

Abbildung 1: Hangman auf dem ESP32

Diesen Beitrag gibt es auch als:

[PDF in deutsch](#)

This episode is also available as:

[PDF in english](#)

In the previous episodes ([Ringmaster2](#) and [Bandit](#)) on the topic of "Games on ESP32 with MicroPython", neopixel rings played the main role. In today's episode, it would be conceivable to use them to display colorful results, but the focus is clearly on the OLED display with its text and graphic display. The latter leaves out the LCD keypad as a display option. The 5 keys would be sufficient, but nobody uses an LCD keypad just for the keys. That's why I'm using the 4x4 keypad again. And if chance and luck influenced the outcome of the game at Bandit, then the gift of combination is now the key to success. Welcome, let's program a well-known game together, which is also well suited as a vocabulary trainer

Hangman

Games on ESP32 with MicroPython

When describing the keypad.py module, I had already indicated that the 4x4 keypad in conjunction with three ASCII tables would allow the entire alphabet to be entered. While planning the Hangman, I thought carefully about using this solution to deliver letters to the ESP32. The mere mechanical problem of creating a proper label for the keyboard made me refrain from doing it every time.

Well, with Hangman you don't have to enter very long texts, individual letters are enough. This ultimately led to the programming of a screen keyboard on the OLED display. And because this display does not have a cursor, one was added in the program. You can find out how all of this works and a lot more in this article. Some of the hardware from the last episodes is used again.

1	ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 oder ähnlich
1	0,96 Zoll OLED I2C Display 128 x 64 Pixel - 1x OLED
1	4x4 Matrix Keypad Tastatur - 1x Keypad
1	MCP23017 Serielles Interface Modul
1	Battery Expansion Shield 18650 V3 inkl. USB Kabel
1	Li-Akku Typ 18650
1	LED Ring 5V RGB WS2812B 12-Bit 37mm als Option

Since we want to edit quite extensive word lists, which is logically very memory-intensive, an ESP32 was put into service again. Because the OLED display is addressed via I2C, I also reconnected the keyboard in this way. Only if you like things a little more colorful can you use a neopixel ring to indicate the approach to the solution. However, this is not absolutely necessary, just a nice accessory, but could inform you how many twelfths of the total number of letters have already been found.

Das Spiel

I designed the playing field in Libre-Office-Calc. With a lot of imagination you can guess what the later graphic will look like.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	A	B	C	D	E	F	G	H	I		A		-	-	-	
2	J	K	L	M	N	O	P	Q	R		B			\		
3	S	T	U	V	W	X	Y	Z			C			O		
4													/		\	
5													/		\	
6														-	-	-

Abbildung 2: Spielfeld

Let's take a look at the OLED keyboard right away. After selecting one of the 9 columns, we accept one of the three characters by pressing the A, B or C key on the 4x4 keypad. The corresponding character is then displayed in column J for about one second. If the letter appears in the solution word, it appears everywhere instead of the underscore. If it was a mistake, a piece of the gallows is set up. We all know that from our school days, when we played hangman on paper behind the back of the person in front in dreary geography or German lessons.

At the beginning of each game round we get 12 points on our account, they correspond to the 12 parts of the hangman. Every wrong letter not only completes the picture, but also subtracts a point. So it is not only important to have found all the letters of the answer at some point, but with as few wrong letters as possible, because at the end the remaining points are added to the total score. The latter is displayed until the next round is pressed at the touch of a button.

Again a random number is determined and, with this as an index, a new word is fetched from the word list. The word list resides as a file in the flash of the ESP32 until it is loaded into the RAM during a cold start. The words can have up to 36 characters, whereby umlauts and sharp "s" are noted as double sounds, as they do not appear in the display alphabet. Since the flash memory is almost inexhaustible, you could also store several such files there, for example on different topics. Hangman.py keeps a record of this so that words are not loaded multiple times during a session. Only when the whole list (60 terms) is through do repetitions inevitably occur. The word lists are created with any editor and saved as a text file in the workspace. With Thonny they are transferred from there to the ESP32.

So now you know what to expect. It all sounds very simple, but I'm sure you'd much rather know how to do this in MicroPython. This can be met, especially since this time it will be about a feature of the Python language in general. We will deal with this in the next chapter, where we will also discuss the other interesting parts of the hangman.py program. Then I will introduce the required software. Finally, it is about hardware construction and commissioning.

Tricks und Infos zu MicroPython

You are familiar with functions and have already used them many times to concentrate code that is used several times in a program in one place. This saves memory space and increases the readability of a program.

You also already know that objects are local in a function, unless they are declared as such in the function body by global. Local objects can only be referenced in the scope of the function. When MicroPython has left the function, it can no longer remember the local variables. The same applies to objects that were passed to the function as parameters.

Now it happens that further functions are defined within a function, as in the next example.

```
# alltest.py
def outer(x):
    n=3
    def mal():
        produkt=n*x
        return produkt
    return x+produkt
```

Start with F5 makes the function outer () and its content known to the global namespace (aka Scope). The syntax is correct, but the call returns an error message.

```
>>> outer (7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in outer
NameError: name 'product' isn't defined
```

Of course, outer (x) doesn't know product, because the variable is local to the inner function mal (). Of course, the global scope knows neither x nor n and certainly not product.

```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' isn't defined
```

```
>>> product
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'product' isn't defined
```

We get a similar error message for n.

Let's change the last line to get what we want.

```
def outer(x):
    n=3
    def mal():
        produkt=n*x
        return produkt
    return x+mal()
```

F5 and outer (7) return 28. As expected, the inner function mal () can read the n defined outside and also the x passed as a parameter. It looks different if we want to change the value of n within mal ().

```
# alltest.py
def outer(x):
    n=3
    def mal():
        n=n*x
        return produkt
    return x+mal()
```

F5 und outer(7):

```
>>> outer(7)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 7, in outer

File "<stdin>", line 5, in mal

NameError: local variable referenced before assignment

n is now interpreted by MicroPython as local to mal () due to the left side of the assignment. On the other hand, it is referenced in the product n * x before a value has been assigned to n. But even n = 8 * x does not deliver the desired result at this point, because after leaving mal (), n is still 3 instead of 56.

```
def outer(x):
    n=3
    def mal():
        n=8*x
        produkt=n*x
        print("innen",n)
        return produkt
    ergebnis=x+mal()
    print("außen",n)
    return ergebnis
```

F5 und outer(7):

```
innen 56
```

```
außen 3
```

```
399
```

However, none of this changes the fact that after exiting outer () none of the contents of the functions are known, let alone can be restored.

I mentioned the global keyword above, which makes it possible to tell a function about the higher-level scope of a variable. This makes it possible to change the value of a function permanently. While global refers to the top level, there is something comparable with nested functions through the keyword nonlocal.

```

def outer(x):
    n=3
    def mal():
        nonlocal n
        print("vorher",n)
        n=8*x
        produkt=n*x
        print("innen",n)
        return produkt
    ergebnis=x+mal()
    print("außen",n)
    return ergebnis

```

F5 und outer(7):

```

>>> outer(7)
vorher 3
innen 56
außen 56
399

```

As you can see, it is now possible to assign a new value to the variable `n` lying outside `mal()` from the `mal()` function.

Now we go one step further and ensure that the inner function `mal()` can still be called after exiting `outer()`. Even more, we also ensure that the inner function also remembers the non-local variable `n` and even the parameter `x` of the outer function `outer()`. This "secret" from MicroPython is called Closure (pronounced: "Klouscher"). What's behind it?

Functions can be treated as values in MicroPython. That means you can assign a function to a variable, pass it as a function parameter or return it as the result of a function. Please note that I'm not talking about a function value here, but the function itself. Let's examine the next example.

```

def outer(x):
    n=0
    def mal():
        nonlocal n
        print("vorher",n)
        n=n+1
        produkt=n*x
        print("nachher",n)
        return produkt
    return mal

```

F5 und h=outer(2)

```

>>> h=outer(2)
>>> h()
vorher 0
nachher 1
2
>>> h()
vorher 1
nachher 2
4
>>> h()
vorher 2
nachher 3
6

```

Did you notice that `outer()` doesn't return a function value this time like it did in the earlier examples? The outer function returns the inner function `mal` that we assign to the variable `h`. This means that the function `mal` can be called, even if we have already left the scope of `outer()`. And two more things are absolutely noteworthy. `h ()` alias `mal ()` remembers both the parameter `2`, which we passed to `outer ()` in `x`, and the variable `n` located outside `mal`. Much more, it is even possible, to increase the value of `n` with each call automatically.

You can create the multiplication table by 5 in a similar way.

```

>>> k=outer(5)
>>> k()
vorher 0
nachher 1
5
>>> k()
vorher 1
nachher 2
10
>>> k()
vorher 2
nachher 3
15

```

The inner function is a closure. Calling `h` or `k` without the function brackets also tells us so.

```

>>> h
<closure>

```

In general, this is a function that is surrounded by another and has an extended namespace, in which the non-local variables of the surrounding function are included if they occur in the inner function. A local variable such as `produkt` in the scope of the inner function is called bound (aka bound variable), a function that only contains bound variables is a closed expression (aka closed term). Free variables (aka free variables), i.e. non-local variables, are those that are defined outside of a function. A function that contains free variables is an open term. Closure is a function that quasi

captures its non-local variables and thus turns the open term into a closed one, hence the name closure.

What can we do with closures now? Now we have to draw a stick figure on the gallows. This consists of 12 line elements. I have stored the commands for this as strings in the elements of the list `gallows`, which are together with a counter variable `n`, both as free objects, in the function `initGalgens()`. The inner function `nextPart()` is also in the scope of `initGalgens()`. For `nextPart`, `gallows` and `n` are non-local, i.e. free variables. With the return of the `nextpart` function by `initGalgens()`, the inner function becomes a closure that remembers the captured `gallows` and `n` when called repeatedly. Since `n` is also shown as `nonlocal extra`, this free variable can also be changed with an external effect using `nextPart()`. We use this to implement an automatic counting function. If we now assign the return of `initGalgens` to the variable `draw`, we can automatically draw the next line of our picture with each call to `draw()` without having to worry about a sequence. Furthermore, `drawing()` tells us with the return value `False` that the round has expired, what more do we want?

Note:

The ability to remember the value of a local variable can be achieved in C and the Arduino IDE by declaring the variable as `static` in the function.

```
def initGalgens():
    n=0
    gallens=[
        "d.fill_rect(d.width-25,d.height-3,24,3,1)", # Boden
        "d.fill_rect(d.width-3,d.height//2,3,d.height//2,1)",
        "d.fill_rect(d.width-3,0,3,d.height//2,1)", # Mast
        "d.fill_rect(d.width-25,0,24,3,1)", # Balken
        "d.line(d.width-18,2,d.width-2,18,1)", # Stuetze
        "d.fill_rect(d.width-22,0,3,15,1)", # Strick
        "d.writeAt('O',13,1,False)", # Kopf
        "d.fill_rect(d.width-22,19,3,20,1)", # Body
        "d.line(d.width-22,18,d.width-26,33,1)", # Arm links
        "d.line(d.width-19,18,d.width-14,33,1)", # Arm rechts
        "d.line(d.width-22,37,d.width-26,50,1)", # Bein links
        "d.line(d.width-19,37,d.width-14,50,1)", ]# Bein rechts
    def nextPart():
        nonlocal n
        exec(gallens[n])
        d.show()
        n+=1
        if n<=11:
            return True
        else:
            return False
    return nextPart
```

```
>>> zeichnen = initGalgens()
>>> zeichnen()
>>> zeichnen() ...
```

The command `exec(gallows[n])` is, in itself, a piece of candy that you have to let melt in your tongue. It enables the execution of MicroPython code that is available as

a string. Again just right for our project. However, this instruction also has an increased risk potential, namely when the string can be entered by a user using an input command or even via a web portal.

The creation of the search word list and the word selection from the same, including repetition protection, is also packaged as a closure. As long as there are still new words available, the latter is guaranteed by the numbers list by appending the currently rolled index for the word list to numbers. Indices in this list are not used a second time for as long as possible until the next cold start. You can very probably already explain the structure and function of `initSuperWord ()` yourself. But what about the explanation of the following statement?

```
>>> initSuperWord>()  
0  
'Ouverture'
```

Let's now discuss the rest of the program, three more functions and the rather short main program are still waiting for it.

The `showString ()` function brings the term you are looking for in its development stages onto the display during the game. The `OLED` class is used to control it, for which there are some innovations compared to the previous version. There is a new instance variable `yOffset`, which with the methods `writeAt ()` and `clearFT ()` shifts the reproduction or deletion of the text lines by up to 4 pixel positions down. This creates space for a cursor line at the top of the display. This cursor marks the selection column for characters from the screen alphabet.

```
cursor (x, h, y = 0, cset = True, show = True)
```

The parameter list includes the text column number and the line width `h` (1..4). `cset = True` sets the Cursor, `False` deletes it and `show = True` writes immediately into the display.

The `blinkDisplay (cnt, off = 0.2, on = 0.8)` method does what its name says. With `cnt` you indicate the number of flashes.

In order to understand why the output of the answer word looks so complicated, a short detour into the main program is recommended. The search word `s` is read into a list as a string from the "superwords.txt" file. Strings are immutable, which means that individual characters of a string cannot simply be changed in the same way as elements of a list.

```
>>> t="test"  
>>> t[2]="x"  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object doesn't support item assignment
```

There is a string method `replace ()`, but it does not work in a position-specific manner, but globally via the string. So you cannot replace individual characters only in a very specific place.

But that is exactly what we have to do. Instead of separating the string at the desired places, replacing the underscore with the character and putting the whole thing back together into a new string, we are going a more memory-saving way. The solution term is initially a string with the same number of "_" as there are characters in the search word. This "_" must now be replaced by the selected letter in all places where it appears in the search term. Therefore we convert the concept of the solution into a byte array, the elements of which can be changed in a targeted manner, even in several places.

```
s = getSuperWord ()
print ("Superword =", s)
s = s.upper ()
U = "_" * len (s)
Ub = bytearray (U.encode ("utf8"))
showString (Ub)
```

`showString ()` does not take the solution string, but the equivalent byte array `Ub`. The byte array `Ub`, which contains the ASCII codes of the characters, is derived from the string `U` via the bytes object `U.encode ("utf8")`. Of course, these codes have to be translated back into ASCII characters for output. This is what the function `chr ()` does. The process is complex but practical.

In addition, the term "solution" must be broken up to three lines. This happens in the two for loops after the integer division of its number of characters and the remainder modulo the line length.

In the `getChar ()` function, we use the "*" and "#" keys to place the cursor in the column from which one of the three characters is to be fetched. If one of the keys "A", "B" or "C" is pressed, the function picks the character from the line string of the alphabet and returns it. The function `eval ()` supports us again in a tricky way.

```
A="ABCDEFGH I "
B="JKLMNOPQR "
C="STUVWXYZ  "
.....
        z=eval (taste) [cursorPos]
        d.writeAt (z, len (A) +1, ord (taste) -65)
        d.writeAt (" ", cursorPos, ord (taste) -65)
```

The code letter of the line, A, B or C is contented in the variable `taste`. `eval ()` turns it into the variable A, B or C and `cursorPos` points to the position in the string. The character is output in front of the line identifier. The line number is cleverly derived from the ASCII code of the line selection. The third line of code removes the character from the alphabet selection for the sake of clarity.

The "D" key ends the entire game. With the delay parameter, we specifically slow down the cursor movement, which makes positioning easier.

The checkChar (c = "") function takes the character we selected and checks whether it appears in the search word string. If this is the case, we look for the positions there, but replace the "_" in the byte array Ub with the ASCII code of the character and return True. That means we can keep guessing. So that Ub can be changed within the function, the byte array must be registered as global.

If we made a mistake, a point will be deducted and the next line will be entered in the drawing. Our Closure nextPart () in initGallows (), which we assigned to the variable gallows, does this reliably. If there are dashes left, gallows () returns True, otherwise False.

```
gallows = initGalgen ()  
.....  
return gallows ()
```

With the closure gallows () you can now surely see the advantage that this function gives us. It is drawn and the status is updated, all automatically with the same call gallows ().

The rest of the main program is quickly explained, in addition to around 20 trivial lines of opening and closing credits, which are largely self-explanatory, there is not much left. The game round continues until there is no "_" left in the solution array or the last line of the hanged man is drawn.

```
while ("_" in Ub) and result == True:  
    character = getChar ()  
    sleep (1)  
    result = checkChar (character)
```

Oh yes - you better remove the following line in the outer while loop when the ESP32 is connected to the PC, because the print command outputs the search term in the terminal.

```
print ("Superword =", s)
```

Here the [hangman.py](#) program comes in one piece

```
# File: hangman.py  
# Author: Jürgen Grzesina  
# Rev. 1.0  
# Stand 16.06.2021  
# *****  
# Importgeschaefte  
# *****  
import os,sys          # System- und Dateianweisungen
```

```

import esp          # nervige Systemmeldungen aus
esp.osdebug(None)

import gc           # Platz fuer Variablen schaffen
gc.collect()
#
from oled import OLED
from ssd1306 import SSD1306_I2C
from machine import Pin,I2C
from keypad import KEYPAD_I2C,KEYPAD
from i2cbus import I2Cbus
from mring import MAGIC_RING
from time import sleep

i2c=I2C(-1,scl=Pin(21),sda=Pin(22))
ibus=I2Cbus(i2c)

d=OLED(i2c,128,64)
keyHwadr=0x20 # HWADR des Portexpanders fuer das 4x4-Pad
kp=KEYPAD_I2C(ibus,keyHwadr) # Hardware Objekt am I2C-Bus
k=KEYPAD(kp,d=d) # hardwareunabhaengige Methoden

mr=MAGIC_RING(neoPin=12,neoCnt=12)

d.yOffset=4
A="ABCDEFGH I"
B="JKLMNOPQR"
C="STUVWXYZ "

s="JOGILOEWSOBERFLASCHENELFVEREIN"
U="_"*len(s)
sb=bytearray(s.encode("utf8"))
Ub=bytearray(U.encode("utf8"))
alphaLength=12

# ***** Funktionen declarieren *****
def showString(q=Ub): # q ist ein bytearray mit den
Stringcodes
    laenge=len(q)
    zeilen=laenge//alphaLength
    rest=laenge%alphaLength
    for line in range(zeilen):
        for pos in range(alphaLength):

d.writeAt(chr(q[line*alphaLength+pos]),pos,line+3,False)
        for pos in range(rest):

d.writeAt(chr(q[zeilen*alphaLength+pos]),pos,zeilen+3,False)
        d.show()

def getChar(delay=0.3):

```

```

global cursorPos
d.cursor(cursorPos, 3, y=0, cset=True, show=True)
d.clearFT(len(A)+1, 0, len(A)+1, 2)
while 1:
    d.cursor(cursorPos, 3, y=0, cset=True, show=True)
    taste=k.waitForKey(timeout=0, ASCII=True)
    sleep(delay)
    if taste=="*":
        d.cursor(cursorPos, 3, y=0, cset=False, show=True)
        cursorPos=(cursorPos-1)%9
    if taste=="+":
        d.cursor(cursorPos, 3, y=0, cset=False, show=True)
        cursorPos=(cursorPos+1)%9
    if taste in "ABC":
        z=eval(taste)[cursorPos]
        d.writeAt(z, len(A)+1, ord(taste)-65)
        d.writeAt(" ", cursorPos, ord(taste)-65)
        return z
    if taste=="\x0d":
        d.clearAll()
        d.writeAt("GAME OVER", 3, 2)
        sys.exit()

def checkChar(c=""):
    global Ub
    global punkte
    pos=[]
    if c in s:
        p=-1
        while 1:
            p=s.find(c, p+1)
            if p!=-1:
                Ub[p]=ord(c) # gefunden und ersetzt
            else:
                break # fertig mit c
        showString(q=Ub) # anzeige updaten
        return True
    else:
        punkte-=1
        return gallows()

def initGalgen():
    n=0
    galgen=[
        "d.fill_rect(d.width-25,d.height-3,24,3,1)", # Boden
        "d.fill_rect(d.width-3,d.height//2,3,d.height//2,1)",
        "d.fill_rect(d.width-3,0,3,d.height//2,1)", # Mast
        "d.fill_rect(d.width-25,0,24,3,1)", # Balken
        "d.line(d.width-18,2,d.width-2,18,1)", # Stuetze
        "d.fill_rect(d.width-22,0,3,15,1)", # Strick
        "d.writeAt('O',13,1,False)", # Kopf
        "d.fill_rect(d.width-22,19,3,20,1)", # Body
    ]

```

```

        "d.line(d.width-22,18,d.width-26,33,1)", # Arm links
        "d.line(d.width-19,18,d.width-14,33,1)", # Arm rechts
        "d.line(d.width-22,37,d.width-26,50,1)", # Bein links
        "d.line(d.width-19,37,d.width-14,50,1)", # Bein rechts
    ]
def nextPart():
    nonlocal n
    exec(galgen[n])
    d.show()
    n+=1
    if n<=11:
        return True
    else:
        return False
return nextPart

def initSuperWord():
    L=[]
    f=open("superwords.txt","r")
    for w in f:
        w = w.strip(" \r\n\t")
        L.append(w)
    f.close()
    numbers=[]
    #print(L)
    def findOne():
        nonlocal numbers
        guess = os.urandom(3)[2]%len(L)
        print(len(numbers))
        while guess in numbers and len(numbers)<len(L):
            guess = os.urandom(3)[2]%len(L)
        if len(numbers)<len(L):
            numbers.append(guess)
        return L[guess]
    return findOne

# ***** Hauptprogramm *****
showString(Ub)
games=0
gesamt=0
getSuperWord=initSuperWord()
while 1:
    punkte=12
    d.clearAll()
    d.writeAt(A+"  A",0,0,False)
    d.writeAt(B+"  B",0,1,False)
    d.writeAt(C+"  C",0,2,True)
    cursorPos=0
    gallows=initGalgen()
    s=getSuperWord()
    print("Superword=",s)
    s=s.upper()

```

```
U=" "*len(s)
Ub=bytearray(U.encode("utf8"))
showString(Ub)
games+=1
ergebnis=True
while ("_" in Ub) and ergebnis==True:
    zeichen=getChar()
    print(zeichen)
    sleep(1)
    ergebnis=checkChar(zeichen)
if ergebnis == True:
    gesamt+=punkte
    d.blinkDisplay(3,0.5,0.5)
    d.clearFT(0,0,12,2)
    d.writeAt("GEWONNEN!!!",0,0,False)
    d.writeAt("PUNKTE {}".format(punkte),0,1,False)
else:
    d.clearFT(0,0,12,2)
    d.writeAt("NEW GAME -",0,0,False)
    d.writeAt("NEW LUCK!!",0,1,False)
    showString(q=sb)
d.writeAt("TOTAL {}".format(gesamt),0,2)
taste=k.waitForKey(0,ASCII=True)
if taste=="\x0d":
    print("Game over")
    d.clearAll()
    d.writeAt("GAME OVER",3,2)
    break
```

The Software

Verwendete Software:

For flashing and programming the ESP32:

[Thonny](#) oder

[µPyCraft](#)

Verwendete Firmware:

[MicropythonFirmware](#)

Please use only stable versions.

MicroPython-Module und Programme

[keypad.py](#) Modul für Tastenfeld-Unterstützung

[mcp.py](#) Modul für Porterweiterungsbaustein MCP23017

[i2cbus.py](#) zum Austausch verschiedener Datentypen

[oled.py](#) die API zur Ansteuerung des OLED-Moduls

[ssd1306.py](#) der Hardwaretreiber für das Display

[mring.py](#) Muster-Treiber für Neopixel-Ringe

[hangman.py](#) Hauptprogramm

You can find [detailed instructions](#) for installing Thonny here. There is also a description of how the [Micropython firmware](#) is burned to the ESP32.

MicroPython is an interpreter language. The main difference to the Arduino IDE, where you always and exclusively flash entire programs, is that you only have to flash the MicroPython firmware once on the ESP32 at the beginning so that the controller understands MicroPython instructions. You can use Thonny, µPyCraft or esptool.py for this. I have described the process for Thonny here

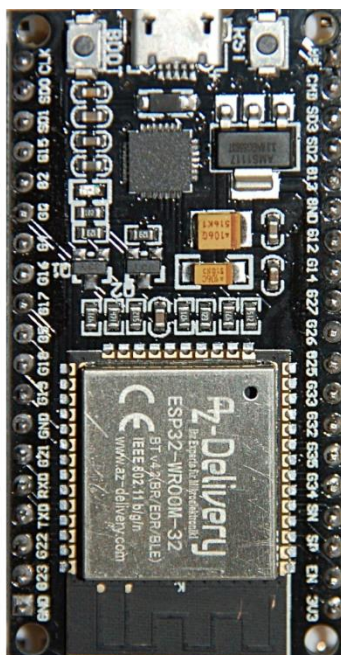
As soon as the firmware is flashed, you can have a casual conversation with your controller, test individual commands and immediately see the answer without first having to compile and transfer an entire program. This is exactly what bothers me about the Arduino IDE. You simply save an enormous amount of time if you can do simple tests of the syntax and hardware through to trying out and refining functions and entire program parts via the command line before you knit a program out of it. For this purpose I also like to create small test programs over and over again. As a kind of macro, they combine recurring commands. From such program fragments, entire applications can develop. If the program is to start autonomously when the controller is switched on, copy the program text into a newly created blank file. Save this file under boot.py in the workspace and upload it to the ESP32 / ESP8266. The program starts automatically the next time it is reset or switched on.

Programs are started manually from the current editor window in the Thonny IDE using the F5 key. This is faster than clicking the start button or using the Run menu. Only the modules used in the program must be in the flash of the ESP32.

If you later want to use the controller together with the Arduino IDE again, simply flash the program in the usual way. However, the ESP32 / ESP8266 then forgot that it ever spoke MicroPython. Conversely, any Espressif chip that contains a compiled program from the Arduino IDE or the AT firmware or LUA can easily be provided with the MicroPython firmware. The process must always be carried out as described [here](#)

Circuit design

The circuit for "Ring Master 2" can also be used for Hangman without modification. If you want to use a 5V power supply instead of the battery holder and the Li battery, you have to connect the 5V to pin 20, Vin, of the ESP32. The 3.3V pin of the ESP32 then supplies the I2C parallel converter for the mi keyboardt.



The image shows a top-down view of the ESP32-DEVKITC_V3 development board. The board is populated with various components including a USB-C port, a USB-A port, a micro-USB port, a battery holder, and an ESP32-WROOM-32 module. The pin headers are clearly visible, and the pinout table is overlaid on the image.

CLK	1	20	Vin
SD0	2	21	CMD
SD1	3	22	SD3
TP3 ADC2-3	G15 4	23	SD2
TP2 ADC2-2	G2 5	24	G13 ADC2-4 TP4
TP4 ADC2-1	G0 6	25	GND
TP0 ADC2-0	G4 7	26	G12 ADC2-5 TP5
	G16 8	27	G14 ADC2-6 TP6
	G17 9	28	G27 ADC2-7 TP7
	G5 10	29	G26 ADC2-9
	G18 11	30	G25 ADC2-8
	G19 12	31	G33 ADC1-5 TP8
GND	13	32	G32 ADC1-4 TP9
G21	14	33	G35 ADC1-7
RXD	15	34	G34 ADC1-6
TXD	16	35	SN/G39 ADC1-3
G22	17	36	SP/G36 ADC1-0
G23	18	37	EN/RST
GND	19	38	3V3

*

Abbildung 3: ESP32-DEVKITC_V3_Pinout

The supply from a 4.5V block of alkaline cells can also be used. The connection also goes to pin 20 of the ESP32. However, if you want to plan a neopixel ring, you should derive a separate 3.3V supply for it. The 3.3V output of the ESP32 doesn't do all of that. An [AMS1117 3.3V power supply module for Raspberry Pi](#) can be used as a component. For supply voltages above 5V, an extra 5V regulator must then be used, because the neopixel ring must not receive more than 5.3V. By the way, old PC power supplies are very suitable for experimenting because they provide 3.3V and 12V in addition to 5V. This means that even hungry power guzzlers can be satisfied

The following figure 4 shows the circuit diagram. You can download a [better readable copy in DIN A4 as a PDF file](#)

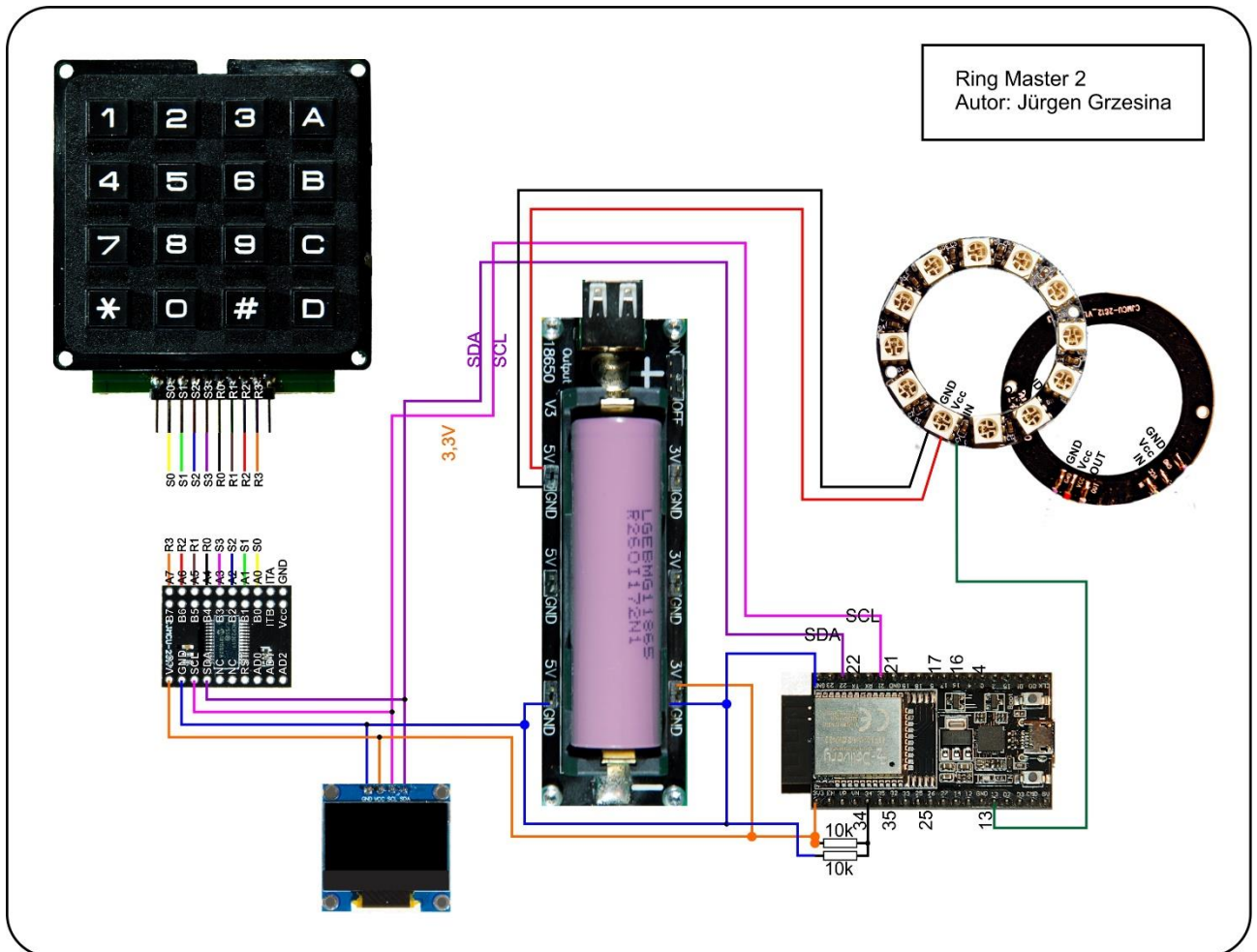


Abbildung 4: Masterring2

The keyboard is connected in such a way that lines of the same color (or lines of the same name) meet those of the MCP23017. So that you can do this with jumper cables, the keyboard board must be provided with an 8-pin (angled) pin header. The two outermost solder pins remain unconnected. The module with the MCP23017 also has two connector strips, namely the two outer rows with the pins up towards the component side, the inner row has a pin or socket strip downwards. If the circuit board is now inserted into a breadboard, the labeled underside of the board points upwards, which makes wiring much easier.

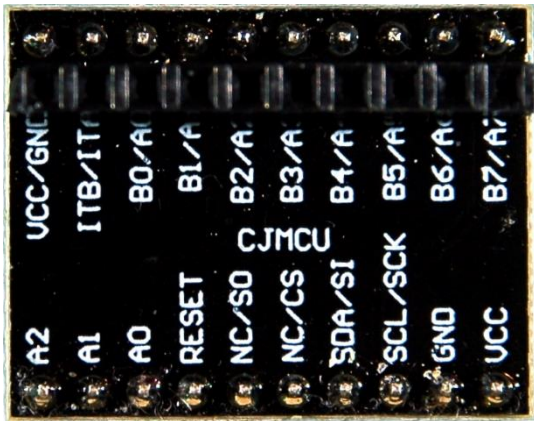


Abbildung 5: MCP23017_Unterseite

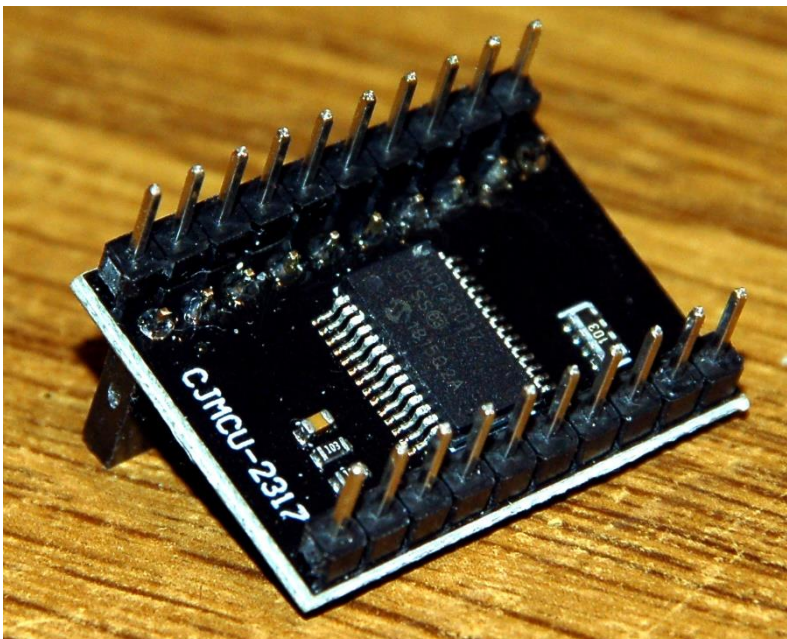


Abbildung 6: MCP23017_Bestückungsseite

Only one step is necessary for autonomous operation without a PC. If we move the program text from hangman.py into the boot.py file and send it back up to the ESP32, the controller automatically starts the game after a reset.

Have fun building, programming and playing.

Links zum Thema Spiele:

[PDF in deutsch](#)

[PDF in english](#)

[Ringmaster2](#) Farbenraten mit Neopixelring
bandido.py der einarmige [Bandit](#)