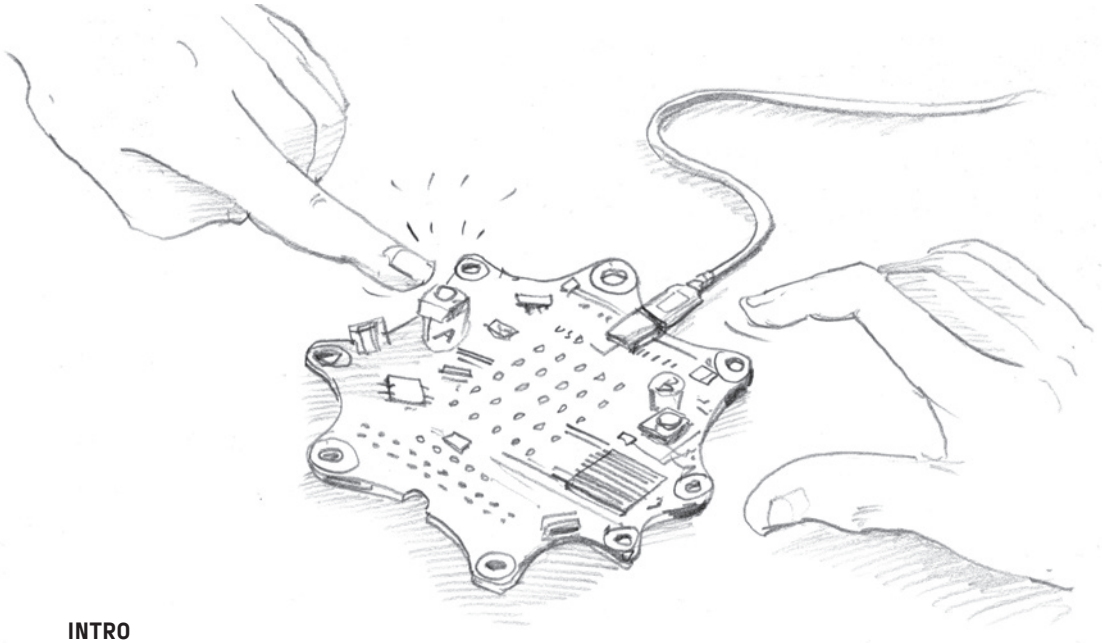
MAKEYOUROWNCODE



INTRO

In diesem Workshop programmieren wir uns ein kleines fetziges Reaktionsspiel, welches wir auf dem Calliope spielen können. Wir werden MakeCode besser kennenlernen, entdecken, dass Schleifen nicht nur an Schnürsenkeln zu finden sind und unseren Programmablauf mit Hilfe von Variablen, Funktionen und Arrays ordentlich in Schwung bringen.

LEVEL

Anfänger*innen mit Vorkenntnissen

DAUER

2-4 h (je nach Vorkenntnissen)

ANZAHL PERSONEN

max. 2 Personen je verfügbarem Computerarbeitsplatz

ALTERSGRUPPE

ab 12 Jahre

NOTWENDIGE VORKENNTNISSE TEILNEHMENDE:

- Grundkenntnisse Computerbedienung
- erste Erfahrungen mit MakeCode[1]
- hilfreich sind erste Erfahrungen mit Calliope-Controller[1]

NOTWENDIGE VORKENNTNISSE WORKSHOPLEITENDE:

- Grundkenntnisse Programmierung
- Grundkenntnisse ▷1. MakeCode
- Grundkenntnisse ▷2. Calliope-Controller

TECHNISCHE AUSSTATTUNG AM WORKSHOP-ORT:

- Computerarbeitsplätze mit Strom und Internet
- Calliope-Controller (falls nicht vorhanden kann auch der online Emulator verwendet werden, macht aber nur halb so viel Spass)

SETUP/VORBEREITUNG:

- Computerarbeitsplätze mit Internetzugang und Browser (z.B. Chrome) bereitstellen
- Es sollte ein Calliope-Controller (inkl. USB-Verbindungskabel) pro Computerarbeitsplatz vorhanden sein, Funktionsfähigkeit der Controller überprüfen, ggf. Batteriepacks bereitstellen.

DAS SPIEL

Die LED des Calliope leuchtet abwechselnd in verschiedenen Farben. Erscheint die Farbe Grün, dann muss Knopf A gedrückt werden. Erscheint Rot, dann muss Knopf B gedrückt werden. Bei jeder anderen Farbe muss der Calliope zur Seite gekippt werden. Ob ihr richtig reagiert habt, oder nicht zeigt die 5×5 LED Anzeige mit + oder – an. Nach Ablauf der Spielrunden läuft „Game over“ als Laufschrift über die LED Anzeige. Per Reset-Knopf wird das Spiel neu gestartet.

▷1.

▷2.

[1] falls nicht vorhanden ggf. in Workshop-Ablauf ausführlichere Vorstellung einplanen

HALLO WELT

0. EINFÜHRUNG:

Nach der Kennenlernrunde der Betreuenden und Teilnehmenden (inkl. Mitteilung der Vorkenntnisse) folgt eine kurze Vorstellung des Calliopes und des Reaktionsspiels, welches wir programmieren werden.

1. MAKECODE UND DIE PROGRAMMIERBAUSTEINE

Die MakeCode Oberfläche besteht aus drei Bereichen: dem Calliope Emulator zum schnellen Testen deines Programmcodes, der Liste verschiedener Rubriken an Programmierbausteinen und der Arbeitsfläche für dein Programm. Die Programmierbausteine ziehst du per drag&drop von der jeweiligen Rubrik auf die Arbeitsfläche. Da die einzelnen Bausteine wie Puzzleteile geformt sind, erkennst du schnell, wo welches Teil passen kann und wo nicht.

↑ Figure 1: Oberfläche MakeCode: (von links nach rechts)
Calliope Emulator, Programmierbausteine, Programmcode

Wichtig für deinen Programmablauf sind die beiden Bausteine „beim Start“ und „dauerhaft“. Während alles innerhalb „beim Start“ definierte nur einmal zu Programmbeginn ausgeführt wird, werden die Elemente in „dauerhaft“ permanent während des Programmablaufs aufgerufen. In unserem kleinen Beispiel zeigen wir einmal zur Begrüßung ein „hallo“ als Laufschrift auf der 5x5 LED-Matrix und lassen zusätzlich dauerhaft die LED rot blinken.

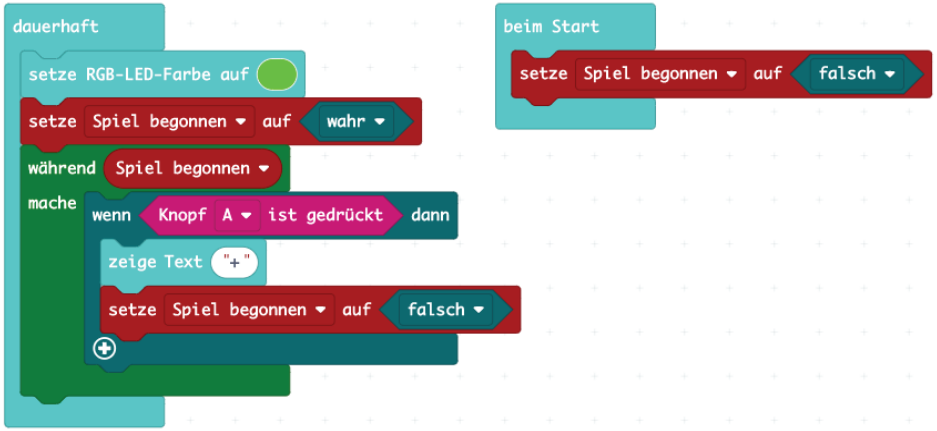
Schaut euch die einzelnen Bausteine in den Rubriken an und probiert ein bisschen rum. Zum Testen eures Programms könnt ihr entweder den Emulator auf der linken Seite nutzen oder ihr spielt euer Programm direkt auf den Calliope. Das Testen wird auch im weiteren Verlauf immer wieder hilfreich sein.

LOS GEHT'S

2. GRUNDELEMENTE REAKTIONSSPIEL

Jetzt beginnen wir unser Spiel-Grundgerüst aufzubauen. Dazu nutzen wir die Bausteine aus den Rubriken „Grundlage“, „Eingabe“, „Variablen“, „Logik“ und „Schleifen“.
Zunächst erstellen wir uns eine Variable namens „Spiel begonnen“ über den Button „ Erstelle eine Variable..“ der Rubrik „Variablen“. *Variablen in der Programmierung sind eine Art Behälter, dessen Wert sich ändern kann. Sie werden genutzt, um Daten und Abläufe zu verwalten.*
Unsere Variable wird entweder den Wert „wahr“ oder „falsch“ haben und steuert damit die Spielrunden. Beim Start setzen wir den Wert zunächst auf „falsch“ und sobald wir die Farbe der LED gesetzt haben auf „wahr“ (die Bausteine für die Werte findet ihr in der Rubrik „Logik“). Jetzt kann das Spiel beginnen!

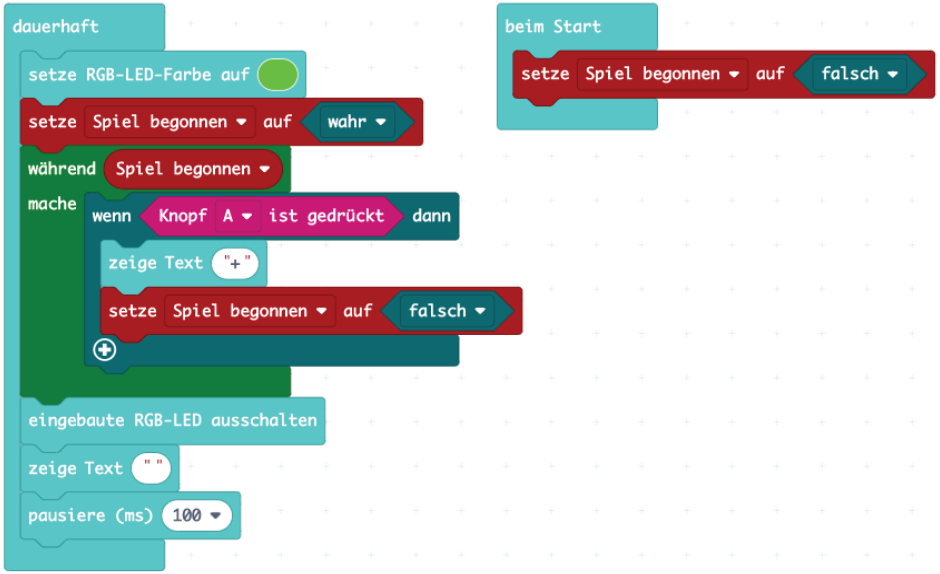
Nun wählen wir uns aus der Rubrik „Schleifen“ den „während“ Baustein aus und verknüpfen unsere „Spiel begonnen“ Variable als Bedingung. *Eine Schleife ist eine Kontrollstruktur der Programmierung, um Abläufe zu steuern. Bestimmend hierbei ist die Schleifen-Bedingung. Solange diese gültig ist, wird alles, was innerhalb des Schleifen-Blocks definiert ist, wiederholt.* In unserem Fall: solange die Variable „Spiel begonnen“ den Wert „wahr“ hat. Bei Grün soll Knopf A gedrückt werden. Wir brauchen jetzt also einen Baustein, der uns hilft zu unterscheiden. Dies nennt man in der Programmierung *„bedingte Anweisung“ - eine Verzweigung, die festlegt, welcher Programmabschnitt ausgeführt wird.* Diese finden wir in der Rubrik Logik als „wenn... dann“ Baustein. „Diesen ergänzen wir wie folgt: „Wenn Knopf A ist gedrückt“ (Rubrik „Eingabe“), dann wird „+“ angezeigt und „Spiel begonnen“ auf „falsch“ gesetzt, um in die nächste Runde zu gelangen (Figure 2).



↑ Figure 2: Zwischenstand, Grundelemente Reaktionsspiel

WÄSCHELEINE

Wenn ihr das Spiel jetzt testet, wird euch auffallen, dass gar nicht richtig zu unterscheiden ist, in welcher Spielrunde wir uns befinden. Daher fügen wir noch drei Bausteine hinzu: LED ausschalten, leeren Text anzeigen und pausieren. Das setzt nun nach jeder Spielrunde den Zustand zurück, bevor die Farbe erneut aufleuchtet (Figure 3).



↑ Figure 3: Ergebnis Teil 2, Grundelemente Reaktionsspiel

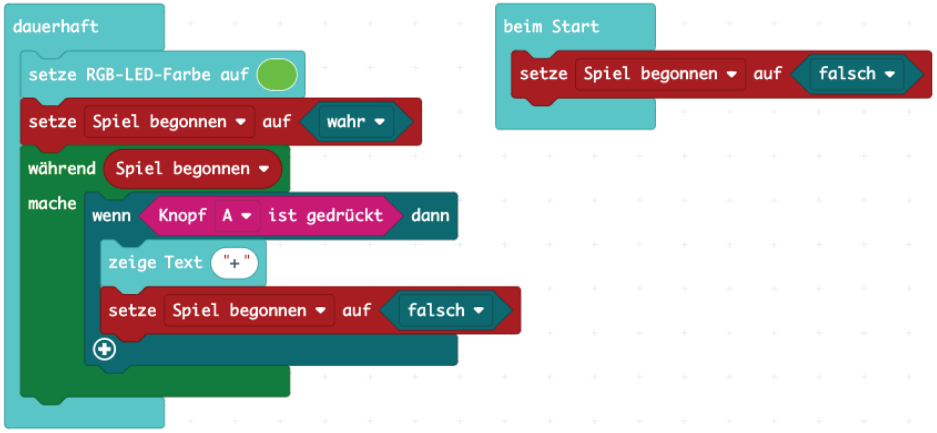
3. EINE WÄSCHELEINE AUS FARBEN UND DEN ZUFALL INS SPIEL BRINGEN

Bisher wird immer nur die Farbe Grün angezeigt, was ziemlich langweilig ist. Eigentlich wollen wir unterschiedliche Farben sehen, zufällig gewählt aus einem Pool an Farben. Der Zufall wird uns helfen! Wir lernen Bausteine aus den Rubriken „Arrays“ und „Mathematik“ kennen.

Was wir brauchen, ist eine neue Variable namens „LedFarben“, welche wir anschließend mit Farben füllen. Um in einer Variable mehrere Werte auf einmal zu speichern, lassen wir sie auf ein Array zeigen. *Ein Array in der Programmierung ist wie eine Art Wäscheleine, auf die wir hintereinander verschiedene Kleidungsstücken hängen, in unserem Fall die Farben.* Dazu wählen wir in der Rubrik „Array“ (unter „Fortgeschritten“) den Baustein „setze... auf“ und weisen beim Start unserer Variable „LedFarben“ die einzelne Farbtöne Grün und Rot (Rubrik „Grundlagen“) zu. Die Anzeige der Farbe Grün erfolgt beispielsweise, indem wir den Wert der „LedFarben“ an Position 0 abrufen (Arrays fangen immer mit 0 an zu zählen). Die Position eines Wertes innerhalb des Arrays wird in der Programmierung auch „index“ genannt.

ZUFALL!

Noch spannender wird es nun, wenn wir die Farbe zufällig wählen. Dafür gibt es in der Rubrik „Mathematik“ den Baustein „wähle eine zufällige Zahl von.. bis..“, die wir mit den Positionen unseres Farb-Arrays verknüpfen: vom index des ersten Wertes bis zum index des letzten. Da die erste Position in Arrays immer bei 0 liegt, entspricht der index des letzten Wertes dementsprechend der Länge des Arrays minus 1. Um die Länge des Arrays abzurufen, gibt es in der Rubrik „Array“ den Baustein „Array-Länge“. Zur besseren Übersicht legen wir den Wert unserer Zufallsberechnung in eine eigene Variable namens „zufälligerWert“ und setzen die Farbe unserer LED auf den Farbwert an der entsprechenden Position unseres LedFarben-Arrays (Figure 4).



↑ Figure 4: Zwischenstand, Farb-Array beim Start erstellen und Farbe per zufälligem Wert abrufen

Nun müssen wir die Interaktion mit den Knöpfen noch korrigieren, denn gerade gilt Knopf A immer als richtig, egal welche Farbe erscheint. Die Variable „zufälligerWert“ hilft uns zu unterscheiden, welcher Knopf in der jeweiligen Spielrunde als richtig oder falsch gilt. Wir ergänzen unsere Spielrunden-Schleife also mit einem weiteren „wenn...dann“ Baustein (Rubrik „Logik“) und nutzen als Unterscheidungsbedingung einen Vergleich, nämlich ob der Inhalt der Variable „zufälligerWert“ der Position 0 (Grün) oder Position 1 (Rot) im Farb-Array entspricht. Die Reaktion auf die jeweils gedrückten Knöpfe ergänzen wir entsprechend (Figure 5, siehe nächste Seite).

EINFACHER

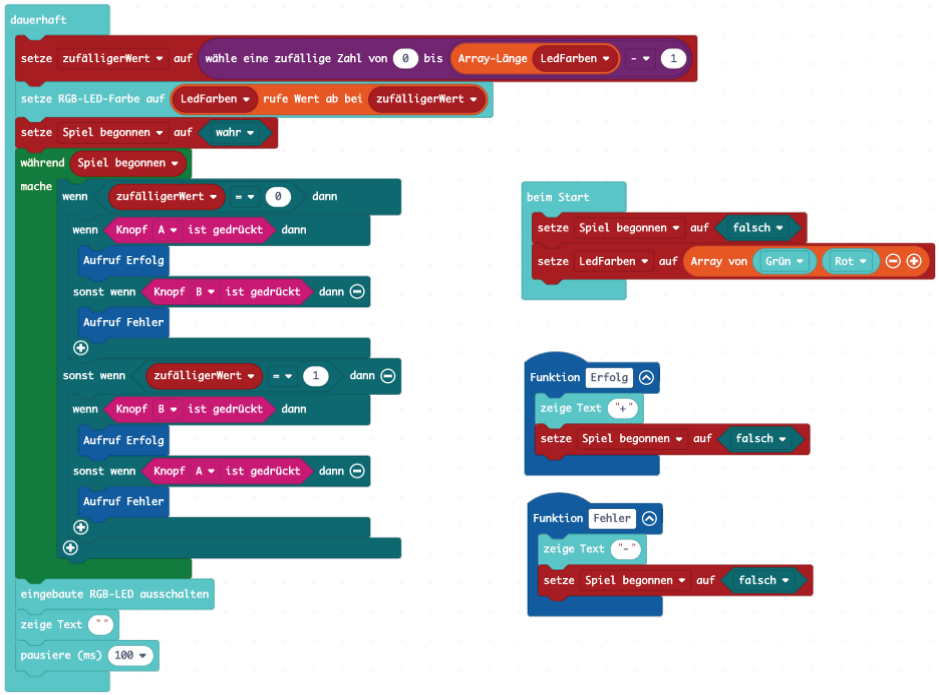
↑ Figure 5: Ergebnis Teil 3, Zufällige Farbwahl und Knopf Interaktion mit Feedback

4. CODE VEREINFACHEN

Wenn wir unseren Programmcode anschauen, fällt auf, dass wir an einigen Stellen exakt das Gleiche wiederholt schreiben. Hier können wir uns Platz und Schreibaufwand sparen, indem wir Funktionen verwenden. *Funktionen in der Programmierung sind kleine Programmteile, die helfen den Code zu strukturieren und einfach wiederzuverwenden.* Das spart nicht nur Zeit, sondern hilft auch Fehler zu vermeiden und schafft Übersichtlichkeit. Um unseren Programmcode zu richtig oder falsch gedrückten Knöpfen auszulagern, legen wir uns zwei Funktionen an (Rubrik „Funktionen“): Erfolg und Fehler. Funktionen können auch sogenannte Parameter haben. Das sind Werte, die wir der Funktion zur Berechnung mitgeben können. Für unsere beiden Funktionen brauchen wir das nicht.
Die Erfolgs-Funktion zeigt das „+“, und die Fehlerfunktion das „-“ als Feedback an.

FARBEN!

Beide Funktionen setzen die „Spiel begonnen“ Variable zurück. Da, wo vorher unser (wiederholter) Programmcode stand, rufen wir jetzt einfach die jeweilige Funktion auf und sehen damit auch viel deutlicher, was an der Stelle eigentlich passiert (Figure 6).



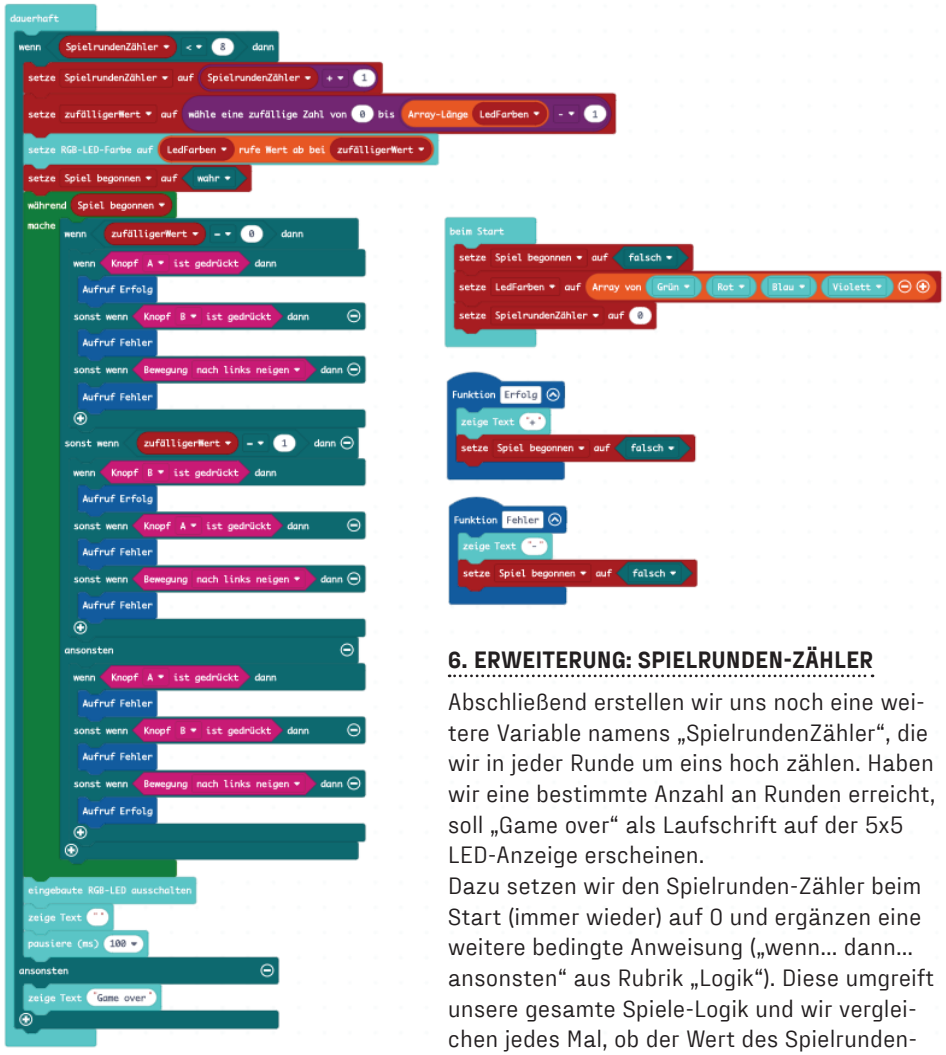
↑ Figure 6: Ergebnis Teil 4, Mehr Übersicht durch Nutzung von Funktionen

5. ERWEITERUNG: MEHR FARBEN

Unsere Spielbasis steht. Jetzt können wir noch mehr Farben ins Spiel bringen. Füge dazu mehr Farben ins Array ein. Die Spiele-Logik ergänzen wir wie folgt: erscheint eine andere Farbe als Grün oder Rot soll der Calliope zur Seite gekippt werden. Dazu fügen wir eine weitere Auswahlbedingung bzgl. des zufälligen Wertes ein, nämlich „ansonsten“ (Klick auf das „+“ der Anweisung). Darin definieren wir das Folgende: weder Knopf A noch B führen zum Erfolg, sondern nur „Bewegung nach links neigen“ (Rubrik „Eingabe“). Ergänze die Kippbewegung auch als Fehleroption für die Farben Grün und Rot (Figure 7, siehe nächste Seite).

ZÄHLEN

↓ Figure 7: Ergebnis Teil 5 und 6, mehr Farben und Spielrunden-Zähler



6. ERWEITERUNG: SPIELRUNDEN-ZÄHLER

Abschließend erstellen wir uns noch eine weitere Variable namens „SpielrundenZähler“, die wir in jeder Runde um eins hoch zählen. Haben wir eine bestimmte Anzahl an Runden erreicht, soll „Game over“ als Laufschrift auf der 5x5 LED-Anzeige erscheinen.
Dazu setzen wir den Spielrunden-Zähler beim Start (immer wieder) auf 0 und ergänzen eine weitere bedingte Anweisung („wenn... dann... ansonsten“ aus Rubrik „Logik“). Diese umgreift unsere gesamte Spiele-Logik und wir vergleichen jedes Mal, ob der Wert des Spielrunden-Zählers unsere maximale Spielrunden-Anzahl (Abbruchbedingung, z.B. 8) übersteigt. Wenn nicht wird weitergespielt und der Zähler eins hochgezählt, ansonsten wird „Game over“ angezeigt.
Per Reset-Knopf beginnt das Spiel von vorn (Figure 8).

WEITERE VORSCHLÄGE ZU ERWEITERUNGEN

Wer Lust hat sein Spiel noch weiter auszubauen, kann noch den Spielstand ermitteln und ausgeben. Eine weitere Idee ist, den Schwierigkeitsgrad zu erhöhen, indem die Knopfbelegung wechselt, wenn z.B. Blau erscheint. Dann muss bei Grün Knopf B gedrückt werden und bei Rot Knopf A. Ihr könnt auch die Geschwindigkeit im Verlauf des Spiels erhöhen.

Lasst eurer Kreativität freien Lauf und habt Spaß!

WEITERLERNEN

Im Internet gibt es zahlreiche Beispiele und Anleitungen für den Calliope und MakeCode.

Microsoft ▷3. Microsoft und ▷4. Calliope stellen eine Reihe an Projekten und Lernideen zusammen.

Die ▷5. tuduu-PLattform liefert weitere Inspirationen.

Wenn du dich weiter mit Programmierung beschäftigen möchtest findest du zahlreiche Übungen, Tutorials und kostenlose Kurse auf ▷6. Start-Coding und ▷7. code.org.

▷3.

▷4.

▷5.

▷6.

▷7.

FABMOBIL

RAUM FÜR ANGEWANDTE ZUKUNFT

PROJEKTBETEILIGTE

KONZEPTION / ART DIRECTION

Constitute e.V.

AUTOR*INNEN

Julius Plüschke, Robert Kunz, Tommy Schönherr,
Kristine Weißbarth, Annett Löser

ILLUSTRATION / LAYOUT

Daniel Stolle

GEFÖRDERT VON:

KULTURSTIFTUNG DES BUNDES

SACHSEN

Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Kulturstiftung des Freistaates Sachsen

PwC-Stiftung
Jugend • Bildung • Kultur

WWW.THECONSTITUTE.ORG
CENTER FOR APPLIED FUTURE

ZEIT-Stiftung
Ebelin und Gerd Bucerius

DIRK OELBERMANN STIFTUNG

GEMEINNÜTZIGER VEREIN E.V.
SCHORNDORF
APOTHEKER DR. MED. MARIA
UND JOHANN-PHILIPP PALM
PALM-STIFTUNG