Reaktionsspiel

Dokumentation zur Projektaufgabe
im Fach Programmieren I bei
Thorsten Wagener

Inhaltsverzeichnis

1. Aufgabenstellung

Mit Hilfe der Programmiersprache Python sollte ein Code zur Ansteuerung von LEDs geschrieben und in einem realen Projekt auf einem Raspberry PI ausgeführt werden.

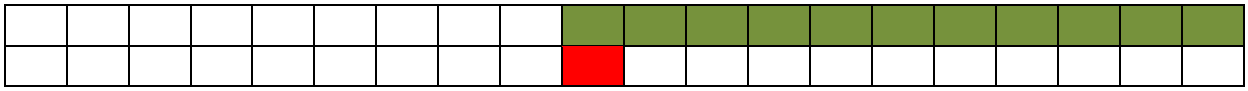
2. Idee

Programmieren 1 Projekt: Reaktionsspiel

Idee: Auf einem LED-Streifen leuchten die LEDs von rechts nach links auf. Dem Spieler geht es darum, das Signal an einer durch einen Pointer markierten Stelle zu stoppen. Dies wird in zwanzig aufeinanderfolgenden Leveln gespielt und immer schwieriger, da die Zeitabstände zwischen den Leveln variieren und die Geschwindigkeit des durchlaufenden Signals immer schneller wird.

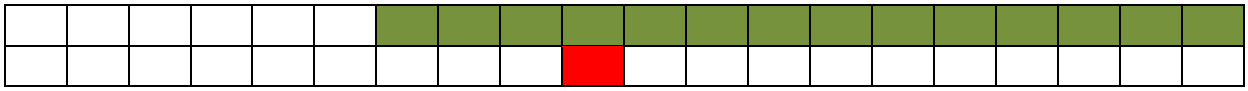
1. Runde (Signal läuft durch):

Treffer:



Fehler – zu früh:

Fehler – zu spät:



zufällige Neuplatzierung des Markers:

2. Runde – 20. Runde:

Neustart durch Knopfdruck

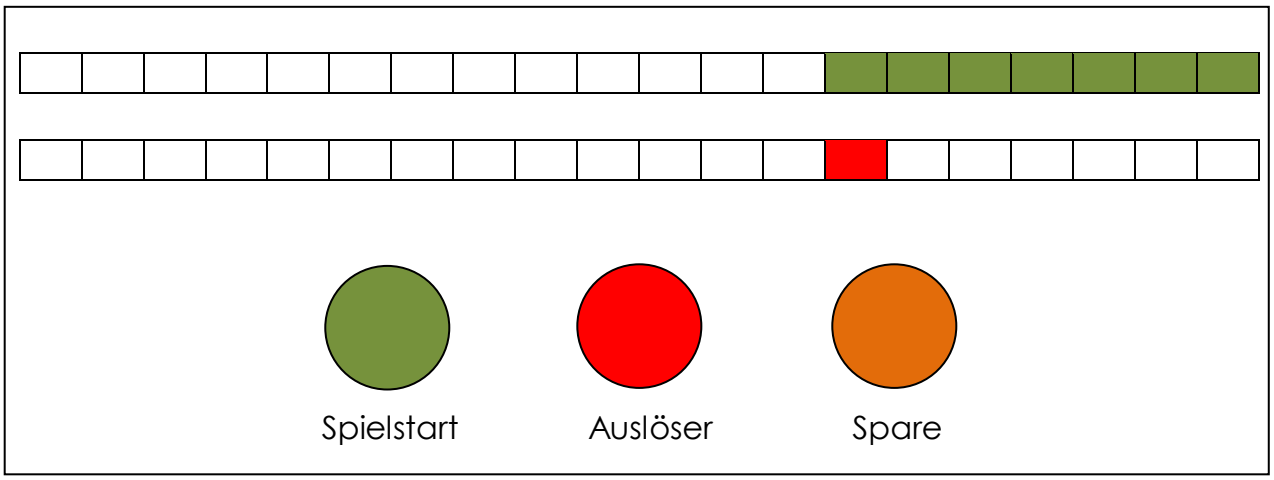
3. Material und Kosten

Knöpfe:

1x Spielstart	grün	
1x Auslöser	rot	
1x Spare	orange	(für die Einführung neuer Spielmodi)

Materialkosten:		Preis
1x	Raspberry PI Model 2	25 €
3x	Arcade Retro Button (drei verschiedene Farben)	11 €
40x	WS2812 LED	15 €
1x	LED-Netzteil 12V DC	10 €
1x	Gehäuse für das Spiel (3D-Druck)	05 €
		66 €

Oberseite:



LED-Nummerierung:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20

4. Probleme und Lösungen

Autostart-Problem:
Die Software muss beim Einstecken der Stromversorgung automatisch starten, damit ein Eingriff über den System-Editor zum manuellen Start der Software nicht notwendig ist.

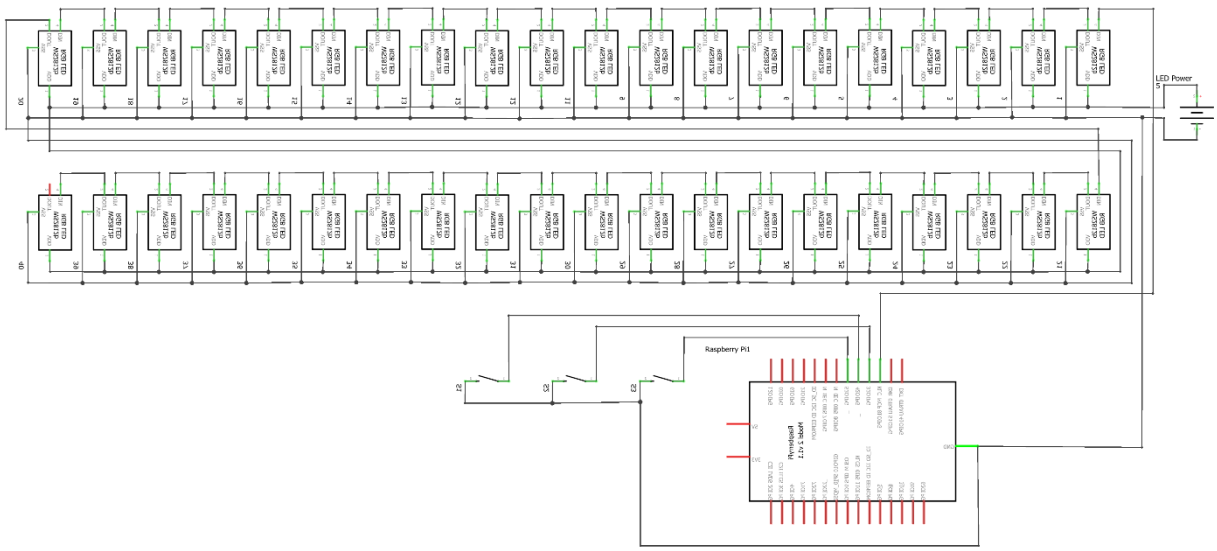
Das Problem wurde über die Einbindung von *sudo python /home/pi/ReactionGameFinal.py* in die Datei */etc/rc.local* gelöst.

5. weitere Ideen

Ausschalten des Pointers nach kurzem Aufblinken:
Nach kurzem Aufblinken wird der Pointer abgeschaltet. Der Spieler muss den Punkt, an welchem er den Streifen stoppen muss, aus dem Gedächtnis rekapitulieren.

2-Spieler-Streifen-Spiel:
Zwei Lichtstreifen laufen gegenläufig aufeinander zu. Eine auf den oberen zwanzig LEDs, die andere auf den unteren zwanzig LEDs. Das Ziel des Spiels ist es, die Schlangen möglichst nah am Mittelpunkt zu stoppen, aber die zweite Schlange nicht zu treffen/zu überschneiden. Das Spiel ist für zwei Spieler ausgelegt, wobei der orange Knopf die Funktion des Auslösers für den zweiten Spieler übernimmt.

6. Schaltplan



7. Code

```
import random                # Import des Random-Moduls zur Generierung von Zufalls-Zahlen
import time                  # Import des Zeit-Moduls für die time.sleep-Funktion (entspricht einem Delay)
from enum import Enum        # Import des Enum-Moduls (Set aus symbolischen Namen für konstante
                             Werte) für spätere Spielvariationen
import RPi.GPIO as GPIO      # Import des GPIO-Moduls (GPIO = General Purpose Input Output) zur
                             Ansteuerung von Inputs und Outputs
from neopixel import *       # Import alles Funktionen des Neopixel-Moduls zur Ansteuerung des
                             LED-Strips

LED_COUNT = 40               # Anzahl aller verwendeten LEDs auf dem Strip
LED_PIN = 18                 # Verwendeter PIN zur Steuer-Verbindung mit den LEDs
LED_FREQ_HZ = 800000         # LED-Signal-Frequenz in Hertz
LED_DMA = 5                  # DMA-Kanal zur Signals-Generierung
LED_BRIGHTNESS = 155         # Helligkeit der LEDs in 8-bit
LED_INVERT = False           # Invertierung des Signal (geschieht nur bei LED-Invert = True)
LED_CHANNEL    = 0           # Kanal der LEDs

GPIO.setmode(GPIO.BCM)       # Zuordnung der Nummerierung der GPIO-Pins, in diesem Fall werden
                             die Eingangsnamen verwendet (bspw. GPIO23 ist Eingang 23),
                             alternativ wäre eine Nummerierung nach Position des Pins auf der
                             Platine möglich (GPIO.setmode(GPIO.BOARD))

GPIO.setup(23,GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Konfiguration des GPIO23 als Input und
                                                    Aktivierung des Pull-Up-Widerstands (Standart-
                                                    Wert des Inuts ist HIGH)
GPIO.setup(24,GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Konfiguration des GPIO24 als Input und
                                                    Aktivierung des Pull-Up-Widerstands (Standart-
                                                    Wert des Inuts ist HIGH)

current_level = 1            # das aktuelle Level wird auf 1 gesetzt (erstes Level)
game_alive = True            # der Spielzustand wird auf True gesetzt (das Spiel kann beginnen)


strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT, LED_BRIGHTNESS,
LED_CHANNEL)                 # Erstellung des LEDs-Objekts und Konfiguration
strip.begin()                # Initialisierung der Bibliothek/des Moduls


class Game:                  # Definition der Klasse "Game"

    def __init__(self):      # Definition von "__init__"
        self.current_position = [0]     # die akutelle Position des Licht-Streifens ist 0 (erste Position)
        self.current_pointer_pos = random.randint(23,38)    # die Postion des Pointers ist eine Zufallszahl
                                                            zwischen 23 und 38
        self.current_level = 1          # das aktuelle Level wird innerhalb der Klasse noch einmal auf 1
                                        gesetzt (erstes Level)
        self.row_one = range(0,19)      # Definition der oberen LED-Reihe mit LEDs der Nummern 0 bis 19
```

```
    self.alive = True       # der aktuelle Spielzustand wird innerhalb der Klasse noch einmal auf True
                            gesetzt (das Spiel kann beginnen)


def set_speed(self, current_level):     # Definiton von "set_speed" (funktioniert als Delay)
    speed = (0.75 / current_level)      # Definition der Rechenoperation zur Berechnung der aktuellen
                                        Spiel-Geschwindigkeit
    return speed                        # Rückgabe des Speed-Wertes an die Funktion

def level_move(self):           # Definition von level_move (die Abfolge des eigentlichen Spiels)


    for i in range(19):             # for-Schleife, welche von 0 bis 19 zählt
        head_pos = self.current_position[-1]        # Definition von "head_pos" als letzten Eintrag des
                                                    Arrays "current_position" aus "__init__"
        head_pos += 1                   # Addition von +1 zum Wert von "head_pos"
        move_position = head_pos        #Defintion von "move_position" als Wert von "head_pos"

        strip.setPixelColorRGB(self.current_pointer_pos, 0,255,0)       # setzt den Pixel (also den Pointer) des
                                                                        Nummern-Wertes von
                                                                        "current_pointer_pos" aus "__init__" auf
                                                                        die Farbe rot
        strip.show()        # gibt die Farbe des Pixels auf dem LED-Strip aus
        self.current_position.append(move_position)         # fügt dem Array "current_position" aus
                                                            "__init__" den Wert von "move_position" als
                                                            letzten Eintrag neu hinzu
        strip.setPixelColorRGB(head_pos, 0,0,255)           # setzt den Pixel des Nummern-Wertes von
                                                            "head_pos" auf die Farbe blau -> eigentliche
                                                            Animation des oberen Farbstreifens, da der
                                                            Wert von "head_pos" bei jedem Durchgang
                                                            der for-Schleife um 1 erhöht wird
        strip.show()                                    # gibt die Farbe des Pixels auf dem LED-Strip aus

        speed = self.set_speed(current_level)           # setzt "speed" auf den Wert von "set_speed"
                                                        (abhängig vom jeweiligen Level)
        time.sleep(speed)                               # Verzögerung um den Wert von "speed" in Sekunden

        if GPIO.input(23) == False:                     # if-Schleife für den Fall, dass der Button "Auslöser"
                                                        gedrückt wird
            for i in range (0,39):                      # for-Schleife, welche von 0 bis 39 zählt
                strip.setPixelColorRGB(i, 0,0,0)        # setzt den Pixel des Nummerwertes "i" auf 0, also
                                                        schaltet ihn ab -> da die Schleife von 0 bis 39 läuft,
                                                        werden alle LEDs abgeschaltet
            break                                       # Abbruch der gesamten for-Schleife
```

```
# hier startet der eigentliche Programm-Code:

while True:        # while True-Schleife -> in diesem Falle eine Endlos-Schleife

    while not GPIO.input(24) == False:      # while not-Schleife für den GPIO24 -> falls dieser nicht
                                            gedrückt wird, läuft eine Endlos-Schleife und nichts passiert
                                            (dies ist der Startmodus des Spiels und der Modus nach allen
                                            abgeschlossenen Levels)

        time.sleep(0.0001)          # Verzögerung um den Wert 0.0001 in Sekunden
        game_alive = True           # der Spielzustand wird auf True gesetzt (das Spiel kann beginnen)
        current_level = 1           # das aktuelle Level wird auf 1 gesetzt (erstes Level)

    else:                           # falls der Button GPIO24 "Spielstart" gedrückt wird, wird diese Option
                                    ausgeführt
        while game_alive == True:   # while-Schleife mit der Bedingung, dass game_alive == True ist (wird im
                                    while not-Zustand sowie am Anfang definiert)

            game = Game()           # Initialisierung und Definiton von "game" der Klasse "Game()"
            current_level +=1       # Addition von +1 zum Wert von "current_level"
            if current_level == 20: # if-Schleife für den Fall, dass "current_level" den Wert von 20 erreicht ->
                                    Überprüfung, dass das Level noch nicht das letzte Level ist
                game_alive = False  # Definition von "game_alive" als False -> Abbruch der while-Schleife ->
                                    das Spielende ist erreicht und der Code springt zurück in den Warte-
                                    Zustand der "while not GPIO.input(24) == False-Schleife", bis erneut
                                    Spielstart gedrückt wird

            game.level_move()
```

8. Fotos