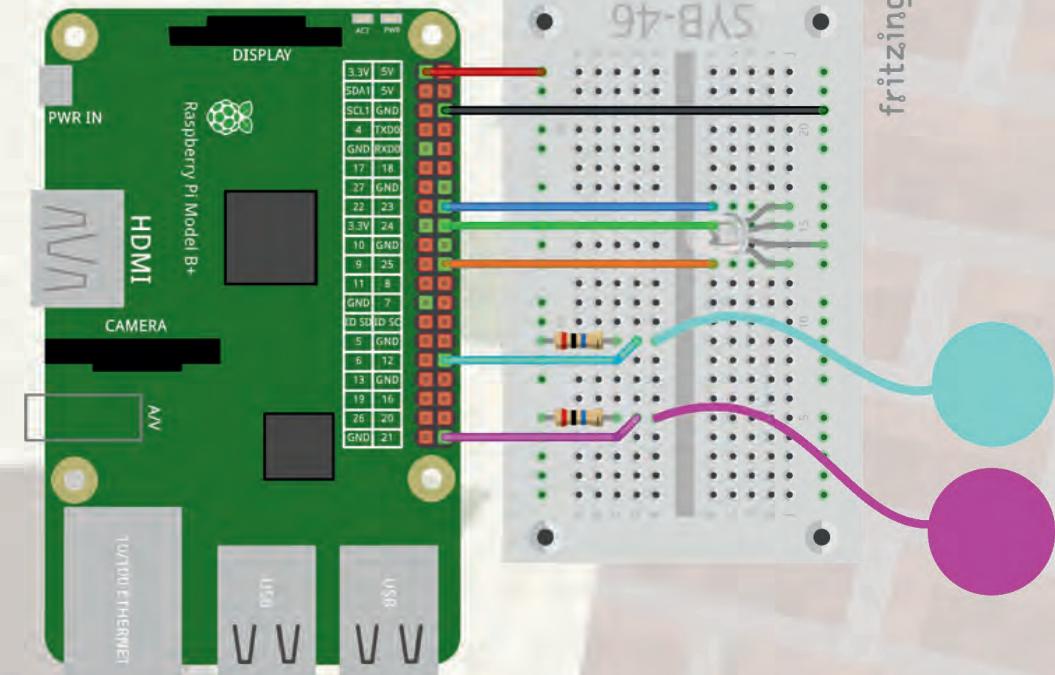
Im Experiment des 12. Tages steuern die Spielfigur Steve und zwei Knetkontakte die RGB-LED. Auf dem Weihnachtsmarkt befindet sich eine freie quadratische Fläche mit blauen und grünen Blöcken. Je weiter die Spielfigur in die blaue Richtung geht, desto heller leuchtet die blaue Farbe der RGB-LED. Je weiter die Spielfigur in die grüne Richtung geht, desto heller leuchtet die grüne Farbe der RGB-LED. Die rote Farbe lässt sich schrittweise über zwei Knetkontakte heller und dunkler einstellen.



Hier steuert die Spielfigur zwei Farben der RGB-LED.

Bauteile

- 1 Steckbrett
- 1 RGB-LED mit Vorwiderstand
- 2 20-MOhm-Widerstände (rot-schwarz-blau)
- 7 GPIO-Verbindungskabel
- 2 Knetkontakte



Zwei Sensorkontakte und RGB-LED.

Das Programm

Das Programm [12mc.py](#) fragt die Position der Spielfigur sowie die beiden Knetkontakte ab und stellt danach die Farben der RGB-LED ein.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
r = 25
g = 24
b = 23
k1 = 12
k2 = 21
x = 3
z = 8

GPIO.setmode(GPIO.BCM)
GPIO.setup(r, GPIO.OUT, initial=False)
GPIO.setup(g, GPIO.OUT, initial=False)
GPIO.setup(b, GPIO.OUT, initial=False)
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

rp = GPIO.PWM(r, 50)
gp = GPIO.PWM(g, 50)
bp = GPIO.PWM(b, 50)
rp.start(0)
gp.start(0)
bp.start(0)
pwm = 0

try:
    while True:
        p = mc.player.getTilePos()
        if p.x >= x-10 and p.x <= x and p.z >= z and p.z <= z+10:
            gp.ChangeDutyCycle((x-p.x)*10)
            bp.ChangeDutyCycle((p.z-z)*10)
        if GPIO.input(k1) == False and pwm<100:
            pwm += 10
            rp.ChangeDutyCycle(pwm)
            time.sleep(0.1)
        if GPIO.input(k2) == False and pwm>0:
            pwm -= 10
            rp.ChangeDutyCycle(pwm)
            time.sleep(0.1)

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

```
r = 25  
g = 24  
b = 23
```

Die Variablen **r**, **g** und **b** enthalten die Nummern der verwendeten GPIO-Pins für die RGB-LED.

```
k1 = 12  
k2 = 21
```

k1 und **k2** enthalten die Nummern der verwendeten GPIO-Pins für die Knetkontakte.

```
x = 3  
z = 8
```

x und **z** sind die Koordinaten der linken vorderen Ecke des durch die farbigen Blöcke beschriebenen Quadrats. Da die Fläche eben ist, spielt die y-Koordinate keine Rolle.

```
rp = GPIO.PWM(r, 50)  
gp = GPIO.PWM(g, 50)  
bp = GPIO.PWM(b, 50)  
rp.start(0)  
gp.start(0)  
bp.start(0)  
pwm = 0
```

Nachdem die GPIO-Pins initialisiert wurden, werden drei PWM-Objekte für die drei Farben angelegt und mit **0** gestartet. Die RGB-LED ist ausgeschaltet.

```
try:  
    while True:  
        p = mc.player.getTilePos()  
        if p.x >= x-10 and p.x <= x and p.z >= z and ↴  
        p.z <= z+10:
```

Die Endlosschleife fragt ständig die Position der Spielfigur ab und prüft, ob sich diese im Bereich der x-/z-Koordinaten der markierten Fläche befindet.

```
gp.ChangeDutyCycle((x-p.x)*10)  
bp.ChangeDutyCycle((p.z-z)*10)
```

Ist das der Fall, werden die PWM-Werte für Grün und Blau entsprechend den Koordinaten eingestellt. Die Fläche misst in beiden Achsen 10 Felder, und ein Rasterfeld entspricht 10 Einheiten, da der Maximalwert für PWM-Werte 100 beträgt. Die Berechnungsformeln unterscheiden sich, weil die Spielfigur von der vorderen linken Ecke in Richtung negativer x-, aber positiver z-Koordinaten läuft.

```
if GPIO.input(k1) == False and pwm<100:  
    pwm += 10  
    rp.ChangeDutyCycle(pwm)  
    time.sleep(0.1)
```

Wenn der Knetkontakt an GPIO-Pin **k1** berührt wird und der in der Variablen **pwm** gespeicherte PWM-Wert für Rot den Wert 100 noch nicht erreicht hat, wird er um 10 Einheiten erhöht, das PWM-Objekt **rp** entsprechend

geändert und 0,1 Sekunden gewartet, um versehentlich zu langes Drücken abzufangen.

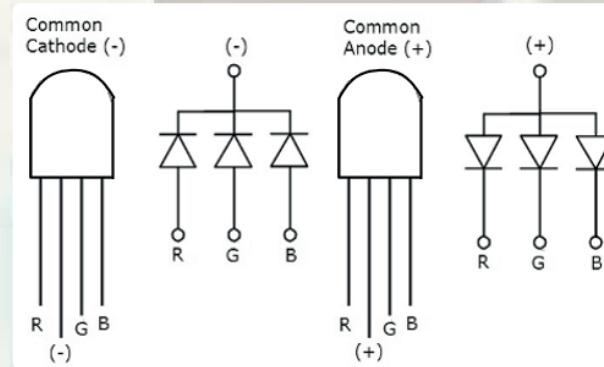
```
if GPIO.input(k2) == False and pwm>0:  
    pwm -= 10  
    rp.ChangeDutyCycle(pwm)  
    time.sleep(0.1)
```

Nach dem gleichen Schema wird beim Berühren des anderen Knetkontakte des PWM-Wert schrittweise bis auf 0 gesenkt.

Schon gewusst ...?

Es gibt zwei Arten von RGB-LEDs: gemeinsame Kathode (CC = Common Cathode) und gemeinsame Anode (CA = Common Anode). Im Adventskalender werden RGB-LEDs mit gemeinsamer Kathode verwendet. Bei diesen leuchten die einzelnen Farben, wenn am entsprechenden Pin ein High-Signal anliegt.

RGB-LEDs mit gemeinsamer Anode brauchen eine umgekehrte Programmierlogik. Die Anode wird an ein +5-V-Signal angeschlossen. Damit eine Farbe leuchtet, muss der entsprechende Pin (R, G oder B) mit Masse verbunden sein, also auf Low gesetzt werden. Pins, an denen ein High-Signal liegt, bewirken, dass die entsprechende Farbe nicht leuchtet.



Für Ihre Notizen



13. TAG

Heute im Adventskalender

- #### ■ 2 GPIO-Verbindungskabel

Verkehrs- und Fußgängerampel mit Knetekontakt

Das Experiment des 13. Tages steuert eine Verkehrs- und Fußgängerampel über einen Knetekontakt. Die Fußgängerampel wird über eine RGB-LED dargestellt. Zusätzlich leuchten Ampeln in der Minecraft-Welt.



Verkehrsampel links und Fußgängerampel rechts

Bauteile

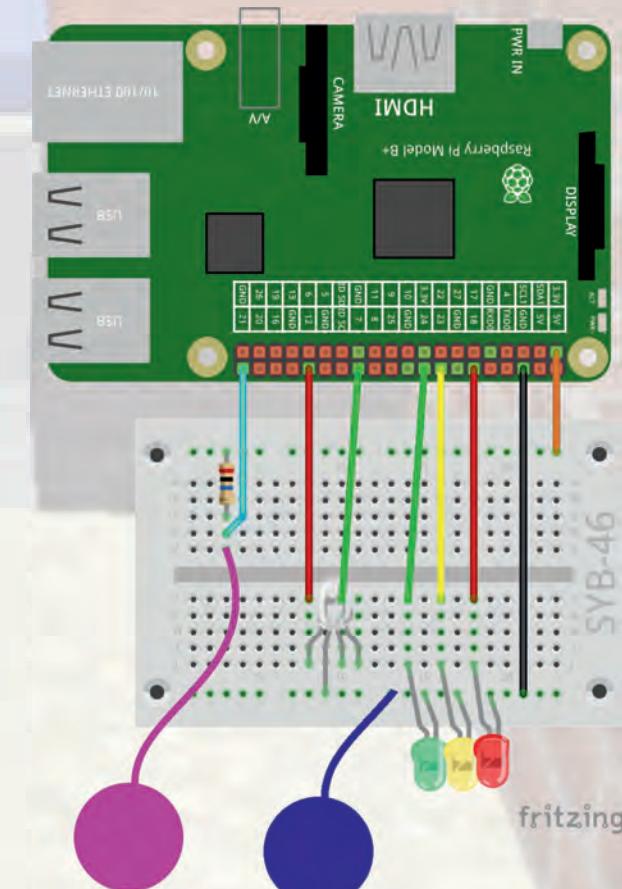
- 1 Steckbrett
 - 1 LED rot mit Vorwiderstand
 - 1 LED gelb mit Vorwiderstand
 - 1 LED grün mit Vorwiderstand
 - 1 RGB-LED mit Vorwiderstand
 - 1 20-MOhm-Widerstand (rot-schwarz-blau)
 - 8 GPIO-Verbindungskabel
 - 2 Knetekkontakte

Das Programm

Das Programm `13mc.py` steuert die Fußgängerampel über einen Knetekontakt.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
```

The image shows a Raspberry Pi Model B+ board connected to a breadboard. The breadboard is populated with three LEDs (red, green, yellow) and two large circular components (pink and blue). Colored wires connect the Pi pins to the breadboard. A Fritzing diagram is overlaid on the bottom right, showing the breadboard and the connections between the Pi pins and the LEDs.



Ampel aus drei LEDs und RGB-LED mit Knetekontakten auf dem Steckbrett. Der zweite Knetekontakt an der Masseschiene wird nicht in jedem Fall benötigt.

So funktioniert das Programm

```
r = 18  
o = 23  
g = 24  
t1 = 21  
rf = 12  
gf = 7
```

Auch hier werden am Anfang Bibliotheken importiert und Variablen angelegt. **r**, **o** und **g** enthalten die Nummern der verwendeten GPIO-Pins der Verkehrsampel (Rot, Orange, Grün). **t1** ist der GPIO-Pin für den Taster, die Variablen **rf** und **gf** enthalten die Nummern der verwendeten GPIO-Pins der Fußgängerampel.

```
r1 = 14  
o1 = 4  
g1 = 5  
sw = 15
```

Die Variablen **r1**, **o1** und **g1** enthalten die Farbnummern der eingeschalteten Blöcke der Ampeln (Rot, Orange, Grün). **sw** ist die Farbe Schwarz, die für alle ausgeschalteten Blöcke verwendet wird.

```
ax = 9  
az = 32  
fx = 6  
fz = 32
```

ax und **az** sind die Koordinaten der Verkehrsampel, **fx** und **fz** die Koordinaten der Fußgängerampel.

```
mc.setBlock(ax, 4, az, block.WOOL.id, sw)  
mc.setBlock(ax, 3, az, block.WOOL.id, sw)  
mc.setBlock(ax, 2, az, block.WOOL.id, g1)  
mc.setBlock(fx, 3, fz, block.WOOL.id, r1)  
mc.setBlock(fx, 2, fz, block.WOOL.id, sw)
```

Nachdem alle verwendeten GPIO-Pins als Ausgänge gesetzt wurden – wobei die grüne LED der Verkehrsampel und die rote LED der Fußgängerampel eingeschaltet werden –, werden noch zwei Blöcke auf den Ampeln farbig eingeschaltet, das Grün der Verkehrsampel und das Rot der Fußgängerampel. Die anderen Blöcke werden aus dem schwarzen Material der ausgeschalteten Lichter gebaut.

```
try:  
    while True:  
        if GPIO.input(t1)==False:
```

Die Endlosschleife wartet darauf, dass die Taste gedrückt wird. Wenn das geschieht, wird der Ampelzyklus ausgelöst.

```
mc.setBlock(ax, 3, az, block.WOOL.id, o1)  
GPIO.output(o, True)  
mc.setBlock(ax, 2, az, block.WOOL.id, sw)  
GPIO.output(g, False)  
time.sleep(0.6)
```

Jetzt bekommt der mittlere Block auf y-Höhe 3 der Verkehrsampel die Farbe **o1**. Er leuchtet orange. Das Gelände hat an dieser Stelle die Höhe 1, die Ampel liegt auf den Höhen 2 bis 4. Gleichzeitig wird die gelbe LED an GPIO-Pin **o** eingeschaltet. Unmittelbar danach wird der untere Block auf y-Höhe 2 auf die Farbe **sw** gesetzt und damit ausgeschaltet. Die grüne LED an GPIO-Pin **g** wird ebenfalls ausgeschaltet. Das Programm wartet 0,6 Sekunden bis zum nächsten Schaltzyklus.

```
mc.setBlock(ax, 4, az, block.WOOL.id, r1)  
GPIO.output(r, True)  
mc.setBlock(ax, 3, az, block.WOOL.id, sw)  
GPIO.output(o, False)  
time.sleep(0.6)
```

Nach dem gleichen Schema werden die rote Lampe der Verkehrsampel auf y-Höhe 4 und die rote LED eingeschaltet. Der gelbe Block und die gelbe LED werden im gleichen Schaltzyklus ausgeschaltet.

Zwischen allen Schaltzyklen wartet die Ampel 0,6 Sekunden. Nur wenn die grüne LED der Fußgängerampel und die rote LED der Verkehrsampel leuchten, dauert die Wartezeit 2 Sekunden.

Nachdem am Ende des Ampelzyklus die Verkehrsampel wieder auf Grün und die Fußgängerampel auf Rot steht, startet die Endlosschleife neu und wartet wieder darauf, dass die Taste gedrückt wird.

```
except KeyboardInterrupt:  
    mc.setBlock(ax, 2, az, block.WOOL.id, sw)  
    mc.setBlock(fx, 3, fz, block.WOOL.id, sw)  
    GPIO.cleanup()
```

Bricht man das Programm mit der Tastenkombination **Strg**+**C** ab, werden der grüne Block der Verkehrsampel und der rote der Fußgängerampel auf Schwarz gesetzt und die GPIO-Pins zurückgesetzt.

Der Ampelzyklus

Farbe(n) Verkehrsampel	Farbe Fußgängerampel	Zeit
Gelb	Rot	0.6 sek.
Rot	Rot	0.6 sek.
Rot	Grün	2.0 sek.
Rot	Rot	0.6 sek.
Rot/Gelb	Rot	0.6 sek.
Grün	Rot	2.0 sek.

Für Ihre Notizen





Heute im Adventskalender

- 1 RGB-LED mit Vorwiderstand

Zwei RGB-LEDs leuchten in zufälligen Farben

Laufen Sie mit der Spielfigur über das Spielfeld, leuchten zwei RGB-LEDs in unterschiedlichen Farben auf. Zusätzlich zeigt das Programm die aktuelle Position und die Position vor dem letzten Schritt im Minecraft-Fenster an. Lassen Sie jemanden versuchen, einen Punkt in der Spielwelt zu finden, an dem beide RGB-LEDs die gleiche Farbe zeigen. Das Ganze ist jedoch ein Scherzprogramm, bei dem sich die Farben der RGB-LEDs gar nicht beeinflussen lassen!



Die Spielfigur läuft durch die Minecraft-Welt und lässt die RGB-LEDs in unterschiedlichen Farben aufleuchten.

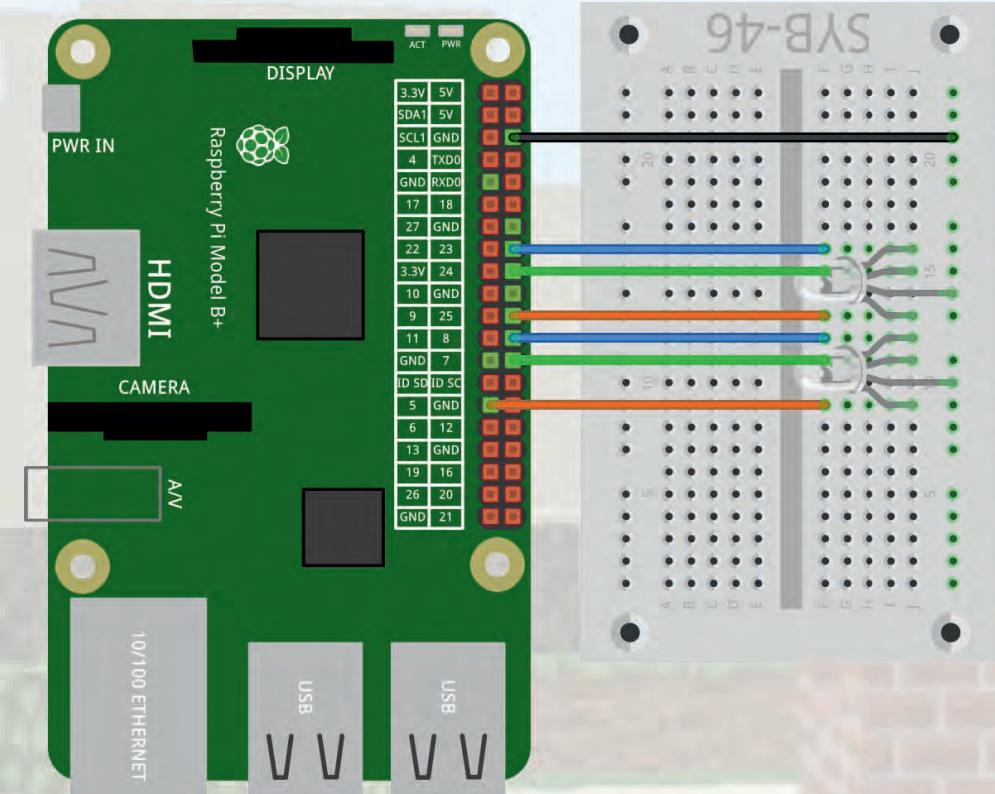
Bauteile

- 1 Steckbrett
- 2 RGB-LED mit Vorwiderstand
- 7 GPIO-Verbindungskabel

Das Programm

Das Programm `14mc.py` lässt die RGB-LEDs in unterschiedlichen Farben leuchten, wenn die Spielfigur sich bewegt.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import RPi.GPIO as GPIO
```



Zwei RGB-LEDs auf dem Steckbrett.

```
import random

mc = minecraft.Minecraft.create()
rgb1 = [25,24,23]
rgb2 = [5,7,8]
x = 0
z = 0

GPIO.setmode(GPIO.BCM)
for i in range(3):
    GPIO.setup(rgb1[i], GPIO.OUT, initial=False)
    GPIO.setup(rgb2[i], GPIO.OUT, initial=False)

try:
    while True:
        p = mc.player.getTilePos()
        if p.x != x:
            i = random.randrange(3)
            GPIO.output(rgb1[i],True)
            i = random.randrange(3)
            GPIO.output(rgb1[i],False)
            i = random.randrange(3)
            GPIO.output(rgb2[i],True)
            i = random.randrange(3)
            GPIO.output(rgb2[i],False)
```

```
mc.postToChat("p.x:"+str(p.x)+" x:"+
"+str(x)+" p.z:"+str(p.z)+" z:"+str(z))
x = p.x
if p.z != z:
    i = random.randrange(3)
    GPIO.output(rgb2[i],True)
    i = random.randrange(3)
    GPIO.output(rgb2[i],False)
    mc.postToChat("p.x:"+str(p.x)+" x:"+
"+str(x)+" p.z:"+str(p.z)+" z:"+str(z))
    z = p.z

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

```
import time, random
```

Bei der Initialisierung wird zusätzlich das Modul `random` importiert, das die Möglichkeit bietet, Zufallszahlen zu erzeugen.

Wie entstehen Zufallszahlen?

Gemeinhin denkt man, in einem Programm könne nichts zufällig geschehen – wie also kann ein Programm dann in der Lage sein, zufällige Zahlen zu generieren? Teilt man eine große Primzahl durch einen beliebigen Wert, ergeben sich ab der x-ten Nachkommastelle Zahlen, die kaum noch vorhersehbar sind. Sie ändern sich auch ohne jede Regelmäßigkeit, wenn man den Divisor regelmäßig erhöht. Dieses Ergebnis ist zwar scheinbar zufällig, lässt sich aber durch ein identisches Programm oder den mehrfachen Aufruf des gleichen Programms jederzeit reproduzieren. Nimmt man jetzt aber eine aus einigen dieser Ziffern zusammengebaute Zahl und teilt sie wiederum durch eine Zahl, die sich aus der aktuellen Uhrzeitsekunde oder dem Inhalt einer beliebigen Speicherstelle des Computers ergibt, kommt ein Ergebnis heraus, das sich nicht reproduzieren lässt und daher als Zufallszahl bezeichnet wird.

```
rgb1 = [25,24,23]
rgb2 = [5,7,8]
```

Die Listen `rgb1` und `rgb2` enthalten die Nummern der GPIO-Pins für die beiden RGB-LEDs. Diese werden als Ausgänge definiert und ausgeschaltet.

```
x = 0
z = 0
```

Die Variablen `x` und `z` werden am Anfang auf 0 gesetzt. Hier werden im Laufe des Spiels die letzten Koordinaten der Spielfigur gespeichert.

```
try:
    while True:
        p = mc.player.getTilePos()
```

Die Endlosschleife speichert in jedem Durchlauf die Position der Spielfigur.

```
if p.x != x:
    i = random.randrange(3)
    GPIO.output(rgb1[i],True)
    i = random.randrange(3)
    GPIO.output(rgb1[i],False)
```

Hat sich die Spielfigur in x-Richtung bewegt, ist also die aktuelle x-Koordinate anders als die letzte gespeicherte Koordinate, wird eine zufällig gewählte Farbe der RGB-LED `rgb1` eingeschaltet und eine ebenfalls zufällig gewählte Farbe der gleichen RGB-LED ausgeschaltet.

Die Funktion `random.randrange(3)` generiert eine ganzzahlige Zufallszahl kleiner als 3, kann also die Werte `0`, `1` oder `2` ausgeben. Das verwendete Verfahren kann zufällig die drei Grundfarben Rot, Grün und Blau, die Mischfarben Gelb (Rot + Grün), Cyan (Grün + Blau) und Violett (Blau + Rot) sowie Weiß als Mischung aller drei Farben erzeugen. Außerdem ist es möglich, dass die RGB-LED ganz ausgeschaltet wird oder zweimal hintereinander die gleiche Farbe zeigt.

```
mc.postToChat("p.x:"+str(p.x)+" x: " + str(x) + " p.z:"+str(p.z)+" z:"+str(z))
```

Die Funktion `mc.postToChat()` schreibt einen beliebigen Text in den Chatbereich auf dem Bildschirm. Bei weiteren Ausgaben scrollen die älteren nach oben und verschwinden nach einiger Zeit automatisch.

Das Programm zeigt hier nach jeder Bewegung die aktuellen und die zuletzt gespeicherten Koordinaten an. Die anzuzeigende Zeichenkette wird mit dem Operator `+` aus mehreren Zeichenketten zusammengesetzt. Zahlenwerte werden zuvor mit der Funktion `str()` in Zeichenketten umgewandelt, da die Zahlen nicht addiert, sondern hintereinander ausgegeben werden sollen.

`x = p.x`

Zum Schluss wird die aktuelle x-Koordinate der Spielfigur in der Variablen `x` gespeichert.

Nach dem gleichen Schema ändert sich die Farbe der zweiten RGB-LED zufällig, wenn sich die Spielfigur in z-Richtung bewegt.

Schon gewusst ...?

Minecraft™ ist in verschiedenen Editionen für zahlreiche Hardwareplattformen erhältlich.

Edition	Plattform	Aktuelle Version
Windows 10	Windows 10	1.0.5
Java	Windows	1.11.2
Java	Mac	1.11.2
Java	Linux	1.11.2
Konsole	Xbox One	CU41
Konsole	Xbox 360	TU51
Konsole	Playstation 3	1.44
Konsole	Playstation 4	1.44
Konsole	Playstation Vita	1.44
Konsole	Wii U	Patch 20
Pocket	Android	1.0.5
Pocket	iOS	1.0.5
Pocket	Windows Phone	1.0.5
Pocket	Kindle Fire	1.0.5
Pocket (VR)	Gear VR	1.0.5
Pocket	Apple TV	1.0.5
Pi	Raspberry Pi	0.1.1

Für Ihre Notizen



15. Tag

Heute im Adventskalender

- 1 Stück Knete

Sensorkontakte steuern den Farbverlauf von RGB-LEDs

Zwei Sensorkontakte lassen auf einer RGB-LED einen Farbverlauf mit 100 verschiedenen Mischfarben erscheinen. Mit der Knete der zweiten Farbe können Sie den Massekontakt (falls Sie ihn benötigen) besser unterscheidbar machen.



Farbiges Licht in einer Weihnachtsmarkthütte.

Bauteile

- 1 Steckbrett
- 2 RGB-LEDs mit Vorwiderstand
- 2 20-MOhm-Widerstände (rot-schwarz-blau)
- 7 GPIO-Verbindungskabel
- 3 Knetkontakte

HSV- und RGB-Farbsystem

Das RGB-Farbsystem, das bisher in allen Programmen verwendet wurde, beschreibt Farben als Mischung der drei Bestandteile Rot, Grün und Blau. Dabei ist es für einen Menschen relativ schwierig, sich eine Mischfarbe vorzustellen. Im Gegensatz dazu beschreibt das HSV-Farbsystem die Farben durch Farbwert (Hue), Sättigung (Saturation) und Helligkeitswert (Value).

Durch einfache Veränderung des H-Wertes können alle Farben des Farbspektrums in voller Intensität beschrieben werden, wenn man die beiden anderen Werte auf Maximum einstellt.

Das Programm

Im Programm `15mc.py` werden über zwei Knetkontakte verschiedene Farbwerte des Spektrums von der RGB-LED angezeigt. Ein farbiges Licht in einer der Weihnachtsmarkthütten wechselt ebenfalls die Farbe, allerdings nicht kontinuierlich, sondern nur in fünf Stufen.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time, colorsys

mc = minecraft.Minecraft.create()
GPIO.setmode(GPIO.BCM)
LED = [25,24,23]
p = [0,0,0]
h = 0
k1 = 12
k2 = 21
x = 10
y = 3
z = 14
c = [3,5,4,1,6]
```

```
for i in range(3):
    GPIO.setup(LED[i], GPIO.OUT)

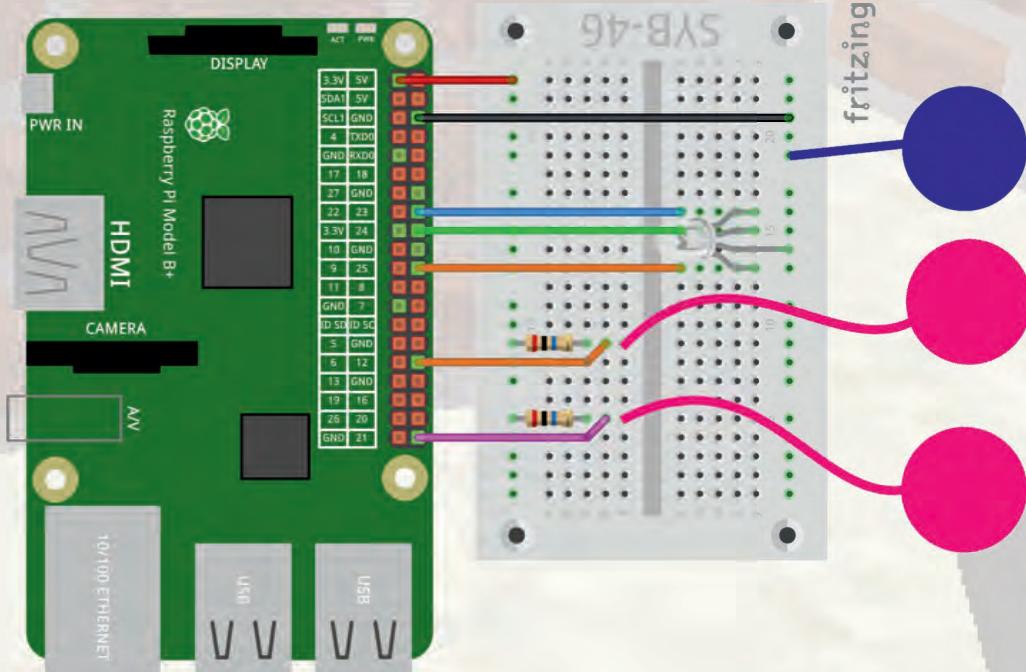
for i in range(3):
    p[i] = GPIO.PWM(LED[i], 50)
    p[i].start(0)

GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)

mc.setBlock(x,y,z,block.WOOL.id,c[0])

try:
    while True:
        if GPIO.input(k1) == False and h<95:
            h += 5
        if GPIO.input(k2) == False and h>0:
            h -= 5
        rgb = colorsys.hsv_to_rgb(h/100, 1, 1)
        for i in range(3):
            p[i].ChangeDutyCycle(rgb[i])
        mc.setBlock(x,y,z,block.WOOL.id,c[int(h/20)])
        time.sleep(0.1)

except KeyboardInterrupt:
    for i in range(3):
        p[i].stop()
```



```
mc.setBlock(x,y,z,block.AIR.id)
GPIO.cleanup()
```

So funktioniert das Programm

```
import time, colorsys
```

Zusätzlich zu den bereits bekannten Bibliotheken wird die Bibliothek `colorsys` zur Umrechnung zwischen HSV- und RGB-Farbsystem importiert.

```
LED = [25,24,23]
p = [0,0,0]
```

Die Liste `LED[]` enthält die Pin-Nummern der drei Farbkomponenten der RGB-LED, die Liste `p[]` drei PWM-Objekte dafür.

```
x = 10
y = 3
z = 14
c = [3,5,4,1,6]
```

Die Variablen `x, y, z` enthalten die Koordinaten des farbigen Blocks in der Minecraft-Welt, die Liste `c[]` die fünf verwendeten Farbnummern des Materials `WOOL`.

```
for i in range(3):
    p[i] = GPIO.PWM(LED[i], 50)
    p[i].start(0)
```

Die PWM-Signale werden mit der Frequenz 50 Hz eingerichtet und mit dem Wert 0 gestartet. Alle drei Farben der RGB-LED sind ausgeschaltet.

```
mc.setBlock(x,y,z,block.WOOL.id,c[0])
```

Der farbige Block wird auf die Farbe `c[0]` gesetzt.

```
try:
    while True:
        if GPIO.input(k1) == False and h<95:
            h += 5
        if GPIO.input(k2) == False and h>0:
            h -= 5
```

Die Farben der RGB-LED werden zunächst im HSV-System erzeugt. Wird der Knetekontakt `k1` berührt, wird der H-Wert um 5 erhöht; wird der Knetekontakt `k2` berührt, so wird er um 5 gesenkt.

```
rgb = colorsys.hsv_to_rgb(h/100, 1, 1)
```

Zur Umrechnung in das RGB-System für die RGB-LEDs wird die Funktion `colorsys.hsv_to_rgb()` verwendet. Sie benötigt drei Parameter im Bereich zwischen 0 und 1. Deshalb wird der H-Wert durch 100 dividiert. Die Werte S und V für Farbsättigung und Helligkeit stehen auf Maximum. Die Funktion gibt eine Liste aus drei Werten zurück, die in der Variablen `rgb` gespeichert wird.

```
for j in range(3):
    p[j].ChangeDutyCycle(rgb[j])
```

Eine Schleife legt die PWM-Signale der drei Farbkomponenten für die errechneten RGB-Werte fest.

```
mc.setBlock(x,y,z,block.WOOL.id,c[int(h/20)])
time.sleep(0.1)
```

Die Funktion `mc.setBlock()` setzt die Farbe des Blocks in der Weihnachtsmarkthütte entsprechend dem eingestellten H-Wert auf eine der fünf Farben aus der Liste `c[]`. Danach wartet das Programm 0,1 Sekunden, bevor die nächsten Farben errechnet werden.

```
except KeyboardInterrupt:
    for i in range(3):
        p[i].stop()
    mc.setBlock(x,y,z,block.AIR.id)
    GPIO.cleanup()
```

Bricht der Benutzer mit der Tastenkombination `Strg + C` das Programm ab, werden die PWM-Signale gestoppt und die GPIO-Pins zurückgesetzt.

Farbnummern für Blöcke vom Typ WOOL

Nummer	Farbe
0	Weiß
1	Orange
2	Magenta
3	Hellblau
4	Gelb
5	Lime
6	Pink
7	Grau
8	Hellgrau
9	Cyan
10	Lila
11	Blau
12	Braun
13	Grün
14	Rot
15	Schwarz

Für Ihre Notizen




```

for i in range(3):
    p[i].stop()
    mc.setBlocks(x,y,z[i],x,y,z[i]+1,block.AIR.id)
GPIO.cleanup()

```

So funktioniert das Programm

Dieses Programm verwendet mehr Variablen als die letzten Programme. Auch hier werden am Anfang Bibliotheken importiert und Variablen angelegt.

```

LED1 = [25,24,23]
LED2 = [5,7,8]
p = [0,0,0]
kr = 21
kg = 16
kb = 12

```

Die Listen `LED1[]` und `LED2[]` enthalten die Nummern der verwendeten GPIO-Pins für die beiden RGB-LEDs, `p[]` die drei PWM-Signale und `kr`, `kg` und `kb` die Nummern der GPIO-Pins für die Knetekkontakte.

```

r = 0
g = 0
b = 0

```

Die Variablen `r`, `g` und `b` enthalten die PWM-Werte für die drei Farben der RGB-LED und sind zu Anfang alle gleich 0.

```

x = 10
y = 3
z = [13,18,23]

```

Die Lage der Blöcke wird ebenfalls in Variablen gespeichert. Alle Farben haben die gleichen x- und y-Koordinaten, aber die drei verschiedenen z-Koordinaten werden in einer Liste gespeichert.

```
c = [14,5,11,15]
```

Die Liste `c[]` enthält die Farben der drei eingeschalteten Blöcke sowie Schwarz für ausgeschaltete Blöcke.

```

for i in range(3):
    GPIO.setup(LED1[i], GPIO.OUT)
    GPIO.setup(LED2[i], GPIO.OUT)

for i in range(3):
    p[i] = GPIO.PWM(LED1[i], 50)
    p[i].start(0)

```

Nach der Definition der Variablen werden die GPIO-Pins initialisiert, und für jede Farbe der RGB-LED `LED1[]` wird ein PWM-Objekt gestartet.

```

for i in range(3):
    mc.setBlocks(x,y,z[i],x,y,z[i]+1,block.WOOL. ↴
id, c[3])

```

Die drei farbigen Blöcke in den Weihnachtsmarkthütten werden auf Schwarz gesetzt. Da jeder dieser Blöcke aus zwei nebeneinanderliegenden Minecraft-Klötzten besteht, wird die Funktion `mc.setBlocks()` verwendet.

```

try:
    while True:
        if GPIO.input(kr) == False:
            if r < 10:
                r += 1
            else:
                r = 0

```

Die Endlosschleife prüft als Erstes, ob der rote Knetek kontakt berührt wurde. Hat die Variable `r` den Maximalwert 10 noch nicht erreicht, wird sie um 1 erhöht. Beim nächsten Antippen des Knetekkontakte nach dem Wert 10 wird sie auf 0 zurückgesetzt.

```
p[0].ChangeDutyCycle(r*10)
```

Jetzt wird der PWM-Wert der roten Farbe der RGB-LED auf das Zehnfache des eingestellten Wertes gesetzt. Die durch die Knetekkontakte generierten Werte bewegen sich zwischen 0 und 10, PWM-Werte liegen dagegen im Bereich von 0 bis 100.

```

mc.setBlocks(x,y,z[0],x,y,z[0]+1,block. ↴
WOOL.id, c[0])
mc.setBlocks(x,y,z[1],x,y,z[1]+1,block. ↴
WOOL.id, c[3])
mc.setBlocks(x,y,z[2],x,y,z[2]+1,block. ↴
WOOL.id, c[3])

```

Der rote Block in der ersten Weihnachtsmarkthütte wird eingeschaltet, da der Knetek kontakt der roten Farbkomponente berührt wurde. Die anderen beiden Blöcke werden ausgeschaltet.

```

GPIO.output(LED2[0],True)
time.sleep(0.1)
GPIO.output(LED2[0],False)

```

Zusätzlich keuchtet die RGB-LED `LED2[]` für 0,1 Sekunden rot. Diese Zeit wird auch verwendet, um zu lange Berührungen des Knetekkontakte nicht mehrfach auszuwerten. Bei Bedarf können Sie diese Zeit auch verlängern.

Nach dem gleichen Schema werten zwei weitere `if`-Abfragen die Knetekkontakte für die grüne und blaue Farbe aus.

Wenn der Benutzer das Programm mit `Strg+C` beendet, werden die PWM-Signale beendet, die farbigen Blöcke gelöscht und die GPIO-Ports geschlossen.

Für Ihre Notizen





Heute im Adventskalender

- 1 Piezo-Summer

Töne mit dem Piezo-Summer

Der im Paket enthaltene Piezo-Summer macht elektrische Schwingungen hörbar. Legt man eine pulsierende Gleichspannung an den beiden Polen des Summers an, wird dieser in Schwingungen versetzt. Piezo-Summer werden ähnlich wie gedimmte LEDs mit PWM-Signalen angesteuert. Je nach Frequenz sind einzelne Klicks oder ein durchgängiger Ton zu hören. Frequenzen von wenigen Hertz (Schwingungen pro Sekunde) hört das menschliche Ohr noch als einzelne Töne, Frequenzen zwischen etwa 20 Hertz und 16 Kilohertz werden als durchgehender Ton unterschiedlicher Höhe wahrgenommen.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 2 RGB-LEDs mit Vorwiderstand
- 1 Piezo-Summer
- 7 GPIO-Verbindungskabel
- 1 Drahtbrücke

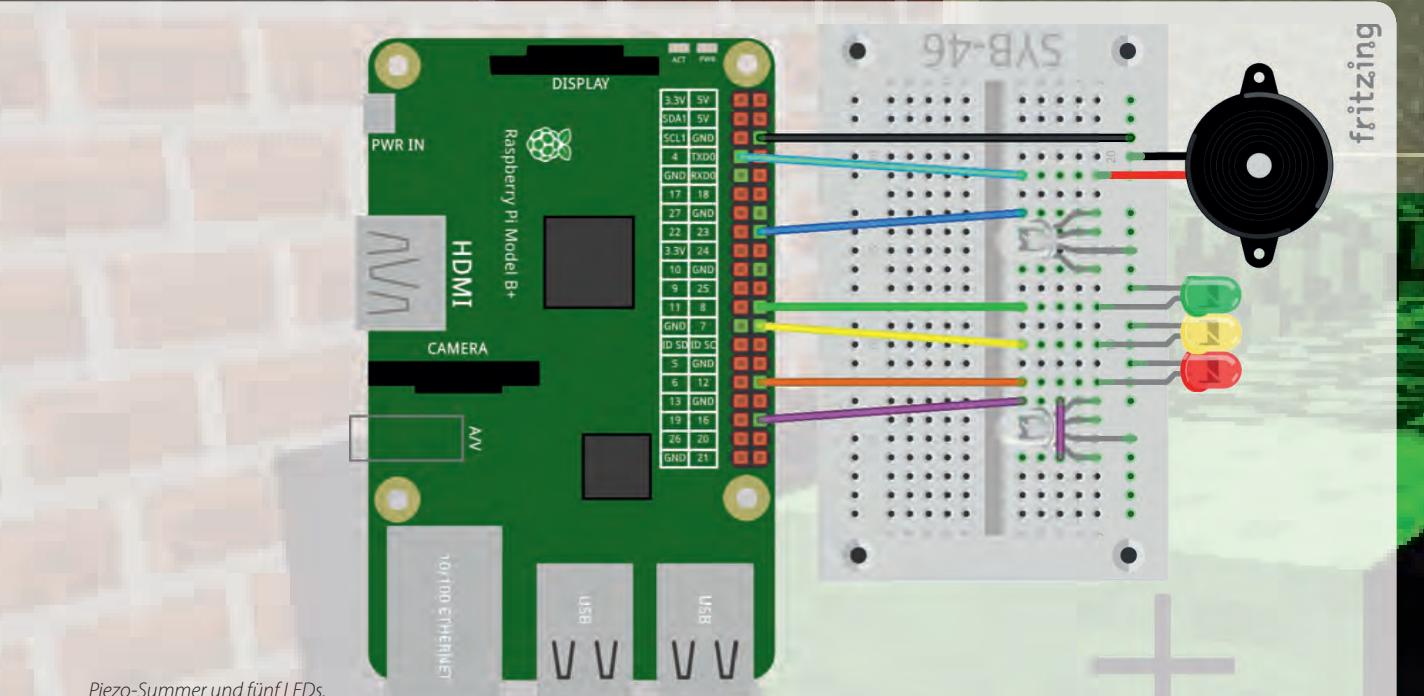
Zwei RGB-LEDs werden zur Anzeige der Farben Lila und Blau verwendet, Farben, für die bis jetzt keine einzelnen LEDs zur Verfügung stehen. Bei der RGB-LED links sind die Anschlüsse für Rot und Blau mit einer Drahtbrücke verbunden, damit sie lila leuchtet, bei der RGB-LED rechts ist nur der Kontakt für Blau angeschlossen.

Das Programm

Das Programm `17mc.py` baut fünf Blöcke in einem Schwarz-Weiß-Muster. Schlägt die Spielfigur mit dem Schwert auf einen davon, leuchtet dieser farbig auf und erzeugt einen Ton. Zusätzlich leuchtet die LED in der entsprechenden Farbe.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

mc = minecraft.Minecraft.create()
LED = [16,12,7,8,23]
```



Piezo-Summer und fünf LEDs.



Fünf Blöcke, die verschiedene Töne erzeugen.

```
pi = 4
TON = [261,293,329,369,415]

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
    GPIO.setup(pi, GPIO.OUT, initial=False)
    p = GPIO.PWM(pi, 1)
```



Beim Anschlagen eines Blocks leuchten dieser und die zugehörige LED farbig auf.

```
x1 = 9
y1 = 2
z1 = 7
c1 = [3,5,4,1,6]
c0 = [15,0,15,0,15]

for i in range(5):
    mc.setBlock(x1, y1, z1+i, block.WOOL.id, c0[i])
```

```

try:
    while True:
        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit.→
pos.y, hit.pos.z)
            for i in range(5):
                if hit.pos.x == xl and hit.pos.z == zl+i:
                    mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c1[i])
                    GPIO.output(LED[i],True)
                    p.ChangeFrequency(TON[i])
                    p.start(1)
                    time.sleep(0.2)
                    mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c0[i])
                    GPIO.output(LED[i],False)
                    p.stop()

except KeyboardInterrupt:
    mc.setBlocks(xl, yl, zl, xl, yl, zl+4, block.→
AIR.id)
    p.stop()
    GPIO.cleanup()

```

So funktioniert das Programm

```

LED = [16,12,7,8,23]
pi = 4
TON = [261,293,329,369,415]

```

Nachdem die Bibliotheken importiert sind, werden die fünf GPIO-Pins der LEDs in einer Liste gespeichert und die für den Piezo-Summer in einer einzelnen Variable abgelegt. Die Liste `TON` enthält die fünf Töne, die beim Schlag auf die Klötze abgespielt werden. Die Pins werden als Ausgänge initialisiert und ausgeschaltet.

```
p = GPIO.PWM(pi, 1)
```

Danach wird ein PWM-Signal auf dem GPIO-Pin des Piezo-Summers eingerichtet.

```

xl = 9
yl = 2
zl = 7
c1 = [3,5,4,1,6]
c0 = [15,0,15,0,15]

```

Die Variablen `xl`, `yl` und `zl` geben die Position des ersten der fünf Blöcke an. Die beiden Listen `c1[]` und `c0[]` enthalten die Farben der fünf Blöcke im eingeschalteten und ausgeschalteten Zustand.

```
for i in range(5):
    mc.setBlock(xl, yl, zl+i, block.WOOL.id, c0[i])
```

Eine Schleife baut die fünf Blöcke abwechselnd schwarz und weiß nach den Farben aus der Liste `c0[]`, die den ausgeschalteten LEDs entsprechen.

```

try:
    while True:
        for hit in mc.events.pollBlockHits():
            bl = mc.getBlockWithData(hit.pos.x, hit.→
hit.pos.y, hit.pos.z)
            for i in range(5):
                if hit.pos.x == xl and hit.pos.z == zl+i:
                    mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c1[i])
                    GPIO.output(LED[i],True)
                    p.ChangeFrequency(TON[i])
                    p.start(1)
                    time.sleep(0.2)
                    mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c0[i])
                    GPIO.output(LED[i],False)
                    p.stop()

except KeyboardInterrupt:
    mc.setBlocks(xl, yl, zl, xl, yl, zl+4, block.→
AIR.id)
    p.stop()
    GPIO.cleanup()

```

Die Hauptschleife des Programms wartet nach bekanntem Schema darauf, dass die Spielfigur auf einen Block schlägt, und speichert diesen dann im Objekt `bl`.

```

for i in range(5):
    if hit.pos.x == xl and hit.pos.z == zl+i:
        mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c1[i])
        GPIO.output(LED[i],True)

        p.ChangeFrequency(TON[i])
        p.start(1)

        time.sleep(0.2)
        mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c0[i])
        GPIO.output(LED[i],False)
        p.stop()

```

Als Nächstes wird geprüft, ob der getroffene Block einer der zuvor gebauten fünf Blöcke ist. Eine Schleife prüft die x-Koordinate, die bei allen diesen Blöcken gleich ist, sowie nacheinander die fünf z-Koordinaten.

```

mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c1[i])
GPIO.output(LED[i],True)

p.ChangeFrequency(TON[i])
p.start(1)

```

Handelt es sich um einen der Blöcke, wird er durch einen Block aus dem der Blocknummer entsprechenden farbig leuchtenden Material aus der Liste `c1[]` überschrieben. Zusätzlich wird die zugehörige LED eingeschaltet.

```

p.ChangeFrequency(TON[i])
p.start(1)

```

Die Frequenz des PWM-Signals wird auf die in der Liste `TON[]` für den angeschlagenen Block gespeicherte Frequenz gesetzt und das PWM-Signal gestartet. Dadurch gibt der Piezo-Summer einen Ton aus.

```

time.sleep(0.2)
mc.setBlock(xl, yl, zl+i, block.WOOL.→
id, c0[i])
GPIO.output(LED[i],False)
p.stop()

```

Das Programm wartet 0,2 Sekunden. Danach wird alles zurückgesetzt. Der Block erscheint wieder im ausgeschalteten Zustand, die LED wird ausgeschaltet und das PWM-Signal gestoppt.

```

except KeyboardInterrupt:
    mc.setBlocks(xl, yl, zl, xl, yl, zl+4, block.→
AIR.id)
    p.stop()
    GPIO.cleanup()

```

Wenn der Benutzer das Programm mit `[Strg]+[C]` beendet, werden die fünf Blöcke gelöscht, das PWM-Signal beendet und die GPIO-Ports geschlossen.

Hörbare Töne

Der tiefste hörbare Ton liegt bei 16 bis 20 Hz, der höchste hörbare Ton hängt vom Lebensalter des Menschen ab. Bei Kleinkindern liegt er etwa bei 20 kHz, bei alten Menschen unter 10 kHz, wobei es allerdings von Mensch zu Mensch große Schwankungen gibt.

Typische Frequenzen

Schallquelle	Frequenzbereich (Hz)
Menschliche Stimme	80 - 1200
Klavier	27,5 - 4186
Violine	196 - 3726
Flöte	267 - 4698
Basstuba	43 - 349
Klassischer Telefonwählton	425
Telefontastentöne (MFV)	697 - 1633
Wechselstrombrummen	50
Hundepfeife	16.000 - 22.000

Für Ihre Notizen



18. TAG

Heute im Adventskalender

- 2 GPIO-Verbindungskabel

Ein neues Haus auf dem Weihnachtsmarkt

Mit einem Knopfdruck bauen Sie weitere Hütten auf dem Weihnachtsmarkt, anstatt sie mühsam aus einzelnen Klötzen zusammenzusetzen.

Bauteile

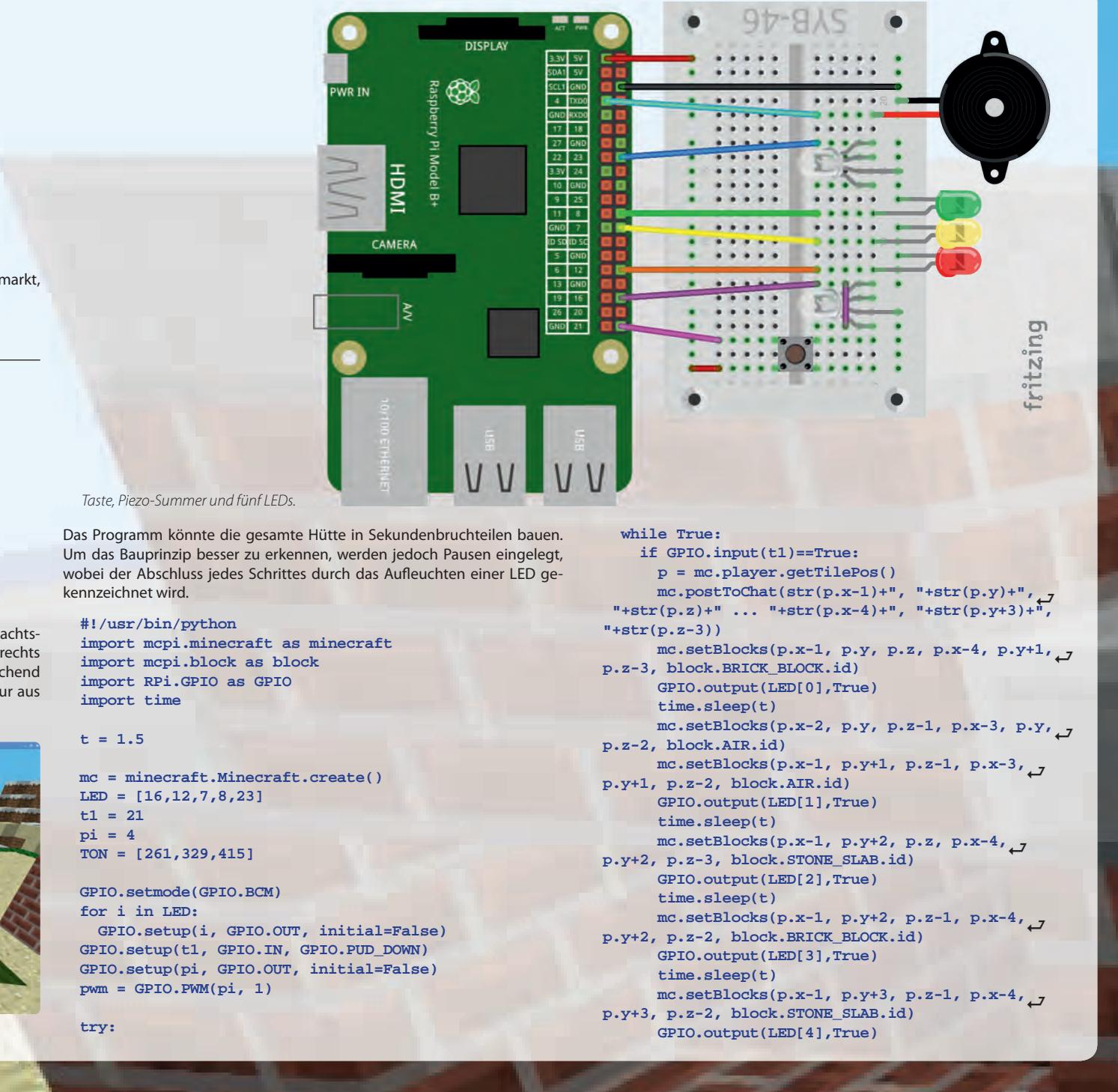
- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 2 RGB-LEDs mit Vorwiderstand
- 1 Taster
- 1 Piezo-Summer
- 9 GPIO-Verbindungskabel
- 2 Drahtbrücken

Das Programm

Das Programm [18mc.py](#) errichtet auf Tastendruck eine neue Weihnachtsmarkthütte direkt vor der Spielfigur. Das Programm ist dafür gedacht, rechts des Weges, vom Eingang aus gesehen, Hütten zu bauen. Dementsprechend sind die Koordinaten in Richtung der negativen x-Achse von der Figur aus ausgerichtet.



Neue Weihnachtsmarkthütten auf Knopfdruck bauen.



```

for i in TON:
    pwm.ChangeFrequency(i)
    pwm.start(1)
    time.sleep(0.2)
    pwm.stop()
    time.sleep(t)
    for i in LED:
        GPIO.output(i,False)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert das Programm

`t = 1.5`

Die Variable `t` legt die Wartezeit zwischen den einzelnen Bauschritten fest. Diese wird technisch nicht benötigt, sondern dient nur dazu, den Baufortschritt mitverfolgen zu können. Anschließend werden nacheinander nach bekanntem Schema die GPIO-Pins für LEDs, Taster, Piezo-Summer sowie das PWM-Signal eingerichtet.

```

try:
    while True:
        if GPIO.input(t1)==True:

```

Die Hauptschleife des Programms wartet, bis der Taster gedrückt wird.

```

p = mc.player.getTilePos()
mc.postToChat(str(p.x-1)+" , "+str(p.y)+" , "
"+str(p.z)+" ... "+str(p.x-4)+" , "+str(p.y+3)+" , "
"+str(p.z-3))

```

Jetzt wird die Position der Spielfigur in der Variablen `p` gespeichert. Daraus ergeben sich die Koordinaten der zu bauenden Hütte, die im Chat angezeigt werden.

```

mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ←
p.z-3, block.BRICK_BLOCK.id)
GPIO.output(LED[0],True)
time.sleep(t)

```

Im ersten Schritt werden zwei massive Lagen Blöcke aus dem Ziegelmaterial `BRICK_BLOCK` gebaut. Dabei leuchtet die erste LED.

```

mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ←
p.z-2, block.AIR.id)
mc.setBlocks(p.x-1, p.y+1, p.z-1, p.x-3, ←
p.y+1, p.z-2, block.AIR.id)
GPIO.output(LED[1],True)
time.sleep(t)

```

Mit dem Material `AIR` wird in der unteren Ebene eine quadratische Aussparung und in der oberen Ebene eine größere angelegt, die das Fenster in das Gebäude schneidet. Dabei leuchtet die zweite LED auf.

```

mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4, ←
p.y+2, p.z-3, block.STONE_SLAB.id)
GPIO.output(LED[2],True)
time.sleep(t)

```

Um nicht die beiden Dachhälften in der unteren der beiden Dachebenen einzeln bauen zu müssen, wird im nächsten Bauschritt zunächst eine durchgehende Flachdachplatte in der Ebene `y+2` des Gebäudes angelegt. In diesem Schritt leuchtet die dritte LED auf.

```

mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4, ←
p.y+2, p.z-2, block.BRICK_BLOCK.id)
GPIO.output(LED[3],True)
time.sleep(t)

```

In der Mitte der `y+2`-Ebene des Gebäudes werden zwei Reihen Blöcke aus dem Ziegelmaterial `BRICK_BLOCK` gebaut, die die Flachdachblöcke in diesem Bereich ersetzen. In diesem Schritt leuchtet die vierte LED auf.

```

mc.setBlocks(p.x-1, p.y+3, p.z-1, p.x-4, ←
p.y+3, p.z-2, block.STONE_SLAB.id)
GPIO.output(LED[4],True)

```

In der Mitte der `y+3`-Ebene des Gebäudes werden wieder zwei Reihen Flachdachblöcke gesetzt, die den Bau abschließen.

```

for i in TON:
    pwm.ChangeFrequency(i)
    pwm.start(1)
    time.sleep(0.2)
    pwm.stop()

```

Zum Schluss erklingt eine Tonfolge aus dem Piezo-Summer. Die Frequenz des PWM-Signals nimmt nacheinander die in der Liste `TON[]` gespeicherten Werte an.

```

time.sleep(t)
for i in LED:
    GPIO.output(i,False)

```

Nach einer weiteren Wartezeit werden die LEDs wieder ausgeschaltet. Die Schleife startet von Neuem. Bewegen Sie die Spielfigur an die gewünschte Position der nächsten Weihnachtsmarkthütte und drücken Sie wieder den Taster.

Für Ihre Notizen



Heute im Adventskalender

- 1 20-MOhm-Widerstand (rot-schwarz-blau)

Gamepad aus Knete steuert Steve

Die Spielfigur Steve in Minecraft lässt sich zwar mit der Maus drehen, aber nicht bewegen. In der Konsoleversion des Spiels funktioniert das über Tasten auf dem Gamepad. Im Experiment des 19. Tages stellen Sie ein Gamepad mit vier Sensorkontakten aus Knete her, mit dem Sie die Spielfigur steuern können.

Bauteile

- 1 Steckbrett
- 4 20-MOhm-Widerstände (rot-schwarz-blau)
- 5 GPIO-Verbindungskabel
- 4 Knetekkontakte

Legen Sie die vier Knetekkontakte in der in der Abbildung gezeigten Anordnung auf den Tisch. Sie entsprechen den vier Richtungstasten auf einem Gamepad.

Da die Python-Schnittstelle in Minecraft nur die Möglichkeit bietet, die Spielfigur auf eine bestimmte Koordinate zu versetzen, nicht aber mithilfe der Tasten vorwärts, rückwärts oder seitwärts relativ zur aktuellen Blickrichtung zu gehen, installieren wir ein zusätzliches Python-Modul, das Tastaturreingaben simuliert. Dazu ist eine Internetverbindung auf dem Raspberry Pi nötig.

Klicken Sie zur Installation der zusätzlichen Pakete zunächst auf dieses Symbol in der Startleiste, um ein Linux-Terminalfenster zu öffnen. Geben Sie dort Folgendes ein:

```
sudo pip3 install python3-xlib
```

Nachdem die Installation durchgelaufen ist, was je nach Geschwindigkeit der Internetverbindung einige Sekunden dauern kann, geben Sie Folgendes ein:

```
sudo pip3 install pyautogui
```

Warten Sie auch hier, bis die Installation abgeschlossen ist. Danach können Sie das Terminalfenster wieder schließen.

Das Programm

Das Programm `19mc.py` steuert die Spielfigur, indem es die vier Knetekkontakte abfragt und entsprechende Tastendrücke simuliert. Für dieses Programm gibt es keine spezielle Aufgabe in der Minecraft-Welt, es steuert einfach nur die Spielfigur.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import pyautogui
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
kw = 16
ka = 21
ks = 12
kd = 7

GPIO.setmode(GPIO.BCM)
GPIO.setup(kw, GPIO.IN)
GPIO.setup(ka, GPIO.IN)
GPIO.setup(ks, GPIO.IN)
GPIO.setup(kd, GPIO.IN)

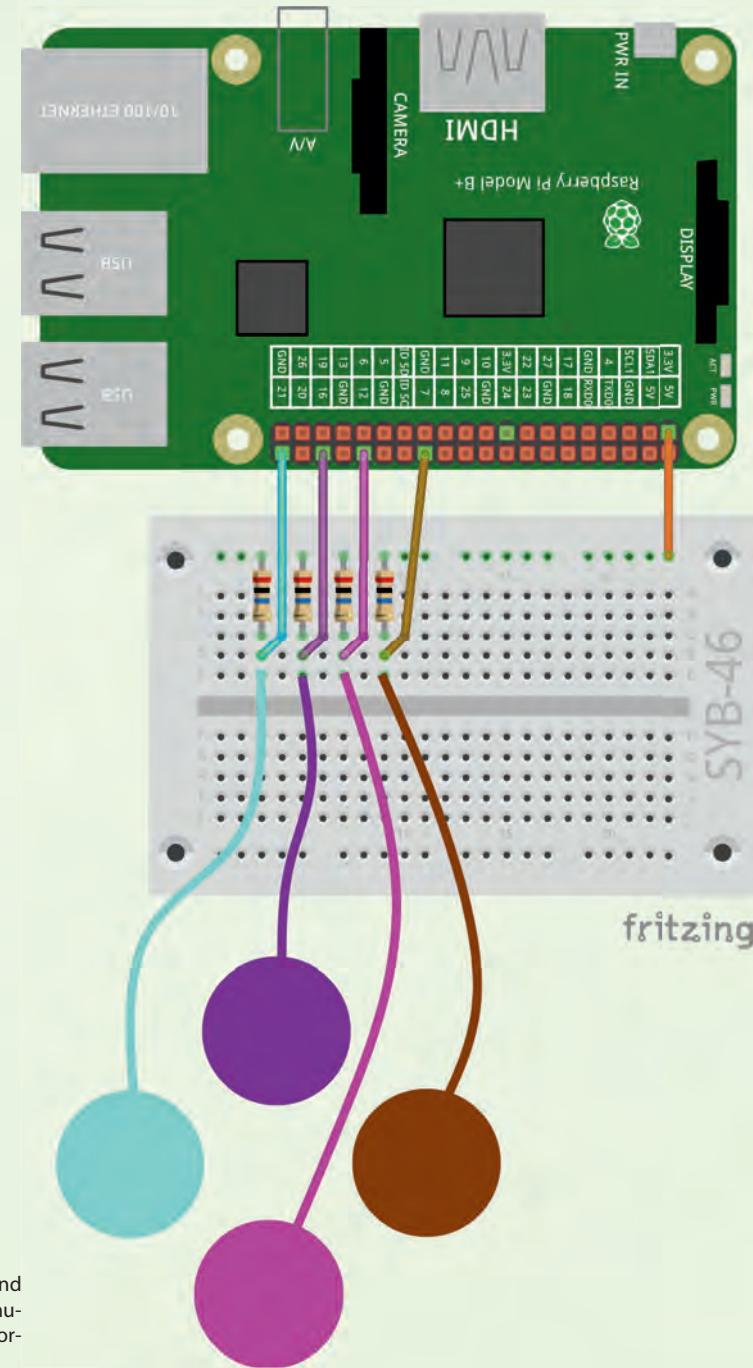
try:
    while True:
        if GPIO.input(kw) == False:
            pyautogui.keyDown('w')
        else:
            pyautogui.keyUp('w')
        if GPIO.input(ka) == False:
            pyautogui.keyDown('a')
        else:
            pyautogui.keyUp('a')
        if GPIO.input(ks) == False:
            pyautogui.keyDown('s')
        else:
            pyautogui.keyUp('s')
        if GPIO.input(kd) == False:
            pyautogui.keyDown('d')
        else:
            pyautogui.keyUp('d')

except KeyboardInterrupt:
    GPIO.cleanup()
```

So funktioniert das Programm

```
import pyautogui
```

Das Modul `pyautogui` ermöglicht unter anderem, das Drücken und Loslassen von Tasten auf der Tastatur über Programmereignisse zu simulieren. Dieses Modul wird am Anfang mit den anderen Modulen importiert.



Ein Gamepad mit vier Sensorkontakten aus Knete steuert die Spielfigur.

```
kw = 16  
ka = 21  
ks = 12  
kd = 7
```

Die Variablen `kw`, `ka`, `ks` und `kd` enthalten die Pin-Nummern der GPIO-Pins für die Knetekkontakte, die die Tasten `w`, `a`, `s` und `d` simulieren.

```
GPIO.setmode(GPIO.BCM)  
GPIO.setup(kw, GPIO.IN)  
GPIO.setup(ka, GPIO.IN)  
GPIO.setup(ks, GPIO.IN)  
GPIO.setup(kd, GPIO.IN)
```

Die vier Pins werden als Eingänge ohne Pulldown-Widerstände eingerichtet.

```
try:  
    while True:  
        if GPIO.input(kw) == False:  
            pyautogui.keyDown('w')
```

Die Hauptschleife des Programms prüft nacheinander, ob einer der Knetekkontakte berührt wird. Wird der Kontakt an Pin `kw` berührt, simuliert `pyautogui.keyDown()` das Drücken der Taste `w`.

```
    else:  
        pyautogui.keyUp('w')
```

Wird der Pin dagegen im Moment der Abfrage nicht berührt, simuliert `pyautogui.keyUp()` das Loslassen der Taste `w`.

```
if GPIO.input(ka) == False:  
    pyautogui.keyDown('a')  
else:  
    pyautogui.keyUp('a')  
if GPIO.input(ks) == False:  
    pyautogui.keyDown('s')  
else:  
    pyautogui.keyUp('s')  
if GPIO.input(kd) == False:  
    pyautogui.keyDown('d')  
else:  
    pyautogui.keyUp('d')
```

Nach dem gleichen Schema werden auch die Tasten `a`, `s` und `d` über Knetekkontakte simuliert.

Für Ihre Notizen



20. TAG

Heute im Adventskalender

- 2 GPIO-Verbindungskabel

Häuser mit Satteldach bauen und wieder löschen

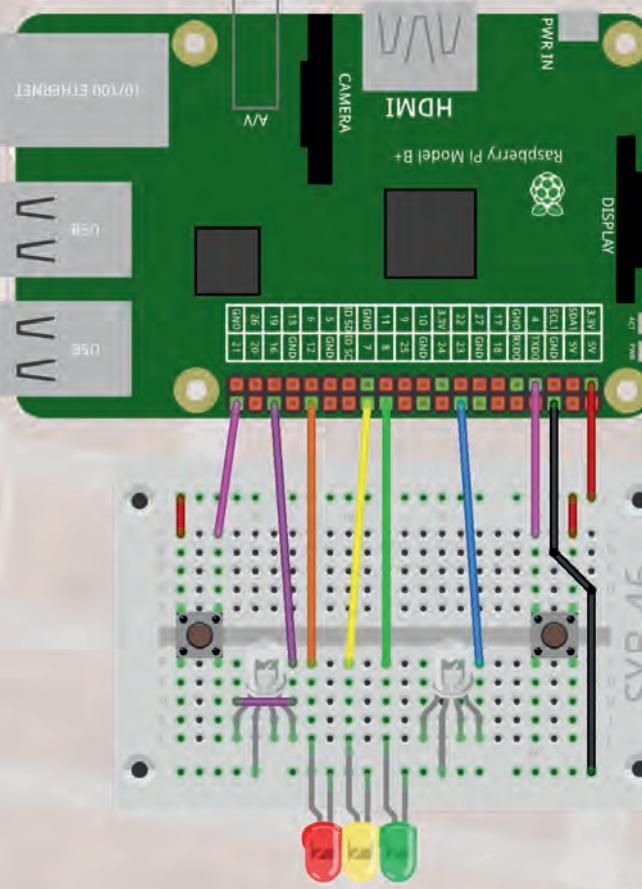
Das Programm des 20. Tages baut auf Knopfdruck Weihnachtsmarkthütten und bietet die Möglichkeit, die zuletzt gebaute Hütte auch einfach per Knopfdruck wieder zu entfernen. Im Gegensatz zur vorherigen Version haben die Hütten diesmal klassische Satteldächer statt Flachdachelementen.



Neue Weihnachtsmarkthütten mit Satteldach auf Knopfdruck bauen.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 2 RGB-LEDs mit Vorwiderstand
- 2 Taster
- 9 GPIO-Verbindungskabel
- 3 Drahtbrücken



Zwei Tasten und fünf LEDs.

Das Programm

Das Programm `20mc.py` basiert auf dem Programm des 18. Tages, allerdings mit zwei wesentlichen Unterschieden: Die Hütten haben statt der Flachdachplatten ein ansprechender aussehendes Satteldach und können durch Betätigung des zweiten Tasters auch wieder gelöscht werden, wenn sie versehentlich an einer falschen Position gebaut wurden.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time
```

```
t = 1.0

mc = minecraft.Minecraft.create()
LED = [16,12,7,8,23]
t1 = 21
t2 = 4

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)

try:
    while True:
        if GPIO.input(t1)==True:
            p = mc.player.getTilePos()
            mc.postToChat(str(p.x-1)+" , "+str(p.y)+" , "+str(p.z)+" ... "+str(p.x-4)+" , "+str(p.y+3)+" , "+str(p.z-3))
            mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, p.z-3, block.BRICK_BLOCK.id)
            GPIO.output(LED[0],True)
            time.sleep(t)
            mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, p.z-2, block.AIR.id)
            mc.setBlocks(p.x-1, p.y+1, p.z-1, p.x-3, p.y+1, p.z-2, block.AIR.id)
            GPIO.output(LED[1],True)
            time.sleep(t)
            mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4, p.y+2, p.z-2, block.BRICK_BLOCK.id)
            GPIO.output(LED[2],True)
            time.sleep(t)
            mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4, p.y+2, p.z, block.STAIRS_WOOD.id,3)
            mc.setBlocks(p.x-1, p.y+2, p.z-3, p.x-4, p.y+2, p.z-3, block.STAIRS_WOOD.id,2)
            GPIO.output(LED[3],True)
            time.sleep(t)
            mc.setBlocks(p.x-1, p.y+3, p.z-1, p.x-4, p.y+3, p.z-1, block.STAIRS_WOOD.id,3)
            mc.setBlocks(p.x-1, p.y+3, p.z-2, p.x-4, p.y+3, p.z-2, block.STAIRS_WOOD.id,2)
            GPIO.output(LED[4],True)
            time.sleep(t)
        for i in LED:
            GPIO.output(i,False)
        if GPIO.input(t2)==True:
            mc.postToChat(str(p.x-1)+" , "+str(p.y)+" , "+str(p.z)+" ... "+str(p.x-4)+" , "+str(p.y+3)+" , "+str(p.z-3))
            mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+3, p.z-3, block.AIR.id)
            time.sleep(t)
```

```
except KeyboardInterrupt:  
    GPIO.cleanup()
```

So funktioniert das Programm

Am Anfang werden wie üblich Bibliotheken importiert und die GPIO-Pins für die LEDs und die beiden Taster eingerichtet. Danach wartet die Hauptschleife des Programms, bis einer der beiden Taster gedrückt wird.

```
if GPIO.input(t1)==True:  
    p = mc.player.getTilePos()  
    mc.postToChat(str(p.x-1)+", "+str(p.y)+",  
"+str(p.z)+" ... "+str(p.x-4)+", "+str(p.y+3)+",  
"+str(p.z-3))
```

Wird der erste Taster gedrückt, speichert das Programm die Position der Spielfigur in den Variablen `p`. Daraus ergeben sich die Koordinaten der zu bauenden Hütte, die im Chat angezeigt werden.

```
mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4,  
p.y+2, p.z-2, block.BRICK_BLOCK.id)  
GPIO.output(LED[2],True)  
time.sleep(t)
```

Die ersten beiden Bauschritte für den Unterbau der Hütte entsprechen denen aus der letzten Programmversion. Da diesmal keine durchgehende Dachplatte in der y+2-Ebene gebaut werden kann, werden diesmal im dritten Schritt zwei Reihen Blöcke aus dem Ziegelmaterial `BRICK_BLOCK` im mittleren Bereich dieser Ebene errichtet.

```
mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4,  
p.y+2, p.z, block.STAIRS_WOOD.id,3)  
mc.setBlocks(p.x-1, p.y+2, p.z-3, p.x-4,  
p.y+2, p.z-3, block.STAIRS_WOOD.id,2)  
GPIO.output(LED[3],True)  
time.sleep(t)
```

Im vierten Schritt entstehen die beiden Dachhälften in der unteren der beiden Dachebenen getrennt voneinander. Das Material `STAIRS_WOOD` stellt die treppenartigen Dachelemente dar. Hier wird ein zusätzlicher Parameter benötigt, der die Richtung der Blöcke angibt. Die beiden verwendeten Richtungen sind 2 und 3. Die Werte 0 und 1 stellen die beiden um 90° versetzten Richtungen dar.

```
mc.setBlocks(p.x-1, p.y+3, p.z-1, p.x-4,  
p.y+3, p.z-1, block.STAIRS_WOOD.id,3)  
mc.setBlocks(p.x-1, p.y+3, p.z-2, p.x-4,  
p.y+3, p.z-2, block.STAIRS_WOOD.id,2)  
GPIO.output(LED[4],True)  
time.sleep(t)
```

Auf die gleiche Weise werden im nächsten Schritt die beiden Blockreihen für das Dach in der y+3-Ebene aufgebaut.

```
time.sleep(t)  
for i in LED:  
    GPIO.output(i,False)
```

Nach einer weiteren Wartezeit werden die LEDs wieder ausgeschaltet.

```
if GPIO.input(t2)==True:  
    mc.postToChat(str(p.x-1)+", "+str(p.y)+",  
"+str(p.z)+" ... "+str(p.x-4)+", "+str(p.y+3)+",  
"+str(p.z-3))  
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+3,  
p.z-3, block.AIR.id)
```

Wird der zweite Taster gedrückt, wird die Hütte durch einen großen Quader aus Blöcken des Materials `AIR` ersetzt und damit gelöscht.

```
time.sleep(t)
```

Nach einer weiteren Wartezeit startet die Schleife von Neuem, damit Sie die nächste Weihnachtsmarkthütte bauen können.

Für Ihre Notizen



21. TAG

Heute im Adventskalender

- 1 Blink-LED mit Vorwiderstand

Blink-LED

Die Blink-LED blinkt automatisch, wenn sie mit Strom versorgt wird, ohne dass ein Programm nötig ist. Solche LEDs werden gern für Statusanzeigen verwendet oder um in wichtigen Situationen die Aufmerksamkeit des Benutzers auf sich zu ziehen.

Häuser in verschiedenen Richtungen bauen

Bei jedem neuen Haus kann vor dem endgültigen Bau die Richtung festgelegt werden. Vier LEDs zeigen diese Richtungen an.



Weihnachtsmarkthütten mit Knetkontakte bauen.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 1 RGB-LED mit Vorwiderstand
- 1 Blink-LED mit Vorwiderstand
- 3 20-MOhm-Widerstände (rot-schwarz-blau)
- 10 GPIO-Verbindungskabel
- 3 Drahtbrücken
- 4 Knetkontakte

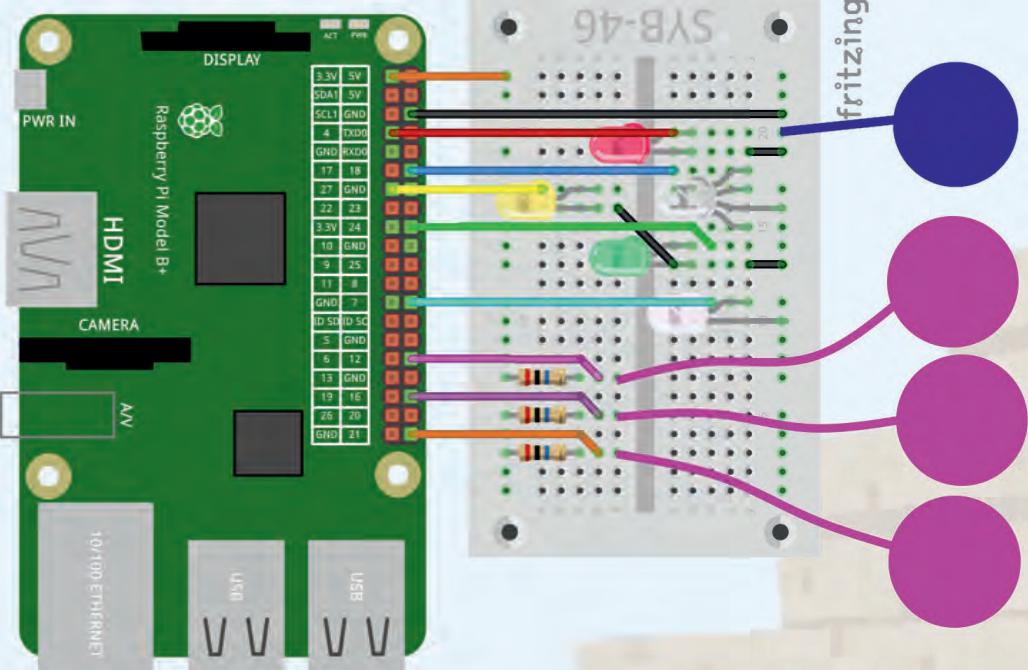
Knetkontakte steuern den Bau von Häusern.

Das Programm

Das Programm [21mc.py](#) baut ähnlich wie das Programm von gestern Häuser mit Satteldach an der Position der Spielfigur. Zur Steuerung werden drei Knetkontakte verwendet. Der erste legt die Position des Hauses fest, mit dem zweiten lässt sich die Ausrichtung in Schritten von 90° angeben. Erst bei Berührung des dritten Knetkontakte wird das Haus gebaut und die Hauptschleife des Programms neu gestartet, sodass weitere Häuser gebaut werden können.



Die drei Sensorkontakte haben unterschiedliche Funktionen beim Hausbau.



```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
import time

t = 0.5

mc = minecraft.Minecraft.create()
LED = [18,24,27,4]
bl = 7
k1 = 21
k2 = 16
k3 = 12

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
GPIO.setup(bl, GPIO.OUT, initial=False)
GPIO.setup(k1, GPIO.IN)
GPIO.setup(k2, GPIO.IN)
GPIO.setup(k3, GPIO.IN)

def haus0():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+3, ↴
    p.z-3, block.AIR.id)
```

```

for i in LED:
    GPIO.output(i,False)

def haus1():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-1, p.y+1, p.z-1, p.x-3, ↴
p.y+1, p.z-2, block.AIR.id)
    mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4, ↴
p.y+2, p.z-2, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4, p.y+2, ↴
p.z, block.STAIRS_WOOD.id,3)
    mc.setBlocks(p.x-1, p.y+2, p.z-3, p.x-4, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,2)
    GPIO.output(LED[0],True)

def haus2():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-2, p.y+1, p.z, p.x-3, p.y+1, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-2, p.y+2, p.z, p.x-3, p.y+2, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-1, p.y+2, p.z, p.x-1, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,1)
    mc.setBlocks(p.x-4, p.y+2, p.z, p.x-4, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,0)
    mc.setBlocks(p.x-2, p.y+3, p.z, p.x-2, p.y+3, ↴
p.z-3, block.STAIRS_WOOD.id,1)
    mc.setBlocks(p.x-3, p.y+3, p.z, p.x-3, p.y+3, ↴
p.z-3, block.STAIRS_WOOD.id,0)
    GPIO.output(LED[1],True)

def haus3():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-2, p.y+1, p.z-1, p.x-4, p.y+1, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4, p.y+2, ↴
p.z-2, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4, p.y+2, ↴
p.z, block.STAIRS_WOOD.id,3)
    mc.setBlocks(p.x-1, p.y+2, p.z-3, p.x-4, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,2)

    mc.setBlocks(p.x-1, p.y+3, p.z-1, p.x-4, p.y+3, ↴
p.z-1, block.STAIRS_WOOD.id,3)
    mc.setBlocks(p.x-1, p.y+3, p.z-2, p.x-4, p.y+3, ↴
p.z-2, block.STAIRS_WOOD.id,2)
    GPIO.output(LED[2],True)

def haus4():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-2, p.y+1, p.z-1, p.x-3, p.y+1, ↴
p.y+1, p.z-3, block.AIR.id)
    mc.setBlocks(p.x-2, p.y+2, p.z, p.x-3, p.y+2, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-1, p.y+2, p.z, p.x-1, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,1)
    mc.setBlocks(p.x-4, p.y+2, p.z, p.x-4, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,0)
    mc.setBlocks(p.x-2, p.y+3, p.z, p.x-2, p.y+3, ↴
p.z-3, block.STAIRS_WOOD.id,1)
    mc.setBlocks(p.x-3, p.y+3, p.z, p.x-3, p.y+3, ↴
p.z-3, block.STAIRS_WOOD.id,0)
    GPIO.output(LED[3],True)

mc.postToChat("Position mit Sensor 1 festlegen")
try:
    while True:
        fertig = False
        d = 1
        if GPIO.input(k1)==False:
            p = mc.player.getTilePos()
            mc.postToChat("Haus mit Sensor 2 drehen ↴
oder entfernen")
            mc.postToChat("Haus mit Sensor 3 bauen")
            haus1()
            while fertig==False:
                GPIO.output(bl,True)
                if GPIO.input(k2)==False:
                    d += 1
                    if d > 4:
                        d = 0
                    if d == 1:
                        haus0()
                        haus1()
                    elif d == 2:
                        haus0()
                        haus2()
                    elif d == 3:
                        haus0()
                        haus3()
                    elif d == 4:
                        haus0()
                        haus4()

```

```

else:
    haus0()
    time.sleep(t)
if GPIO.input(k3)==False:
    fertig = True
    for i in LED:
        GPIO.output(i,False)
        GPIO.output(bl,False)
        mc.postToChat("Position mit Sensor 1
festlegen")
        time.sleep(t)

except KeyboardInterrupt:
    GPIO.cleanup()

```

So funktioniert das Programm

```

LED = [18,24,27,4]
bl = 7
k1 = 21
k2 = 16
k3 = 12

```

Das Programm ähnelt weitgehend dem vorherigen. Am Anfang werden GPIO-Pins für vier LEDs, eine Blink-LED und drei Knetkontakte eingerichtet. Die LEDs zeigen diesmal nicht den Baufortschritt an, sondern die Richtung, in der die zukünftige Weihnachtsmarkthütte steht. Deshalb sind die LEDs im Viereck und nicht in einer Reihe angeordnet.

Um das Programm übersichtlich zu halten, werden Teile in sogenannte Funktionen ausgelagert. Eine Funktion wird einmal mit `def` definiert und bekommt einen Namen, über den sie jederzeit an beliebiger Stelle im Programm aufgerufen werden kann.

```

def haus0():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+3, ↴
p.z-3, block.AIR.id)
    for i in LED:
        GPIO.output(i,False)

```

Die Funktion `haus0()` löscht das Haus an der angegebenen Position und schaltet alle vier LEDs aus.

```

def haus1():
    mc.setBlocks(p.x-1, p.y, p.z, p.x-4, p.y+1, ↴
p.z-3, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-2, p.y, p.z-1, p.x-3, p.y, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-1, p.y+1, p.z-1, p.x-3, p.y+1, ↴
p.z-2, block.AIR.id)
    mc.setBlocks(p.x-1, p.y+2, p.z-1, p.x-4, p.y+2, ↴
p.z-2, block.BRICK_BLOCK.id)
    mc.setBlocks(p.x-1, p.y+2, p.z, p.x-4, p.y+2, ↴
p.z, block.STAIRS_WOOD.id,3)
    mc.setBlocks(p.x-1, p.y+2, p.z-3, p.x-4, p.y+2, ↴
p.z-3, block.STAIRS_WOOD.id,2)

```

```

mc.setBlocks(p.x-1, p.y+3, p.z-1, p.x-4, p.y+3, ←
p.z-1, block.STAIRS_WOOD.id,3)
mc.setBlocks(p.x-1, p.y+3, p.z-2, p.x-4, p.y+3, ←
p.z-2, block.STAIRS_WOOD.id,2)
GPIO.output(LED[0],True)

```

Die Funktion `haus1()` baut ein Haus an der angegebenen Position mit dem Fenster in Richtung der Spielfigur nach dem gleichen Verfahren wie im vorherigen Programm und schaltet die blaue LED ein, die im Viereck nach unten gerichtet ist.

Die Funktionen `haus2()`, `haus3()` und `haus4()` wirken sich ähnlich aus, wobei die Ausrichtung des gebauten Hauses jedoch um jeweils 90° gedreht ist. Die der Ausrichtung entsprechende LED wird eingeschaltet.

```

mc.postToChat("Position mit Sensor 1 festlegen")
try:
    while True:
        fertig = False
        d = 1

```

Das Hauptprogramm gibt als Erstes dem Benutzer eine Anweisung, was zu tun ist. Dann startet die Hauptschleife, die am Anfang zwei Variablen setzt. Die Variable `fertig` wird später verwendet, um die Hauptschleife neu zu starten, wenn ein Haus gebaut wurde. Die Variable `d` gibt die Richtung an, in der ein Haus gebaut wird. `1` bedeutet dabei die Standardrichtung mit dem Fenster zur Spielfigur.

```

if GPIO.input(k1)==False:
    p = mc.player.getTilePos()
    mc.postToChat("Haus mit Sensor 2 drehen ←
oder entfernen")
    mc.postToChat("Haus mit Sensor 3 bauen")

```

Berührt der Benutzer den ersten Knetekontakt, wird die Position der Spielfigur in der Variablen `p` gespeichert. Anschließend erhält der Benutzer weitere Anweisungen zu den Funktionen der anderen beiden Knetekontakte.

```
haus1()
```

Jetzt wird die Funktion `haus1()` aufgerufen, die ein Haus an der Position der Spielfigur baut.

```

while fertig==False:
    GPIO.output(bl,True)
    if GPIO.input(k2)==False:

```

In der Hauptschleife startet eine weitere Schleife, die so lange wiederholt wird, wie die Variable `fertig` den Wert `False` hat. Diese Schleife schaltet die Blink-LED ein, um anzusehen, dass das Programm auf eine Eingabe des Benutzers wartet – genauer gesagt darauf, dass der Benutzer mit dem zweiten Knetekontakt die Richtung des Hauses festlegt oder mit dem dritten das Haus endgültig baut.

```

if GPIO.input(k2)==False:
    d += 1
    if d > 4:
        d = 0

```

Wenn der Benutzer den zweiten Knetekontakt berührt, wird die Variable `d`, die die Richtung angibt, um 1 erhöht. Das Programm verarbeitet vier Richtungen mit den Zahlen 1 bis 4. Würde durch wiederholte Berührung des Knetekontaktes ein größerer Wert als 4 erreicht, wird die Variable `d` auf 0 gesetzt, was bedeutet, dass das Haus gelöscht oder gar nicht erst gebaut wird.

```

if d == 1:
    haus0()
    haus1()
elif d == 2:
    haus0()
    haus2()
elif d == 3:
    haus0()
    haus3()
elif d == 4:
    haus0()
    haus4()
else:
    haus0()

```

Abhängig vom Wert der Variable `d` wird ein Haus in der eingestellten Richtung gebaut. Zuvor wird das zuletzt angezeigte Haus gelöscht, sonst würden sich die Dächer verschiedener Varianten im Firstbereich überlagern.

```
time.sleep(t)
```

Das Programm wartet eine kurze Zeit, um zu verhindern, dass etwas längere Berührungen des zweiten Knetekontaktes das Haus so schnell drehen, dass der Benutzer die gewünschte Richtung nicht auswählen kann.

```

if GPIO.input(k3)==False:
    fertig = True
    for i in LED:
        GPIO.output(i,False)
        GPIO.output(bl,False)
    mc.postToChat("Position mit Sensor 1 ←
festlegen")

```

Berührt der Benutzer den dritten Knetekontakt, wird die Variable `fertig` auf `True` gesetzt, was bedeutet, dass das Haus endgültig gebaut wird. Alle LEDs und die Blink-LED werden ausgeschaltet. Der Benutzer erhält einen Hinweis, dass die Position des nächsten Hauses mit dem ersten Knetekontakt gewählt werden kann. Nach einer kurzen Wartezeit startet die Hauptschleife neu.

Für Ihre Notizen



22. TAG

Heute im Adventskalender

- #### ■ 1 LED orange mit Vorwiderstand

Farbige Beleuchtung in den Weihnachtsmarkthütten

Bauen Sie verschiedenfarbige Lampen in die Weihnachtsmarkthütten und lassen Sie diese gleichzeitig mit den LEDs der entsprechenden Farbe leuchten. Zusätzlich wird bei jedem Ein- und Ausschalten einer LED auch die Blink-LED ein- bzw. ausgeschaltet.



Farbige Blöcke in den Weihnachtsmarkthütten

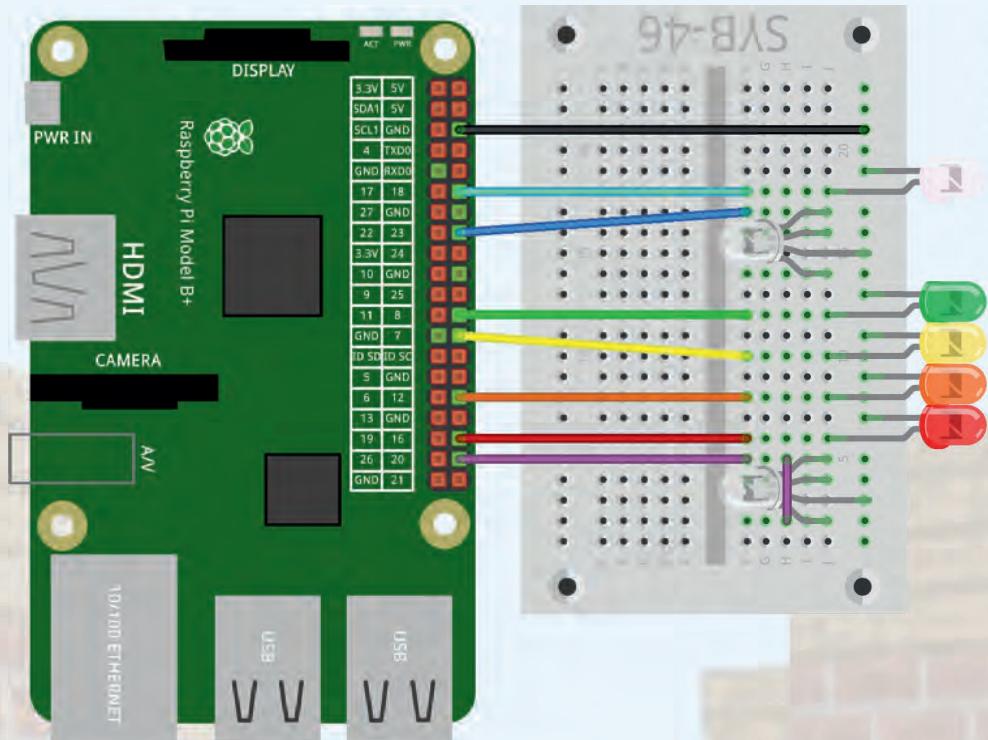
Bauteile

- 1 Steckbrett
 - 1 LED rot mit Vorwiderstand
 - 1 LED orange mit Vorwiderstand
 - 1 LED gelb mit Vorwiderstand
 - 1 LED grün mit Vorwiderstand
 - 2 RGB-LEDs mit Vorwiderstand
 - 1 Blink-LED mit Vorwiderstand
 - 8 GPIO-Verbindungskabel
 - 1 Drahtbrücke

Für die Farben Lila und Blau werden wie in früheren Projekten wieder RGB-LEDs verwendet, bei denen nur einzelne Pins angeschlossen sind.



Diese Materialien können für die farbigen Blöcke verwendet werden.



Sieben LEDs leuchten auf dem
Weihnachtsmarkt

Das Programm

Vor und während der Ausführung des Programms `22mc.py` bauen Sie mit den bekannten Möglichkeiten von Minecraft farbige Klötze in die Fenster der Weihnachtsmarktbuden. Dabei können die Materialien aus der Abbildung verwendet werden.

Die Zahlen bezeichnen die Materialnummern, die für das Programm verwendet werden. Großgeschriebene Farben stehen für eingeschaltete LEDs, kleingeschriebene für ausgeschaltete.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
b = 18
LED = [20,16,12,7,8,23]
ein = [2,6,1,0,5,3]
aus = [10,14,12,4,13,11]

GPIO.setmode(GPIO.BCM)
GPIO.setup(b, GPIO.OUT, initial=False)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)

mc.postToChat("Zuerst farbige Blöcke in die Fenster setzen")
try:
    while True:
        for hit in mc.events.pollBlockHits():
```

```

    b1 = mc.getBlockWithData(hit.pos.x, hit. pos.y, hit.pos.z)
    if b1.id == block.BRICK_BLOCK.id:
        b2 = mc.getBlockWithData(hit.pos.x, hit. pos.y+1, hit.pos.z)
        if b2.id == block.WOOL.id:
            d = b2.data
            for i in range(6):
                if d == aus[i]:
                    mc.setBlock(hit.pos.x, hit.pos.y+1, hit.pos.z, block.WOOL.id, ein[i])
                    GPIO.output(LED[i],True)
                    GPIO.output(b,True)
            for i in range(6):
                if d == ein[i]:
                    mc.setBlock(hit.pos.x, hit.pos.y+1, hit.pos.z, block.WOOL.id, aus[i])
                    GPIO.output(LED[i],False)
                    GPIO.output(b,False)

    except KeyboardInterrupt:
        GPIO.cleanup()

```

So funktioniert das Programm

Im Programm [22mc.py](#) laufen Sie mit der Spielfigur über den Weihnachtsmarkt und schlagen mit dem Schwert auf die Blöcke unterhalb des Fensters der Hütten, als ob die Spielfigur auf die Theke klopft. Befindet sich im Fenster ein farbiger Block, werden dieser und die LED gleicher Farbe zum Leuchten gebracht. Der nächste Schlag schaltet Block und LED wieder aus.

```

b = 18
LED = [20,16,12,7,8,23]
ein = [2,6,1,0,5,3]
aus = [10,14,12,4,13,11]

```

Am Anfang werden GPIO-Pins für die LEDs definiert. **b** ist der Pin der Blink-LED. Die Pins der sechs farbigen LEDs sind in der Liste **LED** gespeichert. Die Listen **ein[]** und **aus[]** enthalten die Farbnummern der eingeschalteten und ausgeschalteten Blöcke.

```

mc.postToChat("Zuerst farbige Blöcke in die Fenster setzen")

```

Das Hauptprogramm erinnert den Spieler als Erstes daran, farbige Blöcke in die Fenster der Weihnachtsmarkthütten zu setzen. Da der Minecraft-Chat keine Umlaute darstellt, wird das ö in **Blöcke** durch **oe** ersetzt.

```

try:
    while True:
        for hit in mc.events.pollBlockHits():
            b1 = mc.getBlockWithData(hit.pos.x, hit. pos.y, hit.pos.z)

```

Eine Endlosschleife wartet darauf, dass der Spieler auf einen Block schlägt. In diesem Fall wird das Material des getroffenen Blocks in der Variablen **b1** gespeichert.

```

            if b1.id == block.BRICK_BLOCK.id:
                b2 = mc.getBlockWithData(hit.pos.x, hit. pos.y+1, hit.pos.z)
                if b2.id == block.WOOL.id:
                    d = b2.data

```

Hat die Spielfigur auf einen Block aus dem Material **BRICK_BLOCK** geschlagen, wird das Material des darüber befindlichen Blocks ausgelesen und in der Variablen **b2** gespeichert. Besteht der Block aus dem Material **WOOL**, handelt es sich wahrscheinlich um einen farbigen Block, der umgeschaltet werden kann. Die Farbnummer wird in der Variablen **d** gespeichert.

```

                    for i in range(6):
                        if d == aus[i]:
                            mc.setBlock(hit.pos.x, hit.pos.y+1, hit.pos.z, block.WOOL.id, ein[i])
                            GPIO.output(LED[i],True)
                            GPIO.output(b,True)

```

Eine Schleife prüft nacheinander, ob die Farbnummer des Blocks einer der in der Liste **aus[]** gespeicherten Farben für ausgeschaltete Lampen entspricht. In diesem Fall bekommt der Block die Farbe der entsprechenden eingeschalteten Lampe aus der Liste **ein[]**. Die LED der gleichen Farbe und die Blink-LED werden eingeschaltet.

```

                    for i in range(6):
                        if d == ein[i]:
                            mc.setBlock(hit.pos.x, hit.pos.y+1, hit.pos.z, block.WOOL.id, aus[i])
                            GPIO.output(LED[i],False)
                            GPIO.output(b,False)

```

Nach dem gleichen Schema werden der Block und die LED ausgeschaltet, wenn die Farbe einer der in der Liste **ein[]** gespeicherten Farben für eingeschaltete Lampen entspricht.

Für Ihre Notizen



23 TAG

Heute im Adventskalender

- #### ■ 1 LED lila mit Vorwiderstand

Minispiel Schatzsuche

In dem Minispiel des 23. Tages soll mit der Spielfigur ein unter der markierten Fläche zufällig versteckter Schatz gefunden werden. Die RGB-LEDs helfen dabei, indem sie den Abstand zum Schatz anzeigen.

Bauteile

- 1 Steckbrett
 - 1 LED rot mit Vorwiderstand
 - 1 LED lila mit Vorwiderstand
 - 1 LED gelb mit Vorwiderstand
 - 1 LED grün mit Vorwiderstand
 - 2 RGB-LEDs mit Vorwiderstand
 - 4 20-MOhm-Widerstände (rot-schwarz-blau)
 - 16 GPIO-Verbindungskabel
 - 3 Drahtbrücken
 - 5 Knetekkontakte

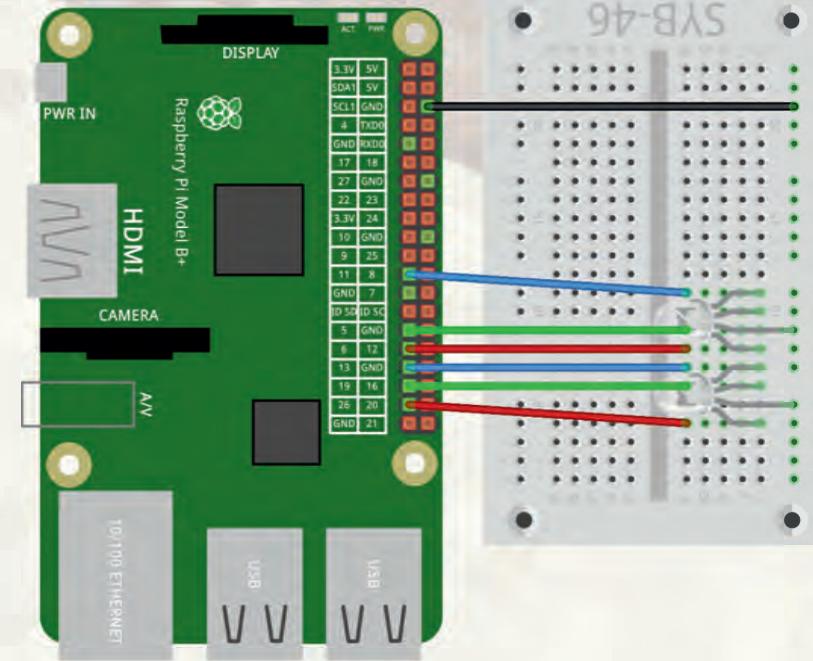
Da es auf dem Steckbrett langsam eng wird, zeigt diese Abbildung den Schaltungsaufbau in zwei Teilen. Die Kabel der RGB-LEDs würden in der Grafik andere Anschlusskabel verdecken.

Das Programm

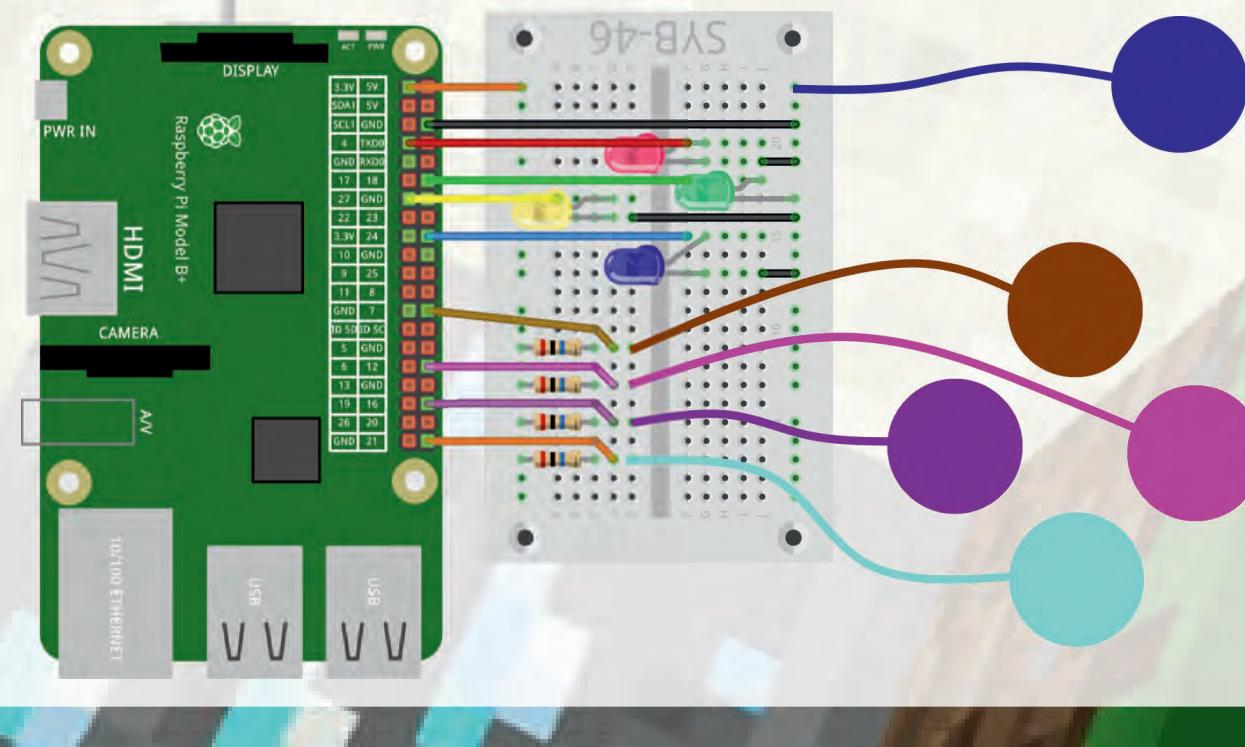
Im Programm `23mc.py` entsteht auf der großen Freifläche auf dem Weihnachtsmarkt eine Wiese, unter der das Programm einen Schatz versteckt.



Unter dieser Wiese wird der Schatz versteckt.



Ein Gamepad mit vier Sensorkontakten aus Knete, vier LEDs und zwei RGB-LEDs.



Bewegen Sie die Spielfigur mit den Knetekontakten. Die Farben der RGB-LEDs zeigen die Entfernung zum Schatz in x- und z-Richtung. Je weiter die Farbe im Spektrum Richtung Blau zeigt, desto weiter weg steht die Spielfigur vom Schatz. Wenn beide RGB-LEDs rot leuchten, stehen Sie genau über dem Schatz. Schlagen Sie dann mit dem Schwert den Grasblock weg, um den Schatz freizulegen. Die vier farbigen LEDs zeigen Berührungen der Knetkontakte an.



Der Schatz wurde gefunden.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import pyautogui, random, time, colorsys
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
LED = [27,24,18,4]
k = [16,21,12,7]
RGB1 = [26,19,13]
RGB2 = [6,5,11]
pwm1 = [0,0,0]
pwm2 = [0,0,0]

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in k:
    GPIO.setup(i, GPIO.IN)
for i in RGB1:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in RGB2:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in range(3):
    pwm1[i] = GPIO.PWM(RGB1[i], 50)
    pwm1[i].start(0)
    pwm2[i] = GPIO.PWM(RGB2[i], 50)
    pwm2[i].start(0)

x1 = -6
```

```
x2 = 2
z1 = 9
z2 = 17
ende = False
x = random.randrange(x1,x2)
z = random.randrange(z1,z2)
mc.setBlocks(x1, 1, z1, x2, 1, z2, block.GRASS.id)
mc.setBlock(x, 0, z, block.DIAMOND_ORE.id)
mc.postToChat("Schatz versteckt")

while ende == False:
    p = mc.player.getTilePos()
    rgb = colorsys.hsv_to_rgb((abs(p.x-x)/(x2-x1)), ↴
    1, 1)
    for i in range(3):
        pwm1[i].ChangeDutyCycle(rgb[i])
    rgb = colorsys.hsv_to_rgb((abs(p.z-z)/(z2-z1)), ↴
    1, 1)
    for i in range(3):
        pwm2[i].ChangeDutyCycle(rgb[i])
    if GPIO.input(k[0]) == False:
        pyautogui.keyDown('w')
        GPIO.output(LED[0],True)
    else:
        pyautogui.keyUp('w')
        GPIO.output(LED[0],False)
    if GPIO.input(k[1]) == False:
        pyautogui.keyDown('a')
        GPIO.output(LED[1],True)
    else:
        pyautogui.keyUp('a')
        GPIO.output(LED[1],False)
    if GPIO.input(k[2]) == False:
        pyautogui.keyDown('s')
        GPIO.output(LED[2],True)
    else:
        pyautogui.keyUp('s')
        GPIO.output(LED[2],False)
    if GPIO.input(k[3]) == False:
        pyautogui.keyDown('d')
        GPIO.output(LED[3],True)
    else:
        pyautogui.keyUp('d')
        GPIO.output(LED[3],False)
    bl = mc.getBlock(x, 1, z)
    if bl == block.AIR.id:
        ende = True

mc.postToChat("Schatz gefunden")
for j in range(10):
    for i in LED:
        GPIO.output(i,True)
    time.sleep(0.1)
    for i in LED:
        GPIO.output(i,False)
```

```
time.sleep(0.1)
for i in range(3):
    pwm1[i].stop()
    pwm2[i].stop()
mc.setBlocks(x1, 0, z1, x2, 1, z2, block.SAND.id)
GPIO.cleanup()
```

So funktioniert das Programm

```
LED = [27,24,18,4]
k = [16,21,12,7]
RGB1 = [26,19,13]
RGB2 = [6,5,11]
pwm1 = [0,0,0]
pwm2 = [0,0,0]
```

Die Liste `LED[]` enthält die GPIO-Pins der vier LEDs, die Liste `k[]` die GPIO-Pins der vier Knetekontakte. In den beiden Listen `RGB1[]` und `RGB2[]` stehen die GPIO-Pins der beiden RGB-LEDs, in den Listen `pwm1[]` und `pwm2[]` die zugehörigen PWM-Signale.

```
x1 = -6
x2 = 2
z1 = 9
z2 = 17
```

Die Variablen `x1`, `x2`, `z1` und `z2` enthalten die Koordinaten zweier gegenüberliegender Ecken des Spielfeldes.

```
ende = False
```

Die Variable `ende` definiert das Spielende. Die Hauptschleife des Programms läuft in diesem Spiel nicht endlos, sondern nur so lange, bis der Schatz gefunden ist.

```
x = random.randrange(x1,x2)
z = random.randrange(z1,z2)
mc.setBlocks(x1, 1, z1, x2, 1, z2, block.GRASS.id)
mc.setBlock(x, 0, z, block.DIAMOND_ORE.id)
mc.postToChat("Schatz versteckt")
```

Die Koordinaten `x` und `y` des Schatzes werden zufällig innerhalb der Spielfläche festgelegt. Die Spielfläche wird als Wiese aus dem Material `GRASS` angelegt. Ein Block aus dem Material `DIAMOND_ORE` wird als Schatz in einer y-Höhe von 0 eingebaut, also eine Einheit unterhalb der Spielfläche. Die Wiese hat an dieser Stelle die y-Höhe 1.

```
while ende == False:
    p = mc.player.getTilePos()
```

Die Hauptschleife des Programms läuft, solange die Endbedingung noch nicht erreicht ist. In jedem Durchlauf wird die Position der Spielfigur in der Variablen `p` gespeichert.

```
rgb = colorsys.hsv_to_rgb((abs(p.x-x)/(x2-x1)), ↴
1, 1)
for i in range(3):
    pwm1[i].ChangeDutyCycle(rgb[i])
```

Der Abstand zwischen Spielfigur und Schatz wird errechnet und anhand der Breite des Spielfeldes so skaliert, dass der Abstand 0 – bei dem die Spielfigur über dem Schatz steht – einem HSV-Wert von 0 (Rot) ergibt. Der maximal mögliche Abstand führt zu dem HSV-Wert am anderen Ende des Farbspektrums (Lila). Der so ermittelte HSV-Wert wird in RGB umgerechnet und mit drei PWM-Signalen von der ersten RGB-LED angezeigt.

```
rgb = colorsys.hsv_to_rgb((abs(p.z-z)/(z2-z1)), ↵
1, 1)
for i in range(3):
    pwm2[i].ChangeDutyCycle(rgb[i])
```

Nach dem gleichen Schema wird der Abstand zwischen Spielfigur und Schatz in z-Richtung errechnet und als RGB-Wert durch die zweite RGB-LED angezeigt.

```
if GPIO.input(k[0]) == False:
    pyautogui.keyDown('w')
    GPIO.output(LED[0],True)
else:
    pyautogui.keyUp('w')
    GPIO.output(LED[0],False)
```

Anschließend werden in jedem Durchlauf der Hauptschleife die vier Knetekontakte abgefragt. Wird einer davon berührt, simuliert `pyautogui` einen entsprechenden Tastendruck und bewegt die Spielfigur. Zusätzlich leuchtet die LED der zugehörigen Richtung so lange, wie der Knetekontakt berührt wird.

```
bl = mc.getBlock(x, 1, z)
if bl == block.AIR.id:
    ende = True
```

Um das Spiel zu beenden, muss der Spieler den Schatz freilegen. Dazu wird in jedem Schleifendurchlauf der Hauptschleife geprüft, ob der Block oberhalb des Schatzes das Material `AIR` hat, also nicht vorhanden ist. In diesem Fall wird die Variable `ende` auf `True` gesetzt und dadurch die Hauptschleife nicht mehr wiederholt.

```
mc.postToChat("Schatz gefunden")
```

Hat der Spieler den Schatz freigelegt, wird die Hauptschleife nicht mehr wiederholt. Jetzt wird kurz der Sieg angezeigt.

```
for j in range(10):
    for i in LED:
        GPIO.output(i, True)
        time.sleep(0.1)
    for i in LED:
        GPIO.output(i, False)
        time.sleep(0.1)
```

Die LEDs blinken zehnmal hintereinander gleichzeitig. Dazu werden alle vier LEDs mehrmals für je 0,1 Sekunden ein- und ausgeschaltet.

```
for i in range(3):
    pwm1[i].stop()
    pwm2[i].stop()
mc.setBlocks(x1, 0, z1, x2, 1, z2, block.SAND.id)
```

Zum Schluss werden die PWM-Signale gestoppt und die komplette Spielfläche einschließlich der tiefer gelegenen Ebene, in der der Schatz lag, wieder mit Sand gefüllt. Auf diese Weise werden auch Blöcke gefüllt, die man weggeschlagen hat, ohne den Schatz zu finden.

Schon gewusst ...?

Interessante Fakten zu Minecraft™

- Die erste Minecraft™-Version wurde am 17.05.2009 veröffentlicht.
- Minecraft™ wurde in allen Editionen insgesamt über 120 Millionen mal verkauft.
- Im September 2014 kaufte Microsoft Minecraft™ einschließlich aller Lizenzen für 2,5 Milliarden US-Dollar.
- Der wichtigste Grund für den Kauf war: Microsoft brauchte ein Vorzeigeprodukt für die technischen Möglichkeiten seiner neuen VR-Brille HoloLens.
- Der Minecraft™-Erfinder Markus Notch Persson tauchte nach dem Verkauf seiner Firma Mojang an Microsoft erstmals auf der Forbes-Liste der reichsten Menschen der Welt auf.
- Die Firma Mojang entwickelte auch noch die Spiele Cobalt und Scrolls, die aber nicht annähernd die Bekanntheit von Minecraft™ erreichten.
- Die Musik der PC- und Konsolenedition stammt vom deutschen Komponisten Daniel Rosenfeld.
- Im Mai 2019 startet der erste offizielle Minecraft™-Film in den Kinos.
- Unter dem Stichwort Minecraft™ gibt es auf YouTube etwa 155.000.000 Videos.
- In der Viktor-Rydberg-Schule in Stockholm ist Minecraft™ Pflichtfach.

Für Ihre Notizen



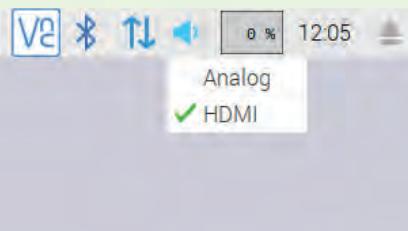
24. TAG

Heute im Adventskalender

- 1 Downloadcode

Weihnachtsmusik auf dem Raspberry Pi

Der Raspberry Pi kann Musik über einen HDMI-Monitor oder über einen externen Lautsprecher oder Kopfhörer an der analogen 3,5-mm-Klinkenbuchse abspielen. Bei Computermonitoren mit DVI-Anschluss, die über den HDMI-Ausgang verbunden sind, muss am analogenen Ausgang ein Lautsprecher angeschlossen werden, da das Audiosignal nicht über das DVI-Kabel oder einen DVI-Adapter übertragen wird. Klicken Sie mit der rechten Maustaste oben rechts auf das Lautsprechersymbol, um den Audioausgang auszuwählen.



Audioausgang auswählen.

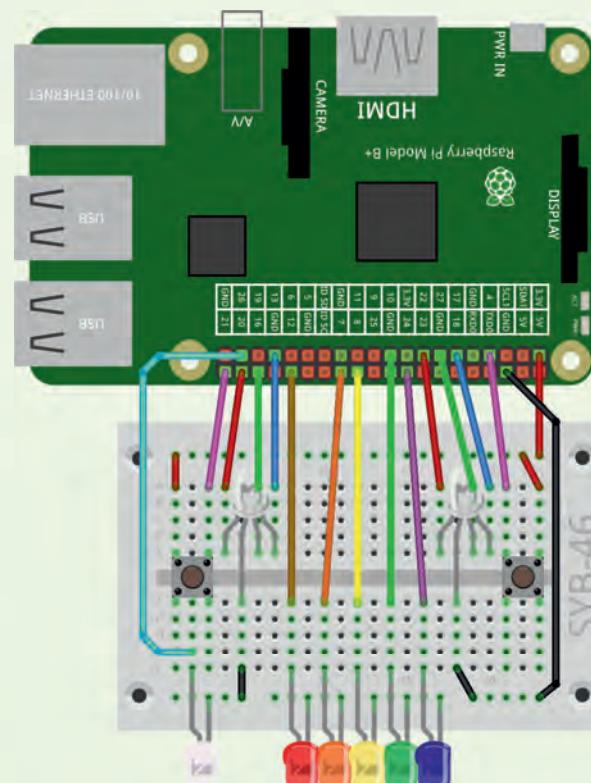


Auf dem Weihnachtsmarkt wird ein Weihnachtsbaum aufgebaut. Am 24. Dezember – und nur an diesem Tag – lassen sich Weihnachtslieder abspielen. Dazu blinken die LEDs und die grauen Klötze rund um den Weihnachtsbaum.

Der Adventskalender enthält an diesem Tag einen Downloadcode für lizenzierte Weihnachtslieder im MP3-Format. Kopieren Sie diese in den Ordner `/home/pi`. Das Programm verwendet zwei der Lieder, die anderen können Sie nach Belieben abspielen oder in das Programm einbauen.

Bauteile

- 1 Steckbrett
- 1 LED rot mit Vorwiderstand
- 1 LED orange mit Vorwiderstand
- 1 LED gelb mit Vorwiderstand
- 1 LED grün mit Vorwiderstand
- 1 LED lila mit Vorwiderstand
- 2 RGB-LEDs mit Vorwiderstand
- 1 Blink-LED mit Vorwiderstand
- 2 Taster
- 16 GPIO-Verbindungskabel
- 4 Drahtbrücken



Fünf LEDs, zwei RGB-LEDs und eine Blink-LED als Weihnachtsbeleuchtung.

Das Programm

Das Programm `24mc.py` spielt Weihnachtslieder im MP3-Format auf dem Raspberry Pi ab. Dazu blinkt eine Weihnachtsbeleuchtung mit allen LEDs des Adventskalenders. Dieses Schauspiel kann man nur am 24. Dezember erleben. An anderen Tagen steht einfach nur ein Weihnachtsbaum auf dem Weihnachtsmarkt in der Minecraft-Welt.

```
#!/usr/bin/python
import mcpi.minecraft as minecraft
import mcpi.block as block
import time, colorsys, subprocess
import RPi.GPIO as GPIO

mc = minecraft.Minecraft.create()
LED = [12,7,8,10,24]
RGB1 = [20,16,13]
RGB2 = [22,27,17]
pwm1 = [0,0,0]
pwm2 = [0,0,0]
b = 26
t1 = 21
t2 = 4

GPIO.setmode(GPIO.BCM)
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in RGB1:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in RGB2:
    GPIO.setup(i, GPIO.OUT, initial=False)
for i in range(3):
    pwm1[i] = GPIO.PWM(RGB1[i], 50)
    pwm1[i].start(0)
    pwm2[i] = GPIO.PWM(RGB2[i], 50)
    pwm2[i].start(0)
GPIO.setup(b, GPIO.OUT, initial=False)
GPIO.setup(t1, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(t2, GPIO.IN, GPIO.PUD_DOWN)

x = -2
y = 1
z = 13
mc.setBlocks(x, y+1, z, x, y+2, z, block.WOOD.id)
mc.setBlocks(x-2, y+3, z-2, x+2, y+4, z+2, block.LEAVES.id)
mc.setBlocks(x-1, y+5, z-1, x+1, y+6, z+1, block.LEAVES.id)
mc.setBlocks(x, y+7, z, x, y+8, z, block.LEAVES.id)
sx = [x-2, x-2, x-2, x, x+2, x+2, x+2, x]
sz = [z-2, z, z+2, z+2, z+2, z, z-2, z-2]
for n in range(8):
    mc.setBlock(sx[n], y+1, sz[n], block.COAL_ORE.id)
```

```

def blink():
    GPIO.output(b,True)
    for k in range(3):
        for j in range(100):
            rgb = colorsys.hsv_to_rgb(j/100, 1, 1)
            for i in range(3):
                pwm1[i].ChangeDutyCycle(rgb[i])
                rgb = colorsys.hsv_to_rgb((100-j)/100, 1, 1)
            for i in range(3):
                pwm2[i].ChangeDutyCycle(rgb[i])
            GPIO.output(LED[j%5],True)
            if j%5 == 0:
                for n in range(8):
                    mc.setBlock(sx[n], y+1, sz[n], block.COAL_ORE.id)
            if j%5 == 2:
                for n in range(8):
                    mc.setBlock(sx[n], y+1, sz[n], block.GOLD_ORE.id)
            time.sleep(0.1)
            GPIO.output(LED[j%5],False)
    GPIO.output(b,False)

try:
    while True:
        if GPIO.input(t1) == True:
            d = time.localtime()
            if d.tm_mon == 12 and d.tm_mday == 24:
                subprocess.Popen(["omxplayer", "Stille_Nacht_Klavier.mp3"])
                blink()
            else:
                mc.postToChat("Heute ist nicht Weihnachten")
        if GPIO.input(t2) == True:
            d = time.localtime()
            if d.tm_mon == 12 and d.tm_mday == 24:
                subprocess.Popen(["omxplayer", "O_Tannenbaum_Klavier.mp3"])
                blink()
            else:
                mc.postToChat("Heute ist nicht Weihnachten")
        time.sleep(0.2)

except KeyboardInterrupt:
    mc.setBlocks(x-2, y+1, z-2, x+2, y+8, z+2, block.AIR.id)
    for i in range(3):
        pwm1[i].stop()
        pwm2[i].stop()
    GPIO.cleanup()

```

So funktioniert das Programm

Am Anfang werden wieder Bibliotheken importiert sowie GPIO-Pins und PWM-Signale eingerichtet. Die neue Bibliothek `subprocess` wird benötigt, um Linux-Kommandozeilenprogramme aus einem Python-Programm heraus aufzurufen.

```

x = -2
y = 1
z = 13
mc.setBlocks(x, y+1, z, x, y+2, z, block.WOOD.id)
mc.setBlocks(x-2, y+3, z-2, x+2, y+4, z+2, block.LEAVES.id)
mc.setBlocks(x-1, y+5, z-1, x+1, y+6, z+1, block.LEAVES.id)
mc.setBlocks(x, y+7, z, x, y+8, z, block.LEAVES.id)

```

Die Variablen `x`, `y` und `z` beschreiben den Fußpunkt des Baumes, der am Anfang des Programms gebaut wird.

```

sx = [x-2, x-2, x-2, x, x+2, x+2, x+2, x]
sz = [z-2, z, z+2, z+2, z+2, z, z-2, z-2]
for n in range(8):
    mc.setBlock(sx[n], y+1, sz[n], block.COAL_ORE.id)

```

Die beiden Listen `sx[]` und `sz[]` beschreiben die Koordinaten der acht Steinblöcke rund um den Baum. Diese werden mit dem Material `COAL_ORE` gebaut und blinken später im Programm.

```

def blink():
    GPIO.output(b,True)
    for k in range(3):
        for j in range(100):
            rgb = colorsys.hsv_to_rgb(j/100, 1, 1)
            for i in range(3):
                pwm1[i].ChangeDutyCycle(rgb[i])
            rgb = colorsys.hsv_to_rgb((100-j)/100, 1, 1)
            for i in range(3):
                pwm2[i].ChangeDutyCycle(rgb[i])

```

Die Funktion `blink()` steuert die Lichteffekte, während ein Lied abgespielt wird. Dabei werden für die beiden RGB-LEDs unterschiedliche PWM-Signale definiert, mit denen die Farben der RGB-LEDs in beiden Richtungen durch das gesamte HSV-Farbspektrum laufen. Die ganze Schleife wird dreimal durchlaufen, was etwa der Länge der abgespielten Weihnachtslieder entspricht.

```
    GPIO.output(LED[j%5],True)
```

Die farbigen LEDs blinken nacheinander auf. Das Programm verwendet den Modulo-Operator `%`, um in der Schleife, die von 0 bis 99 läuft, in regelmäßigen Rhythmus die Zahlen 0 bis 4 für die Nummern der LEDs zu generieren, die aufleuchten sollen.

```

    if j%5 == 0:
        for n in range(8):
            mc.setBlock(sx[n], y+1, sz[n], block.COAL_ORE.id)

```

```

    if j%5 == 2:
        for n in range(8):
            mc.setBlock(sx[n], y+1, sz[n], block.GOLD_ORE.id)

```

In jedem Zyklus leuchten die acht Steinblöcke einmal auf und werden wieder ausgeschaltet.

```

        time.sleep(0.1)
        GPIO.output(LED[j%5],False)

```

Nach einer Wartezeit von 0,1 Sekunden wird die aktuelle LED wieder ausgeschaltet.

```
    GPIO.output(b,False)
```

Am Ende der gesamten Schleife wird auch die Blink-LED ausgeschaltet.

```

try:
    while True:
        if GPIO.input(t1) == True:

```

Die Hauptschleife des Programms wartet, bis der Benutzer einen Taster drückt.

```

            d = time.localtime()
            if d.tm_mon == 12 and d.tm_mday == 24:

```

Wird der Taster `t1` gedrückt, wird die aktuelle Zeit in der Datenstruktur `d` gespeichert und daraus ermittelt, ob gerade der 24. Dezember ist.

```

                subprocess.Popen(["omxplayer", "Stille_Nacht_Klavier.mp3"])
                blink()

```

Ist dies der Fall, werden über das vorinstallierte Linux-Kommandozeilenprogramm `omxplayer` das Weihnachtslied „Stille Nacht“ und die Lichteffekte aus der Funktion `blink()` abgespielt.

```

            else:
                mc.postToChat("Heute ist nicht Weihnachten")

```

An allen anderen Tagen erscheint nur eine Meldung im Minecraft-Chat, aber es gibt kein Weihnachtslied und auch keine Lichteffekte. Drückt der Spieler den zweiten Taster, wird das Weihnachtslied „O Tannenbaum“ abgespielt.

```

        except KeyboardInterrupt:
            mc.setBlocks(x-2, y+1, z-2, x+2, y+8, z+2, block.AIR.id)
            for i in range(3):
                pwm1[i].stop()
                pwm2[i].stop()
            GPIO.cleanup()

```

Das Programm läuft, bis es mit der Tastenkombination `[Strg]+[C]` abgebrochen wird. Danach werden der Baum gelöscht, die PWM-Signale gestoppt und die GPIO-Pins zurückgesetzt.

Frohe Weihnachten!



Liebe Kunden!

Dieses Produkt wurde in Übereinstimmung mit den geltenden europäischen Richtlinien hergestellt und trägt daher das CE-Zeichen.
Der bestimmungsgemäße Gebrauch ist in der beiliegenden Anleitung beschrieben.



Bei jeder anderen Nutzung oder Veränderung des Produktes sind allein Sie für die Einhaltung der geltenden Regeln verantwortlich.
Bauen Sie die Schaltungen deshalb genau so auf, wie es in der Anleitung beschrieben wird.
Das Produkt darf nur zusammen mit dieser Anleitung weitergegeben werden.

Das Symbol der durchkreuzten Mülltonne bedeutet, dass dieses Produkt getrennt vom Hausmüll als Elektroschrott dem Recycling zugeführt werden muss. Wo Sie die nächstgelegene kostenlose Annahmestelle finden, sagt Ihnen Ihre kommunale Verwaltung.



Achtung! Augenschutz und LEDs:

Blicken Sie nicht aus geringer Entfernung direkt in eine LED, denn ein direkter Blick kann Netzhautschäden verursachen! Dies gilt besonders für helle LEDs im klaren Gehäuse sowie in besonderem Maße für Power-LEDs. Bei weißen, blauen, violetten und ultravioletten LEDs gibt die scheinbare Helligkeit einen falschen Eindruck von der tatsächlichen Gefahr für Ihre Augen. Besondere Vorsicht ist bei der Verwendung von Sammellinsen geboten. Betreiben Sie die LEDs so wie in der Anleitung vorgesehen, nicht aber mit größeren Strömen.

Alle in diesem Buch vorgestellten Schaltungen und Programme wurden mit der größtmöglichen Sorgfalt entwickelt, geprüft und getestet. Trotzdem können Fehler im Buch und in der Software nicht vollständig ausgeschlossen werden. Verlag und Autor haften in Fällen des Vorsatzes oder der groben Fahrlässigkeit nach den gesetzlichen Bestimmungen. Im Übrigen haften Verlag und Autor nur nach dem Produkthaftungsgesetz wegen der Verletzung des Lebens, des Körpers oder der Gesundheit oder wegen der schulhaften Verletzung wesentlicher Vertragspflichten. Der Schadensersatzanspruch für die Verletzung wesentlicher Vertragspflichten ist auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht ein Fall der zwingenden Haftung nach dem Produkthaftungsgesetz gegeben ist.

Minecraft ist eine Marke der Mojang Synergies AB mit Sitz in Stockholm, Schweden.

Kein offizielles Minecraft-Produkt. Nicht von Mojang genehmigt oder mit Mojang verbunden.

© 2018 Franzis Verlag GmbH, 85540 Haar bei München

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträgern oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt.

Die meisten Produktbezeichnungen von Hard- und Software sowie Firmennamen und Firmenlogos, die in diesem Werk genannt werden, sind in der Regel gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

GTIN 4019631150233



FRANZIS
**ADVENTSKALENDER
 PROGRAMMIEREN MIT
 MINECRAFT™**



**DER ADVENTSKALENDER FÜR
 MINECRAFT™-FANS UND SOLCHE,
 DIE ES WERDEN WOLLEN**

**STEUERN SIE
 ELEKTRONIK MIT
 MINECRAFT™**

**In 24 Tagen vom Spieler zum Programmierer:
 So sinnvoll war Minecraft™ spielen noch nie.**

Du bist Minecraft™-Fan oder interessierst dich für das Spiel? Dann ist dieser Adventskalender genau das Richtige für dich: Verkürze dir die Wartezeit auf Weihnachten mit 24 tollen Minecraft™-Projekten.

Mit Minecraft™ verbringst du nicht nur viele kurzweilige Stunden mit spielen, sondern kannst auch programmieren lernen. Erstelle deine eigenen Minispiele und steige so in die Python-Programmierung ein. Im Adventskalender wird die Raspberry-Pi-Edition von Minecraft™ eingesetzt, mit der sich an der GPIO wunderbar Elektronik aus Minecraft™ heraus ansteuern lässt. Auch eine Eingabe über Elektronik ist möglich, z. B. wirst du Steve mit Knete steuern.

Über einen am Raspberry Pi angeschlossenen Piezo kannst du aus Minecraft™ heraus Töne wiedergeben.

Mit Hilfe des farbig illustrierten Handbuchs lassen sich alle 24 Experimente auch ganz ohne Vorkenntnisse aufbauen und programmieren. Alle notwendige Software wird kostenlos zum Download angeboten, darunter eine eigens für den Adventskalender erstellte Minecraft™-Welt.

Die Experimente funktionieren mit dem Raspberry Pi 3 oder dem Raspberry Pi 3 Modell B+. Der Raspberry Pi ist nicht enthalten. Für die Durchführung der Experimente ist ein Internetzugang erforderlich.

Zusätzlich benötigt:
 Raspberry Pi 3 oder Raspberry Pi 3 Modell B+

Minecraft ist eine Marke der Mojang Synergies AB
 mit Sitz in Stockholm, Schweden.

Dies ist kein offizielles Minecraft-Produkt.
 Nicht von Mojang genehmigt oder mit Mojang verbunden.

Für Kinder unter 14 Jahren nicht geeignet!

Bestellnummer: 1662788

Made in China

© 2018 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, D-85540 Haar, Germany
 Innovation, Irrtümer und Druckfehler vorbehalten
 2018/01



WEEE-REG.-NR.:
 DE 21445697

GTIN 4019631150233



4 019631 150233