Handreichung PhC

1. Kurzvorstellung

Simulation eines Morsetasters und Decodierung von Morsecodes

2. Einordnung in die Lehrpläne

Oberschule	Gymnasium	Berufliches Gymnasium
Klassenstufe 8, Lernbereich 2: *Informationen verarbeiten: Modell – Algorithmus – Lösung* • Algorithmusbegriff • Grundlegende Programmstrukturen • Problemlöseprozess		
Klassenstufe 10, Lernbereich 2: *Arbeit in Projekten* • Gestalten eines eigenen Projektes	Klassenstufen 9/10, Lernbereich 4: *Algorithmen und Programme* • Algorithmusbegriff • Algorithmische Grundstrukturen • Problemlöseprozess	
	Jahrgangsstufen 11/12 (Grundkurs), Lernbereich 8 B: *Technische Informatik – Hardware und Prozessdatenverarbeitung* • Messen	Jahrgangsstufen 12/13 (Grundkurs), Lernbereich 3: *Algorithmen und Programme* • Grundbegriffe • Grundstrukturen Lernbereich 4B: *Projekt Technische Informatik* • Gestalten eines Projektes zur Steuerung und Regelung von Prozessen

Max Böhm

3. Lernziele

Kognitive Lernziele: Die Schüler:innen…

- wenden gelernte Grundlagen der Programmierung und Algorithmierung an.
- nutzen die algorithmischen Grundstrukturen Sequenz, Verzweigung und Zyklus.
- nutzen sinnvoll Variablen und Wertzuweisungen.
- durchlaufen die Phasen des Problemlöseprozesses (Analyse, Modellierung, Implementierung, Test/Fehlerbehebung) für ein gegebenes Problem.
- testen systematisch eine Implementierung auf korrekte Funktionalität, auch in Bezug auf Sonderfälle. (Bildungsstandards, S. 6)

Psychomotorische Lernziele: Die Schüler:innen…

- implementieren einen Algorithmus im MakeCode-Editor.

Affektive Lernziele: Die Schüler:innen…

- beachten vereinbarte Konventionen bezüglich der Werkzeuge, Techniken und Dokumente. (Bildungsstandards, S. 8)
- kooperieren bei der Lösung informatischer Probleme. (Bildungsstandards, S. 8)

4. Voraussetzungen

Fachlich:

Grundwissen über

- Algorithmen + algorithmische Grundstrukturen
  - Sequenz
  - Verzweigung
  - Zyklus
- eine Programmierumgebung für Calliope oder MicroBit
  - insbesondere auch Funkübertragung

Technisch:

- je 1 Kit Calliope oder MicroBit pro Schüler:in bzw. Gruppe
- falls Morsetaster zusätzliche Taste sein soll, dann noch
  - Steckbrett
  - Krokodilsklemmen
  - Verbindungskabel Steckbrett
  - Tastschalter
  - Widerstand (für Pull-Up/-Down)
- für die Programmierung
  - Computer (empfohlen) mit Internetverbindung oder entsprechende Entwicklungsumgebung oder
  - Smart-Device der Wahl mit entsprechend installierter App

Materiell:

- Arbeitsblatt / Ausdruck mit Morsealphabet
- ggf. später Codeschnipsel für komplette Fallunterscheidung der Morsecodes

5. Kurzdarstellung

Die Grundidee ist, einen Taster für die Kommunikation über Morsecode mit Calliope oder MicroBit zu simulieren. Je nachdem, wie lange der jeweilige Knopf gedrückt wird, soll entweder ein kurzes Signal (.) oder ein langes Signal (-) in eine String-Variable gespeichert werden, welche dann per Funk übertragen werden kann. Diese soll beim Empfangsgerät direkt ausgewertet werden und der entsprechende Buchstabe / das entsprechende Symbol auf der LED-Matrix angezeigt werden.

Wie genau das Programm letztlich aussieht / aussehen soll, ist von den Anforderungen abhängig. So können beispielsweise visuelle (versch. Farben RGB-LED, LED-Matrix) oder auditive (kurzer/langer Ton) Rückmeldungen eingebaut werden, oder auch „Quality-of-Life"-Verbesserungen (z.B. Programm wartet, bis Taster losgelassen wird).

Auch die Schwierigkeit kann verändert werden bzw. kann man die Schwierigkeit aufbauend gestalten, z.B.:

1. Funkübertragung kennenlernen; je eine Taste für . / - ; Empfangsgerät zeigt nur . bzw. –
2. Statt 2 separaten Tasten für . / - soll nun nur noch eine Taste genutzt werden
3. **Statt einzelner Signale jetzt Zeichenkette für einen Buchstaben übertragen; Decodierung empfangener Zeichenketten**
4. Mehrere Zeichenketten von Buchstaben (z.B. mit festem Trennzeichen) für ein Wort übertragen; entsprechende Anpassung in Decodierung
   (für „einfache" Lösungen benötigt man Stringmanipulation und Listen)

Es bietet sich also an, mit einem Grundprogramm zu starten und dieses schrittweise mit weiteren Aufgaben zu erweitern, diese Unterteilung bietet sich aber auch für Differenzierung an. Für komplexere Aufgaben wie 3./4. bietet sich vorherige grobe Modellierung (Struktogramm, Flussdiagramm) an.

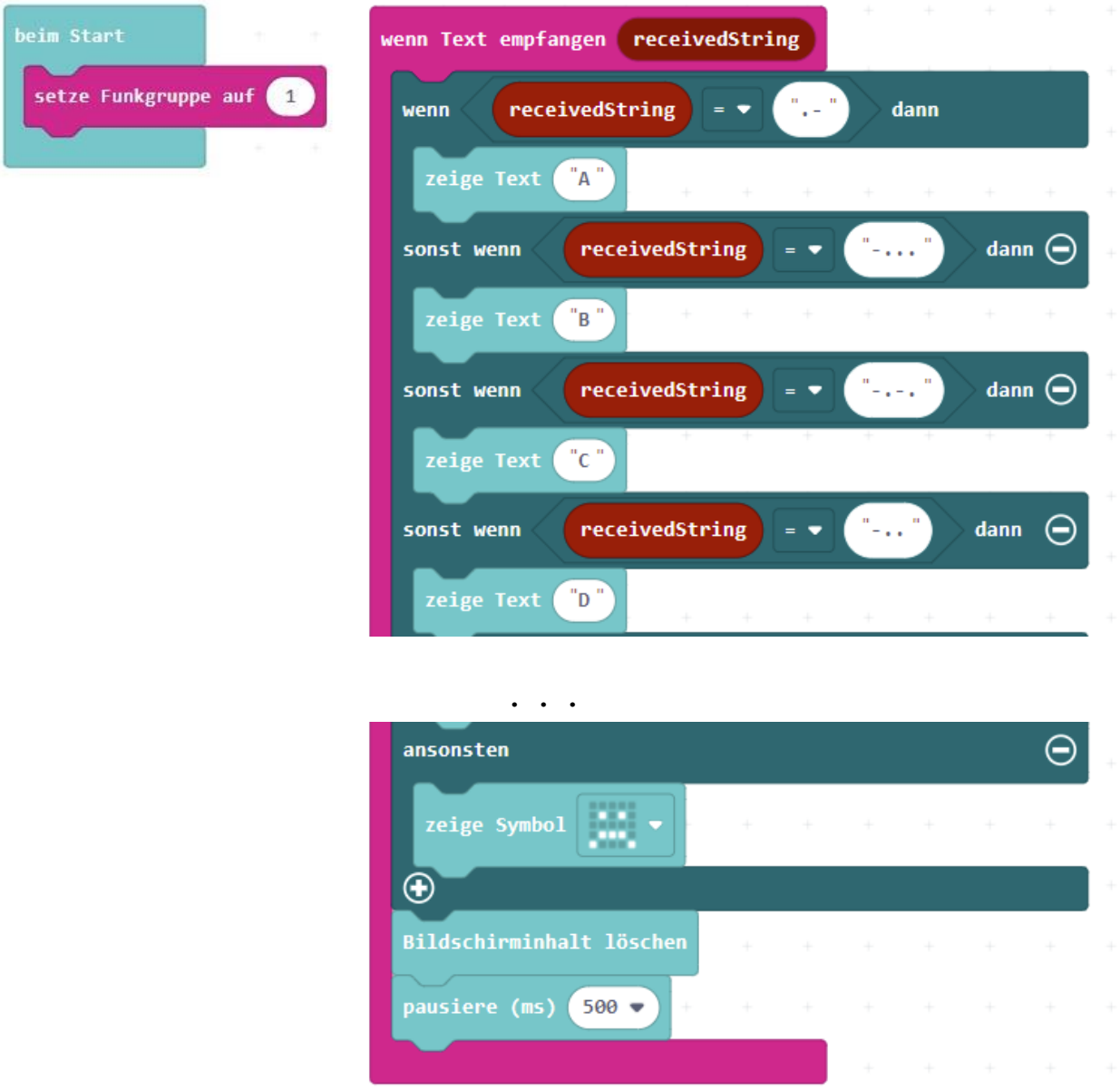
Für die Decodierung sollten die Schüler:innen nur einige ausgewählte Buchstaben in die Fallunterscheidung aufnehmen müssen, dafür aber beispielsweise Symbole/Buchstaben mit unterschiedlicher „Länge der Zeichenkette" (z.B.: A = .- → Länge 2; B = -… → Länge 4; E = . → Länge 1 usw.). Bei entsprechendem Fortschritt kann dann einfach ein vorgefertigter Codeschnipsel mit den restlichen Buchstaben/Symbolen bereitgestellt werden.

Das Morsealphabet muss natürlich in irgendeiner Weise zur Verfügung gestellt werden, beispielsweise durch ein Arbeitsblatt. Dieses könnte auch nur teilweise vorgegeben sein und von den Schüler:innen vervollständigt werden, z.B.:

Buchstabe / Symbol	Morsecode	Eingabe mit der Calliope
A	**. -**	**A B**
B	**- . . .**	**B A A A**
C	**- . - .**	
D		**B A A**

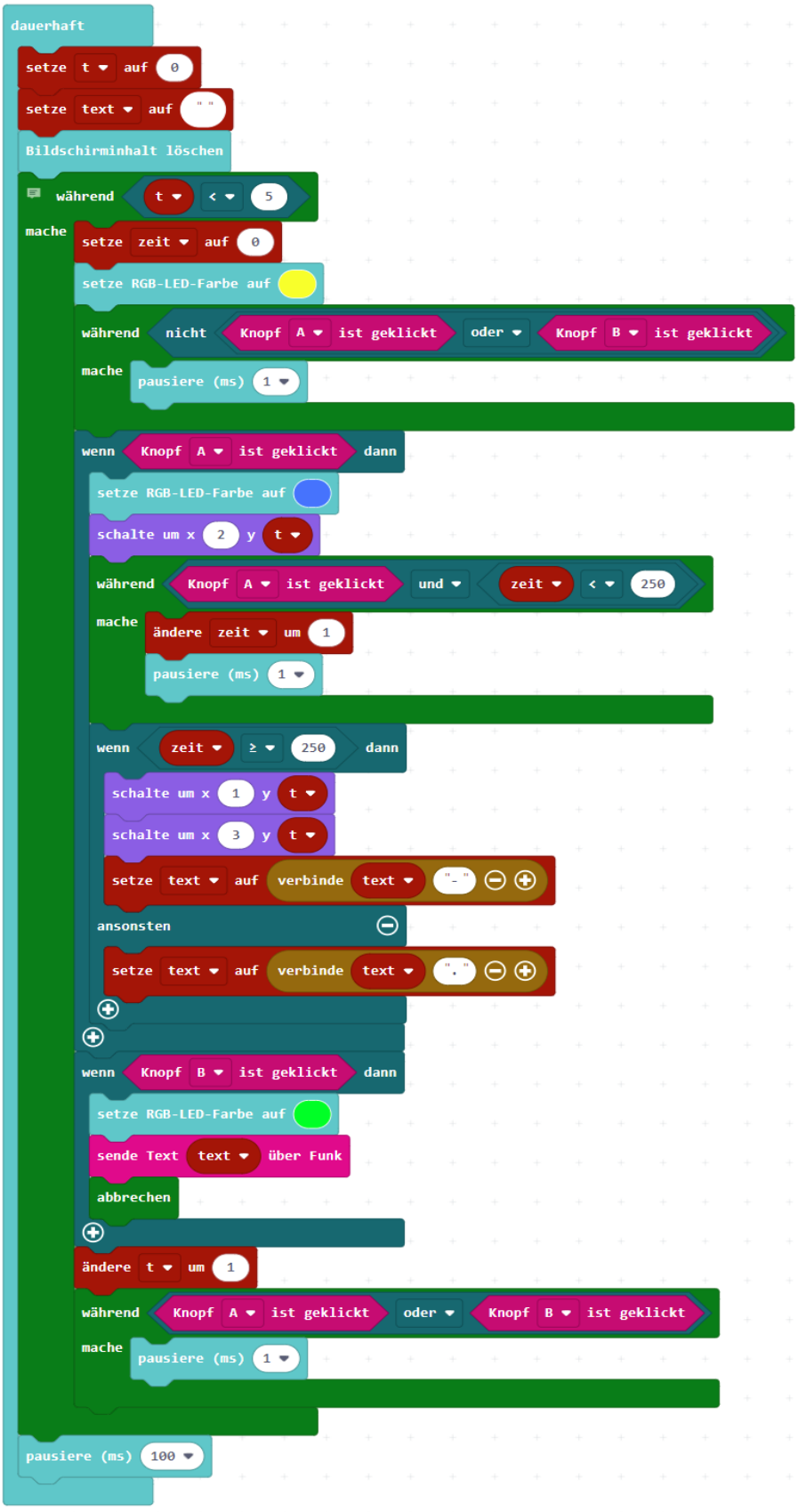
Nachfolgend ist eine Möglichkeit für das Projekt (obige Aufgabe 3) im Makecode-Editor für die Calliope dargestellt.

Initialisierung und Decodierung



- Funkgruppe beim Starten festlegen
- Möglichkeit der Decodierung mittels Verzweigung
    - Extrasymbol (☹) für unbekannte Abfolge

Hauptprogramm zur Simulation des Tasters und zum Absenden



- RGB-LED gibt aktuellen Status an (gelb = „Bereit für Eingabe“; blau = „aktuelle Eingabe lesen“; grün = „Eingabe wird gesendet“)
- Nach Drücken von A-/B-Taste wird gewartet, bis diese wieder losgelassen wird
- **Zeichenkette *kann in diesem konkreten Fall* nicht länger als 4 Signale sein**

Programmcode in Python:

```
def on_received_string(receivedString):
    if receivedString == ".-":
        basic.show_string("A")
    elif receivedString == "-...":
        basic.show_string("B")
    elif receivedString == "-.-.":
        basic.show_string("C")
    elif receivedString == "-..":
        basic.show_string("D")
    else:
        basic.show_icon(IconNames.SAD)
    basic.clear_screen()
    basic.pause(500)
radio.on_received_string(on_received_string)

zeit = 0
text = ""
t = 0
radio.set_group(1)

def on_forever():
    global t, text, zeit
    t = 0
    text = ""
    basic.clear_screen()
    # Diese Abbruchbedingung muss natürlich geändert werden, falls Codes mit mehr als 4
    # Signalen gewollt sind
    # (dann muss auch der Rest, insbes. das Senden angepasst werden)
    while t < 5:
        zeit = 0
        basic.set_led_color(0xffff00)
        while not (input.button_is_pressed(Button.A) or input.button_is_pressed(Button.B)):
            basic.pause(1)
        if input.button_is_pressed(Button.A):
            basic.set_led_color(0x007fff)
            led.toggle(2, t)
            while input.button_is_pressed(Button.A) and zeit < 250:
                zeit += 1
                basic.pause(1)
            if zeit >= 250:
                led.toggle(1, t)
                led.toggle(3, t)
                text = "" + text + "-"
            else:
                text = "" + text + "."
        if input.button_is_pressed(Button.B):
            basic.set_led_color(0x00ff00)
            radio.send_string(text)
            break
        t += 1
        while input.button_is_pressed(Button.A) or input.button_is_pressed(Button.B):
            basic.pause(1)
    basic.pause(100)
basic.forever(on_forever)
```

Programmcode in JavaScript:

```
radio.onReceivedString(function on_received_string(receivedString: string) {
    if (receivedString == ".-") {
        basic.showString("A")
    } else if (receivedString == "-...") {
        basic.showString("B")
    } else if (receivedString == "-.-.") {
        basic.showString("C")
    } else if (receivedString == "-..") {
        basic.showString("D")
    } else {
        basic.showIcon(IconNames.Sad)
    }
    basic.clearScreen()
    basic.pause(500)
})
let zeit = 0
let text = ""
let t = 0
radio.setGroup(1)
basic.forever(function on_forever() {

    t = 0
    text = ""
    basic.clearScreen()
    //  Diese Abbruchbedingung muss natürlich geändert werden, falls Codes mit mehr als 4
    // Signalen gewollt sind
    //  (dann muss auch der Rest, insbes. das Senden angepasst werden)
    while (t < 5) {
        zeit = 0
        basic.setLedColor(0xffff00)
        while (!(input.buttonIsPressed(Button.A) || input.buttonIsPressed(Button.B))) {
            basic.pause(1)
        }
        if (input.buttonIsPressed(Button.A)) {
            basic.setLedColor(0x007fff)
            led.toggle(2, t)
            while (input.buttonIsPressed(Button.A) && zeit < 250) {
                zeit += 1
                basic.pause(1)
            }
            if (zeit >= 250) {
                led.toggle(1, t)
                led.toggle(3, t)
                text = "" + text + "-"
            } else {
                text = "" + text + "."
            }
        }
        if (input.buttonIsPressed(Button.B)) {
            basic.setLedColor(0x00ff00)
            radio.sendString(text)
            break
        }

        t += 1
        while (input.buttonIsPressed(Button.A) || input.buttonIsPressed(Button.B)) {
            basic.pause(1)
        }
    }
    basic.pause(100)
})
```