Programmieren mit Blöcken

Spiele, Bilder und Musik mit PBC+ Studio



AB 10 JAHREN

DIE PROGRAMMIERREIHE FÜR DEN PICOBOY COLOR PLUS

Dieses Buch wurde mit künstlicher Intelligenz (KI) generiert. Die Programme wurden getestet – Fehler sind trotzdem möglich.

Bevor es losgeht

In diesem Buch lernst du, deinen PicoBoy Color Plus zu programmieren. Du brauchst dafür keine Vorkenntnisse. Du baust deine Programme aus **Blöcken**, die du wie Puzzleteile aneinanderhängst. Am Anfang lässt du Texte und Bilder erscheinen, später bewegen sich Figuren, es piept und blinkt – und am Ende baust du dein eigenes Spiel.

So ist das Buch aufgebaut

Jedes Kapitel hat denselben Aufbau:

- **Worum geht's?** – sagt dir, was du in dem Kapitel lernst.
- **Erklärungen** mit Bildern der Blöcke, genau so, wie du sie in PBC+ Studio siehst.
- **Ausprobieren** – ein etwas größeres Programm zum Nachbauen.
- **Übungsaufgaben** – mit Sternen: ⭐ zum Einsteigen, ⭐⭐ wenn du weiter willst, ⭐⭐⭐ zum Knobeln. Die Lösungen stehen in Anhang C. Schau erst nach, wenn du es selbst versucht hast!
- **Das Wichtigste in Kürze** – zum Wiederholen.

Unterwegs findest du farbige Kästen:

> **Tipp:** Hier steht, wie etwas leichter geht.

> **Achtung:** Hier steht, worauf du aufpassen musst.

> **Info:** Hier steht etwas, das gut zu wissen ist.

Was du brauchst

- einen Computer mit dem Browser **Chrome** oder **Edge**
- die Seite **pbcstudio.de**
- einen **PicoBoy Color Plus** mit USB-Kabel – fast alles geht aber auch ohne Gerät im **Emulator**, einem PicoBoy auf dem Bildschirm

Alle Programme aus dem Buch gibt es auch als Dateien zum Öffnen in PBC+ Studio (Endung `.pbcp`). Den Dateinamen findest du jeweils beim Programm.

Die Teile des Buchs

Teil	Darum geht es
I – Los geht's	Gerät und PBC+ Studio kennenlernen, das erste Programm
II – Grundlagen	zeichnen, warten, wiederholen, Variablen, Entscheidungen, Tasten, LEDs und Töne
III – Grafik und Bewegung	Turtle, Canvas, Sprites und Kollisionen
IV – Größere Programme	Funktionen, Listen, Texte, Mathe, Kippen
V – Projekte	zwei Spiele Schritt für Schritt – und dann dein eigenes

Viel Spaß beim Programmieren!

Inhalt

Teil I – Los geht's

1 Der PicoBoy Color Plus und PBC+ Studio 5
2 Das erste Programm 10

Teil II – Grundlagen

3 Zeichnen und Farben 16
4 Schritt für Schritt: Reihenfolge und Zeit 24
5 Wiederholungen 30
6 Variablen 37
7 Entscheidungen 44
8 Eingabe: Tasten und Joystick 51
9 Ausgabe: LEDs und Töne 58

Teil III – Grafik und Bewegung

10 Turtle-Grafik 64
11 Der Canvas – ein Hintergrund im Speicher 72
12 Sprites – Figuren, die sich bewegen 79
13 Kollisionen 86

Teil IV – Größere Programme bauen

14 Funktionen 94
15 Listen 103
16 Texte und Konsole 112
17 Mathe-Werkzeuge 118
18 Der Beschleunigungssensor 125

Teil V – Projekte

19 Projekt: Reaktionstest 131
20 Projekt: Schatzjagd 139
21 Ein eigenes Spiel entwickeln 148

Anhang

A Blockreferenz 157
B Fehlermeldungen verstehen und Fehler suchen 169
C Lösungen der Übungsaufgaben 175
D Glossar 252

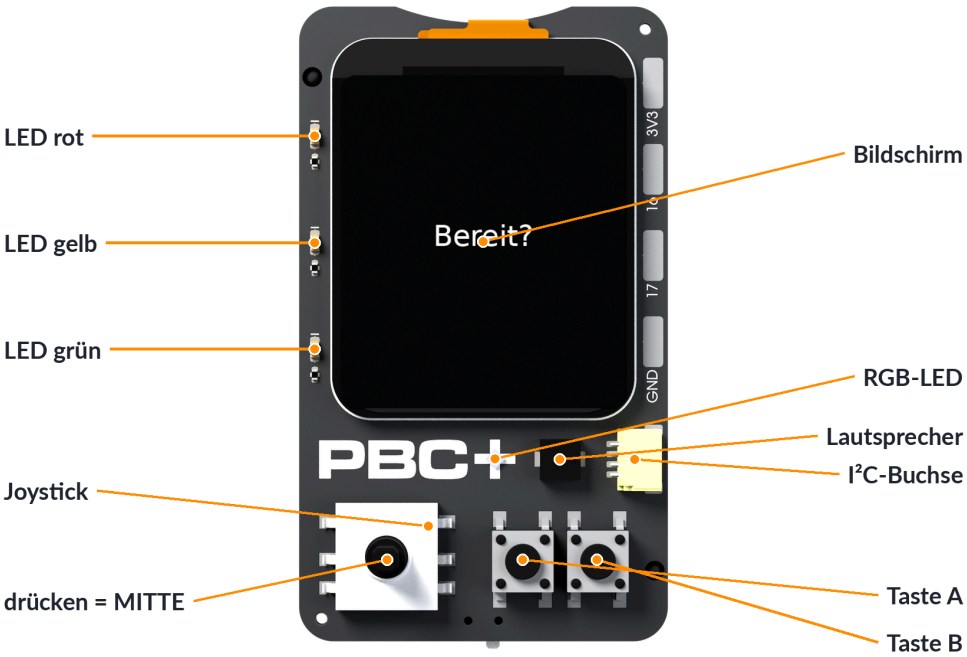
KAPITEL 1

Der PicoBoy Color Plus und PBC+ Studio

Worum geht's? Bevor du programmierst, lernst du dein Werkzeug kennen: den PicoBoy Color Plus – einen kleinen Computer mit Bildschirm und Tasten – und PBC+ Studio, das Programm im Browser, mit dem du ihn programmierst.

1.1 Das Gerät

Der **PicoBoy Color Plus** (kurz: PBC+) ist ein kleiner Computer zum Programmieren. Er sieht aus wie eine Spielkonsole – und du kannst damit auch eigene Spiele bauen.

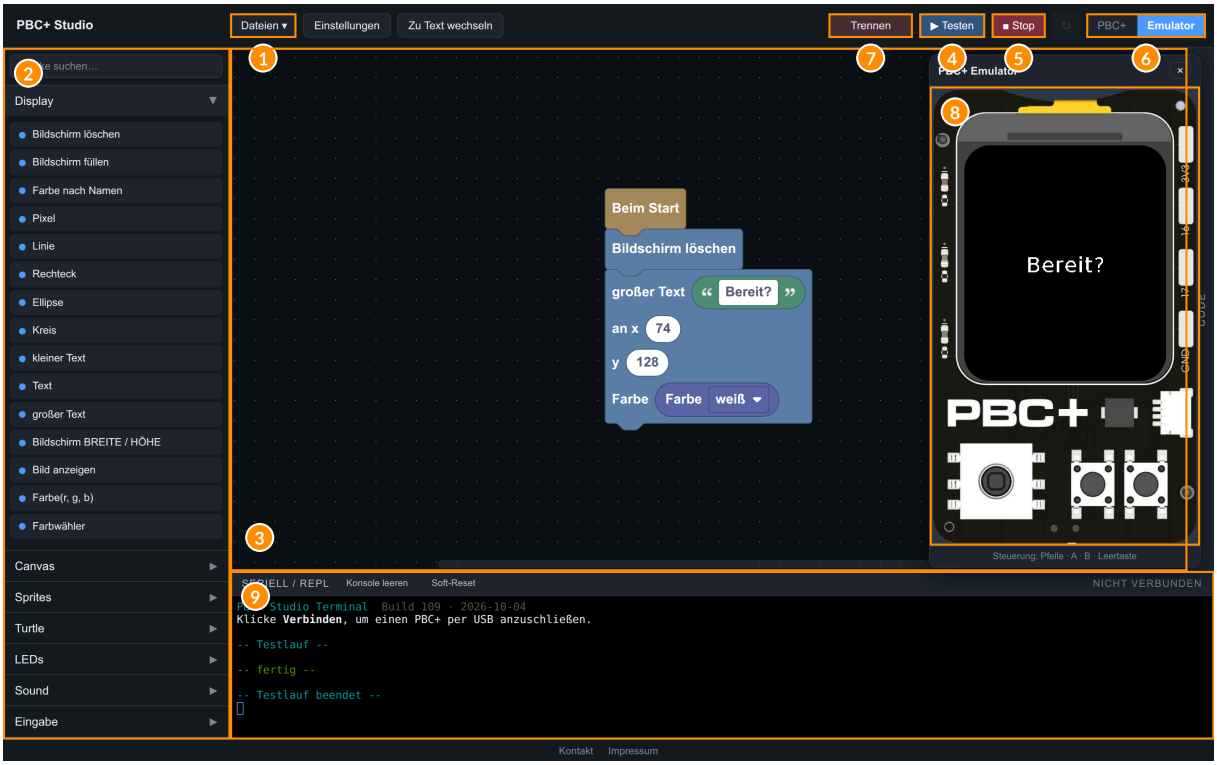


Teil	Wofür?
Bildschirm	ein Farbbildschirm, 240 Pixel breit und 280 Pixel hoch
Joystick	ein kleiner Steuerknüppel für hoch, runter, links und rechts. Drückst du ihn nach unten, ist das die Taste **MITTE**.
Taste A und Taste B	zwei Knöpfe, z. B. zum Springen oder Schießen
LEDs rot, gelb, grün	drei kleine Lämpchen links neben dem Bildschirm
RGB-LED	ein Lämpchen, das in jeder Farbe leuchten kann
Lautsprecher	spielt Töne und kleine Melodien
Beschleunigungssensor	spürt, wie du das Gerät hältst und kippst
I²C-Buchse	weiße Buchse, passend für STEMMA QT und Qwiic: Sensoren und andere I²C-Module anstecken, ohne Löten

Mit dem Computer verbindest du den PBC+ über ein **USB-Kabel**.

1.2 PBC+ Studio

Programmiert wird im Browser mit **PBC+ Studio**. Öffne dafür die Seite **pbcstudio.de** – am besten in **Chrome** oder **Edge**. Nur diese Browser können sich mit dem Gerät verbinden.



Nr.	Name	Wofür?
1	**Dateien**	Programme speichern und öffnen, Bilder hochladen
2	**Blockleiste**	Hier liegen alle Blöcke, sortiert in Kategorien wie *Display*, *Sprites* oder *Sound*. Ein Klick auf eine Kategorie klappt sie auf.
3	**Arbeitsfläche**	Hier baust du dein Programm aus Blöcken zusammen.
4	**▶ Testen**	startet dein Programm
5	**■ Stop**	hält dein Programm an
6	**Ziel**	Hier wählst du: Läuft das Programm im **Emulator** oder auf dem echten Gerät (**PBC+**)?
7	**Verbinden / Trennen**	verbindet PBC+ Studio mit dem Gerät
8	**Emulator**	ein PicoBoy auf dem Bildschirm. Er tut so, als wäre er das echte Gerät.
9	**Terminal**	Hier erscheinen Meldungen, zum Beispiel wenn dein Programm einen Fehler hat.

> Der Knopf **„Zu Text wechseln“** zeigt dein Programm als Text in der Programmiersprache Python. In diesem Buch arbeitest du nur mit Blöcken – du brauchst ihn nicht.

1.3 Blöcke holen, verbinden, löschen

Jedes Programm beginnt mit dem Block **„Beim Start“**. Er ist immer da und lässt sich nicht löschen.



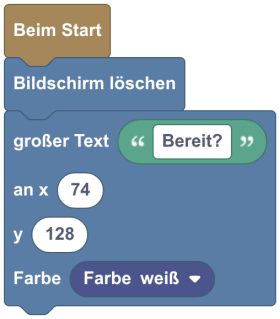
Alles, was **unter** „Beim Start“ hängt, führt der PBC+ aus – von oben nach unten, ein Block nach dem anderen.

- **Block holen:** Klicke in der Blockleiste auf eine Kategorie und dann auf einen Block. Oder ziehe den Block mit der Maus auf die Arbeitsfläche.
- **Blöcke verbinden:** Ziehe einen Block so unter einen anderen, dass die kleine Nase oben in die Kerbe darüber passt. Wenn du loslässt, schnappt er ein.
- **Werte ändern:** Klicke in ein weißes Feld und tippe eine neue Zahl oder einen neuen Text. Bei einem Feld mit Pfeil ▾ wählst du aus einer Liste.
- **Block löschen:** Klicke ihn an und drücke die Taste **Entf**. Nur dieser eine Block verschwindet, die Blöcke darunter rücken nach. Du kannst einen Block auch zurück auf die Blockleiste ziehen – dann verschwinden aber auch alle Blöcke, die unter ihm hängen.

> Ein Block, der **nicht** unter „Beim Start“ hängt, wird nicht ausgeführt. Er liegt nur auf der Arbeitsfläche herum.

1.4 Testen im Emulator oder auf dem Gerät

Wenn du PBC+ Studio zum ersten Mal öffnest, liegt schon ein kleines Programm auf der Arbeitsfläche:





So sieht es im Emulator aus.

Im Emulator testen:

1. Wähle oben rechts als Ziel **Emulator**.
2. Klicke auf ▶ **Testen**. Auf dem Bildschirm des Emulators erscheint „Bereit?“.
3. Mit ■ **Stop** hältst du ein Programm an, das nicht von selbst aufhört.

Im Emulator drückst du die Tasten mit der Tastatur:

Gerät	Tastatur
Joystick hoch, runter, links, rechts	Pfeiltasten
Joystick drücken (MITTE)	Leertaste
Taste A / Taste B	Taste **A** / Taste **B**

Du kannst die Knöpfe im Emulator auch mit der Maus anklicken.

Auf dem Gerät testen:

1. Verbinde den PBC+ mit dem USB-Kabel mit dem Computer.
2. Wähle als Ziel **PBC+** und klicke auf **Verbinden**. Der Browser zeigt eine Liste – wähle dort den PicoBoy aus.
3. Klicke auf ▶ **Testen**. Jetzt läuft dein Programm auf dem echten Gerät.

> **Emulator oder Gerät?** Der Emulator ist praktisch, weil du kein Kabel brauchst. Auf dem echten Gerät macht es aber mehr Spaß: Du hältst dein Programm in der Hand, die Tasten fühlen sich echt an, und Kippen funktioniert nur dort.

1.5 Programme speichern und öffnen

PBC+ Studio merkt sich dein Programm, auch wenn du die Seite neu lädst. Das gilt aber nur in diesem Browser auf diesem Computer. Damit nichts verloren geht, speicherst du dein Programm als Datei:

- **Speichern: Dateien → Auf den Computer…** Das Programm landet als Datei `programm.pbcp` in deinem Download-Ordner. Gib ihr am besten gleich einen besseren Namen, zum Beispiel `hallo.pbcp`.
- **Öffnen: Dateien → Vom Computer…** und die Datei auswählen.
- **Neu anfangen: Dateien → Neu**. Vorher speichern nicht vergessen!

1.6 Ein Programm auf dem Gerät behalten

Mit ▶ **Testen** läuft ein Programm nur so lange, bis du den PBC+ ausschaltest oder ein anderes Programm testest. Soll es auf dem Gerät bleiben, speicherst du es dort:

1. Verbinde das Gerät (Ziel **PBC+**, **Verbinden**).
2. **Dateien → Auf den PBC+…** – lass **Python-Programm (.py)** ausgewählt und gib einen Namen ein, zum Beispiel `hallo`.
3. Einmalig: **Dateien → Startmenü installieren…**

Danach zeigt der PBC+ beim Einschalten ein Menü mit allen gespeicherten Python-Programmen. Mit dem Joystick wählst du eins aus, mit einer Taste startest du es.

Block-Projekte mitnehmen: Ein Python-Programm lässt sich nicht wieder als Blöcke öffnen. Willst du an einem anderen Computer weiterbauen, wähle bei **Dateien → Auf den PBC+…** stattdessen **Block-Projekt (.pbcp)**. Dort verbindest du das Gerät und öffnest das Projekt mit **Dateien → Vom PBC+…** wieder als Blöcke. Starten lässt es sich auf dem PicoBoy nicht, und im Menü steht es auch nicht. Ist als Ziel der **Emulator** gewählt, arbeiten beide Menüpunkte mit dem Emulator statt mit dem Gerät.

Übungsaufgaben

Aufgabe 1 ⭐ **Teile finden** Welche Taste auf der Tastatur ist im Emulator der Joystick-Druck (MITTE)? Und mit welchem Knopf hältst du ein Programm an?

Aufgabe 2 ⭐ **Dein Name** Ändere das Startprogramm: Statt „Bereit?“ soll „Hallo,“ und dein Name erscheinen, in Gelb. Wenn der Text nicht ganz auf den Bildschirm passt, verkleinere die Zahl bei **x**.

Aufgabe 3 ⭐⭐ **Zwei Zeilen** Lass „Hallo,“ und „ich bin …“ in zwei Zeilen untereinander erscheinen, in zwei verschiedenen Farben. *Tipp:* Klicke mit der rechten Maustaste auf den Block „großer Text“ und wähle **Kopieren**. Ändere beim zweiten Block die Zahl bei **y**.

Aufgabe 4 ⭐⭐ **Speichern und wieder öffnen** Speichere dein Programm aus Aufgabe 3 als Datei. Klicke auf **Dateien → Neu** und öffne danach deine Datei wieder. Ist alles noch da?

Aufgabe 5 ⭐⭐⭐ **Falsche Reihenfolge** Ziehe im Startprogramm „Bildschirm löschen“ **unter** den Block „großer Text“. Teste. Was siehst du jetzt? Erkläre, warum.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Der PicoBoy Color Plus hat einen Bildschirm (240 × 280 Pixel), einen Joystick, die Tasten A und B, LEDs, einen Lautsprecher und einen Beschleunigungssensor.
- Du programmierst ihn mit PBC+ Studio im Browser (Chrome oder Edge).
- Jedes Programm beginnt mit „Beim Start“. Nur Blöcke, die darunter hängen, werden ausgeführt – von oben nach unten.
- Mit ▶ **Testen** startest du dein Programm, mit ■ **Stop** hältst du es an.
- Testen kannst du im **Emulator** oder auf dem echten Gerät (**PBC+**).
- Speichere deine Programme als Datei: **Dateien → Auf den Computer…**

KAPITEL 2

Das erste Programm

Worum geht's? Jetzt schreibst du dein erstes eigenes Programm: Es löscht den Bildschirm und schreibt einen Text darauf – genau an die Stelle, die du willst. Dafür lernst du, wie der PBC+ die Punkte auf seinem Bildschirm zählt. Und du erfährst, wie du Notizen in dein Programm schreibst.

2.1 Der Block „Beim Start" – hier beginnt alles

Jedes Programm in PBC+ Studio hat genau einen Block **„Beim Start"**. Er ist immer auf der Arbeitsfläche, und du kannst ihn nicht löschen.



Wenn du auf ▶ **Testen** klickst, schaut der PBC+ nach, welche Blöcke unter „Beim Start" hängen. Dann führt er sie aus: zuerst den obersten, dann den nächsten und so weiter bis zum letzten. Danach ist das Programm fertig.

Ein Programm ist also eine **Liste von Befehlen**, die der Computer der Reihe nach abarbeitet – so ähnlich wie ein Kochrezept.

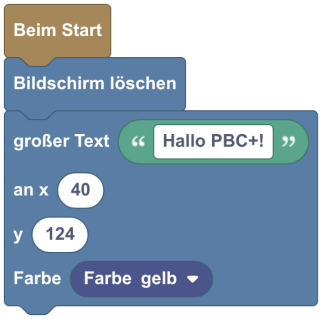
> Blöcke, die irgendwo frei auf der Arbeitsfläche liegen, werden **nicht** ausgeführt. Ein Block muss an „Beim Start" (oder an einem Block darunter) einschnappen.

2.2 Bildschirm löschen und Text schreiben

Für dein erstes Programm brauchst du zwei Blöcke aus der Kategorie **Display**:

Block	Was er tut
	macht den ganzen Bildschirm schwarz
	schreibt einen Text an eine Stelle, in einer Farbe

Baue dieses Programm:





So sieht es im Emulator aus.

1. Klicke in der Blockleiste auf **Display** und dann auf „Bildschirm löschen“. Ziehe den Block unter „Beim Start“, bis er einschnappt.
2. Hole aus **Display** den Block „großer Text“ und hänge ihn darunter.
3. Klicke in das Textfeld und schreibe `Hallo PBC+!` hinein.
4. Trage bei **x** die Zahl 40 und bei **y** die Zahl 124 ein. Wähle bei **Farbe** „gelb“.
5. Klicke auf ▶ **Testen**.

Wenn du einen Text-Block aus der Blockleiste holst, steht in allen Feldern eine **0** und als Farbe „weiß“. Ändere die Zahlen, sonst landet dein Text ganz oben links in der Ecke.

> **Warum zuerst löschen?** Auf dem Bildschirm kann noch etwas von vorher zu sehen sein. „Bildschirm löschen“ sorgt dafür, dass dein Programm auf einem leeren, schwarzen Bildschirm beginnt.

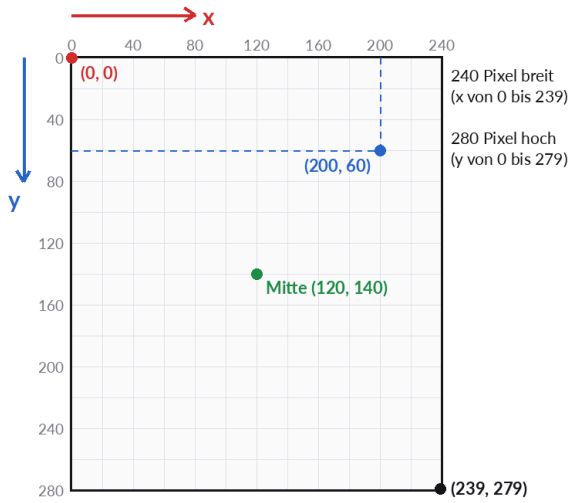
> **Keine Umlaute auf dem Bildschirm!** Die Schrift des PBC+ kennt kein ä, ö, ü und kein ß. Statt dieser Buchstaben erscheinen Fragezeichen. Schreibe deshalb in Text-Blöcken `ae`, `oe`, `ue` und `ss` – also `Gruesse` statt „Grüße“.

2.3 Das Koordinatensystem

Woher weiß der PBC+, wo der Text hin soll? Der Bildschirm besteht aus winzigen Leuchtpunkten, den **Pixeln**. Er ist **240 Pixel breit** und **280 Pixel hoch**. Jedes Pixel hat eine Adresse aus zwei Zahlen:

- **x** sagt, wie weit rechts das Pixel liegt. Ganz links ist x = 0.
- **y** sagt, wie weit unten das Pixel liegt. Ganz oben ist y = 0.

Die beiden Zahlen heißen **Koordinaten**. Man schreibt sie in Klammern, x zuerst: (200, 60) ist das Pixel 200 nach rechts und 60 nach unten.



> **Aufgepasst:** Anders als im Matheunterricht zählt **y nach unten**! Je größer y, desto weiter unten auf dem Bildschirm. Der Punkt (0, 0) ist oben links, nicht unten links.

Die x-Werte gehen von 0 bis 239, die y-Werte von 0 bis 279 – zusammen sind das 240 und 280 Pixel, weil bei 0 angefangen wird. Die Mitte des Bildschirms liegt bei (120, 140).

Dieses Programm schreibt in jede Ecke und in die Mitte ein Wort:



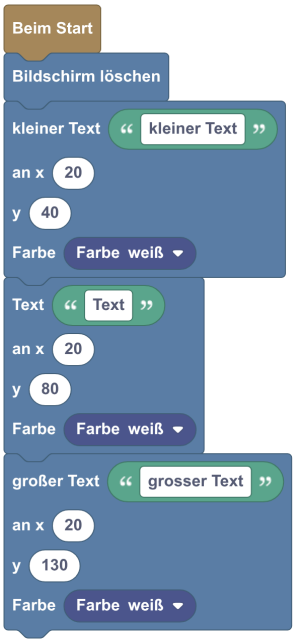


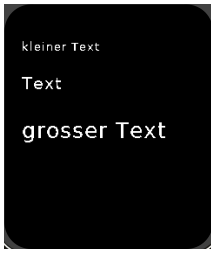
> **Runde Ecken:** Der Bildschirm des PBC+ hat abgerundete Ecken – im Emulator und auf dem echten Gerät. Was ganz in einer Ecke steht, wird ein Stück abgeschnitten. Deshalb haben die Wörter im Beispiel etwas Abstand zum Rand.

Was außerhalb des Bildschirms liegt, wird einfach nicht gezeigt. Schreibst du einen langen Text bei x = 200, siehst du nur seinen Anfang. Eine Fehlermeldung gibt es dabei nicht.

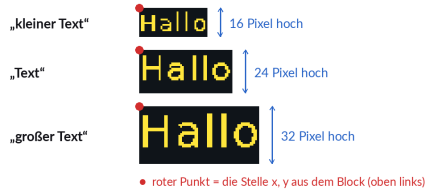
2.4 Drei Textgrößen

In der Kategorie **Display** gibt es drei Text-Blöcke. Sie funktionieren gleich, nur die Schrift ist verschieden groß:





Wichtig ist, **welche Stelle** x und y beim Text meinen: Es ist die Ecke **oben links** des Textes. Der Text wächst von dort nach rechts und nach unten.



Mit diesen Höhen kannst du ausrechnen, wie viel Platz du nach unten lassen musst. Ein großer Text bei y = 100 reicht bis y = 132. Die nächste Zeile sollte also frühestens bei y = 132 beginnen – mit etwas Luft lieber bei 140.

> **Text in die Mitte setzen:** Wie breit ein Text ist, hängt von seinen Buchstaben ab – ein „i“ ist schmaler als ein „M“. Probiere einfach aus: Verändere x so lange, bis links und rechts gleich viel Platz ist.

2.5 Kommentare: Notizen im Programm

Wenn Programme länger werden, verliert man leicht den Überblick. Dafür gibt es **Kommentare**. Ein Kommentar ist eine Notiz für Menschen. Der PBC+ überspringt ihn einfach – er tut nichts.

Den Kommentar-Block findest du ganz unten in der Blockleiste unter **Kommentar**:

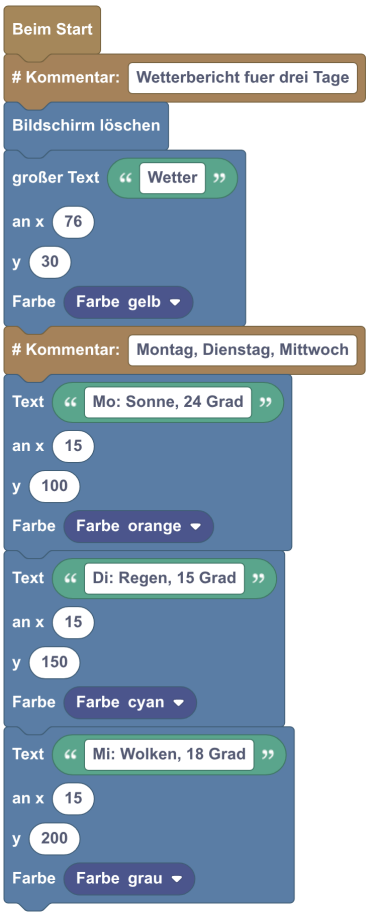


Klicke auf „Notiz“ und schreibe hinein, was der nächste Teil des Programms macht. Kommentare darfst du mit Umlauten schreiben, weil sie nie auf dem Bildschirm erscheinen.

Ausprobieren: Der Wetterbericht

Dieses Programm zeigt das Wetter für drei Tage. Die Überschrift ist groß, die Tage sind mittelgroß. Zwei Kommentare sagen, was die Teile des Programms tun.

Achte auf die y-Werte: Zwischen zwei Zeilen liegen immer 50 Pixel. Der mittlere Text ist 24 Pixel hoch – so bleibt genug Platz dazwischen.





Der Wetterbericht im Emulator

Das Programm findest du als Datei `k02_wetter.pbcp` im Ordner `buch/programme`. Ändere es für deinen Wohnort und die nächsten drei Tage!

Übungsaufgaben

Aufgabe 1 ⭐ **Dein Name in der Mitte** Schreibe deinen Vornamen mit „großer Text“ in die Mitte des Bildschirms. Wähle eine Farbe, die dir gefällt.

Aufgabe 2 ⭐ **Wo landet der Text?** Ein kleiner Text steht bei (180, 10), ein zweiter bei (10, 250). Wo auf dem Bildschirm erscheinen sie? Überlege zuerst, dann probiere es aus.

Aufgabe 3 ⭐⭐ **Die Text-Treppe** Schreibe „Stufe 1“ bis „Stufe 4“ so, dass die Wörter wie eine Treppe nach rechts unten laufen. Beginne das Programm mit einem Kommentar, der sagt, was es tut.

Aufgabe 4 ⭐⭐ **Grüße mit Umlauten** Schreibe „Grüße“ mit einem großen Text auf den Bildschirm. Was erscheint? Repariere den Text so, dass er richtig zu lesen ist.

Aufgabe 5 ⭐⭐⭐ **Genau in der Mitte** Schreibe „ENDE“ mit „großer Text“ so, dass es **genau** in der Mitte des Bildschirms steht – waagerecht und senkrecht. Tipp: Der große Text ist 32 Pixel hoch. Die Breite findest du durch Ausprobieren.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Jedes Programm beginnt bei „Beim Start“. Die Blöcke darunter laufen von oben nach unten.
- „Bildschirm löschen“ macht den Bildschirm schwarz.
- Der Bildschirm ist 240 Pixel breit und 280 Pixel hoch. (0, 0) ist **oben links**, x zählt nach rechts, **y nach unten**.
- Es gibt drei Textgrößen: „kleiner Text“ (16 Pixel hoch), „Text“ (24) und „großer Text“ (32). x und y geben die Ecke oben links des Textes an.
- Die Schrift kennt keine Umlaute: `ae`, `oe`, `ue`, `ss` schreiben.
- Ein Kommentar ist eine Notiz für Menschen. Der PBC+ überspringt ihn.

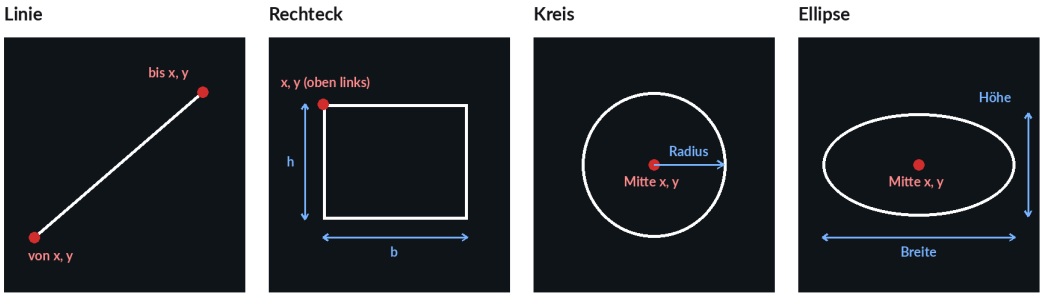
KAPITEL 3

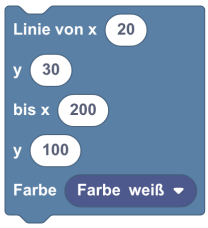
Zeichnen und Farben

Worum geht's? Mit ein paar einfachen Formen – Punkten, Linien, Rechtecken und Kreisen – kannst du erstaunlich viel zeichnen: ein Fenster, ein Gesicht, ein ganzes Haus. Du lernst, wie du jede Farbe mischst, die du dir vorstellen kannst, und wie du fertige Bilder auf den Bildschirm bringst.

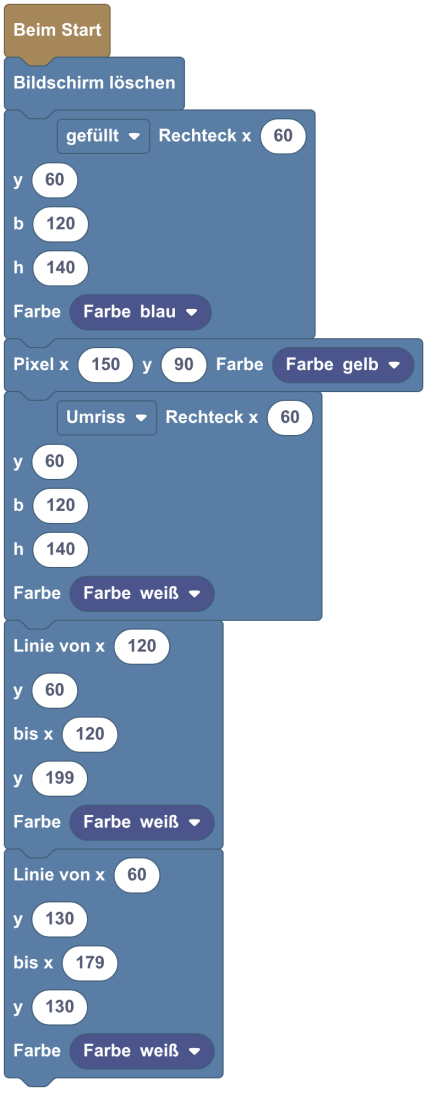
Alle Blöcke in diesem Kapitel findest du in der Kategorie **Display**. Für die Formen brauchst du die Koordinaten aus Kapitel 2: (0, 0) ist oben links, x zählt nach rechts, y nach unten.

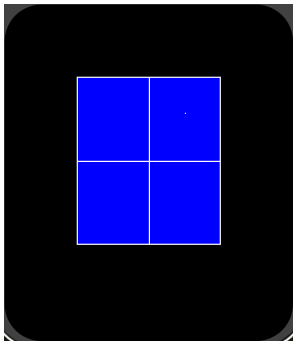
3.1 Pixel, Linie, Rechteck



Block	Was er braucht
	**Pixel** färbt einen einzigen Bildpunkt an der Stelle x, y. Ein Pixel ist so klein, dass du genau hinsehen musst.
	**Linie** zeichnet eine gerade Linie vom Anfang (**von x, y**) bis zum Ende (**bis x, y**).
	**Rechteck** braucht die Ecke **oben links** (x, y), die Breite **b** und die Höhe **h**. Ganz vorne im Block wählst du **Umriss** (nur der Rand) oder **gefüllt** (ganz ausgemalt).

Damit zeichnest du ein Fenster in der Nacht. Der gelbe Pixel ist ein Stern – kannst du ihn entdecken?





Ein Fenster bei Nacht

Das blaue Glas wird zuerst gemalt, danach der weiße Rahmen und das Fensterkreuz. Was später kommt, liegt oben drauf. Mehr dazu in Kapitel 4.

3.2 Ellipse und Kreis

Block	Was er braucht
	Ein **Kreis** braucht seine **Mitte** (x, y) und seinen **Radius** – das ist der Abstand von der Mitte bis zum Rand. Ein Kreis mit Radius 50 ist also 100 Pixel breit.
	Eine **Ellipse** ist ein in die Länge gezogener Kreis. Auch sie braucht ihre Mitte. Dazu kommen ihre ganze **Breite** und ihre ganze **Höhe**.

> **Mitte, nicht Ecke!** Beim Rechteck geben x und y die Ecke oben links an. Bei Kreis und Ellipse ist es die **Mitte**. Und Vorsicht: Beim Kreis gibst du den Radius an (halbe Breite), bei der Ellipse die ganze Breite.

Bei Kreis und Ellipse wählst du **gefüllt** oder **Umriss**. Aus einem gelben Kreis und vier Ellipsen wird ein Smiley:



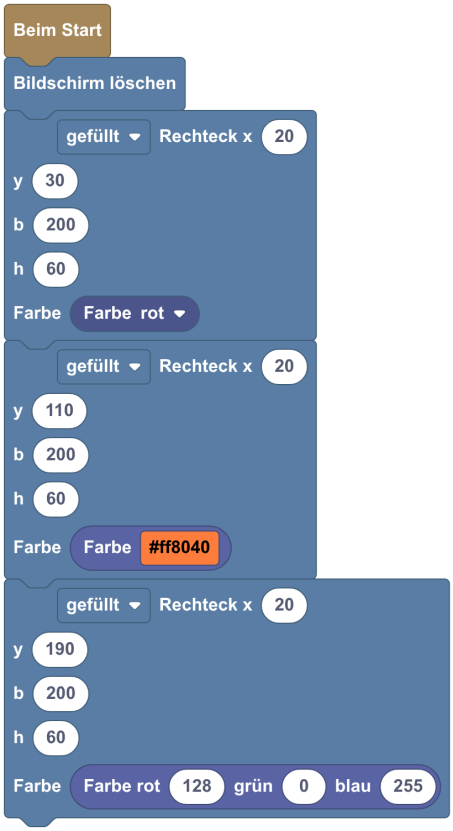


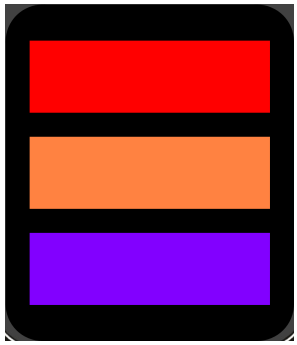
Der Trick steckt im Mund: Zuerst kommt eine **schwarze** Ellipse. Dann malt eine **gelbe** Ellipse, die etwas höher liegt, den oberen Teil wieder zu. Übrig bleibt ein schwarzer Bogen – das Lächeln.

3.3 Farben

Jeder Zeichenblock hat ein Feld **Farbe**. Es gibt drei Arten, eine Farbe festzulegen. Alle drei Blöcke stecken in der Kategorie **Display**, und du kannst sie in jedes Farbfeld ziehen.

Block	So geht's
Farbe rot	**Farbe nach Namen:** Wähle eine von elf Farben aus der Liste: weiß, schwarz, rot, grün, blau, gelb, cyan, magenta, grau, orange, lila.
Farbe #ff8040	**Farbwähler:** Klicke auf das farbige Feld. Es öffnet sich das Fenster „Farbe wählen". Such dir eine Farbe aus und klicke auf **OK**.
Farbe rot 128 grün 0 blau 255	**Farbe(r, g, b):** Du mischst die Farbe selbst aus Rot, Grün und Blau. Jede Zahl geht von 0 (gar nichts) bis 255 (ganz hell).





Drei Rechtecke, drei Arten, die Farbe festzulegen: oben rot nach Namen, in der Mitte Orange aus dem Farbwähler, unten Lila aus Rot 128, Grün 0 und Blau 255.

Wie mischt man mit Licht? Ein Bildschirm mischt Farben aus rotem, grünem und blauem Licht. Das funktioniert anders als mit dem Tuschkasten: Je mehr Licht dazukommt, desto **heller** wird es.

rot	grün	blau	ergibt
255	0	0	Rot
255	255	0	Gelb (Rot + Grün!)
255	255	255	Weiß (alles Licht)
200	80	60	Ziegelrot

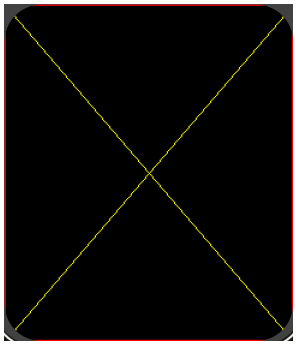
> **Tipp:** Wenn du eine Farbe aus dem Farbwähler genau so in einem anderen Block brauchst, klicke mit der rechten Maustaste auf den Farbblock und wähle **Kopieren**. Die Kopie hat genau dieselbe Farbe.

3.4 Bildschirm BREITE und HÖHE

Der Block „Bildschirm BREITE / HÖHE“ liefert die Zahl **240** (Bildschirmbreite) oder **280** (Bildschirmhöhe). Was du willst, wählst du in der Liste: Bildschirmbreite

Du kannst ihn überall einsetzen, wo eine Zahl stehen darf. Das Programm wird dadurch leichter zu lesen: „Bildschirmbreite“ sagt mehr als „240“.



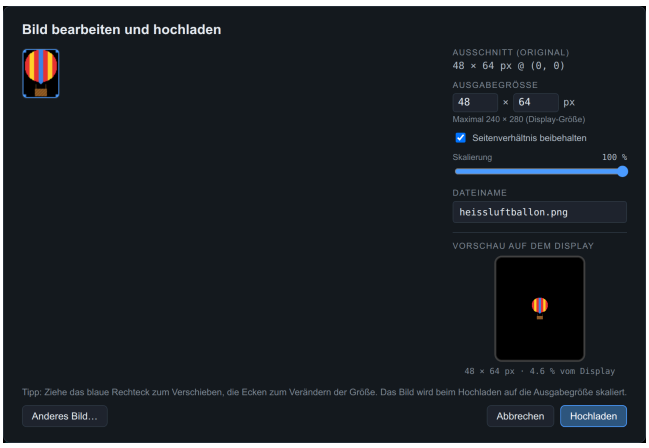


Ein roter Rahmen um den ganzen Bildschirm und zwei Linien quer durch – fertig ist ein großes X. Im Bild siehst du den roten Rahmen in den Ecken nicht ganz, weil die Ecken des Bildschirms abgerundet sind (Kapitel 2).

3.5 Bilder anzeigen

Manches lässt sich mit Formen nur mühsam zeichnen – ein Heißluftballon zum Beispiel. Dafür kannst du fertige Bilder auf den Bildschirm bringen. Die Bilder müssen **PNG-Dateien** sein.

Schritt 1: Bild hochladen. Klicke auf **Dateien → Bilder verwalten…**. Es öffnet sich eine Liste der Bilder, die schon da sind. Klicke auf **PNG hochladen…** und dann auf **Bild von der Festplatte wählen…**.



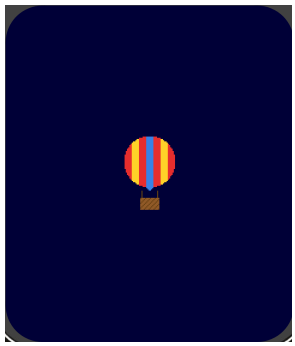
Hier kannst du das Bild noch anpassen:

- **Ausschnitt:** Ziehe das blaue Rechteck, wenn du nur einen Teil des Bildes willst.
- **Ausgabegröße:** So groß wird das Bild auf dem PBC+ in Pixeln. Größer als 240 × 280 geht nicht – ein großes Foto wird automatisch verkleinert.
- **Dateiname:** Unter diesem Namen findest du das Bild später wieder. Verwende keine Leerzeichen und keine Umlaute.

Mit **Hochladen** landet das Bild in der Liste. Im Emulator speichert der Browser das Bild. Arbeitest du mit dem Gerät, wird es auf den PBC+ geladen.

Schritt 2: Bild anzeigen. Hole aus **Display** den Block „Bild anzeigen“. Wähle im Feld neben „Bild“ deine Datei aus. x und y geben die Ecke **oben links** des Bildes an – wie beim Rechteck.

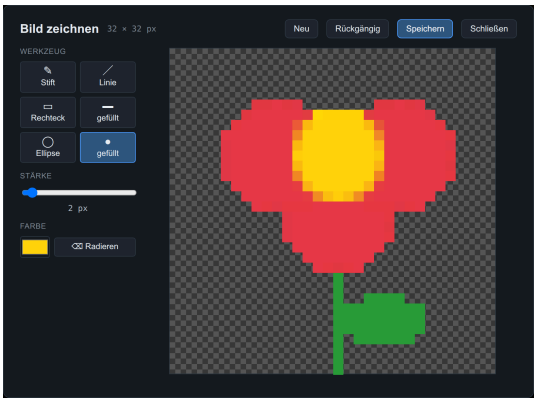




Der Ballon ist 48 Pixel breit und 64 Pixel hoch. Die Stellen um den Ballon herum sind im Bild **durchsichtig**. Deshalb bleibt dort der dunkelblaue Himmel zu sehen. Das Bild `heissluftballon.png` liegt im Ordner `material`.

Ist die Liste im Block „Bild anzeigen“ leer, hast du noch kein Bild hochgeladen. Öffnest du ein Programm auf einem anderen Computer, musst du die Bilder dort auch hochladen.

Mehrere Bilder auf einmal: Packe sie in eine ZIP-Datei und klicke unter **Dateien → Bilder verwalten…** auf **ZIP hochladen…**. Alle PNG-Bilder darin landen in der Liste, so wie sie sind.

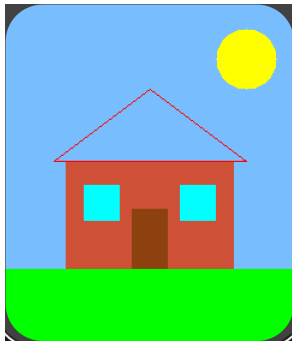


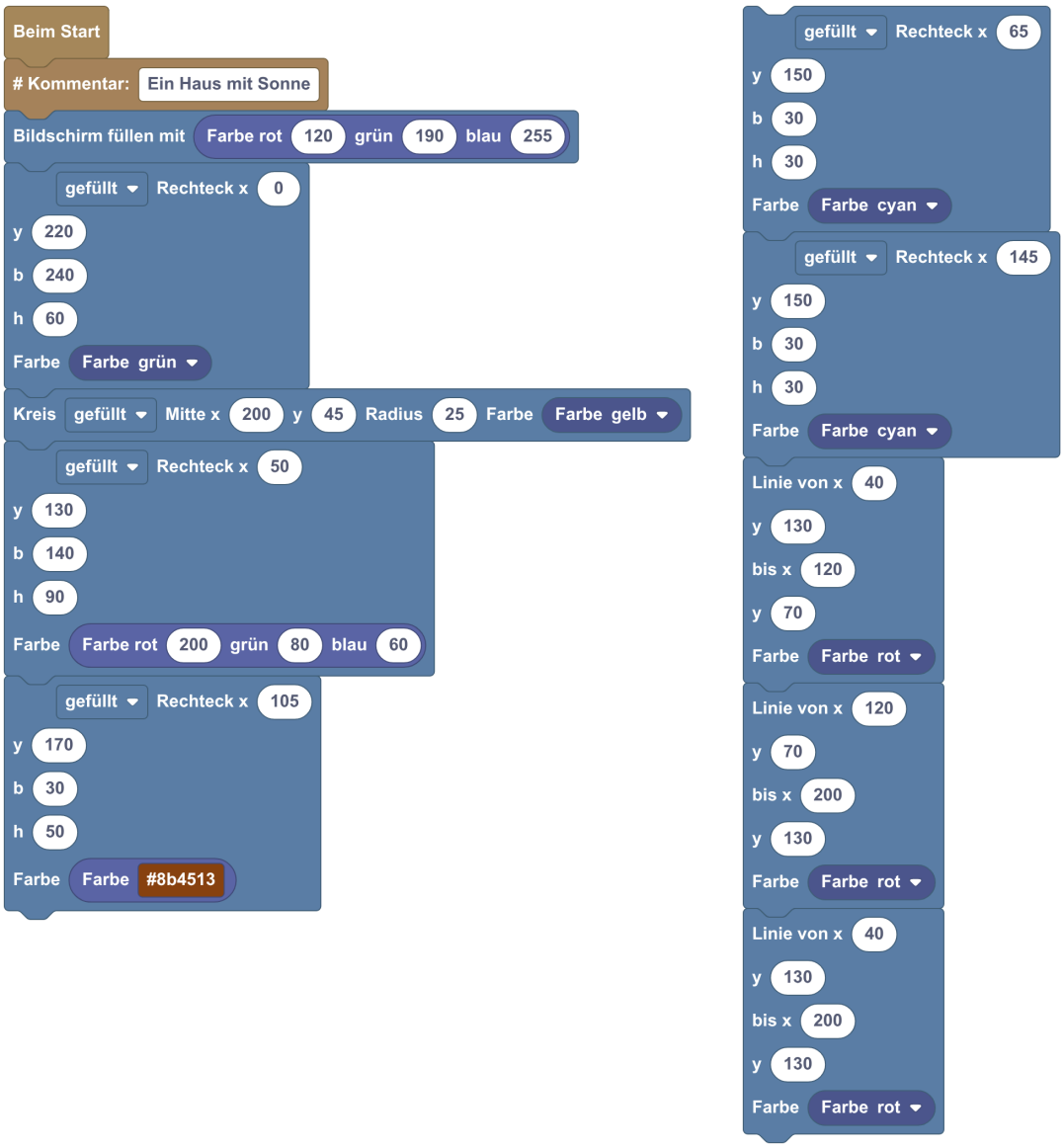
Selbst malen: Mit **Dateien → Bild zeichnen…** malst du ein Bild direkt in PBC+ Studio. Zuerst wählst du die Größe, zum Beispiel 32 × 32 Pixel. Dann malst du mit Stift, Linie, Rechteck und Ellipse. **Radieren** macht Pixel wieder durchsichtig. Mit **Speichern** gibst du dem Bild einen Namen – danach steht es in der Liste wie ein hochgeladenes Bild.

Ausprobieren: Ein Haus mit Sonne

Jetzt kommt alles zusammen: Der Himmel ist hellblau (aus Rot, Grün und Blau gemischt), die Wiese grün, die Sonne gelb. Die Mauer ist ziegelrot, die Tür braun aus dem Farbwähler. Das Dach besteht aus drei Linien.

„Bildschirm füllen“ steht ganz oben. Es malt den ganzen Bildschirm in einer Farbe an. Alles andere kommt danach und liegt darüber. Das Programm ist lang: Der zweite Teil (rechts) hängt direkt unter dem ersten.





... und so geht es weiter.

Das Programm findest du als Datei `k03_haus.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Die Deutschlandfahne** Zeichne die Flagge von Deutschland: drei waagerechte Streifen in Schwarz, Rot und Gold auf weißem Hintergrund. Gold mischst du aus Rot 255, Grün 204, Blau 0.

Aufgabe 2 ⭐ **Der traurige Smiley** Ändere den Smiley aus 3.2 so, dass er traurig schaut. Er soll auch eine hellblaue Träne bekommen.

Aufgabe 3 ⭐⭐ **Eine Ampel** Zeichne eine Ampel: einen grauen Kasten mit drei Lichtern (rot, gelb, grün) und einen grauen Mast darunter. Diese Ampel brauchst du in Kapitel 4 wieder!

Aufgabe 4 ⭐⭐ **Das Zielfernrohr** Zeichne eine waagerechte und eine senkrechte Linie durch die Mitte des Bildschirms, zwei Kreise als Umriss um die Mitte und einen kleinen roten Punkt genau in der Mitte. Verwende für die langen Linien den Block „Bildschirm BREITE / HÖHE“.

Aufgabe 5 ⭐⭐⭐ **Der Schneemann** Zeichne einen Schneemann vor einem hellblauen Himmel: drei weiße Kreise, Augen, eine orangefarbene Karottennase, Knöpfe und einen schwarzen Hut.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- **Pixel** färbt einen Punkt, **Linie** verbindet zwei Punkte.
- **Rechteck:** x und y sind die Ecke **oben links**, dazu Breite b und Höhe h.
- **Kreis** und **Ellipse:** x und y sind die **Mitte**. Der Kreis braucht den Radius, die Ellipse ihre ganze Breite und Höhe.
- Farben gibt es nach Namen, aus dem Farbwähler oder gemischt aus Rot, Grün und Blau (je 0 bis 255).
- „Bildschirmbreite“ ist 240, „Bildschirmhöhe“ ist 280.
- Bilder lädst du mit **Dateien → Bilder verwalten…** hoch oder malst sie mit **Dateien → Bild zeichnen…**. „Bild anzeigen“ zeigt sie an.
- Was später gemalt wird, liegt oben drauf.

KAPITEL 4

Schritt für Schritt: Reihenfolge und Zeit

Worum geht's? Ein Computer ist schnell – so schnell, dass du gar nicht siehst, wie er dein Bild Stück für Stück malt. In diesem Kapitel lernst du, warum die Reihenfolge der Blöcke so wichtig ist, wie du dein Programm warten lässt und wie du misst, wie viel Zeit vergangen ist.

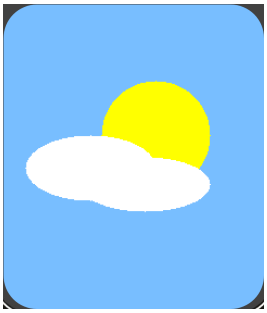
4.1 Blöcke laufen von oben nach unten

Du weißt schon: Der PBC+ führt die Blöcke unter „Beim Start" einen nach dem anderen aus, von oben nach unten. Beim Zeichnen hat das eine wichtige Folge:

> **Was später gemalt wird, liegt oben drauf.**

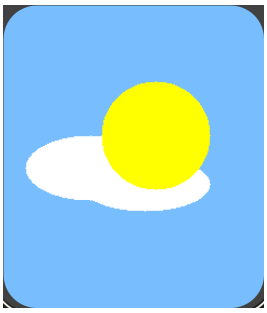
Das ist wie beim Malen mit Deckfarben auf Papier. Sieh dir diese beiden Programme an. Sie enthalten genau dieselben Blöcke – nur die Reihenfolge ist anders:





Erst die Sonne, dann die Wolke





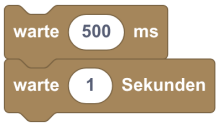
Erst die Wolke, dann die Sonne

Im ersten Programm schiebt sich die Wolke vor die Sonne. Im zweiten Programm wird die Sonne zuletzt gemalt und verdeckt die Wolke.

> **Reihenfolge beim Zeichnen:** Male immer zuerst, was **hinten** ist (Himmel, Hintergrund), und zuletzt, was **vorne** ist. „Bildschirm füllen" gehört deshalb fast immer ganz an den Anfang.

4.2 Warten in Millisekunden und Sekunden

Normalerweise rast der PBC+ durch dein Programm. Willst du, dass etwas eine Weile zu sehen ist, musst du ihn **warten** lassen. Dafür gibt es in der Kategorie **Zeit** zwei Blöcke:



- „warte … ms“ wartet eine Anzahl von **Millisekunden** (kurz: ms). 1000 Millisekunden sind 1 Sekunde. 500 ms sind also eine halbe Sekunde.
- „warte … Sekunden“ wartet ganze Sekunden. Du kannst auch eine Kommazahl eintippen – aber mit **Punkt**: `0.5` ist eine halbe Sekunde.

Während das Programm wartet, passiert nichts. Was auf dem Bildschirm steht, bleibt einfach stehen.

> Ein neuer Warte-Block aus der Blockleiste wartet **0** ms. Vergiss nicht, eine Zahl einzutragen!

Mit Warte-Blöcken baust du einen **Countdown** wie beim Raketenstart:





Nach drei Sekunden: Los!

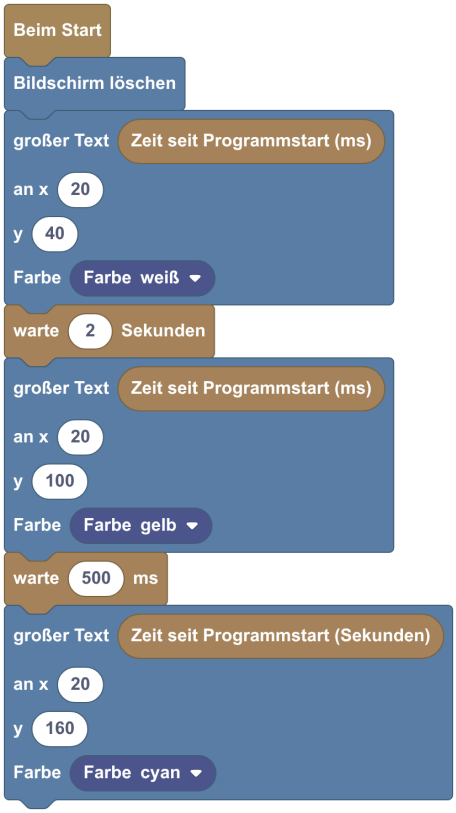
Für jede Zahl gilt dasselbe Muster: Bildschirm löschen, Zahl schreiben, 1 Sekunde warten. Ohne „Bildschirm löschen“ würden die Zahlen übereinander stehen.

4.3 Die Zeit seit Programmstart messen

Der PBC+ hat eine eingebaute Stoppuhr. Sie beginnt beim Programmstart bei 0 und zählt die Millisekunden. In der Kategorie **Zeit** gibt es zwei Blöcke, die diese Uhr ablesen:

- „Zeit seit Programmstart (ms)“ – die Zeit in Millisekunden, eine ganze Zahl
- „Zeit seit Programmstart (Sekunden)“ – dieselbe Zeit in Sekunden, mit Nachkommastellen

Die beiden Blöcke sind **Werte-Blöcke**: Sie haben links eine Nase und passen in ein Feld eines anderen Blocks. Steckst du sie in einen Text-Block, schreibt der PBC+ die Zahl auf den Bildschirm:





Auf dem Bildschirm stehen drei Zahlen. Das sagen sie:

- Schon die erste Zahl ist nicht 0. Bis das Programm den ersten Text schreibt, vergehen ein paar Millisekunden.
- Zwischen der ersten und der zweiten Zahl liegen etwas **mehr** als 2000 ms. Zu den 2 Sekunden Warten kommt die Zeit für das Löschen und Schreiben dazu.
- Die dritte Zahl ist in Sekunden, mit Nachkommastellen. 2.8 bedeutet 2,8 Sekunden – der Computer schreibt Kommazahlen mit Punkt.

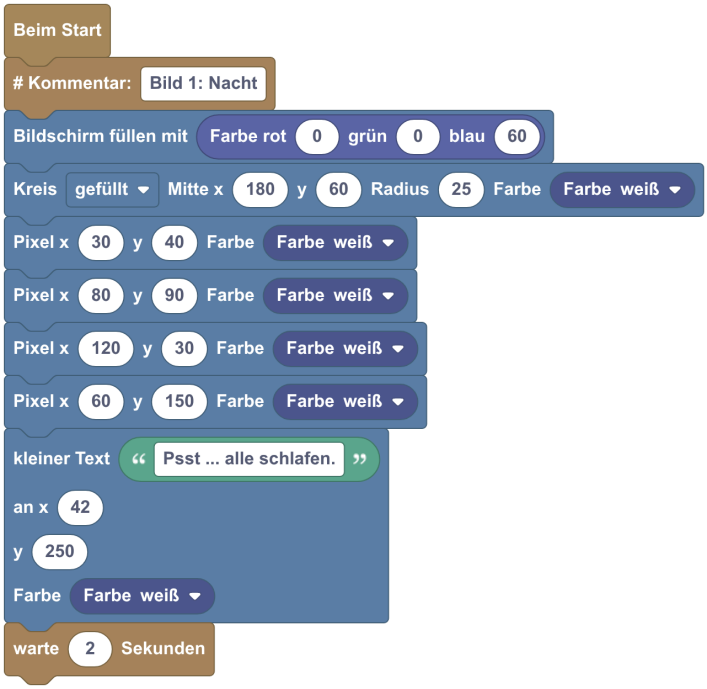
Deine Zahlen werden etwas anders sein – jedes Mal ein bisschen.

> Manchmal zeigt der Sekunden-Block sehr viele Nachkommastellen, zum Beispiel 8.282999999999999. Das ist eine Eigenheit beim Rechnen mit Kommazahlen. Die Millisekunden sind immer ganze Zahlen. Wie du Zahlen rundest, lernst du in Kapitel 17.

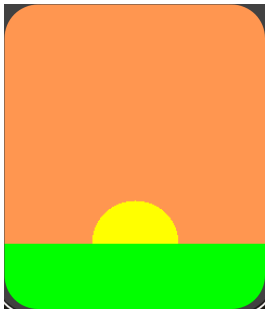
Du wirst die Stoppuhr noch oft brauchen, zum Beispiel für einen Reaktionstest (Kapitel 19).

Ausprobieren: Guten Morgen!

Dieses Programm erzählt eine kleine Geschichte in drei Bildern. Jedes Bild bleibt 2 Sekunden stehen. Am Ende zeigt es, wie viele Millisekunden die Geschichte gedauert hat.

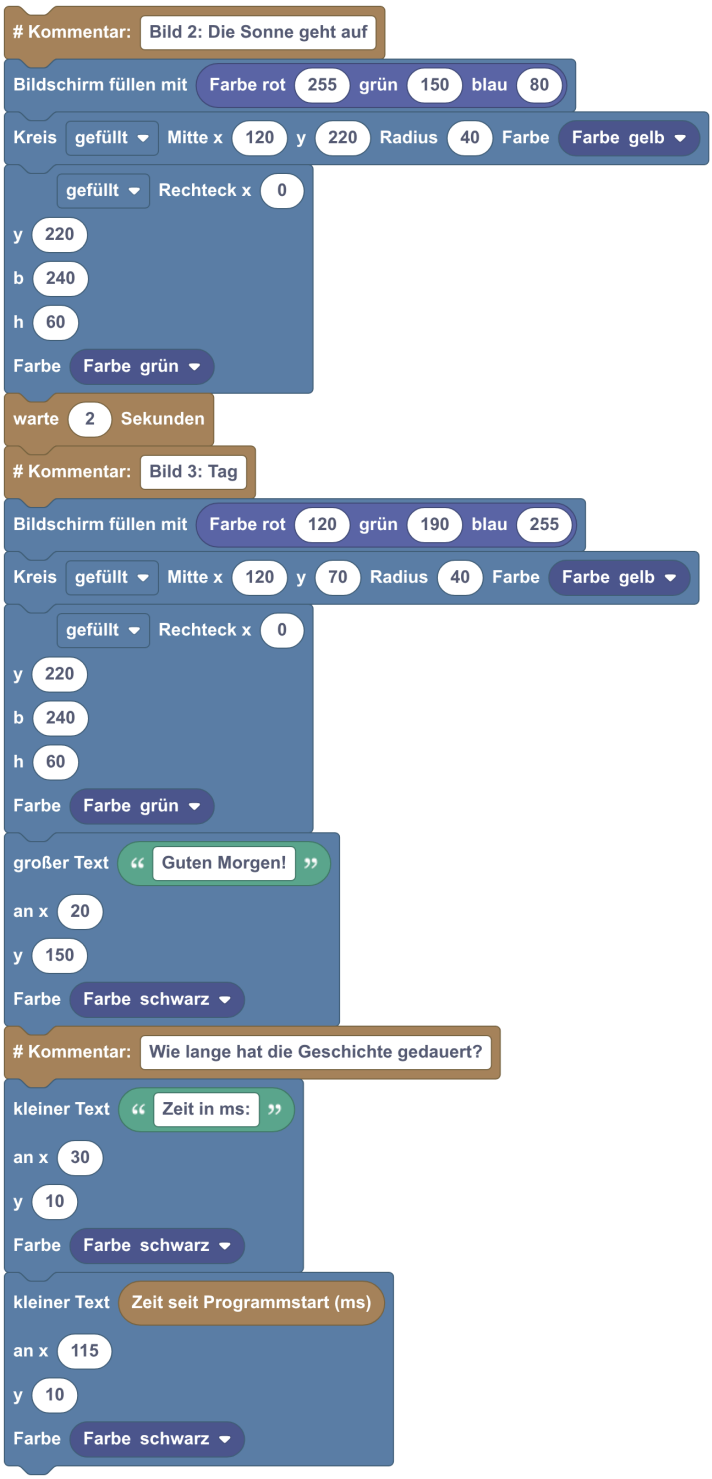








Die Kommentare zeigen, wo ein neues Bild beginnt. Der zweite Teil des Programms (unten) hängt direkt unter dem ersten.



... und so geht es weiter.

Beim Sonnenaufgang wird die Sonne zuerst gemalt und danach die grüne Wiese. Die Wiese verdeckt die untere Hälfte der Sonne – so sieht es aus, als ginge sie gerade auf.

Das Programm findest du als Datei `k04_morgen.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Countdown ab 5** Ändere den Countdown aus 4.2 so, dass er bei 5 beginnt und am Ende „Start!“ zeigt.

Aufgabe 2 ⭐ Wo ist der Smiley? Dieses Programm soll einen Smiley auf blauem Grund zeigen. Doch der Bildschirm bleibt einfach blau. Warum? Repariere das Programm.



Aufgabe 3 ⭐⭐ Die Ampel schaltet Nimm deine Ampel aus Kapitel 3 (Aufgabe 3). Lass sie schalten: rot – rot und gelb – grün – gelb – rot. Ein Licht, das aus ist, malst du dunkelgrau (Rot, Grün und Blau je 60). Wähle passende Wartezeiten.

Aufgabe 4 ⭐⭐ Die Zeitansage Schreibe dreimal untereinander die „Zeit seit Programmstart (ms)“ auf den Bildschirm. Warte dazwischen jeweils 1 Sekunde. Um wie viel wachsen die Zahlen? Warum sind es nicht genau 1000?

Aufgabe 5 ⭐⭐⭐ Achtung, Achtung! Lass „Achtung!“ in Rot dreimal aufblinken: Text zeigen, eine halbe Sekunde warten, Text verschwinden lassen, eine halbe Sekunde warten. Tipp: Den Text lässt du verschwinden, indem du ein schwarzes Rechteck darübermalst. Mit **Kopieren** (rechte Maustaste) sparst du dir Arbeit.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Blöcke laufen der Reihe nach von oben nach unten.
- Was später gemalt wird, liegt oben drauf. Male zuerst den Hintergrund.
- „warte … ms“ wartet Millisekunden, „warte … Sekunden“ ganze Sekunden. 1000 ms = 1 Sekunde.
- Kommazahlen schreibst du mit Punkt: `0.5`.
- „Zeit seit Programmstart“ liest die eingebaute Stoppuhr ab – in Millisekunden oder in Sekunden.
- Löschen, Zeichnen und Schreiben kosten auch ein bisschen Zeit.

KAPITEL 5

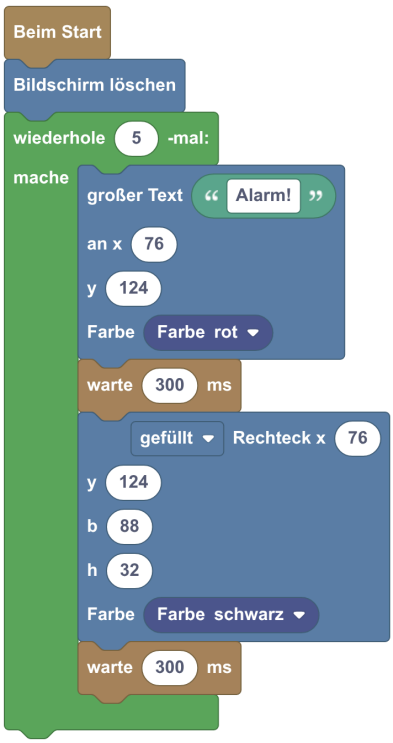
Wiederholungen

Worum geht's? Im letzten Kapitel hast du Blöcke oft kopiert, damit etwas mehrmals passiert. Das geht viel einfacher: mit **Schleifen**. Eine Schleife wiederholt Blöcke – ein paarmal, mit einer Zahl, die mitzählt, oder für immer. Damit zeichnest du Muster und bringst Dinge auf dem Bildschirm in Bewegung.

Alle Schleifen findest du in der Kategorie **Kontrollstrukturen**. Eine Schleife sieht aus wie eine Klammer mit einer Lücke. Die Blöcke, die wiederholt werden sollen, ziehst du **in** die Lücke.

5.1 N-mal wiederholen

Der Block „wiederhole … -mal" führt die Blöcke in seinem Inneren so oft aus, wie die Zahl sagt. Danach geht es unter der Schleife weiter.



So läuft das Programm ab:

1. Der Bildschirm wird gelöscht – **einmal**, denn dieser Block steht vor der Schleife.
2. In der Schleife: „Alarm!" erscheint, 300 ms warten, ein schwarzes Rechteck übermalt den Text, 300 ms warten.
3. Schritt 2 passiert fünfmal. Dann ist das Programm fertig.

Vergleiche mit Aufgabe 5 aus Kapitel 4: Dort brauchtest du für dreimal Blinken zwölf Blöcke. Jetzt sind es fünf – egal, ob es 3-mal, 5-mal oder 100-mal blinken soll.

> Holst du „N-mal wiederholen“ aus der Blockleiste, steht im Zahlenfeld eine **0**. Eine Schleife, die 0-mal wiederholt, tut gar nichts. Trage die Zahl ein, die du brauchst.

5.2 Die Zählschleife „für i von … bis…“

Manchmal soll bei jeder Wiederholung etwas **anders** sein – zum Beispiel die Position. Dafür gibt es die Zählschleife. In der Blockleiste heißt sie „für i von … bis…“:

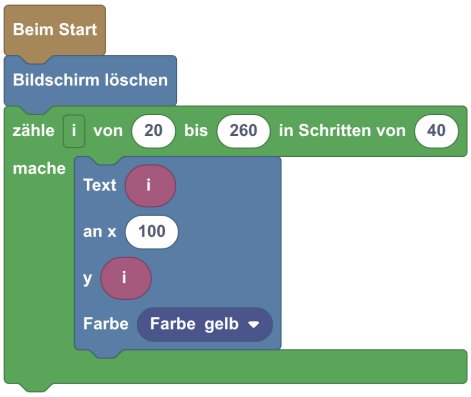


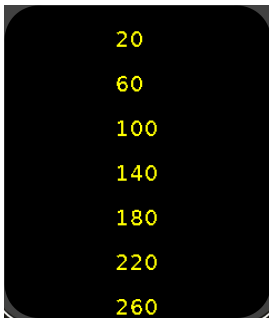
Sie hat einen **Zähler** mit dem Namen `i`. Beim ersten Durchlauf ist `i` so groß wie die Zahl bei „von“. Bei jedem weiteren Durchlauf wird `i` um die Zahl bei „in Schritten von“ größer. Die Schleife hört auf, wenn `i` größer als die Zahl bei „bis“ werden würde.

Den Wert von `i` kannst du in jedes Zahlenfeld stecken. Den Block `i` findest du in der Kategorie **Variablen**, sobald die Zählschleife auf der Arbeitsfläche liegt:



Dieses Programm schreibt den Wert von `i` auf den Bildschirm – und benutzt `i` auch gleich als y-Koordinate:



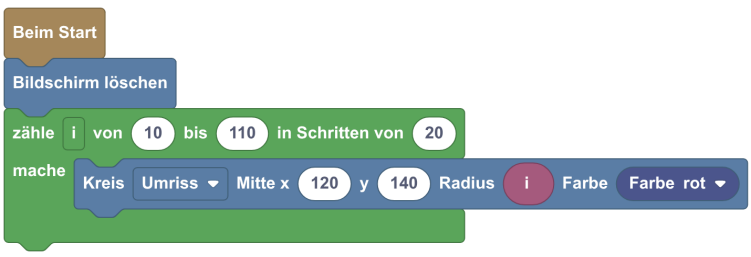


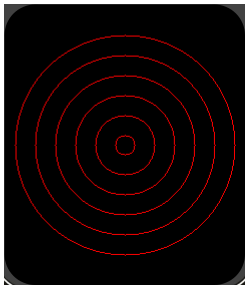
`i` ist nacheinander 20, 60, 100, 140, 180, 220 und 260. Jede Zahl steht genau so weit unten, wie sie groß ist. Nach 260 käme 300 – das ist mehr als 260, also hört die Schleife auf.

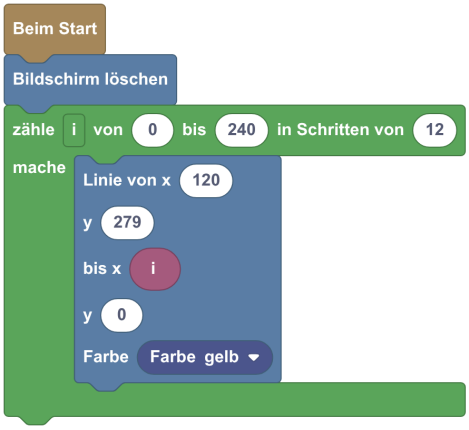
> Ziehe den Block `i` aus der Kategorie **Variablen** in das Feld. Tippst du den Buchstaben i in ein Textfeld, erscheint auf dem Bildschirm einfach der Buchstabe „i“ – und nicht die Zahl.

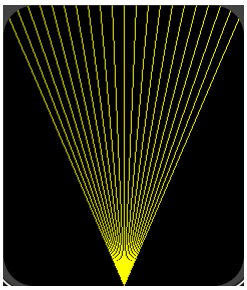
Muster mit der Zählschleife

Mit `i` als Radius entstehen Ringe – eine Zielscheibe. Mit `i` als Endpunkt einer Linie entsteht ein Fächer:







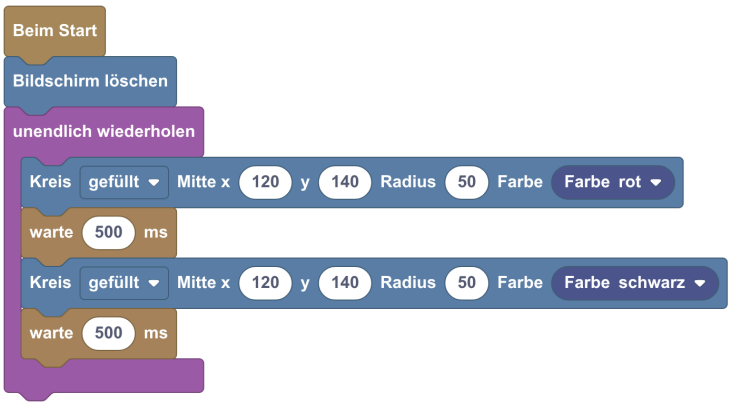


Alle Linien des Fächers beginnen unten in der Mitte bei (120, 279). Ihr Ende wandert am oberen Rand entlang: bei x = 0, 12, 24 … bis 240. Das sind 21 Linien – mit einem einzigen Linien-Block!

> **Rückwärts zählen:** Ist die Zahl bei „von“ größer als die bei „bis“, zählt die Schleife von selbst **rückwärts**. „von 10 bis 1 in Schritten von 1“ ergibt 10, 9, 8 … 1 – praktisch für einen Countdown.

5.3 Unendlich wiederholen

Der Block „unendlich wiederholen“ wiederholt seine Blöcke immer wieder – ohne Ende. Das Programm läuft, bis du auf ■ **Stop** klickst (oder das Gerät ausschaltest).



Dieses Blinklicht leuchtet eine halbe Sekunde, ist eine halbe Sekunde dunkel, leuchtet wieder … für immer.

Fast jedes Spiel hat eine unendliche Schleife: Das Spiel läuft immer weiter, bis man aufhört.

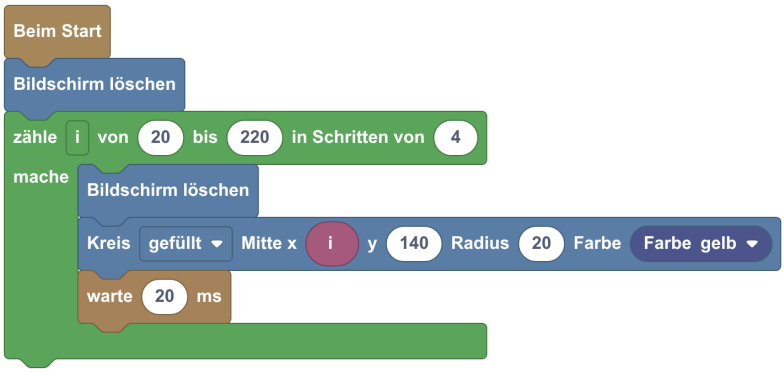
> **Ohne Pause geht's nicht gut.** Enthält eine Endlosschleife keinen Warte-Block, meldet PBC+ Studio beim Testen „Endlosschleife ohne Pause". So eine Schleife hält den PBC+ ständig beschäftigt, und er reagiert vielleicht nicht mehr auf ■ **Stop**. Klicke auf **Abbrechen** und baue einen Block „warte 20 ms" in die Schleife ein.

Blöcke, die du **unter** „unendlich wiederholen" hängst, werden nie ausgeführt – die Schleife hört ja nie auf.

5.4 Animation – und warum es flackert

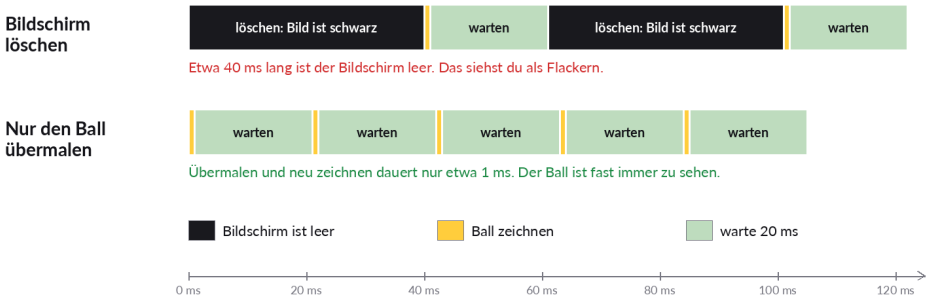
Eine **Animation** ist ein bewegtes Bild. Wie im Daumenkino zeigst du viele Bilder schnell hintereinander. Jedes ist ein kleines bisschen anders.

Ein gelber Ball soll von links nach rechts rollen. Der erste Versuch: In jedem Schritt wird der Bildschirm gelöscht, der Ball an der Stelle `i` gezeichnet und 20 ms gewartet.



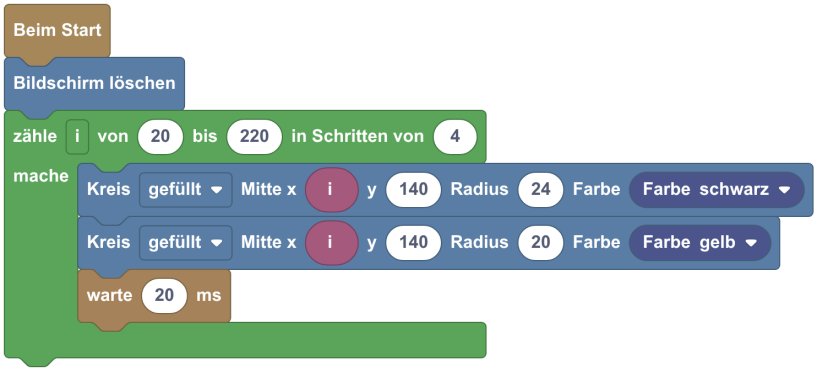
Probiere es aus: Der Ball bewegt sich – aber er **flackert** stark. Mal ist er zu sehen, mal nicht.

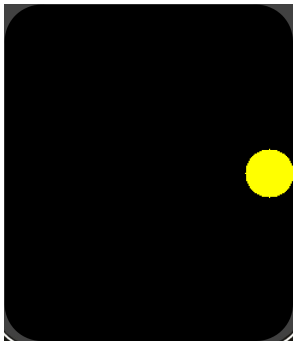
Warum flackert es? Der Bildschirm hat 240 × 280 = 67 200 Pixel. Jedes davon muss beim Löschen einzeln zum Bildschirm geschickt werden. Das dauert eine Weile – etwa 40 Millisekunden. In dieser Zeit ist der Bildschirm schwarz. Der Ball ist danach nur kurz zu sehen, bis das nächste Löschen beginnt. Der Bildschirm ist also **länger leer**, als der Ball zu sehen ist.



Das passiert auf dem echten Gerät genauso wie im Emulator. Der Emulator ahmt extra nach, wie lange das Gerät zum Löschen braucht.

Die Lösung: nur übermalen, was sich bewegt. Statt den ganzen Bildschirm zu löschen, malst du nur die alte Stelle des Balls in der Hintergrundfarbe an. Dazu nimmst du einen schwarzen Kreis, der etwas **größer** ist als der Ball:





Der Ball ist immer zu sehen.

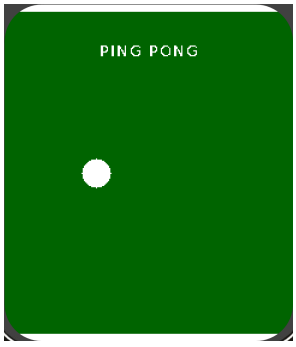
Der Ball hat den Radius 20 und rückt in jedem Schritt 4 Pixel weiter. Der schwarze Kreis hat den Radius 20 + 4 = 24. Er deckt den alten Ball also ganz zu – und gleich danach wird der neue Ball gemalt. Das dauert nur etwa 1 Millisekunde. Der Ball flackert nicht mehr.

> **Regel für Animationen**
>
> Lösche nicht den ganzen Bildschirm. Male nur die alte Stelle in der Hintergrundfarbe an und zeichne sofort danach an der neuen Stelle. Den Hintergrund malst du einmal am Anfang – vor der Schleife.

Ausprobieren: Ping-Pong

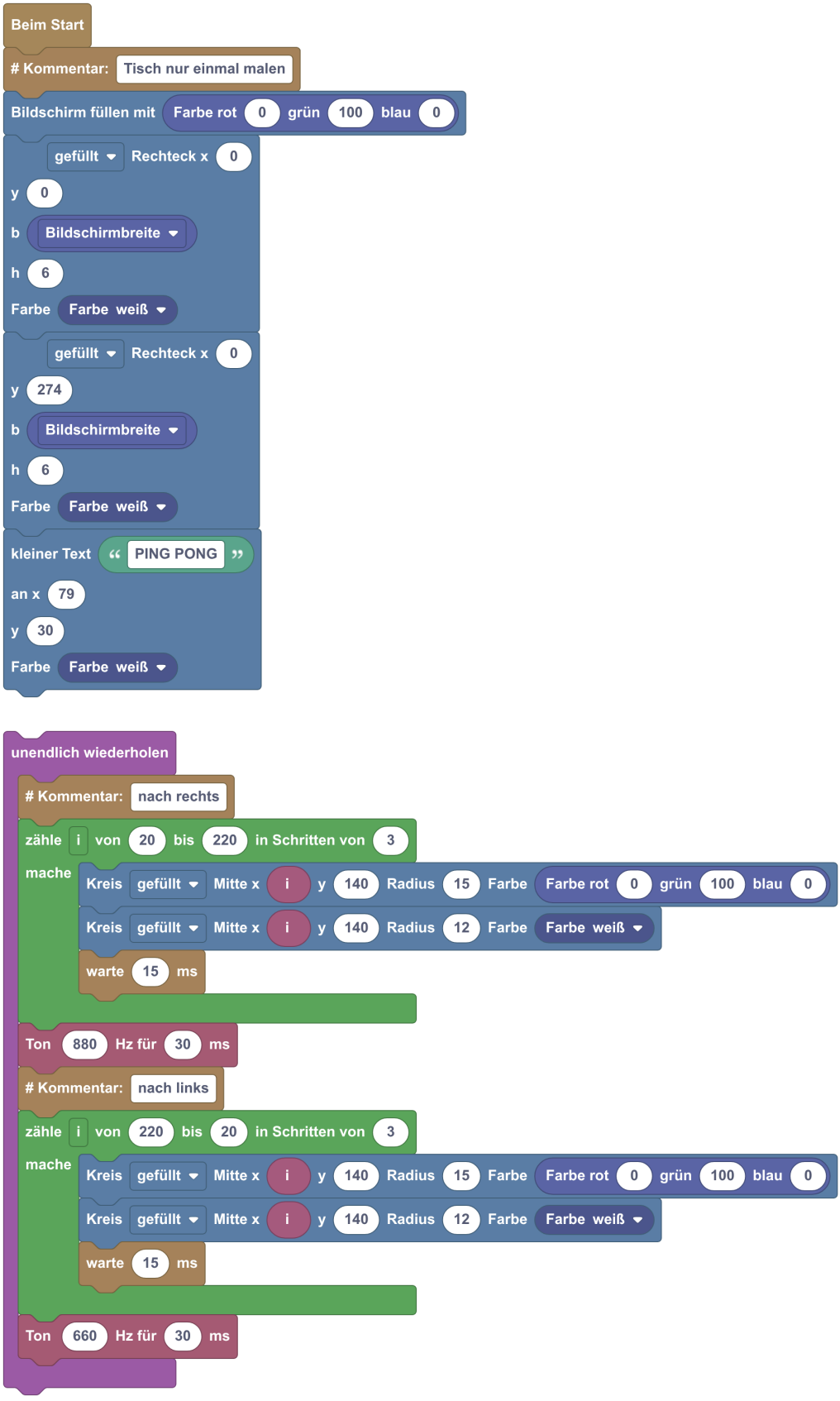
Ein weißer Ball fliegt über einen grünen Tisch – hin und her, für immer. Bei jedem Aufprall ertönt ein kurzer Ton. Hier kommen alle drei Schleifen zusammen:

- Der Tisch wird **einmal** gemalt, vor der Schleife.
- „unendlich wiederholen“ sorgt dafür, dass der Ball nie aufhört.
- Darin laufen zwei Zählschleifen: eine zählt von 20 bis 220 (nach rechts), die andere von 220 zurück bis 20 (nach links).



Der Ball fliegt hin und her.

Der Ball wird mit einem grünen Kreis übermalt, denn der Tisch ist grün. Die grüne Farbe muss **genau** dieselbe sein wie beim „Bildschirm füllen“ – sonst sieht man eine Spur. Der zweite Teil des Programms hängt direkt unter dem ersten.



... und so geht es weiter.

Den Block „Ton ... Hz für ... ms“ findest du in der Kategorie **Sound**. Mehr über Töne erfährst du in Kapitel 9. Das Programm findest du als Datei `k05_pingpong.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Schneller blinken** Ändere das Programm aus 5.1: „Alarm!“ soll zehnmal in Gelb blinken, und zwar schneller.

Aufgabe 2 ⭐ **Enge Ringe** Ändere bei der Zielscheibe aus 5.2 die Schrittweite von 20 auf 5. Was passiert? Wie viele Ringe sind es jetzt?

Aufgabe 3 ⭐⭐ **Karopapier** Zeichne mit zwei Zählschleifen ein Gitter aus blauen Linien: senkrechte Linien im Abstand von 20 Pixeln und waagerechte Linien im Abstand von 20 Pixeln.

Aufgabe 4 ⭐⭐ **Countdown mit Schleife** Baue einen Countdown von 10 bis 1 mit nur **einer** Zählschleife. Jede Zahl steht 1 Sekunde lang groß in der Mitte. Am Ende erscheint „Start!“.

Aufgabe 5 ⭐⭐⭐ **Ein volles Dach** Das Dach des Hauses aus Kapitel 3 ist nur ein Umriss. Fülle es mit einer Zählschleife aus: Zeichne viele Linien von der Dachspitze (120, 70) zu den Punkten (i, 130), wobei i von 40 bis 200 läuft.

Aufgabe 6 ⭐⭐⭐ **Regentropfen** Ein blauer Tropfen soll immer wieder von oben nach unten fallen – ohne zu flackern. Tipp: Lass `i` bei −10 beginnen und bis 290 laufen. Dann startet der Tropfen oberhalb des Bildschirms und verschwindet unten ganz.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Eine Schleife wiederholt die Blöcke in ihrem Inneren.
- „wiederhole … -mal“ wiederholt eine feste Anzahl von Malen.
- „für i von … bis… in Schritten von …“ zählt mit. Den Zähler `i` holst du aus **Variablen** und steckst ihn in Zahlenfelder.
- Ist „von“ größer als „bis“, zählt die Schleife rückwärts.
- „unendlich wiederholen“ hört nie auf – nur ■ **Stop** beendet sie. Ein Warte-Block gehört hinein.
- Für Animationen nicht den ganzen Bildschirm löschen, sondern nur die alte Stelle übermalen. Sonst flackert es.

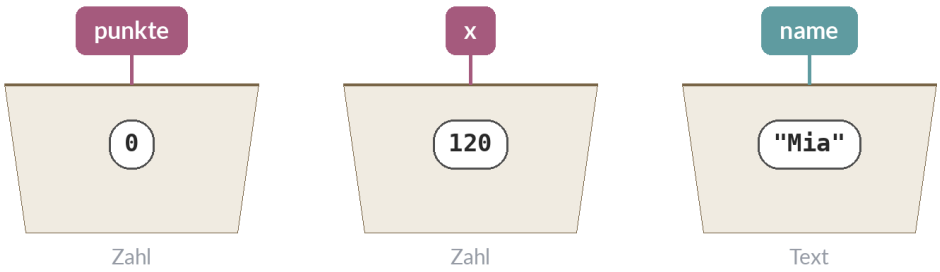
KAPITEL 6

Variablen

Worum geht's? Ein Programm muss sich vieles merken: den Punktestand, wo der Ball gerade ist, wie die Spielerin heißt. Dafür gibt es **Variablen**. In diesem Kapitel legst du Variablen an, rechnest mit ihnen, lässt den Zufall mitspielen und speicherst auch Wörter.

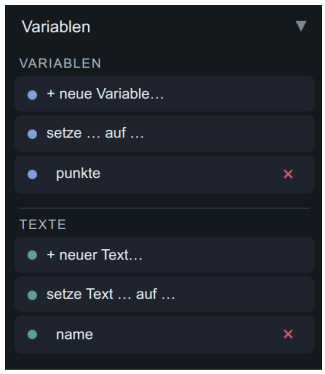
6.1 Was ist eine Variable?

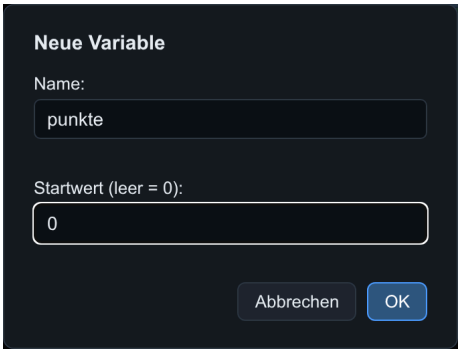
Eine **Variable** ist wie eine Schachtel mit einem Namensschild. In der Schachtel liegt ein Wert, zum Beispiel eine Zahl. Über den Namen findet das Programm die Schachtel wieder. Den Inhalt kann es jederzeit herausholen (lesen) oder austauschen (setzen).



Eine Variable anlegen

1. Klappe in der Blockleiste die Kategorie **Variablen** auf.
2. Klicke auf **„+ neue Variable…“**. Ein Fenster öffnet sich.
3. Gib bei **Name** einen Namen ein, zum Beispiel `punkte`.
4. Bei **Startwert** kannst du eine Zahl eintragen. Lässt du das Feld leer, beginnt die Variable mit 0.
5. Klicke auf **OK**.





Jetzt steht `punkte` in der Kategorie **Variablen**. Ein Klick darauf holt einen Block, der den Wert der Variable liest.

Hast du einen Startwert eingetragen, hängt PBC+ Studio den passenden Block „setze punkte auf 0“ automatisch direkt unter „Beim Start“.

Setzen und lesen

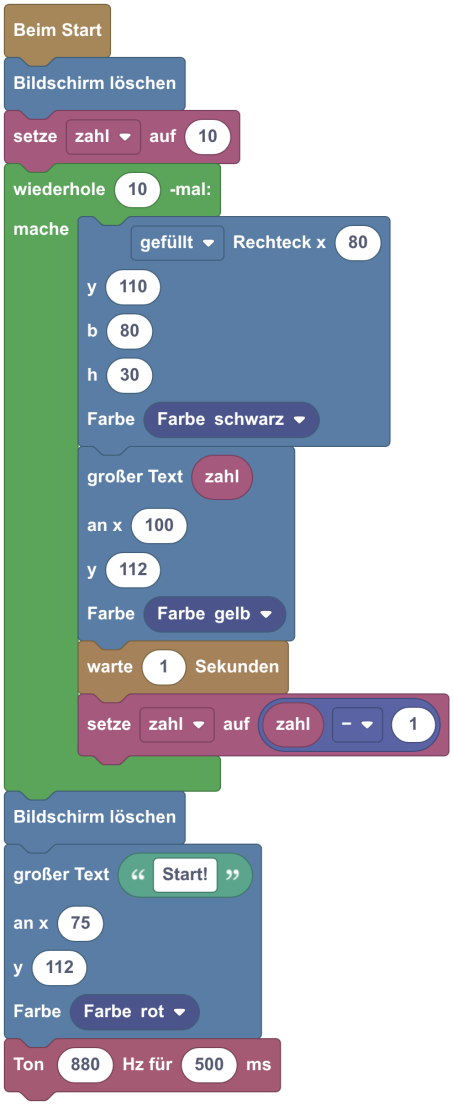


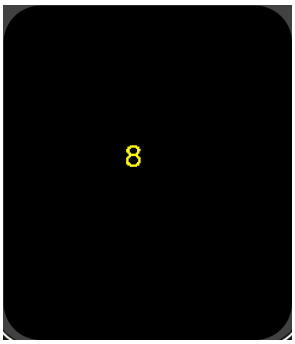
- **„setze … auf…“** legt einen neuen Wert in die Schachtel. Der alte Wert ist danach weg. Im Feld ▾ wählst du, welche Variable gemeint ist.
- Der **runde Block mit dem Namen** liest den Wert. Du kannst ihn überall einsetzen, wo sonst eine Zahl steht: bei x und y, bei einer Wartezeit oder in einem Text-Block.

Das Besondere: Eine Variable kann sich **selbst verändern**. Dieser Block nimmt den alten Wert von `zahl`, zieht 1 ab und legt das Ergebnis wieder in `zahl`:



Liest man ihn laut, klingt er seltsam: „setze zahl auf zahl minus 1“. Er bedeutet aber einfach: **Mach `zahl` um 1 kleiner.** Damit baust du einen Countdown für eine Rakete:





`zahl` beginnt bei 10. In jedem Durchlauf der Schleife zeigt das Programm `zahl` an, wartet eine Sekunde und macht `zahl` um 1 kleiner. Nach zehn Durchläufen erscheint „Start!“.

> **Warum das schwarze Rechteck?** Ein Text-Block malt nur die Buchstaben. Den alten Text darunter löscht er nicht. Ohne das Rechteck stünde die 9 über der 10, und du könntest nichts mehr lesen. Deshalb übermalst du zuerst die alte Zahl mit einem schwarzen Rechteck und schreibst dann die neue.

> **Gute Namen:** Ein Name sagt, was in der Schachtel liegt: `punkte`, `leben`, `x`. Er besteht aus Buchstaben, Ziffern und dem Unterstrich `_` und beginnt nicht mit einer Ziffer. Schreibe ihn ohne Umlaute und ß: `groesse` statt `größe`. Wörter wie `print` oder `if` sind verboten, weil die Programmiersprache sie selbst braucht – das Fenster sagt dir dann Bescheid.

6.2 Rechnen

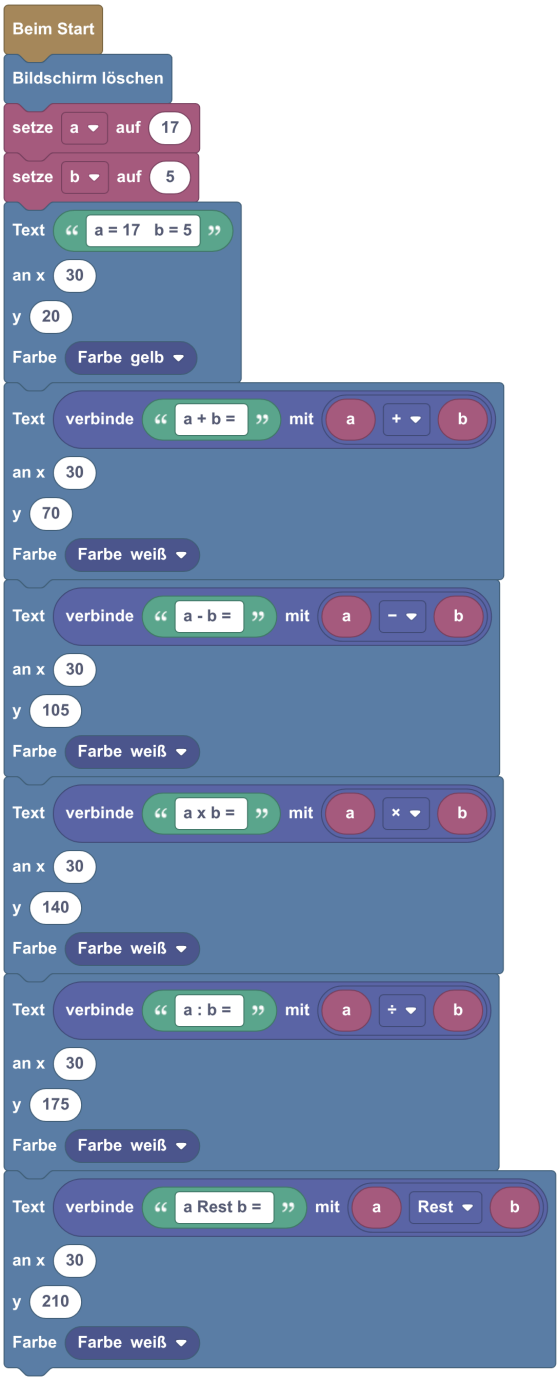
In der Kategorie **Mathe** findest du den Rechen-Block. Im Feld ▾ in der Mitte wählst du die Rechenart: **+**, **−**, **×**, **÷** oder **Rest**.

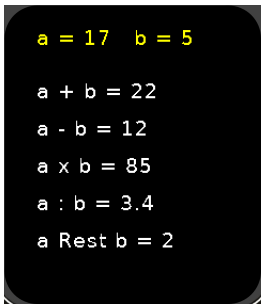


In die beiden Felder kannst du Zahlen, Variablen oder weitere Rechen-Blöcke stecken. Ein Block, der in einem anderen steckt, wird **zuerst** ausgerechnet. Dieser Block rechnet also erst 2 + 3 = 5 und dann 5 × 4 = 20:



Das Programm unten rechnet mit den Variablen `a` = 17 und `b` = 5:





Zwei Rechenarten verdienen einen zweiten Blick:

- **÷ (geteilt)** liefert immer eine Kommazahl. Der PBC+ schreibt sie, wie im Englischen, mit einem **Punkt**: 17 ÷ 5 ergibt `3.4`. Sogar 6 ÷ 2 ergibt `3.0` und nicht `3`. Brauchst du eine ganze Zahl, steck das Ergebnis in den Block „... runden“ (Kategorie **Mathe**).
- **Rest** ist der Rest beim Teilen. 17 geteilt durch 5 ist 3, Rest **2** – denn 3 × 5 = 15, und bis 17 fehlen noch 2. Mit dem Rest findest du heraus, ob eine Zahl gerade ist: Bei geraden Zahlen ist „zahl Rest 2“ gleich 0.

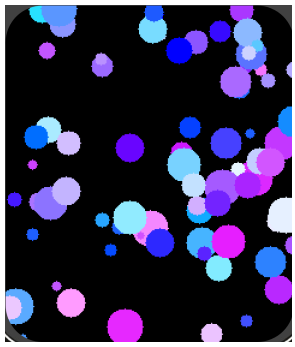
> **Umlaute auf dem Bildschirm:** Die Schrift des PBC+ kennt keine Umlaute und kein ß. Statt „Würfe“ erscheint „W?rfe“. Schreibe in Text-Blöcken deshalb „Wuerfe“, „gross“ und „heisse“. Auch × und ÷ fehlen in der Schrift – nimm `x` und `:`.

6.3 Zufallszahlen

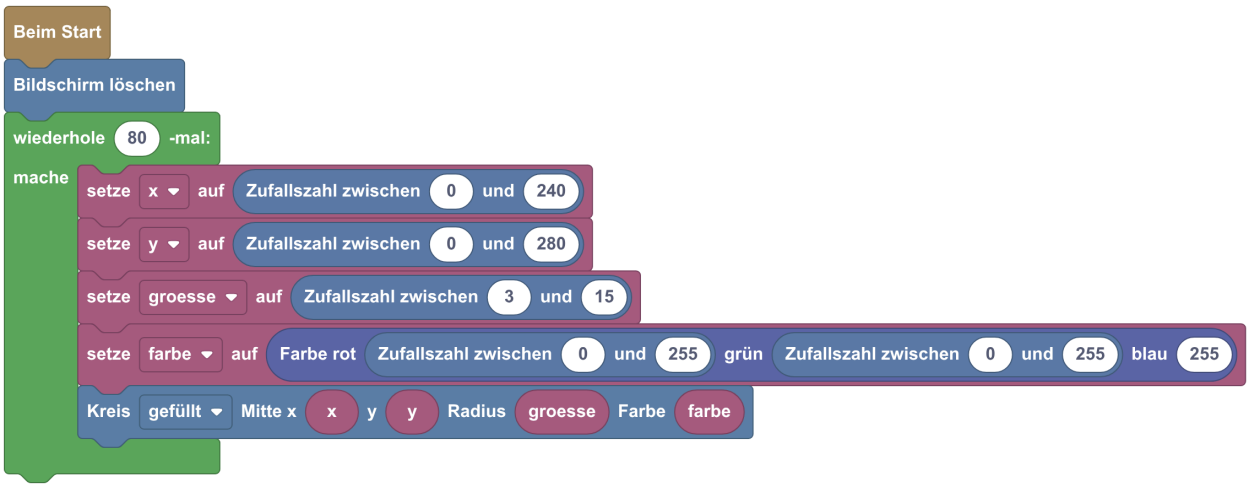
Der Block „Zufallszahl zwischen … und …“ aus der Kategorie **Mathe** würfelt dir eine ganze Zahl. Beide Grenzen sind möglich: Dieser Block liefert 1, 2, 3, 4, 5 oder 6.



Jedes Mal, wenn das Programm an diesem Block vorbeikommt, entsteht eine neue Zahl. So malt das folgende Programm 80 Kreise an zufällige Stellen, in zufälligen Größen und Farben. Starte es mehrmals – das Bild ist jedes Mal anders.



Konfetti



> **Eine Zufallszahl merken:** Den Radius würfelt das Programm zuerst in die Variable `groesse`. Jeder Block „Zufallszahl“ würfelt nämlich neu, sobald er an der Reihe ist. Allgemein gilt: **Brauchst du dieselbe Zufallszahl mehr als einmal, speichere sie in einer Variable.**

6.4 Texte als Variablen

Eine Variable kann auch ein Wort oder einen Satz speichern. So legst du sie an:

1. Klicke in der Kategorie **Variablen** im Abschnitt **Texte** auf **„+ neuer Text…“**.
2. Gib einen Namen ein, zum Beispiel `name`, und als Startwert ein Wort, zum Beispiel `Mia`.

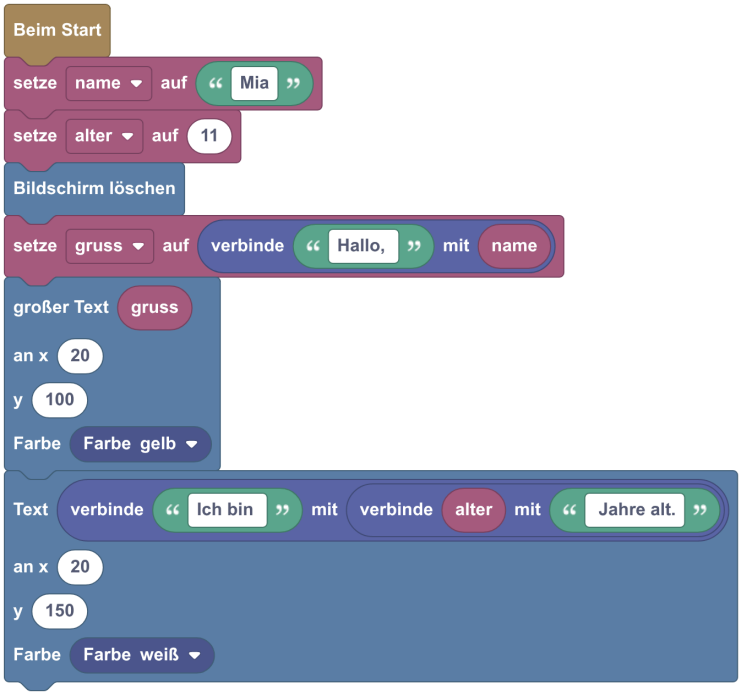
PBC+ Studio hängt dann „setze name auf „Mia““ unter „Beim Start“. Ein Text steht im Block immer zwischen den Anführungszeichen „ und “.



Mit „verbinde … mit…“ aus der Kategorie **Text** hängst du zwei Teile aneinander. Das klappt mit Texten und mit Zahlen:



Ist `punkte` gerade 7, ergibt das „Punkte: 7“. Achte auf das Leerzeichen hinter dem Doppelpunkt, sonst steht da „Punkte:7“. Für mehr als zwei Teile steckst du ein „verbinde“ in ein anderes:

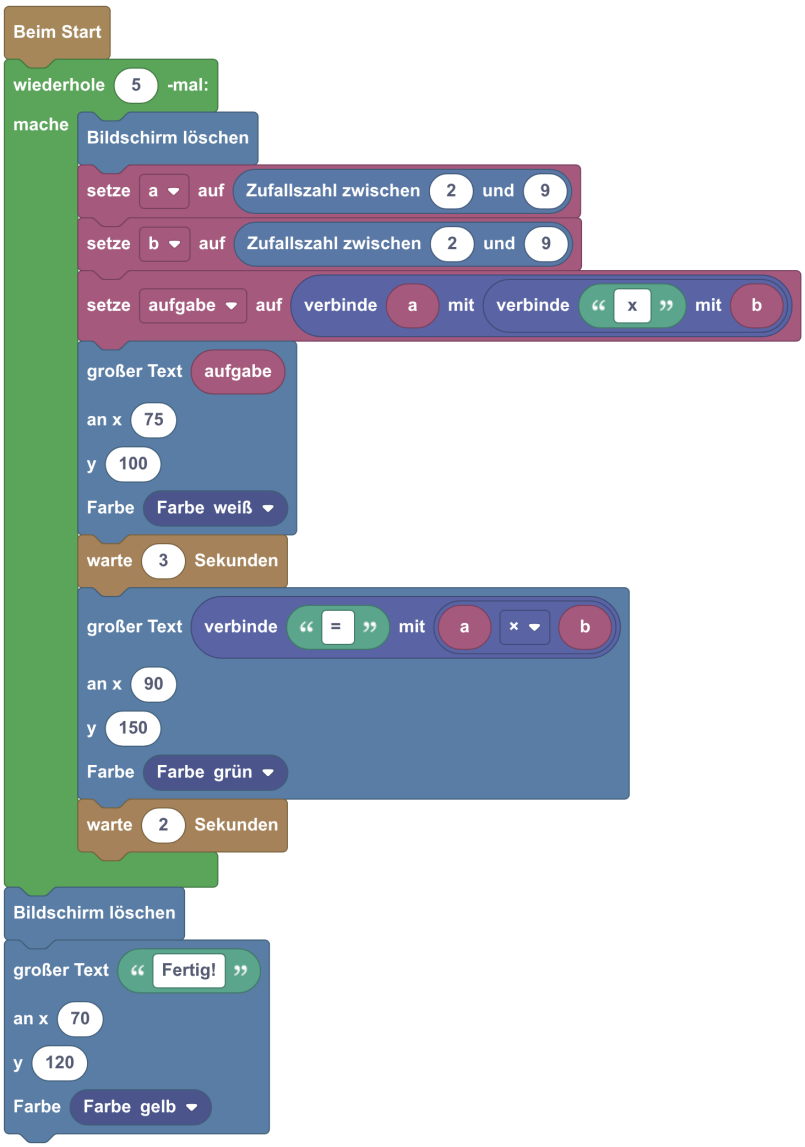




> **3 + 4 oder „3“ und „4“?** Der Rechen-Block rechnet 3 + 4 = 7. Der Block „verbinde 3 mit 4“ hängt die Zeichen aneinander und ergibt „34“. Willst du rechnen, nimm den Rechen-Block.

Ausprobieren: Der Rechentrainer

Der PBC+ stellt dir fünf Malaufgaben. Rechne im Kopf, bevor nach drei Sekunden das Ergebnis erscheint. Das Programm würfelt die beiden Zahlen `a` und `b`, baut daraus den Text `aufgabe` (zum Beispiel „7 x 8“) und rechnet das Ergebnis mit dem Rechen-Block aus. `aufgabe` ist eine Text-Variable, `a` und `b` sind Zahl-Variablen.



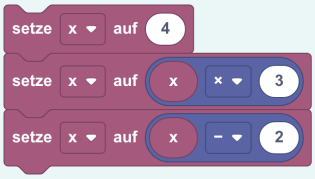


Das Programm findest du als Datei `k06_rechentrainer.pbcp` im Ordner `buch/programme`. Ändere die Zufallszahlen, wenn dir die Aufgaben zu leicht sind.

Übungsaufgaben

Aufgabe 1 ⭐ **Countdown in Zweierschritten** Ändere den Countdown aus 6.1: Er soll bei 20 beginnen und in Zweierschritten rückwärts zählen: 20, 18, 16 … 0. Wie oft muss die Schleife jetzt laufen?

Aufgabe 2 ⭐ **Was kommt heraus?** Rechne im Kopf: Welchen Wert hat `x` nach diesen drei Blöcken? Prüfe deine Antwort mit einem Text-Block.



Was zeigt ein Text-Block mit „verbinde 3 mit 4“ an, und was einer mit „3 + 4“?

Aufgabe 3 ⭐⭐ Sechs Würfe Lass sechs Würfelzahlen untereinander erscheinen. Verwende eine Variable `y` für die Zeile. Sie beginnt bei 20 und wird nach jedem Wurf um 40 größer.

Aufgabe 4 ⭐⭐ Mein Steckbrief Lege die Text-Variablen `name` und `hobby` und die Zahl-Variable `alter` an. Das Programm soll anzeigen: „Ich heisse …“, „Ich mag …“, „Ich bin … Jahre alt.“ und „In 5 Jahren bin ich …“ – die letzte Zahl soll das Programm selbst ausrechnen.

Aufgabe 5 ⭐⭐⭐ Zeit umrechnen Würfle eine Zahl `sekunden` zwischen 60 und 600. Zeige an, wie viele Minuten und Sekunden das sind, zum Beispiel „347 Sekunden sind 5 min 47 s“. *Tipp:* Die Sekunden, die übrig bleiben, liefert der Rest-Block. Ziehst du sie von `sekunden` ab, lässt sich der Rest genau durch 60 teilen.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Eine **Variable** ist eine Schachtel mit Namen, in der ein Wert liegt.
- Anlegen: Kategorie **Variablen** → „+ neue Variable…“ (Zahl) oder „+ neuer Text…“. Namen ohne Umlaute und Leerzeichen.
- „setze … auf…“ legt einen neuen Wert hinein; der runde Block mit dem Namen liest ihn. „setze zahl auf zahl + 1“ macht `zahl` um 1 größer.
- Rechnen mit **+ − × ÷ Rest**. ÷ liefert eine Kommazahl mit Punkt (3.4).
- „Zufallszahl zwischen 1 und 6“ würfelt bei jedem Durchlauf neu. Brauchst du dieselbe Zahl mehrmals, merke sie dir in einer Variable.
- „verbinde … mit…“ hängt Texte und Zahlen aneinander. Auf dem Bildschirm keine Umlaute verwenden.

KAPITEL 7

Entscheidungen

Worum geht's? Bisher hat dein Programm immer dasselbe getan. Jetzt lernt es, Fragen zu stellen und je nach Antwort etwas anderes zu tun: Ist es kalt? Ist Wochenende? Ist schon eine Sechs gefallen? Dafür gibt es „falls", Vergleiche und die „solange"-Schleife.

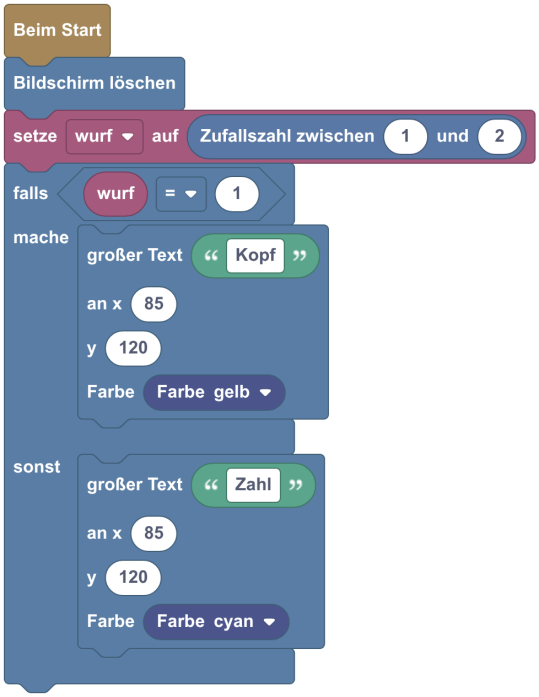
7.1 „falls" und „falls / sonst"

In der Kategorie **Kontrollstrukturen** findest du zwei Entscheidungs-Blöcke: „falls" und „falls / sonst". Oben in die Lücke mit der spitzen Form kommt eine **Bedingung** – eine Frage, auf die es nur zwei Antworten gibt: **wahr** oder **falsch**.



Ist die Bedingung wahr, führt der PBC+ die Blöcke bei **mache** aus. Ist sie falsch, überspringt er sie einfach. Hier ertönt der Ton also nur, wenn `wurf` gleich 1 ist.

Beim Block „falls / sonst" gibt es zwei Wege. Ist die Bedingung wahr, laufen die Blöcke bei **mache**, sonst die Blöcke bei **sonst**. Einer der beiden Wege wird immer genommen – nie beide. Das passt zu einem Münzwurf:



Die Zufallszahl `wurf` ist 1 oder 2. Bei 1 zeigt der PBC+ „Kopf", sonst „Zahl". Starte das Programm ein paarmal mit ▶ **Testen**.

> Einen Block „sonst falls" gibt es nicht. Willst du zwischen drei Möglichkeiten wählen, steckst du ein zweites „falls / sonst" in den **sonst**-Teil des ersten (siehe 7.2).

7.2 Vergleichen: = ≠ < >

Bedingungen baust du meist mit dem **Vergleich**-Block aus der Kategorie **Mathe**. Er vergleicht zwei Werte und antwortet mit wahr oder falsch.



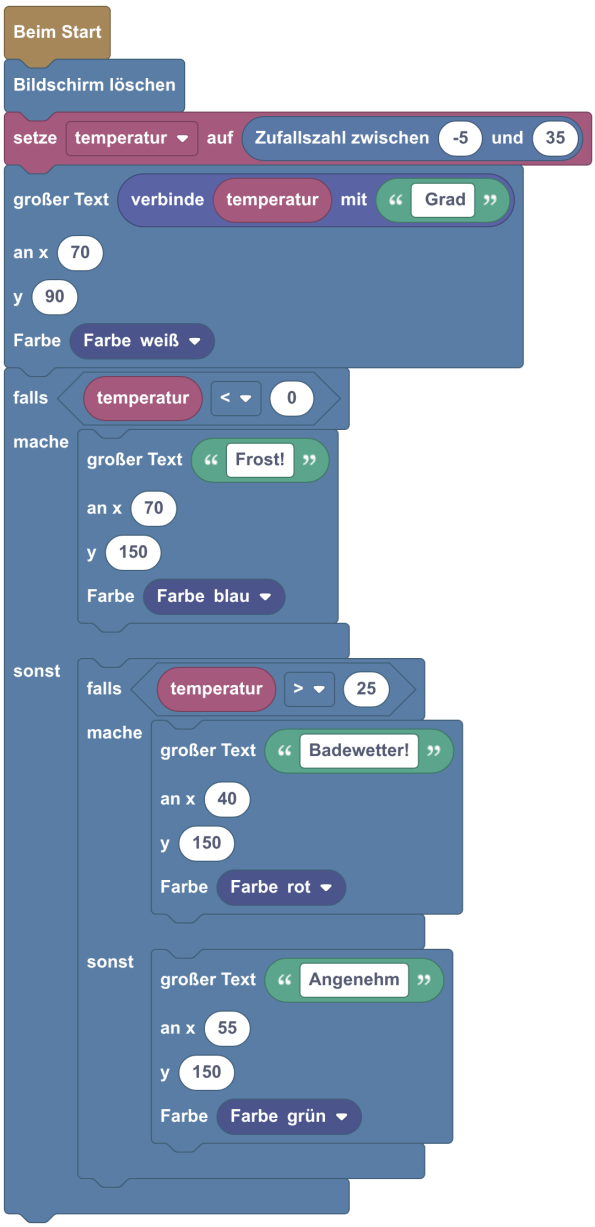
Im Feld ▾ in der Mitte wählst du, wie verglichen wird:

Zeichen	bedeutet	Beispiel	Antwort
=	ist gleich	4 = 4	wahr
≠	ist nicht gleich	4 ≠ 4	falsch
<	ist kleiner als	3 < 5	wahr
≤	ist kleiner oder gleich	5 ≤ 5	wahr
>	ist größer als	3 > 5	falsch
≥	ist größer oder gleich	6 ≥ 5	wahr

> **Eselsbrücke:** Die offene Seite von < und > zeigt immer zur größeren Zahl. Wie ein Krokodil, das die größere Portion fressen will.

> **= ist nicht „setze“!** Der Vergleich „zahl = 5“ **fragt** nur, ob `zahl` gerade 5 ist. Er ändert nichts. Willst du `zahl` auf 5 bringen, nimmst du den Block „setze zahl auf 5“.

Der Wetterbericht würfelt eine Temperatur zwischen −5 und 35 Grad und wählt aus drei Möglichkeiten. Das zweite „falls / sonst“ steckt im **sonst**-Teil des ersten. Es wird also nur gefragt, wenn es nicht friert.





Temperatur	Anzeige
unter 0	Frost!
0 bis 25	Angenehm
über 25	Badewetter!

7.3 und, oder, nicht – wahr und falsch

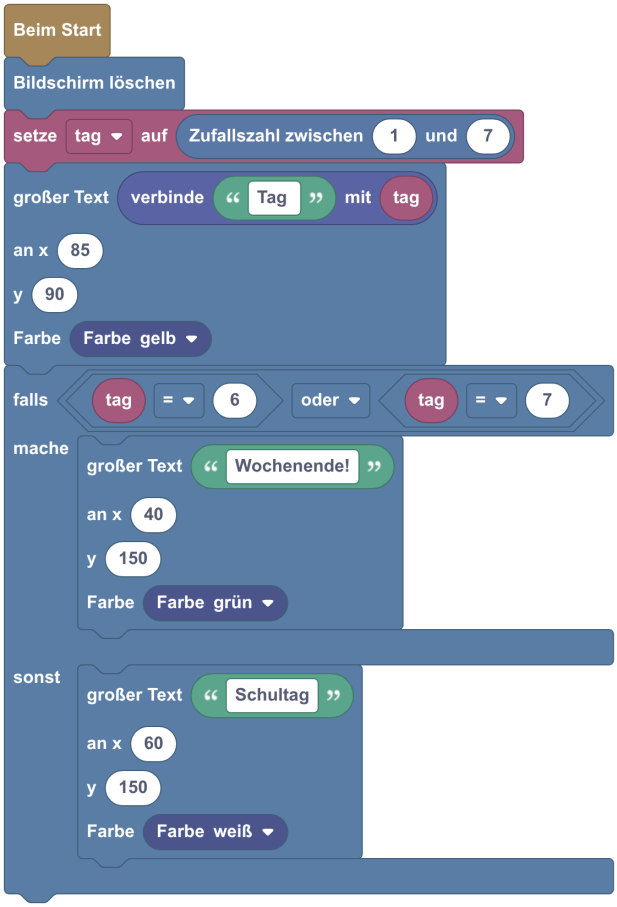
Manchmal hängt eine Entscheidung von zwei Fragen ab. Dafür gibt es in der Kategorie **Mathe** den Block „und / oder“. In beide Lücken steckst du eine Bedingung:



- **und:** Die Antwort ist nur wahr, wenn **beide** Bedingungen wahr sind. Der Block oben fragt: Liegt `zahl` zwischen 10 und 20?
- **oder:** Die Antwort ist wahr, wenn **mindestens eine** Bedingung wahr ist.

A	B	A und B	A oder B
wahr	wahr	wahr	wahr
wahr	falsch	falsch	wahr
falsch	wahr	falsch	wahr
falsch	falsch	falsch	falsch

Ein Tag der Woche ist eine Zahl von 1 (Montag) bis 7 (Sonntag). Wochenende ist am Tag 6 **oder** am Tag 7:



Der Block **„nicht“** dreht eine Antwort um: aus wahr wird falsch und aus falsch wird wahr. „nicht tag = 7“ heißt also: Es ist nicht Sonntag.



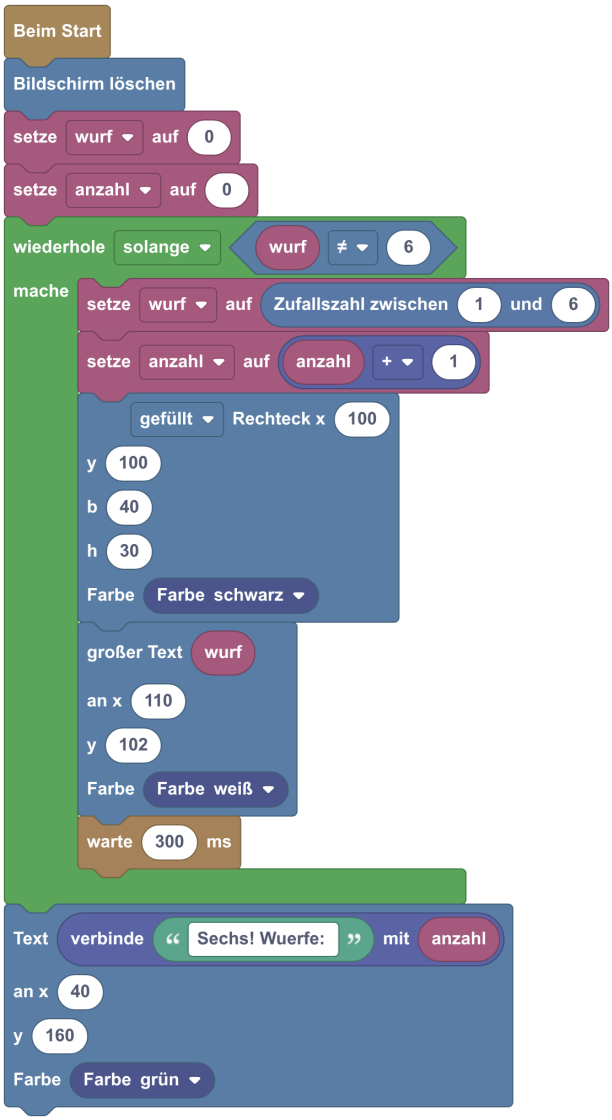
Der Block **„wahr / falsch“** ist eine feste Antwort. Im Feld ▾ wählst du wahr oder falsch. Du brauchst ihn selten – zum Beispiel, um eine Bedingung zum Ausprobieren immer wahr zu machen.



7.4 Die „solange“-Schleife

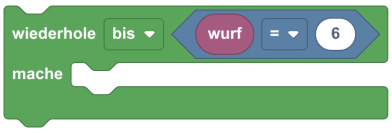
„wiederhole 10-mal“ läuft genau zehnmal. Oft weißt du aber vorher nicht, wie oft etwas wiederholt werden muss. Wie oft muss man würfeln, bis eine Sechs kommt? Mal einmal, mal zwanzigmal.

Dafür gibt es in der Kategorie **Kontrollstrukturen** den Block **„wiederhole solange … mache“**. Vor jedem Durchlauf prüft er die Bedingung. Ist sie wahr, läuft die Schleife noch einmal. Ist sie falsch, geht das Programm unter der Schleife weiter.



`wurf` beginnt bei 0. So ist die Bedingung „wurf ≠ 6“ am Anfang wahr, und die Schleife startet. In jedem Durchlauf wird neu gewürfelt und `anzahl` um 1 erhöht. Kommt eine 6, ist „wurf ≠ 6“ falsch, die Schleife endet, und das Ergebnis erscheint.

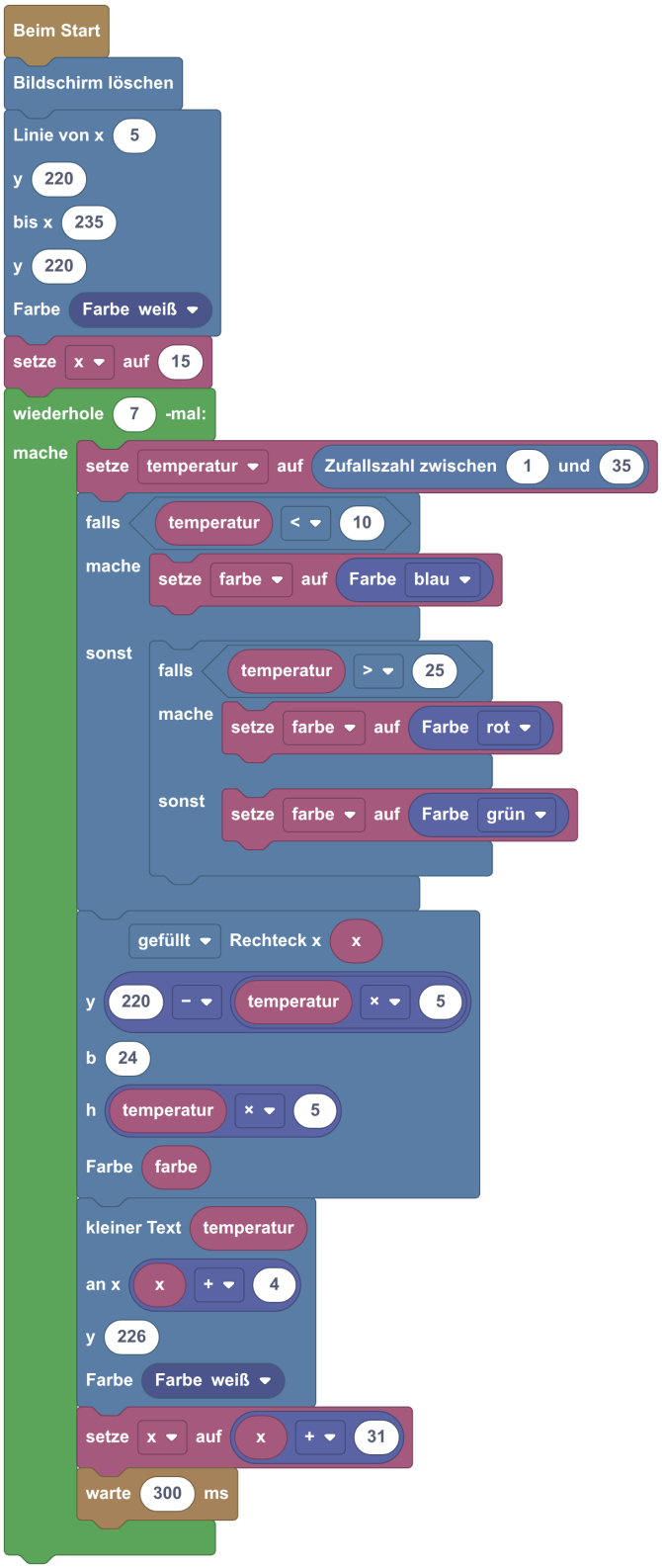
Im Feld ▾ des Blocks kannst du statt **solange** auch **bis** wählen. Dann läuft die Schleife, **bis** die Bedingung wahr wird. Diese Schleife tut dasselbe wie die oben:

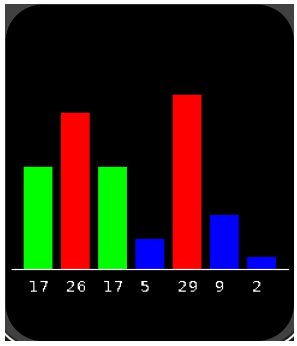


> **Endlosschleife:** Wird die Bedingung nie falsch, hört die Schleife nie auf. Vergisst du zum Beispiel, in der Schleife neu zu würfeln, bleibt `wurf` immer 0 – und die Schleife läuft ewig. Dann hilft nur ■ **Stop**. Prüfe deshalb: Ändert sich in der Schleife etwas, das in der Bedingung vorkommt?

Ausprobieren: Das Wetter der Woche

Der PBC+ erfindet für jeden Tag der Woche eine Temperatur und malt sie als Säule. Kalte Tage (unter 10 Grad) werden blau, heiße Tage (über 25 Grad) rot, alle anderen grün.





Neu ist hier: Die Farbe steckt in einer **Variable** `farbe`. Den Block „Farbe … “ kannst du nämlich auch in „setze … auf…“ stecken. Das „falls / sonst“ wählt nur die Farbe aus. Gemalt wird danach mit einem einzigen Rechteck-Block.

Die Säule ist `temperatur` × 5 Pixel hoch und endet an der weißen Linie bei y = 220. Darum beginnt sie bei y = 220 – `temperatur` × 5. Nach jeder Säule rückt `x` um 31 Pixel nach rechts.

Das Programm findest du als Datei `k07_wetterwoche.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Münzwurf mit Ton** Erweitere den Münzwurf aus 7.1: Bei „Kopf“ soll ein hoher Ton erklingen, bei „Zahl“ ein tiefer.

Aufgabe 2 ⭐ **Wahr oder falsch?** Was antworten diese Bedingungen? Überlege erst, dann prüfe mit einem „falls / sonst“, das „wahr“ oder „falsch“ anzeigt. a) 7 > 3 b) 5 = 6 c) 4 < 4
d) nicht (2 > 5)
e) 3 < 5 und 8 < 6 f) 3 < 5 oder 8 < 6

Aufgabe 3 ⭐⭐ **Gerade oder ungerade?** Würfle eine Zahl zwischen 1 und 100. Zeige sie an und darunter „gerade“ oder „ungerade“. *Tipp:* Erinnere dich an den Rest-Block aus Kapitel 6.

Aufgabe 4 ⭐⭐ **Punkte und Noten** Würfle eine Punktzahl von 0 bis 100. Ab 90 Punkten zeigt der PBC+ „sehr gut“, ab 75 „gut“, ab 50 „bestanden“, sonst „weiter ueben“.

Aufgabe 5 ⭐⭐⭐ **Pasch!** Würfle mit zwei Würfeln `w1` und `w2`, bis beide dieselbe Zahl zeigen (ein „Pasch“). Zähle die Würfe mit. *Tipp:* Welche Startwerte brauchen `w1` und `w2`, damit die Schleife überhaupt startet?

Aufgabe 6 ⭐⭐⭐ **Hitzewelle** Erweitere das Wetter der Woche: Zähle in einer Variable `heiss` die roten Tage und zeige am Ende zum Beispiel „3 heisse Tage“ an.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Eine **Bedingung** ist eine Frage mit der Antwort **wahr** oder **falsch**.
- **falls:** Nur wenn die Bedingung wahr ist, laufen die Blöcke bei „mache“. **falls / sonst:** sonst laufen die Blöcke bei „sonst“.
- Mehr als zwei Möglichkeiten: ein „falls / sonst“ in das „sonst“ stecken.
- Vergleiche: = ≠ < ≤ > ≥. Der Vergleich „=“ fragt nur, „setze“ ändert.
- **und:** beide wahr. **oder:** mindestens eins wahr. **nicht:** dreht um.
- **wiederhole solange** läuft, solange die Bedingung wahr ist; **bis** läuft, bis sie wahr wird. In der Schleife muss sich etwas ändern, sonst läuft sie ewig.

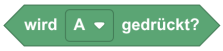
KAPITEL 8

Eingabe: Tasten und Joystick

Worum geht's? Jetzt bestimmst du mit, was passiert! Der PBC+ hat einen Joystick und die Tasten A und B. In diesem Kapitel fragt dein Programm ab, welche Taste gedrückt ist, wartet auf einen Tastendruck und lässt dich einen Punkt über den Bildschirm steuern.

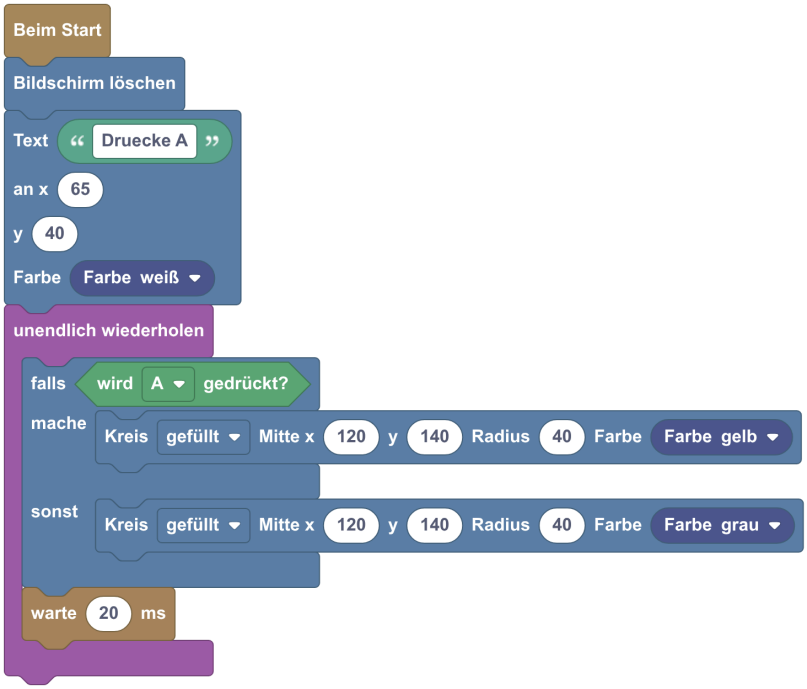
Alle Blöcke dieses Kapitels findest du in der Kategorie **Eingabe**. Im Feld ▾ wählst du die Taste: **HOCH**, **RUNTER**, **LINKS**, **RECHTS** (den Joystick in diese Richtung drücken), **MITTE** (den Joystick nach unten drücken), **A** oder **B**. Im Emulator nimmst du die Pfeiltasten, die Leertaste für MITTE und die Tasten A und B der Tastatur.

8.1 „wird … gedrückt?“ – solange die Taste unten ist



Dieser Frage-Block antwortet **wahr**, solange die Taste in diesem Moment heruntergedrückt ist, und sonst **falsch**. Du steckst ihn in ein „falls“.

Das Programm fragt nur in dem Augenblick, in dem es an dem Block vorbeikommt. Damit es die Taste immer wieder prüft, gehört die Frage in eine Schleife. Dieser Leuchtknopf ist gelb, solange du A drückst, und sonst grau:

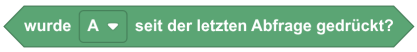


Etwa 50-mal pro Sekunde fragt die Schleife: Wird A gedrückt? Je nach Antwort malt sie den Kreis gelb oder grau.

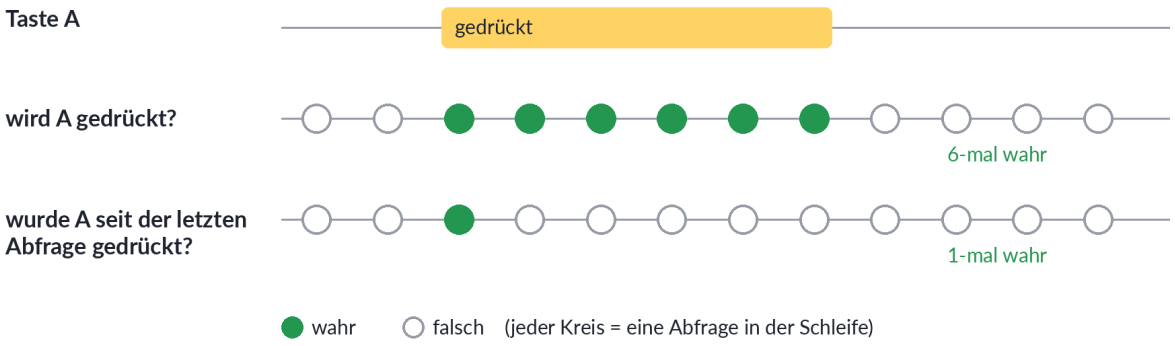
Das „warte 20 ms“ am Ende gibt dem PBC+ eine kleine Pause. Ohne Pause würde die Schleife so schnell rasen wie möglich. PBC+ Studio warnt dich, wenn in einer Endlosschleife kein „warte“ steht.

8.2 „wurde … gedrückt?“ – genau einmal pro Druck

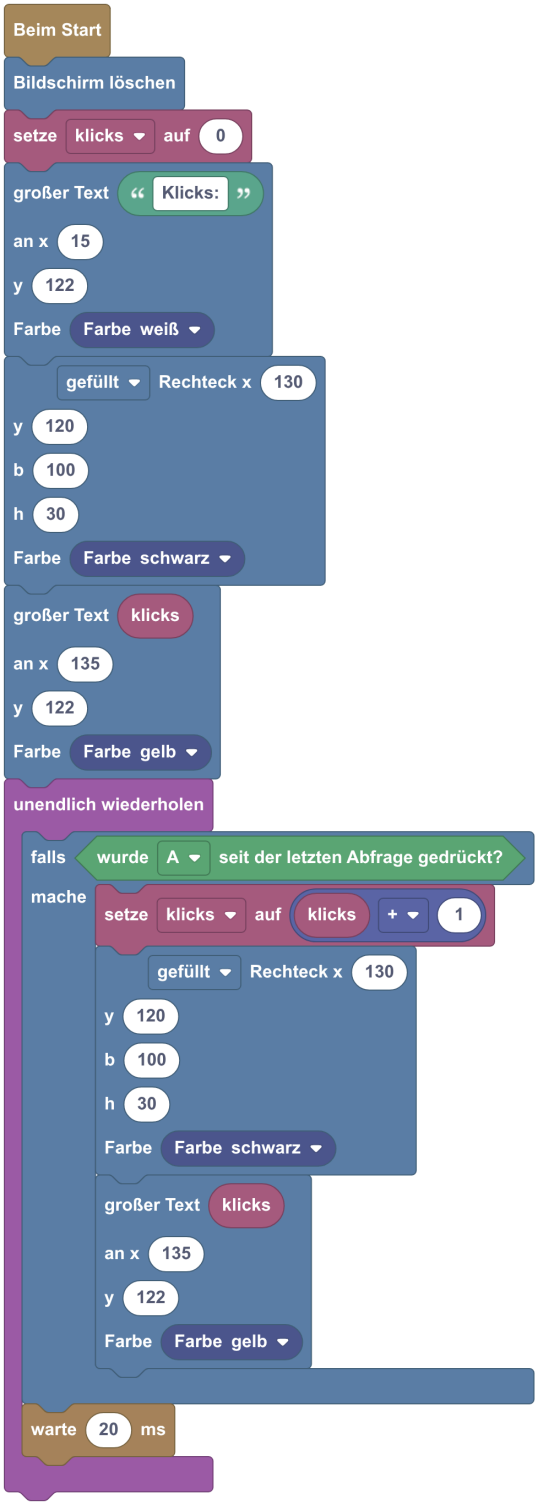
Jetzt soll ein Zähler mitzählen, wie oft du A drückst. Mit „wird A gedrückt?“ klappt das nicht: Selbst ein kurzer Druck dauert eine Zehntelsekunde. In dieser Zeit fragt die Schleife mehrmals – und zählt jedes Mal mit. Für so etwas gibt es einen zweiten Block:



Er antwortet nur **ein einziges Mal** mit wahr, wenn die Taste neu gedrückt wurde. Danach antwortet er falsch, bis du die Taste loslässt und noch einmal drückst. Das Bild zeigt den Unterschied:



So sieht der Klick-Zähler aus. Die Zahl wird nur neu gemalt, wenn sich etwas geändert hat:





3-mal A gedrückt



mit „wird“: einmal eine halbe Sekunde lang

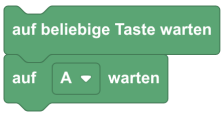
Probiere es selbst: In der Datei `k08_zaehler_wird.pbcp` steht „wird A gedrückt?“ statt „wurde …“. Halte A kurz fest – der Zähler springt um viele Zahlen weiter.

> **Warum „seit der letzten Abfrage“?** Der PBC+ merkt sich einen Tastendruck, bis das Programm danach fragt. Drückst du A, während das Programm gerade eine Sekunde wartet, geht der Druck nicht verloren: Die nächste Frage „wurde A … gedrückt?“ antwortet wahr. Jede Taste hat dabei ihr eigenes Gedächtnis.

Du willst …	dann nimm
etwas tun, **solange** die Taste gedrückt ist (laufen, Dauerton)	„wird … gedrückt?“
etwas **einmal pro Druck** tun (zählen, springen, Farbe wechseln)	„wurde … seit der letzten Abfrage gedrückt?“

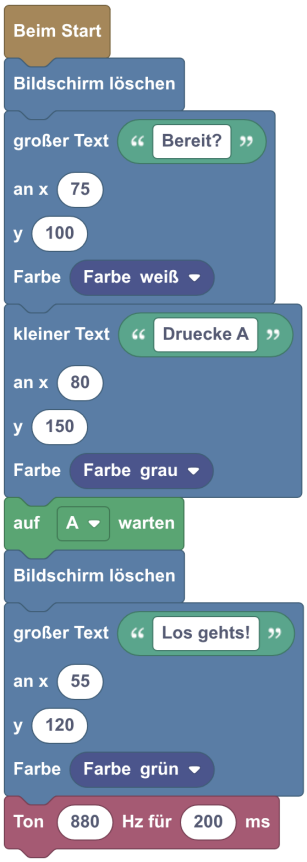
8.3 Auf eine Taste warten

Manchmal soll das Programm einfach anhalten, bis du bereit bist. Dafür gibt es zwei Blöcke:



- **„auf beliebige Taste warten“** wartet, bis irgendeine Taste gedrückt und wieder losgelassen wurde.
- **„auf A warten“** wartet auf eine bestimmte Taste. Im Feld ▾ wählst du welche. Auch hier geht es erst weiter, wenn du die Taste wieder loslässt.

Solange das Programm wartet, passiert sonst nichts. Für einen Startbildschirm ist das genau richtig:

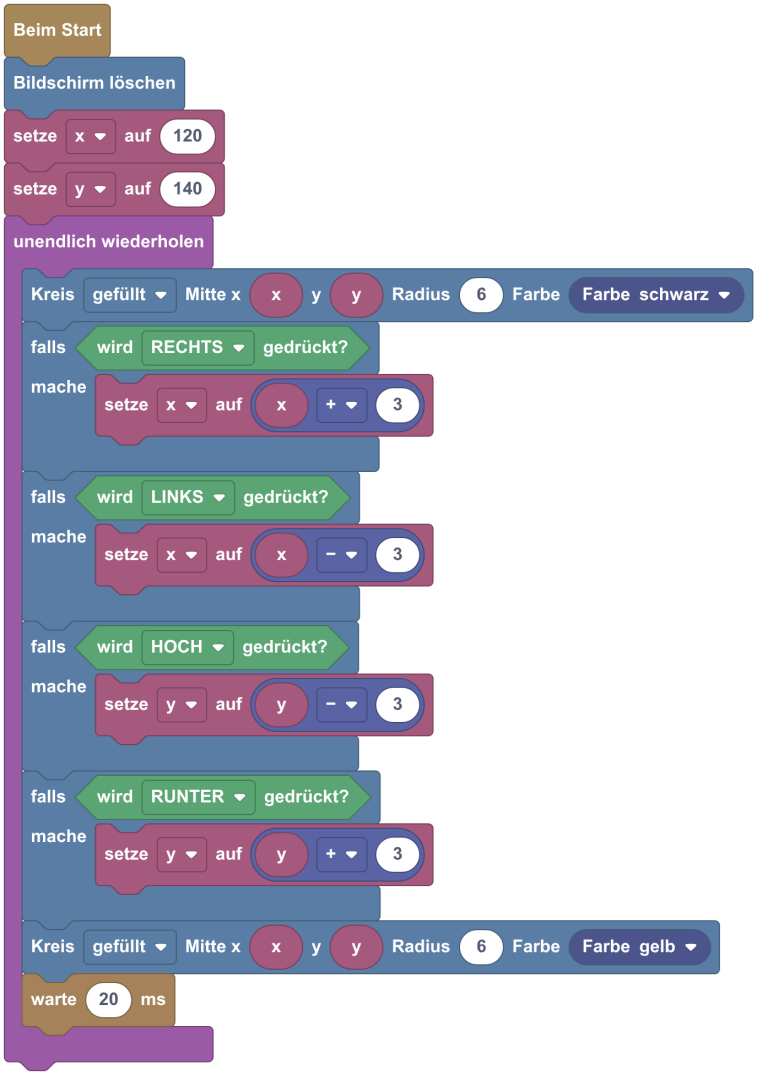


Das Programm zeigt „Bereit?“ und bleibt bei „auf A warten“ stehen. Erst wenn du A drückst und loslässt, geht es weiter.

In einem Spiel brauchst du dagegen „wird … gedrückt?“ oder „wurde … gedrückt?“: Dort soll sich ja alles weiterbewegen, auch wenn du gerade keine Taste drückst.

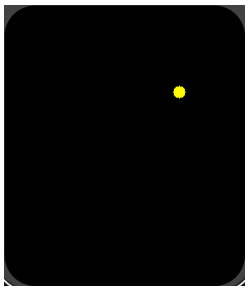
8.4 Einen Punkt mit dem Joystick steuern

Ein gelber Punkt soll sich mit dem Joystick bewegen. Seine Position merkst du dir in den Variablen `x` und `y`. Drückst du nach rechts, wird `x` um 3 größer; drückst du nach oben, wird `y` um 3 kleiner (denn y zählt nach unten).

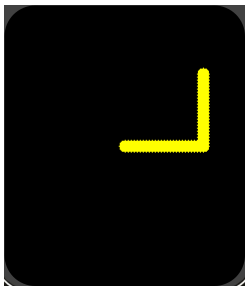


Jede Runde der Schleife hat drei Schritte:

1. Den Punkt an der **alten** Stelle schwarz übermalen.
2. `x` und `y` je nach Joystick ändern.
3. Den Punkt an der **neuen** Stelle gelb malen.



mit Übermalen



ohne Übermalen

Warum Schritt 1? Lässt du ihn weg, bleibt jeder alte Punkt stehen, und der Punkt zieht eine dicke Spur hinter sich her (Bild rechts).

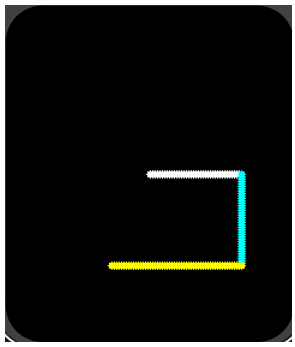
Man könnte auch in jeder Runde den ganzen Bildschirm löschen. Dann flackert es aber stark. Das Löschen des ganzen Bildschirms dauert nämlich so lange, dass du kurz ein schwarzes Bild siehst – auf dem Gerät genauso wie im Emulator. Stell dir ein Daumenkino vor: Statt für jedes Bild ein neues Blatt zu nehmen, radierst du nur den Punkt weg und malst ihn ein Stück weiter wieder hin. Das geht viel schneller.

> **Die Regel für flimmerfreie Bewegung:** Nie den ganzen Bildschirm löschen. Erst die alte Stelle in der Hintergrundfarbe übermalen, dann **sofort** an der neuen Stelle zeichnen.

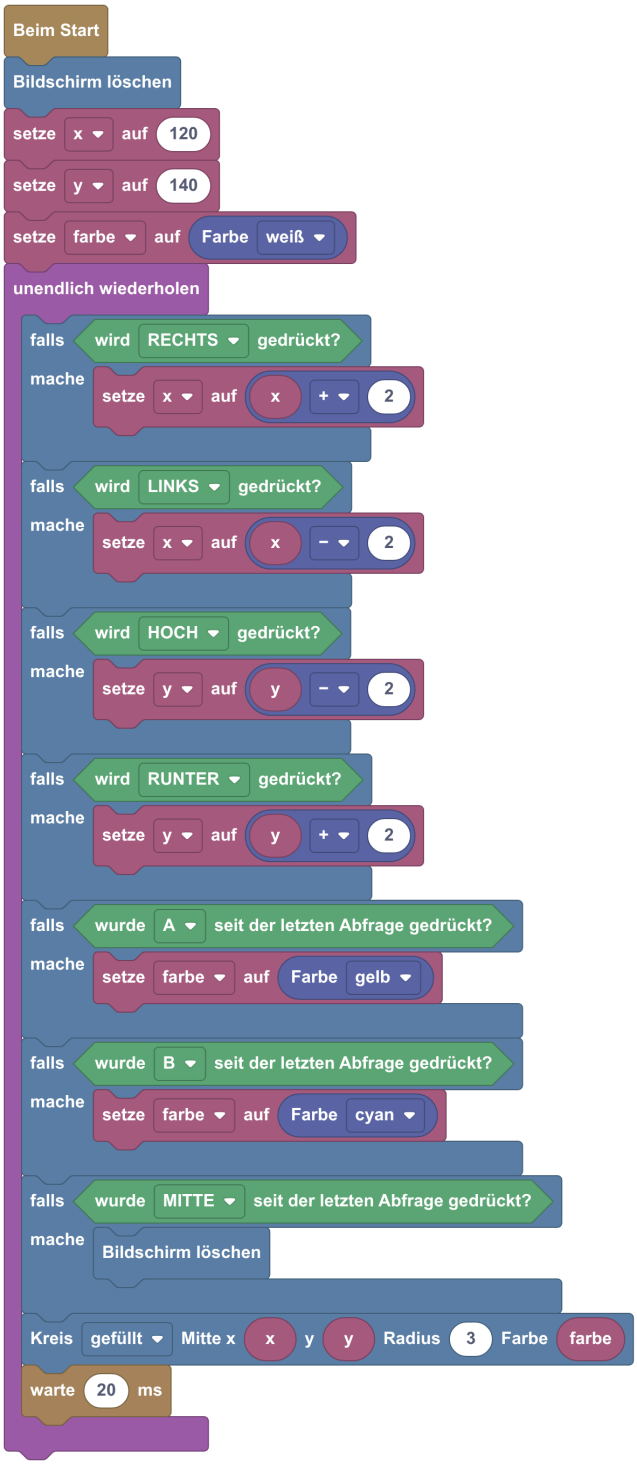
Ausprobieren: Das Zeichenbrett

Hier ist die Spur gewollt: Der Punkt wird nie übermalt, also zeichnet er wie ein Stift. Mit dem Joystick malst du, mit **A** wird der Stift gelb, mit **B** türkis. **MITTE** (Leertaste) löscht das Bild.

Die Stiftfarbe steckt in der Variable `farbe`. Für A, B und MITTE nimmt das Programm „wurde … gedrückt?“, denn die Farbe soll pro Druck genau einmal wechseln. Für den Joystick nimmt es „wird … gedrückt?“: Der Stift soll fahren, solange du drückst.



Weiß, türkis, gelb



Das Programm findest du als Datei `k08_zeichenbrett.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Zwei Leuchtknöpfe** Erweitere den Leuchtknopf aus 8.1: Drückst du B, wird der Kreis rot. *Tipp:* Stecke ein zweites „falls / sonst“ in den sonst-Teil.

Aufgabe 2 ⭐ **Hoch und runter zählen** Erweitere den Klick-Zähler aus 8.2: Mit B zählt er um 1 rückwärts.

Aufgabe 3 ⭐⭐ **Nicht aus dem Bild** Im Programm aus 8.4 kann der Punkt aus dem Bildschirm laufen. Sorge dafür, dass er am Rand stehen bleibt. *Tipp:* Bewege ihn nach rechts nur, wenn RECHTS gedrückt wird **und** `x` noch kleiner als 230 ist.

Aufgabe 4 ⭐⭐ **Turbo** Solange du B gedrückt hältst, soll der Punkt doppelt so schnell fahren. *Tipp:* Lege eine Variable `schritt` an und nimm sie statt der 3.

Aufgabe 5 ⭐⭐⭐ **Tauziehen für zwei** Zwei Spieler ziehen an einem Seil – einer drückt A, einer drückt B. Ein gelber Kreis startet in der Mitte. Jeder Druck auf A zieht ihn 10 Pixel nach links, jeder Druck auf B 10 Pixel nach rechts. Erreicht er den linken oder rechten Rand, hat der Spieler auf dieser Seite gewonnen. *Tipp:* Nimm „wiederhole solange“ mit einer „und“-Bedingung.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Tasten: HOCH, RUNTER, LINKS, RECHTS, MITTE, A, B (Emulator: Pfeiltasten, Leertaste, A, B).
- **„wird … gedrückt?“** ist wahr, solange die Taste unten ist – gut zum Laufen und Steuern.
- **„wurde … seit der letzten Abfrage gedrückt?“** ist einmal pro Druck wahr – gut zum Zählen und Umschalten.
- Tastenfragen gehören in eine Schleife, sonst wird nur einmal gefragt.
- **„auf … warten“** hält das Programm an, bis die Taste gedrückt und wieder losgelassen ist.
- Bewegung ohne Flackern: alte Stelle übermalen, dann sofort neu zeichnen – nie den ganzen Bildschirm löschen.

KAPITEL 9

Ausgabe: LEDs und Töne

Worum geht's? Der PBC+ kann nicht nur malen. Er hat drei kleine Lämpchen, eine bunte RGB-LED und einen Lautsprecher. In diesem Kapitel lässt du es blinken und leuchten, baust eine Sirene und spielst ein Lied.

9.1 Die drei Status-LEDs

Links neben dem Bildschirm sitzen drei LEDs: rot, gelb und grün. LED ist die Abkürzung für „Leuchtdiode", ein kleines, sparsames Lämpchen. Die Blöcke dafür findest du in der Kategorie **LEDs**:



Im ersten Feld ▾ wählst du **rot**, **gelb** oder **grün**, im zweiten **an** oder **aus**. Mit „alle Status-LEDs aus" machst du alle drei auf einmal aus.



Eine LED bleibt so, wie du sie zuletzt eingestellt hast – auch wenn das Programm schon zu Ende ist. Zum Blinken brauchst du also immer zwei Blöcke: an und aus, mit einem „warte" dazwischen. Dieses Lauflicht lässt die drei LEDs nacheinander aufleuchten, immer wieder:



Jede LED geht an, leuchtet 200 ms lang und geht wieder aus. Dann ist die nächste dran. Im Emulator siehst du die LEDs links neben dem Bildschirm.



Alle LEDs an: links rot, gelb und grün, unten die RGB-LED im Plus-Zeichen (hier lila)

9.2 Die RGB-LED: jede Farbe möglich

Die **RGB-LED** sitzt im Plus-Zeichen des PBC±Schriftzugs. Sie enthält drei winzige Lämpchen: **R**ot, **G**rün und **B**lau. Für jedes stellst du ein, wie hell es leuchtet – von 0 (aus) bis 255 (ganz hell). Aus den drei Farben mischt die LED jede andere Farbe.



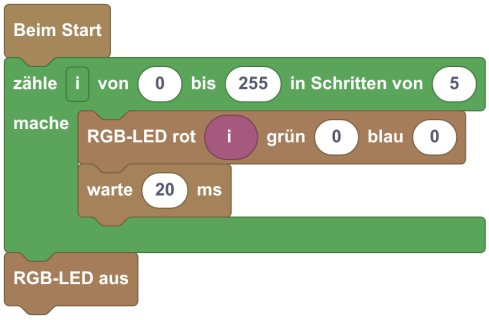
rot	grün	blau	Farbe
255	0	0	rot
0	255	0	grün
0	0	255	blau
255	255	0	gelb
255	255	255	weiß
60	0	0	schwaches Rot

Das Mischen funktioniert anders als beim Malkasten: Rotes und grünes **Licht** zusammen ergeben **Gelb**. Alle drei ganz hell ergeben Weiß.

Bequemer ist manchmal „RGB-LED auf Farbe“. Hier steckst du einen Farb-Block hinein, zum Beispiel „Farbe nach Namen“ oder den Farbwähler. „RGB-LED aus“ schaltet sie wieder aus.



Mit einer Zählschleife (Kapitel 5) wird die LED langsam heller: `i` zählt in Fünferschritten von 0 bis 255, und der Rot-Wert wächst mit.



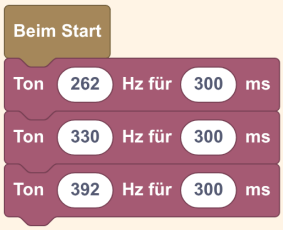
9.3 Töne und Dauertöne

Der Lautsprecher spielt Töne. Wie hoch ein Ton ist, gibst du in **Hertz** (Hz) an. Das ist die Zahl der Schwingungen pro Sekunde. Je größer die Zahl, desto höher der Ton: 200 Hz brummt tief, 2000 Hz piepst hoch. Die Blöcke stehen in der Kategorie **Sound**.



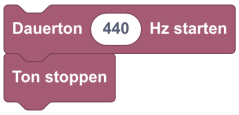
„Ton … Hz für … ms“ spielt einen Ton für eine bestimmte Zeit. Aber Vorsicht:

> **Der Ton-Block wartet nicht!** Er startet den Ton, und das Programm läuft sofort weiter, während der Ton noch klingt. Stehen drei Ton-Blöcke direkt untereinander, hörst du deshalb nur den letzten. Die ersten beiden werden sofort vom nächsten Ton abgelöst.

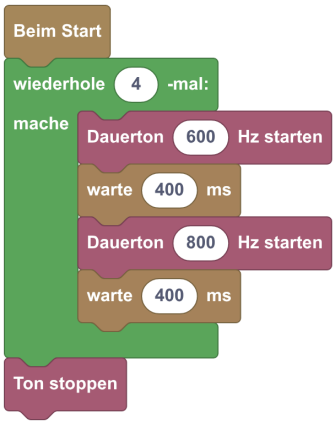


> Setze hinter jeden Ton ein „warte“, das so lang ist wie der Ton.

Für Töne, die so lange klingen sollen, bis etwas passiert, gibt es zwei weitere Blöcke:



„Dauerton … Hz starten“ lässt den Ton immer weiter klingen. Ein neuer Dauerton ersetzt den alten. Erst „Ton stoppen“ macht wieder Ruhe – vergiss ihn nicht! Eine Sirene wechselt zwischen zwei Tönen hin und her:



Die Schleife läuft viermal. In jedem Durchlauf klingen 600 Hz und 800 Hz je 400 ms lang. Danach stoppt der Ton.

Im Emulator hörst du die Töne über die Lautsprecher deines Computers. Ist nichts zu hören, prüfe die Lautstärke.

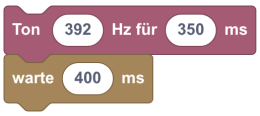
9.4 Eine Melodie programmieren

Jeder Ton der Tonleiter hat seine Frequenz. Diese Tabelle brauchst du für Lieder:

Ton	c’	d’	e’	f’	g’	a’	h’	c’’
Hz	262	294	330	349	392	440	494	523

Das c’’ am Ende ist genau doppelt so hoch wie das c’ am Anfang: Seine Zahl ist fast genau doppelt so groß.

Eine Note besteht aus zwei Blöcken: dem Ton und einem „warte“. Den Ton machst du 50 ms **kürzer** als das Warten. So entsteht zwischen zwei Noten eine winzige Pause, und zwei gleiche Noten hintereinander klingen nicht wie eine lange.



Für die Länge der Noten legst du fest: Eine **Viertelnote** dauert 400 ms, eine **halbe Note** 800 ms. Das Lied „Hänschen klein“ beginnt so:

Silbe	Häns-	chen	klein	ging	al-	lein
Ton	g’	e’	e’	f’	d’	d’
Hz	392	330	330	349	294	294
Länge (ms)	400	400	800	400	400	800



Das sind die ersten sechs Noten. Weiter geht es mit „in die wei-te Welt hin-ein“: c’ d’ e’ f’ g’ g’ (je 400 ms) und noch einmal g’ (800 ms).

Die ganze erste Zeile steht in der Datei `k09_haenschen.pbcp`.

> **Tipp:** Baue erst eine Note (Ton + warte). Dann klicke mit der rechten Maustaste auf den Ton-Block und wähle **Kopieren** – das geht schneller, als jeden Block neu zu holen.

Ausprobieren: Das Mini-Klavier

Jetzt wird der PBC+ zum Musikinstrument. Jede Taste spielt einen Ton, solange du sie drückst, und die RGB-LED leuchtet dazu in einer passenden Farbe:



Taste	LINKS	RUNTER	RECHTS	HOCH	A	B
Ton	c’	d’	e’	f’	g’	a’

Am Anfang jeder Runde setzt das Programm `note` auf 0. Die sechs „falls“-Blöcke prüfen die Tasten. Ist eine gedrückt, bekommt `note` deren Frequenz. Ist danach `note` größer als 0, klingt ein Dauerton, sonst ist Ruhe. Für den Grün-Wert der RGB-LED rechnet das Programm `note` − 250: tiefe Töne leuchten blau, hohe eher türkis.



Das Programm findest du als Datei `k09_klavier.pbcp` im Ordner `buch/programme`. Kannst du „Alle meine Entchen“ darauf spielen?

Übungsaufgaben

Aufgabe 1 ⭐ **Lauflicht rückwärts** Ändere das Lauflicht aus 9.1: Es soll von Grün über Gelb nach Rot laufen und doppelt so schnell sein.

Aufgabe 2 ⭐ **Mischfarben** Lass die RGB-LED nacheinander je eine Sekunde lang gelb, türkis, lila und weiß leuchten. Welche Zahlen brauchst du für Türkis (Grün + Blau) und Lila (Rot + Blau)?

Aufgabe 3 ⭐⭐ **Hänschen klein, zweite Zeile** Die zweite Zeile heißt „Stock und Hut steht ihm gut, ist gar wohlgemut“. Die Töne: g’ e’ e’ (halbe) · f’ d’ d’ (halbe) · c’ e’ g’ g’ · c’ (lang, 1200 ms). Hänge sie an die erste Zeile an.

Aufgabe 4 ⭐⭐ **Alarmanlage** Eine Alarmanlage heult abwechselnd mit 1000 Hz und 700 Hz, und die rote LED blinkt dazu – bis jemand A drückt. Dann ist Ruhe, die LEDs gehen aus, und „Alarm aus“ erscheint. *Tipp:* „wiederhole solange nicht wurde A … gedrückt?“

Aufgabe 5 ⭐⭐ **Heulende Sirene** Lass einen Dauerton mit einer Zählschleife langsam von 400 Hz auf 1200 Hz steigen und wieder fallen. Wiederhole das dreimal.

Aufgabe 6 ⭐⭐⭐ **Das Tempo in einer Variable** Lege eine Variable `takt` an (zum Beispiel 300) und baue die ersten sechs Noten von „Hänschen klein“ so, dass alle Längen aus `takt` berechnet werden. Wenn du `takt` änderst, wird das ganze Lied schneller oder langsamer.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Die drei **Status-LEDs** (rot, gelb, grün) schaltest du mit „setze … LED an/aus“. Sie bleiben so, bis du sie umschaltest.
- Die **RGB-LED** mischt jede Farbe aus Rot, Grün und Blau (je 0 bis 255).
- Tonhöhe in **Hertz**: je größer die Zahl, desto höher der Ton.
- „Ton … für … ms“ **wartet nicht** – setze ein „warte“ dahinter.
- „Dauerton“ klingt, bis „Ton stoppen“ kommt.
- Melodie: pro Note ein Ton (etwas kürzer) und ein „warte“. c’ = 262 Hz, a’ = 440 Hz, c’’ = 523 Hz.

KAPITEL 10

Turtle-Grafik

Worum geht's? Eine kleine Schildkröte mit einem Stift läuft über den Bildschirm und malt dabei Linien. Du sagst ihr, wie weit sie laufen und wie weit sie sich drehen soll. Mit wenigen Blöcken und einer Schleife entstehen so Vielecke, Sterne und bunte Muster. Zum Schluss lernst du, wie die Turtle Farben „sieht".

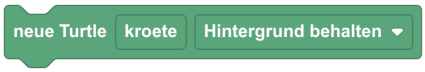
10.1 Eine Schildkröte mit Stift

Turtle ist das englische Wort für Schildkröte. Die Turtle ist ein kleiner Pfeil auf dem Bildschirm. Die Spitze zeigt, wohin sie blickt. Wenn sie läuft, zieht sie mit ihrem Stift eine Linie hinter sich her.

Eine Turtle erstellen

Bevor die Turtle etwas tun kann, musst du sie erstellen:

1. Öffne in der Blockleiste die Kategorie **Turtle**.
2. Klicke auf **„neue Turtle erstellen"**. PBC+ Studio fragt nach einem Namen. Gib zum Beispiel `kroete` ein (ohne Umlaut!).
3. Hänge den neuen Block direkt unter „Beim Start".



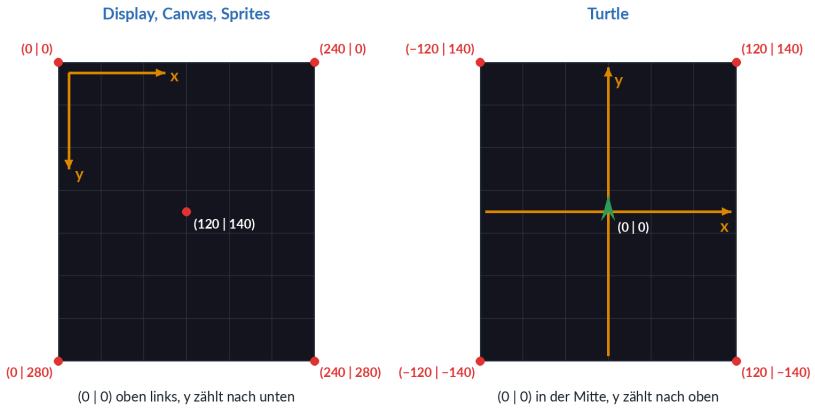
In der Kategorie **Turtle** findest du nun alle Befehle. Jeder beginnt mit dem Namen der Turtle: „kroete vorwärts 50 Pixel".

> „neue Turtle" muss **über** allen anderen Turtle-Blöcken stehen. Fehlt der Block, meldet PBC+ Studio „Turtle fehlt".

Die Turtle rechnet anders

Du kennst das Koordinatensystem des Bildschirms: (0 | 0) ist oben links, und y zählt nach unten. Die Turtle hat ihr **eigenes** Koordinatensystem, so wie im Mathe-Unterricht:

- (0 | 0) ist die **Mitte** des Bildschirms.
- x zählt nach rechts, **y zählt nach oben**.
- Am Anfang steht die Turtle in der Mitte und **blickt nach oben**.



(0 | 0) oben links, y zählt nach unten

(0 | 0) in der Mitte, y zählt nach oben

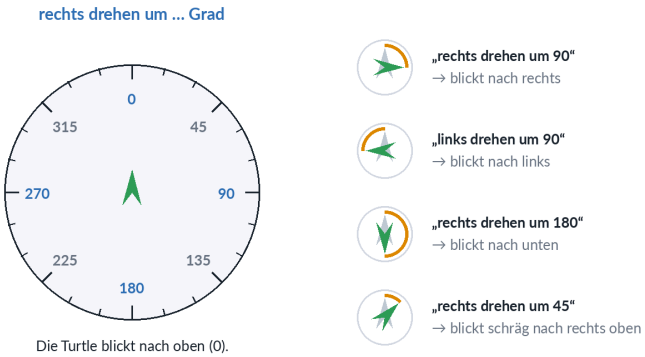
Der Bildschirm reicht für die Turtle von x = −120 (linker Rand) bis x = 120 (rechter Rand) und von y = −140 (unten) bis y = 140 (oben). Läuft die Turtle darüber hinaus, ist die Linie einfach nicht mehr zu sehen. Probier es aus: Hänge „kroete vorwärts 80 Pixel“ unter „neue Turtle“ und teste. Die Turtle läuft von der Mitte aus nach oben.

10.2 Vorwärts, rückwärts, drehen

Mit vier Blöcken bewegst du die Turtle:

Block	Was tut die Turtle?
kroete vorwärts 50 Pixel	läuft 50 Pixel in Blickrichtung
kroete rückwärts 50 Pixel	läuft 50 Pixel rückwärts, blickt aber weiter nach vorn
kroete rechts drehen um 90 Grad	dreht sich auf der Stelle nach rechts (im Uhrzeigersinn)
kroete links drehen um 90 Grad	dreht sich auf der Stelle nach links

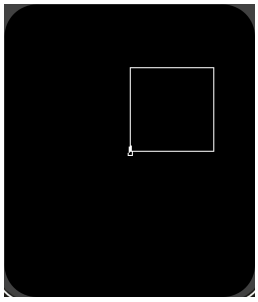
Beim Drehen malt die Turtle nichts. Sie ändert nur ihre Blickrichtung. Wie weit sie sich dreht, gibst du in **Grad** an. Eine ganze Umdrehung sind 360 Grad, eine Vierteldrehung 90 Grad.



Die Turtle blickt nach oben (0).

Läuft die Turtle 80 Pixel, dreht sich um 90 Grad nach rechts und macht das viermal, entsteht ein Quadrat:





Das Quadrat im Emulator

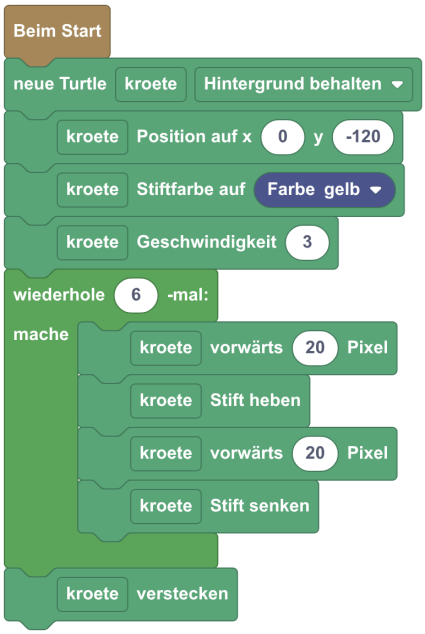
Tipp: Spiel selbst Turtle! Geh ein paar Schritte, mach eine Vierteldrehung nach rechts … So findest du schnell die richtigen Befehle.

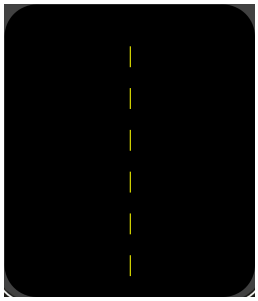
10.3 Stift, Farbe und Geschwindigkeit

Diese Blöcke ändern, **wie** die Turtle malt:

Block	Bedeutung
kroete Stift heben	Die Turtle hebt den Stift. Sie läuft weiter, malt aber nicht.
kroete Stift senken	Der Stift ist wieder unten: Die Turtle malt.
kroete Stiftfarbe auf Farbe gelb	neue Farbe für den Stift (am Anfang: weiß)
kroete Geschwindigkeit 5	1 = langsam, 10 = schnell, **0 = sofort fertig**. Am Anfang ist es 5.
kroete Position auf x 0 y -120	Die Turtle springt an diese Stelle – ohne zu malen.
kroete zurück zum Ursprung	Die Turtle springt zurück in die Mitte (0 | 0).
kroete verstecken / kroete zeigen	Der Pfeil verschwindet oder ist wieder zu sehen. Die Linien bleiben.
kroete Leinwand löschen	Alles Gemalte wird gelöscht, der Bildschirm wird schwarz.

Ein Beispiel: Die Turtle malt den gestrichelten Mittelstreifen einer Straße. Sie springt zuerst nach unten (y = −120). Dann malt sie 20 Pixel, hebt den Stift, läuft 20 Pixel ohne zu malen und senkt den Stift wieder – sechsmal.





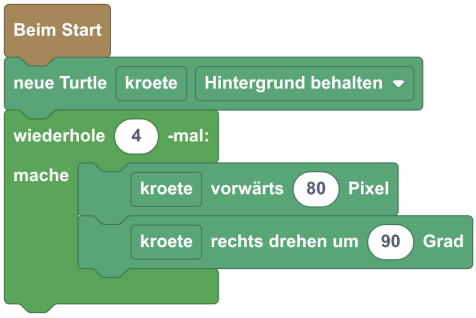
Der Mittelstreifen

Auf dem echten PBC+ ist die Turtle bei Geschwindigkeit 1 oder 2 sehr langsam. Im Emulator ist der Unterschied kleiner: Er zeichnet jede lange Linie in höchstens etwa zwölf Schritten.

> **„Richtung auf … Grad“ ist etwas anderes als „drehen“:** Dieser Block dreht die Turtle nicht ein Stück weiter, sondern stellt ihre Blickrichtung fest ein – wie bei einem Kompass: 0 = oben, 90 = rechts, 180 = unten, 270 = links. Egal, wohin die Turtle vorher geschaut hat: Nach „Richtung auf 90 Grad“ schaut sie nach rechts.

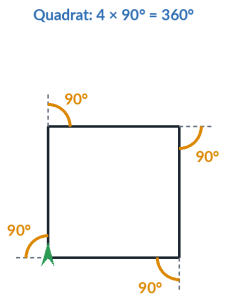
10.4 Vielecke und Muster mit Schleifen

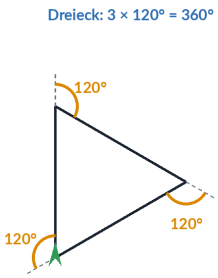
Im Quadrat-Programm stehen dieselben zwei Blöcke viermal hintereinander. Das geht kürzer mit der Schleife „wiederhole … -mal“ aus Kapitel 5:



Warum dreht sich die Turtle eigentlich um 90 Grad? Wenn sie einmal ganz herum gelaufen ist, blickt sie wieder in die Richtung, in der sie gestartet ist. Zusammen hat sie sich also **einmal ganz gedreht: 360 Grad**. Beim Quadrat verteilen sich die 360 Grad auf 4 Ecken, beim Dreieck auf 3 Ecken:

Daraus folgt eine einfache Regel: **Drehwinkel = 360 ÷ Anzahl der Ecken.**

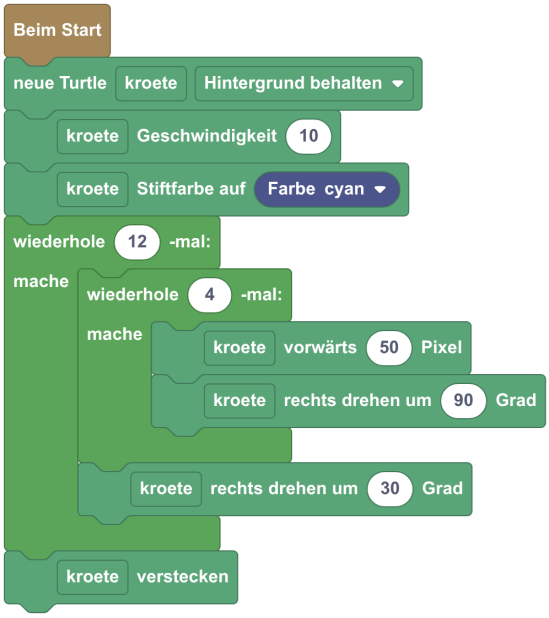


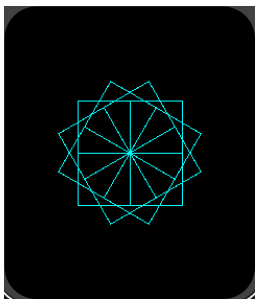


Vieleck	Ecken	Drehwinkel
Dreieck	3	120°
Quadrat	4	90°
Fünfeck	5	72°
Sechseck	6	60°
Achteck	8	45°

Muster: Schleife in der Schleife

Zeichnest du ein Quadrat, drehst die Turtle ein Stück und zeichnest das nächste Quadrat, entsteht ein Muster. Hier sind es 12 Quadrate. Nach jedem Quadrat dreht sich die Turtle um 30 Grad – 12 × 30° = 360°, also einmal herum.





12 Quadrate, jedes um 30° gedreht

Die innere Schleife malt ein Quadrat. Die äußere Schleife wiederholt das Ganze zwölfmal.

10.5 „Turtle berührt Farbe“ – die Turtle sieht nach vorn

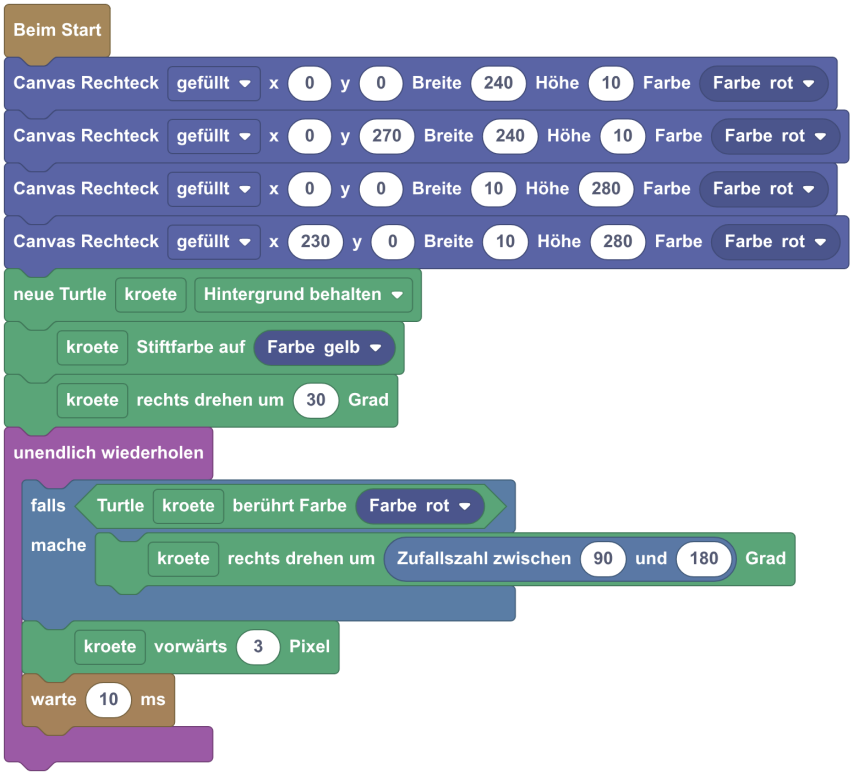
Die Turtle kann Farben erkennen. Dafür gibt es einen Frage-Block mit spitzen Enden. Er antwortet mit **wahr** oder **falsch**:

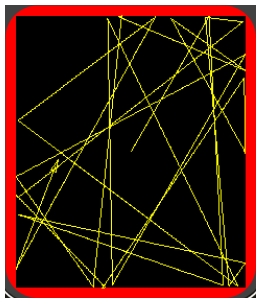


Er prüft **den Punkt 3 Pixel vor der Spitze der Turtle**. Ist dieser Punkt genau rot, ist die Antwort wahr. Die Turtle schaut also nach vorn – wie jemand, der beim Gehen kurz vor die Füße sieht.

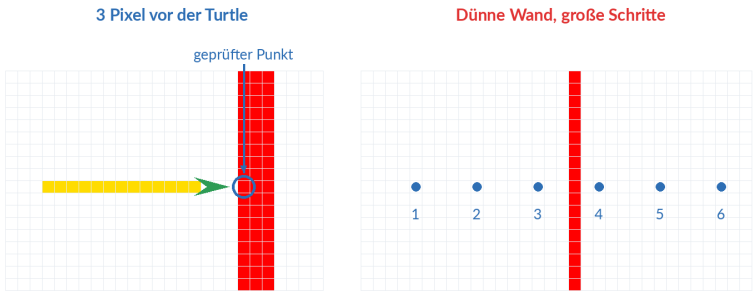
Der Staubsauger-Roboter

Ein Staubsauger-Roboter fährt geradeaus, bis er an eine Wand stößt. Dann dreht er sich und fährt weiter. Das kann die Turtle auch. Zuerst malst du vier rote Wände an den Rand. Dafür nimmst du die Rechteck-Blöcke aus der Kategorie **Canvas** (mehr dazu in Kapitel 11). Dann läuft die Turtle in kleinen Schritten von 3 Pixeln. Vor jedem Schritt fragt sie: Ist vor mir Rot? Dann dreht sie sich um eine zufällige Zahl zwischen 90 und 180 Grad.





Nach 20 Sekunden



Schritt 3 prüft vor der Wand, Schritt 4 schon dahinter.

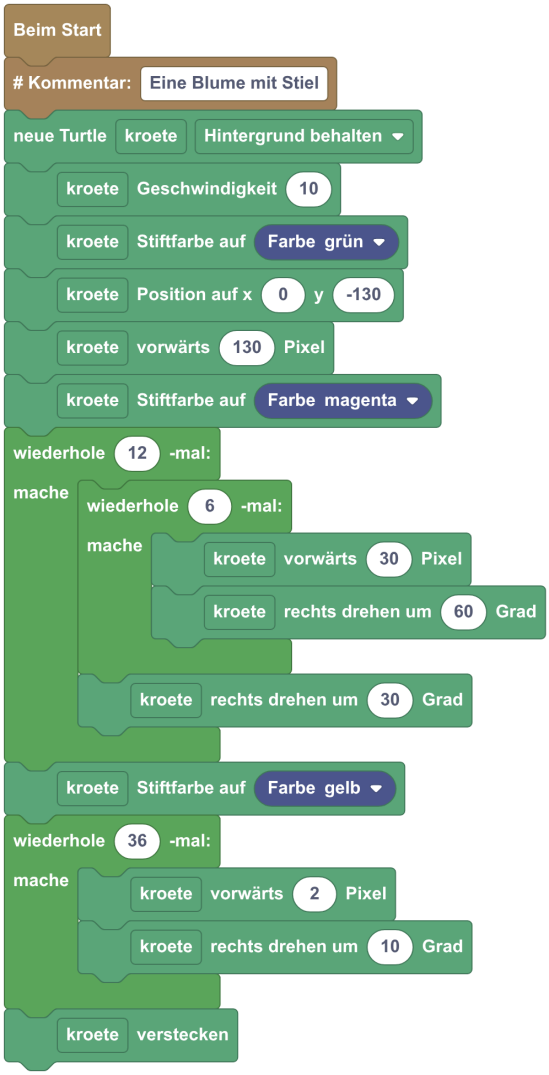
Damit die Turtle die Wand sicher sieht:

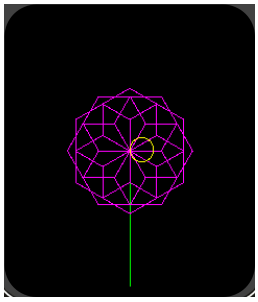
- Male Wände und Hindernisse **dicker als ein Schritt** der Turtle. Sonst prüft sie einmal kurz vor der Wand und beim nächsten Schritt schon dahinter (siehe Bild oben, rechts). Die Wände hier sind 10 Pixel dick.
- Die Farbe muss **genau** gleich sein. Male mit „Farbe rot“ und frage nach „Farbe rot“.
- Wähle für den Stift eine **andere Farbe** als für die Wand. Sonst hält die Turtle ihre eigenen Linien für eine Wand.
- Die Turtle sieht nur, was auf dem Canvas liegt: Linien, Canvas-Formen und Bilder.

Übrigens: Fehlt in „unendlich wiederholen“ ein „warte“-Block, fragt PBC+ Studio vor dem Start nach („Endlosschleife ohne Pause“).

Ausprobieren: Eine Blume

Die Turtle malt zuerst den grünen Stiel von unten bis zur Mitte. Dann malt sie 12 Sechsecke in Magenta, jedes um 30 Grad gedreht – das sind die Blütenblätter. Zum Schluss malt sie in der Mitte einen kleinen gelben Kreis: 36-mal 2 Pixel vorwärts und 10 Grad drehen.





Die Blume im Emulator

Das Programm findest du als Datei `k10_blume.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Dreieck** Lass die Turtle ein Dreieck mit 100 Pixel langen Seiten malen. Benutze eine Schleife.

Aufgabe 2 ⭐ **Ein Stern** Ändere dein Dreieck-Programm: 5 Wiederholungen, 100 Pixel vorwärts, 144 Grad nach rechts drehen, Stiftfarbe gelb. Was entsteht? Wie oft hat sich die Turtle insgesamt gedreht?

Aufgabe 3 ⭐⭐ **Ein Kreis** Die Turtle kann keine Kreise malen – oder doch? Lass sie 36-mal 8 Pixel vorwärts laufen und sich jedes Mal um 10 Grad drehen. Erkläre, warum das fast wie ein Kreis aussieht.

Aufgabe 4 ⭐⭐ **Drei Quadrate** Male drei Quadrate (Abschnitt 10.2) nebeneinander, mit 20 Pixeln Abstand. Die Turtle darf zwischen den Quadraten nichts malen. *Tipp:* Starte bei x = −100, y = 0. Zwischen zwei Quadraten: Stift heben, nach rechts drehen, 60 Pixel vorwärts, nach links drehen, Stift senken.

Aufgabe 5 ⭐⭐ **Zehn Stöße** Ändere den Staubsauger-Roboter: Er soll nach 10 Wandberührungen anhalten und sich verstecken. *Tipp:* Zähle in einer Variable `stoesse` mit. Nimm statt „unendlich wiederholen“ die Schleife „wiederhole solange stoesse < 10“.

Aufgabe 6 ⭐⭐⭐ **Die Vieleck-Maschine** Lass die Turtle nacheinander ein Dreieck, Viereck, Fünfeck … bis zum Zehneck malen, jedes in einer anderen Farbe. Alle starten an derselben Stelle. *Tipp:* Nimm die Zählschleife „zähle ecken von 3 bis 10“. Den Drehwinkel rechnest du mit dem Mathe-Block aus: 360 ÷ `ecken`. Für die Farbe nimmst du „Farbe rot … grün … blau …“ und setzt bei Rot `ecken` × 25 ein.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Zuerst kommt „neue Turtle“ unter „Beim Start“, danach die Befehle.
- Die Turtle startet in der Mitte (0 | 0) und blickt nach oben. y zählt bei ihr **nach oben**.
- „vorwärts“ und „rückwärts“ malen eine Linie, „rechts drehen“ und „links drehen“ ändern nur die Blickrichtung.
- Mit gehobenem Stift läuft die Turtle, ohne zu malen. „Position auf“ springt immer, ohne zu malen.
- Geschwindigkeit 0 = sofort fertig, 1 = langsam, 10 = schnell.
- Für ein Vieleck gilt: Drehwinkel = 360 ÷ Anzahl der Ecken.
- „Turtle berührt Farbe“ prüft den Punkt 3 Pixel vor der Turtle. Wände müssen dicker sein als ein Schritt.

KAPITEL 11

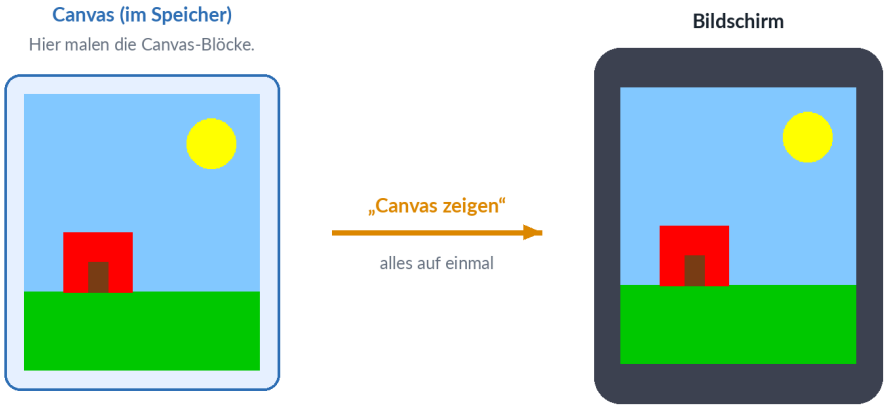
Der Canvas – ein Hintergrund im Speicher

Worum geht's? Ein Maler malt sein Bild zuerst auf eine Leinwand und hängt es erst auf, wenn es fertig ist. Genauso arbeitet der **Canvas**: Du malst ein Bild im Speicher und zeigst es dann auf einmal. So entstehen Hintergründe, Postkarten und Bilder, die nicht flackern. Der Canvas ist auch das Blatt, auf dem die Turtle malt.

11.1 Canvas anlegen, füllen, zeigen

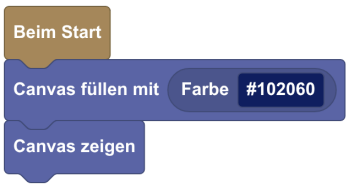
Canvas ist das englische Wort für Leinwand. Der Canvas ist ein Bild im Speicher des PBC+, genau so groß wie der Bildschirm: 240 × 280 Pixel. Die Koordinaten sind dieselben wie beim Bildschirm: (0 | 0) ist oben links.

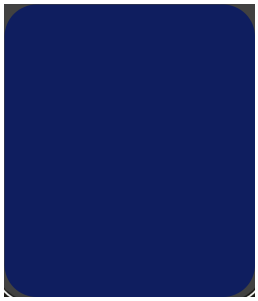
Was du auf den Canvas malst, siehst du **noch nicht**. Erst der Block „Canvas zeigen" bringt das ganze Bild auf einmal auf den Bildschirm.



Alle Blöcke dafür findest du in der Kategorie **Canvas**:

Block	Was passiert?
Canvas füllen mit Farbe blau	Der ganze Canvas bekommt eine Farbe.
Canvas zeigen	Der Canvas erscheint auf dem Bildschirm.
Canvas leeren	Der Canvas wird wieder schwarz.
Canvas anlegen	Ein neuer, schwarzer Canvas wird angelegt.





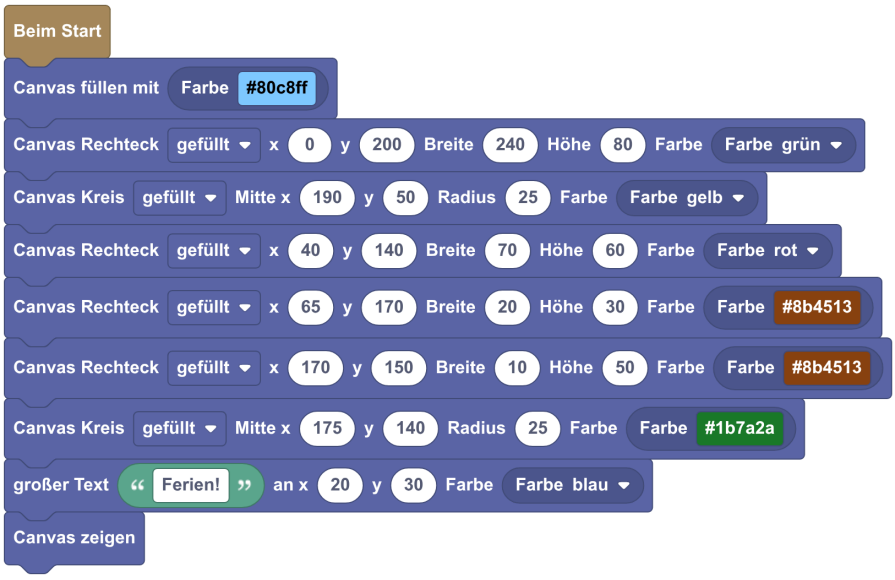
Ein dunkelblauer Nachthimmel

> **Es gibt genau einen Canvas.** PBC+ Studio legt ihn automatisch an, sobald dein Programm einen Canvas-Block oder eine Turtle benutzt. Am Anfang ist er schwarz. Den Block „Canvas anlegen“ brauchst du deshalb nicht. Wenn du ihn trotzdem benutzt, muss er **vor** allen anderen Canvas-Blöcken stehen – sonst meldet das Programm einen Fehler (`NameError`).

11.2 Auf dem Canvas zeichnen und schreiben

In der Kategorie **Canvas** gibt es Pixel, Linie, Rechteck, Ellipse und Kreis sowie Text in drei Größen. Sie funktionieren genau wie die Blöcke aus Kapitel 3, malen aber auf den Canvas.

> Die Text-Blöcke für den Canvas sehen fast genauso aus wie die Text-Blöcke für das Display. Sie haben nur eine andere Farbe. Nimm sie aus der Kategorie **Canvas**! Rechteck, Kreis und Co. beginnen dagegen mit dem Wort „Canvas“.





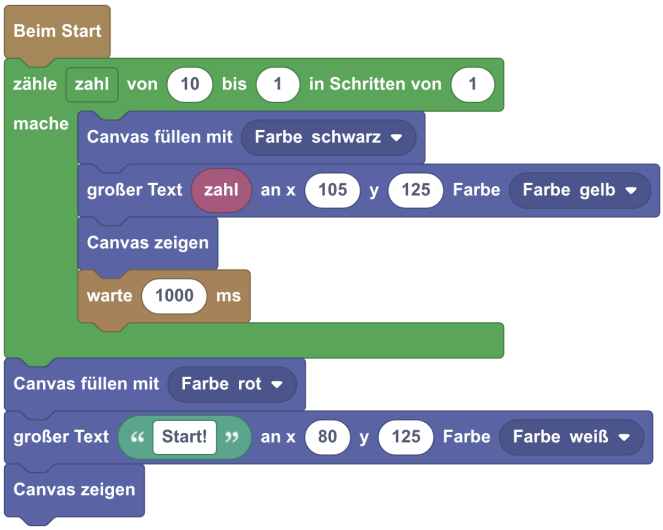
Erst „Canvas zeigen“ macht das Bild sichtbar.

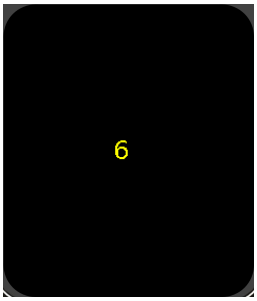
Die Reihenfolge ist wichtig: Was später gemalt wird, liegt oben. Deshalb kommt der Himmel zuerst und das Haus danach.

> **Tipp:** Umlaute und ß kann der PBC+ auf dem Bildschirm nicht darstellen – er zeigt dafür ein „?“. Schreibe in Texten auf dem Bildschirm „ue“ statt „ü“ oder wähle ein anderes Wort.

Ohne Flackern

Willst du ein Bild immer wieder ändern, löschst du normalerweise den Bildschirm und malst neu. Dabei ist der Bildschirm kurz leer – er flackert. Mit dem Canvas passiert das nicht: Du malst das neue Bild im Speicher, und „Canvas zeigen“ ersetzt das alte Bild auf einmal. Ein Countdown von 10 bis 1:

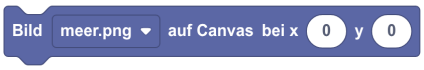




> **Emulator und Gerät:** Im Emulator dauert das Bemalen des Canvas mit Blöcken länger als auf dem echten PBC+. Ein Rechteck so groß wie der ganze Bildschirm braucht im Emulator etwa 0,2 Sekunden. Für Spiele, die in jedem Schritt das ganze Bild neu malen, ist das zu langsam. „Canvas füllen“ und „Bild auf Canvas“ sind dagegen auch im Emulator schnell.

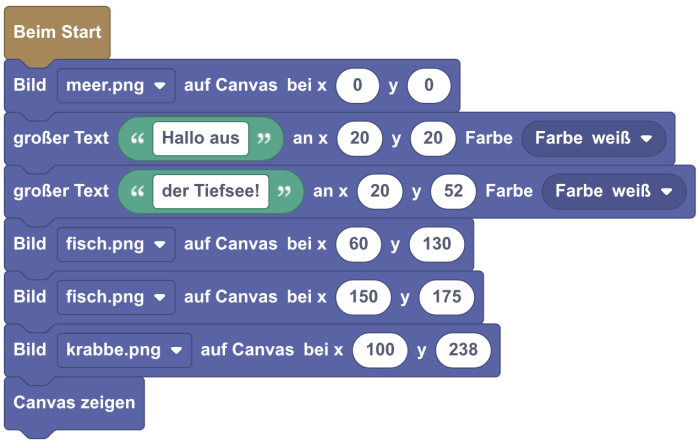
11.3 Ein Bild auf den Canvas legen

Mit „Bild auf Canvas“ legst du ein Bild auf den Canvas. Das Bild musst du vorher hochladen (**Dateien → Bilder verwalten…**) oder selbst malen (**Dateien → Bild zeichnen…**). Im Block wählst du es aus der Liste aus.



- x und y geben an, wo die **linke obere Ecke** des Bildes liegt.
- Ein Bild, das 240 × 280 Pixel groß ist, füllt bei x = 0, y = 0 den ganzen Canvas – ein Hintergrundbild.
- Kleine Bilder kannst du so oft auf den Canvas legen, wie du willst. Durchsichtige Stellen bleiben durchsichtig.

Eine Postkarte aus der Tiefsee: Hintergrund `meer.png`, darauf ein Gruß und drei kleine Bilder.

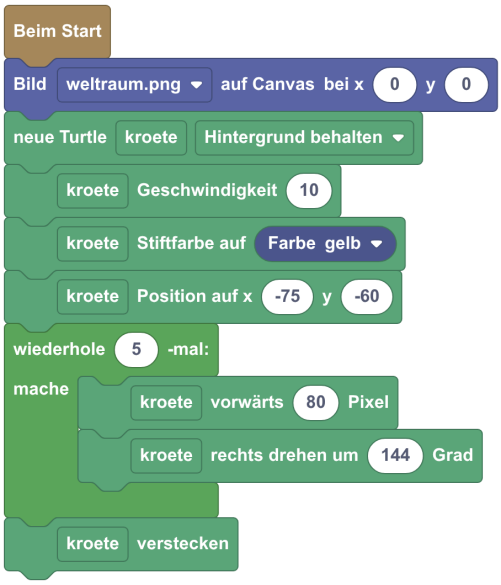


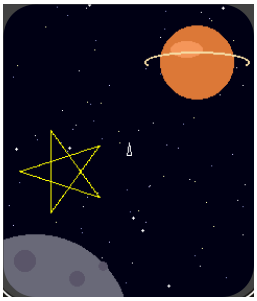


11.4 Canvas und Turtle

Die Turtle aus Kapitel 10 malt **auf den Canvas**. Deshalb kannst du zuerst einen Hintergrund auf den Canvas legen und dann die Turtle darauf malen lassen. Im Block „neue Turtle“ stellst du ein, was mit dem Hintergrund passiert:

- **Hintergrund behalten:** Die Turtle malt auf das, was schon auf dem Canvas ist.
- **neuer Hintergrund:** Der Canvas wird zuerst schwarz.





Ein Stern im Weltall

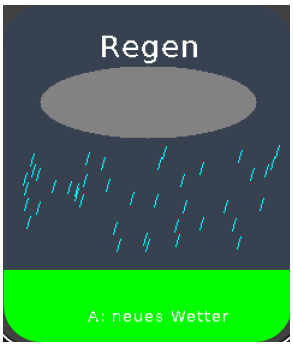
Hier brauchst du kein „Canvas zeigen“: Wenn die Turtle erstellt wird, zeigt sie den Canvas mit allem, was schon darauf ist.

> **Tipp:** Male den Hintergrund **bevor** du die Turtle erstellst. Malst du später noch etwas auf den Canvas, dann hänge „Canvas zeigen“ an. Nimm in Turtle-Programmen nur Canvas-Blöcke, keine Display-Blöcke – die Turtle malt sonst über deine Display-Bilder hinweg.

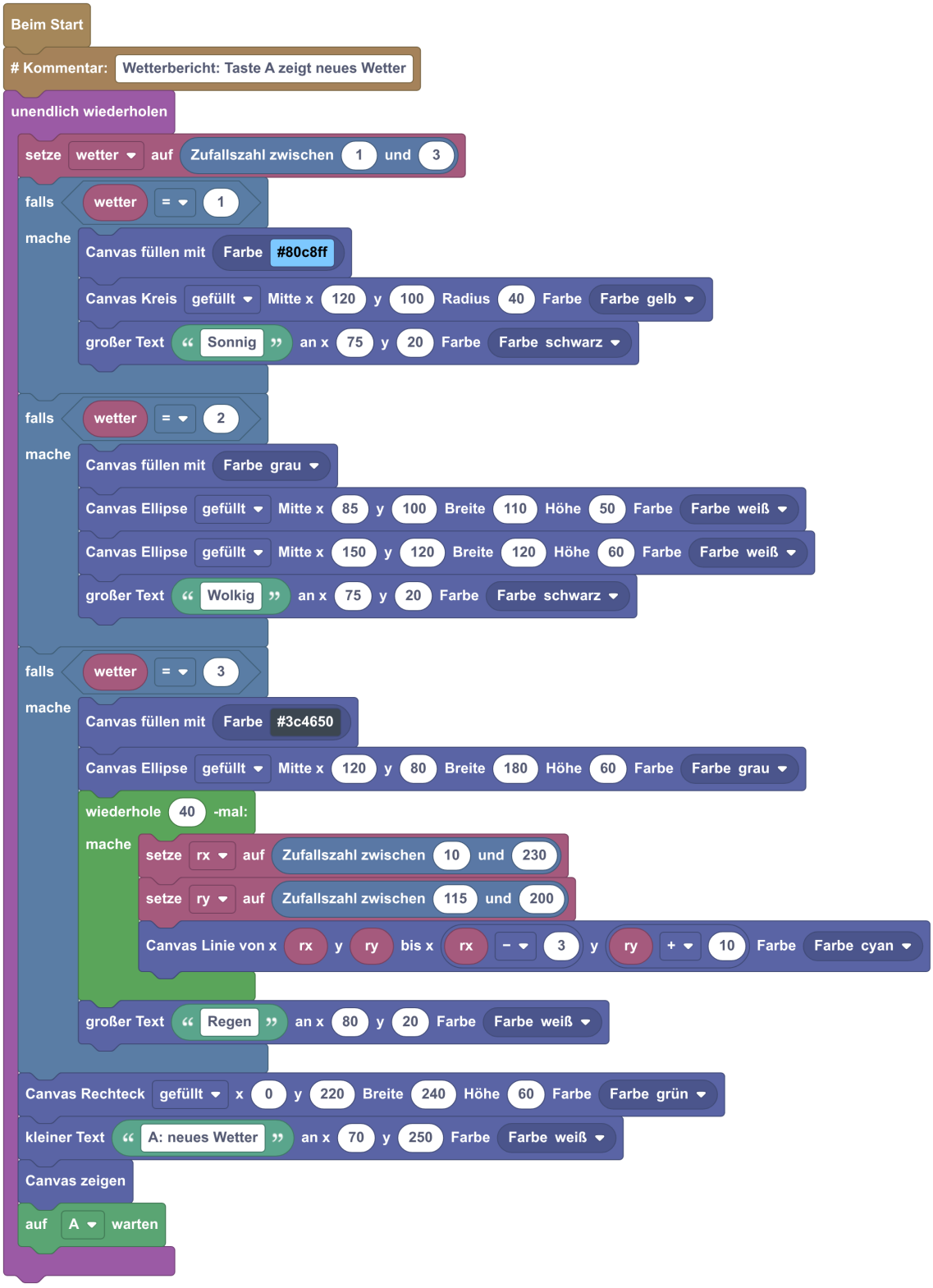
Ausprobieren: Der Wetterbericht

Der PBC+ würfelt das Wetter aus: 1 = Sonne, 2 = Wolken, 3 = Regen. Je nach Zahl malt er ein anderes Bild auf den Canvas. Für den Regen malt er 40 kurze schräge Linien an zufällige Stellen. Unten liegt bei jedem Wetter eine grüne Wiese. Dann zeigt er den Canvas und wartet auf die Taste A. Weil das neue Bild erst fertig gemalt und dann gezeigt wird, springt es ohne Flackern um.

Das Programm findest du als Datei `k11_wetter.pbcp` im Ordner `buch/programme`.



Heute: Regen



Übungsaufgaben

Aufgabe 1 ⭐ **Eine Flagge** Male die deutsche Flagge auf den Canvas: drei gleich hohe Streifen in Schwarz, Rot und Gold. Für Gold nimmst du den Farbwähler. *Tipp:* 280 ÷ 3 ist ungefähr 93.

Aufgabe 2 ⭐ Wo ist das Bild? Nimm die Landschaft aus 11.2 und lösche den Block „Canvas zeigen“. Was siehst du jetzt? Erkläre, warum.

Aufgabe 3 ⭐⭐ Postkarte aus dem All Lege das Bild `weltraum.png` auf den Canvas. Male einen weißen Rahmen (Rechteck als Umriss) und schreibe einen Gruß in zwei Zeilen.

Aufgabe 4 ⭐⭐ Sternenhimmel Male 100 weiße Pixel an zufällige Stellen des Canvas und einen gelben Mond. *Tipp:* Nimm „wiederhole 100-mal“ und „Canvas Pixel“ mit zwei Zufallszahlen.

Aufgabe 5 ⭐⭐⭐ Die Turtle malt eine Sonne Male auf den Canvas einen hellblauen Himmel und eine grüne Wiese. Dann malt die Turtle oben rechts eine gelbe Sonne aus 12 Strahlen: vorwärts, rückwärts, um 30 Grad drehen.

Aufgabe 6 ⭐⭐⭐ Fischschwarm Lege `meer.png` auf den Canvas und darauf zehnmal `fisch.png` an zufälliger Stelle. Denk an den Hinweis zum Emulator in 11.2!

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Der Canvas ist ein Bild im Speicher, so groß wie der Bildschirm. Es gibt genau einen; PBC+ Studio legt ihn automatisch an.
- Was du auf den Canvas malst, siehst du erst nach „Canvas zeigen“.
- „Canvas zeigen“ ersetzt das ganze Bild auf einmal – nichts flackert.
- Canvas-Blöcke findest du in der Kategorie **Canvas**. Die Text-Blöcke sehen aus wie die des Displays, haben aber eine andere Farbe.
- „Bild auf Canvas“ legt ein hochgeladenes Bild an die Stelle x, y (linke obere Ecke).
- Die Turtle malt auf den Canvas. Male den Hintergrund, bevor du die Turtle erstellst.

KAPITEL 12

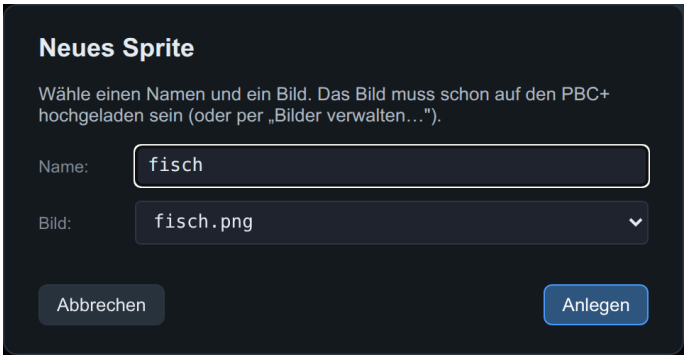
Sprites – Figuren, die sich bewegen

Worum geht's? In Spielen laufen Figuren über den Bildschirm: ein Fisch, eine Rakete, ein Ufo. Solche Figuren heißen **Sprites**. Ein Sprite ist ein kleines Bild mit einem Namen. Du kannst es zeigen, verstecken, an eine Stelle setzen und gleiten lassen – und der Hintergrund bleibt dabei heil.

12.1 Ein Sprite anlegen

Ein Sprite braucht ein Bild. Das Bild muss **vorher** in PBC+ Studio sein:

1. Lade das Bild hoch: **Dateien → Bilder verwalten…** In diesem Kapitel brauchst du `fisch.png`, `krabbe.png`, `rakete.png`, `ufo.png` und `meer.png`. Du kannst auch selbst eins malen: **Dateien → Bild zeichnen…**
2. Öffne in der Blockleiste die Kategorie **Sprites** und klicke auf **„+ neues Sprite…“**.
3. Gib einen Namen ein, zum Beispiel `fisch`, und wähle das Bild aus. Klicke auf **Anlegen**.



Der Dialog „Neues Sprite“



Das neue Sprite in der Kategorie Sprites

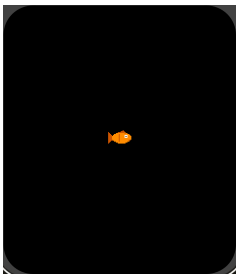
Jetzt steht `fisch` in der Kategorie **Sprites**, und darunter alle Befehle. In jedem Sprite-Block wählst du im Feld mit dem Pfeil ▾, welches Sprite gemeint ist.

- Der Name darf nur Buchstaben, Ziffern und `_` enthalten – keine Umlaute, kein ß, keine Leerzeichen. Sonst meldet PBC+ Studio „Ungültiger Name“.
- Holst du einen Sprite-Befehl, bevor es ein Sprite gibt, meldet PBC+ Studio „Sprite fehlt“.
- Mehrere Sprites dürfen dasselbe Bild haben, zum Beispiel `fisch1` und `fisch2` mit `fisch.png`.

12.2 Zeigen und verstecken

Ein neues Sprite ist am Anfang **unsichtbar** und steht oben links bei (0 | 0). Mit „zeige Sprite“ erscheint es, mit „verstecke Sprite“ verschwindet es wieder.





Der Fisch in der Mitte

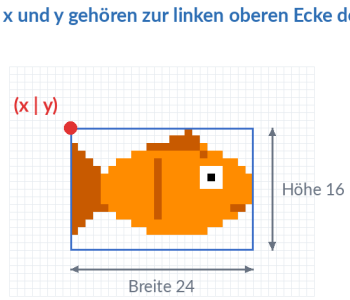
Setze das Sprite immer zuerst an seinen Platz und zeige es danach. Sonst blitzt es kurz oben links auf.

Wechseln sich Zeigen und Verstecken schnell ab, blinkt das Sprite. Das Ufo blinkt fünfmal und bleibt dann sichtbar:

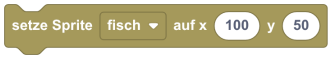
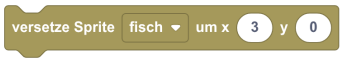


12.3 Setzen, versetzen, gleiten

Der Platz eines Sprites wird mit x und y angegeben – mit denselben Koordinaten wie beim Bildschirm: (0 | 0) ist oben links, y zählt nach unten. x und y gehören zur **linken oberen Ecke** des Bildes.



Vier Blöcke bewegen ein Sprite:

Block	Was passiert?
	Das Sprite springt an die Stelle x = 100, y = 50.
	Das Sprite rückt 3 Pixel nach rechts. Negative Zahlen: nach links (x) oder nach oben (y).
	Das Sprite gleitet in 1000 ms (1 Sekunde) zur Stelle 100, 50.
	Das Sprite gleitet in 1 Sekunde 50 Pixel nach rechts.

Gleiten: Die Rakete startet

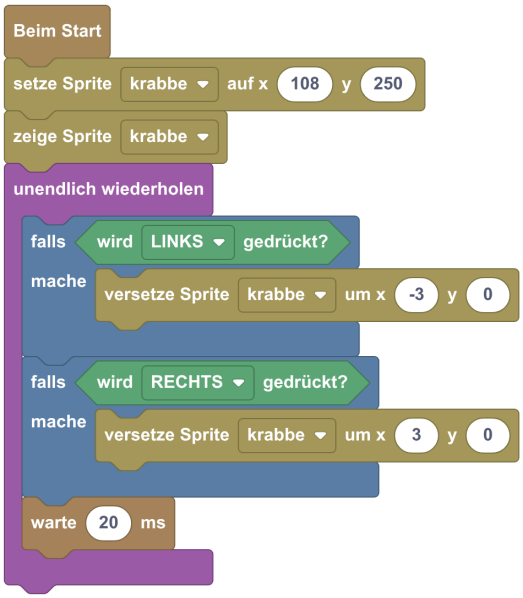
Die Rakete steht unten, wartet eine Sekunde und gleitet dann in 2 Sekunden nach oben aus dem Bild. Die Rakete ist 28 Pixel hoch. Bei y = −40 ist sie also ganz verschwunden.



Wichtig: Während ein Sprite gleitet, **wartet das Programm**. Der nächste Block kommt erst dran, wenn das Sprite angekommen ist. In dieser Zeit reagiert dein Programm auch nicht auf Tasten.

Versetzen: Die Krabbe läuft

Willst du ein Sprite mit dem Joystick steuern, versetzt du es in einer Schleife immer wieder um ein kleines Stück:



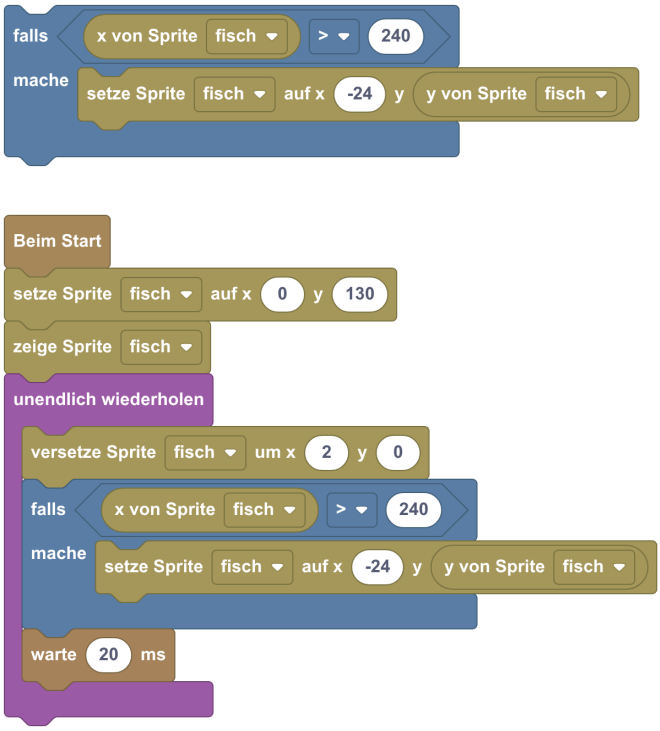
Solange du den Joystick nach rechts drückst, rückt die Krabbe alle 20 ms 3 Pixel nach rechts.

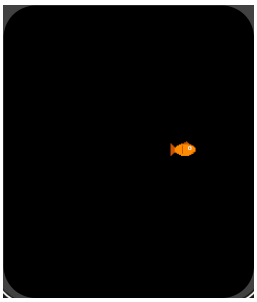
12.4 Die Position abfragen: x und y

Die Blöcke „x von Sprite“ und „y von Sprite“ verraten dir, wo ein Sprite gerade steht. Es sind runde Wert-Blöcke: Du steckst sie in Vergleiche oder Rechnungen.

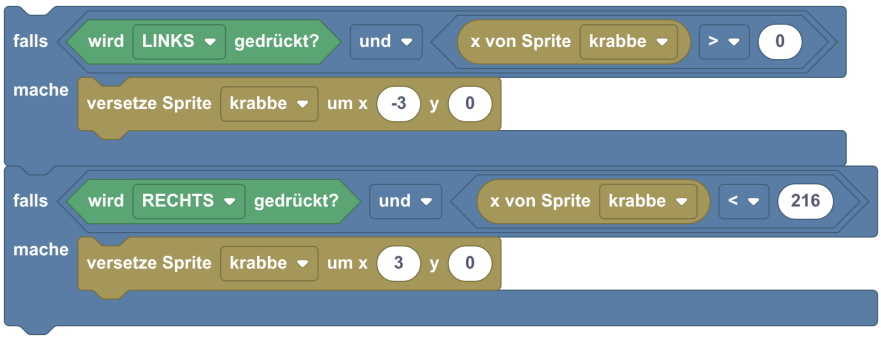


Der Fisch schwimmt immer wieder. Er schwimmt nach rechts. Ist er rechts aus dem Bildschirm verschwunden (x größer als 240), kommt er links wieder herein: bei x = −24 – das ist gerade seine Breite – und auf derselben Höhe wie vorher.





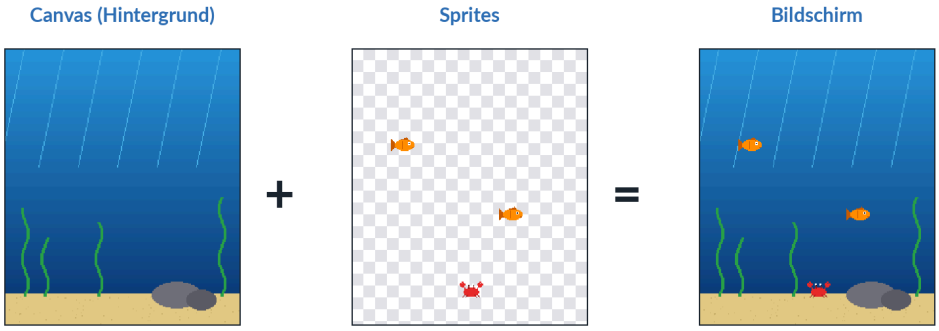
Die Krabbe bleibt auf dem Bildschirm. Sie soll nur nach links laufen, wenn sie noch nicht am linken Rand ist (x > 0), und nur nach rechts, wenn sie noch nicht am rechten Rand ist. Die Krabbe ist 24 Pixel breit, also muss x kleiner als 240 − 24 = 216 sein:



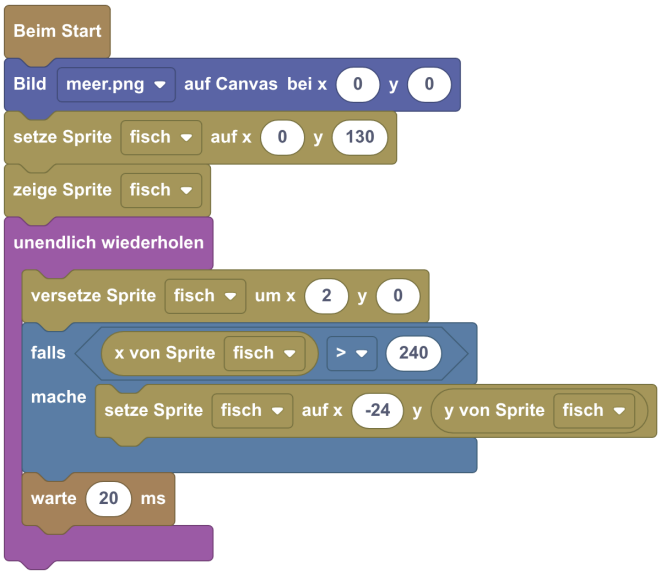
12.5 Sprites vor einem Hintergrund

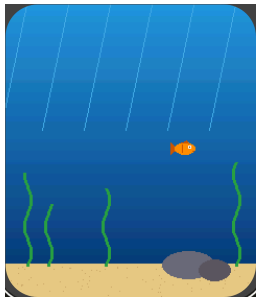
Bei jeder Bewegung eines Sprites malt der PBC+ den **ganzen Bildschirm** neu: zuerst den Hintergrund, dann alle sichtbaren Sprites. Was zuletzt gezeigt wurde, liegt oben.

Ohne Canvas ist der Hintergrund schwarz. Benutzt dein Programm den Canvas aus Kapitel 11, wird der Canvas **automatisch** zum Hintergrund der Sprites. „Canvas zeigen“ brauchst du dafür nicht.



Probier es aus: Hänge im Fisch-Programm aus 12.4 den Block „Bild meer.png auf Canvas“ direkt unter „Beim Start“. Jetzt schwimmt der Fisch durchs Meer.



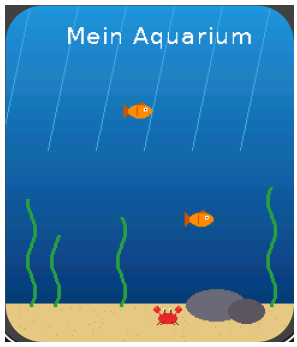


> Was du mit den **Display**-Blöcken malst oder schreibst, verschwindet, sobald sich ein Sprite bewegt – der Bildschirm wird ja neu gemalt. Alles, was stehen bleiben soll (Hintergrund, Titel, Punktestand), malst und schreibst du auf den **Canvas**.

Ausprobieren: Das Aquarium

Zwei Fische schwimmen verschieden schnell durch das Meer. Kommen sie rechts heraus, tauchen sie links in einer zufälligen Höhe wieder auf. Die Krabbe am Boden steuerst du mit dem Joystick. Der Titel steht auf dem Canvas und bleibt deshalb stehen.

Lege drei Sprites an: `fisch1` und `fisch2` mit `fisch.png`, `krabbe` mit `krabbe.png`. Das Programm findest du als Datei `k12_aquarium.pbcp` im Ordner `buch/programme`.



Das Aquarium im Emulator

```
Beim Start
# Kommentar: Aquarium mit Fischen und Krabbe
Bild meer.png auf Canvas bei x 0 y 0
Text " Mein Aquarium " an x 50 y 15 Farbe Farbe weiß
setze Sprite fisch1 auf x 0 y 80
setze Sprite fisch2 auf x 100 y 170
setze Sprite krabbe auf x 108 y 250
zeige Sprite fisch1
zeige Sprite fisch2
zeige Sprite krabbe
unendlich wiederholen
    versetze Sprite fisch1 um x 2 y 0
    versetze Sprite fisch2 um x 1 y 0
    falls x von Sprite fisch1 > 240
    mache setze Sprite fisch1 auf x -24 y Zufallszahl zwischen 40 und 200
    falls x von Sprite fisch2 > 240
    mache setze Sprite fisch2 auf x -24 y Zufallszahl zwischen 40 und 200
    falls wird LINKS gedrückt? und x von Sprite krabbe > 0
    mache versetze Sprite krabbe um x -3 y 0
    falls wird RECHTS gedrückt? und x von Sprite krabbe < 216
    mache versetze Sprite krabbe um x 3 y 0
    warte 20 ms
```

Übungsaufgaben

Aufgabe 1 ⭐ **Ufo-Landung** Das Ufo erscheint oben links und gleitet in 3 Sekunden nach unten in die Mitte (x = 106, y = 250). Dort blinkt es dreimal.

Aufgabe 2 ⭐ **Rundflug** Die Rakete fliegt dreimal im Kreis durch die vier Ecken des Bildschirms. Jede Strecke dauert 1 Sekunde. *Tipp:* Die Rakete ist 16 Pixel breit und 28 Pixel hoch.

Aufgabe 3 ⭐⭐ **Rakete im Weltall** Steuere die Rakete mit dem Joystick in alle vier Richtungen vor dem Hintergrund `weltraum.png`.

Aufgabe 4 ⭐⭐ **Neue Höhe** Ändere das Fisch-Programm aus 12.5: Kommt der Fisch links wieder herein, soll er jedes Mal in einer zufälligen Höhe schwimmen.

Aufgabe 5 ⭐⭐ **Auf und ab** Das Ufo schwebt immer auf und ab. Es bewegt sich in jedem Schritt um `dy` Pixel nach unten. Ist es unten (y > 260) oder oben (y < 10) angekommen, wird aus `dy` die Gegenrichtung: aus 2 wird −2.

Aufgabe 6 ⭐⭐⭐ **Das Ufo verfolgt dich** Du steuerst die Rakete. Das Ufo kommt ihr in jedem Schritt 1 Pixel näher: Ist das Ufo weiter links als die Rakete, rückt es nach rechts – und so weiter für alle vier Richtungen.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Bild hochladen, dann in der Kategorie **Sprites** mit „+ neues Sprite…“ ein Sprite anlegen: Name ohne Umlaute, Bild auswählen.
- Ein neues Sprite ist unsichtbar. Erst setzen, dann zeigen.
- x und y gehören zur linken oberen Ecke des Sprite-Bildes.
- „setze“ springt, „versetze“ rückt ein Stück weiter, „gleite“ bewegt langsam – und das Programm wartet, bis das Sprite angekommen ist.
- „x von Sprite“ und „y von Sprite“ sagen, wo das Sprite gerade steht.
- Der Canvas ist automatisch der Hintergrund der Sprites. Display-Bilder verschwinden, wenn sich ein Sprite bewegt.

KAPITEL 13

Kollisionen

Worum geht's? In fast jedem Spiel muss das Programm merken, ob sich etwas berührt: Hat der Hai den Fisch erwischt? Ist der Ball am Rand angekommen? Ist das Rennauto von der Strecke abgekommen? Dafür gibt es drei Blöcke. In diesem Kapitel lernst du, wie sie funktionieren – und wo sie an ihre Grenzen kommen.

Wenn zwei Dinge zusammenstoßen, nennt man das eine **Kollision**. Für dieses Kapitel brauchst du, was du in Kapitel 12 über Sprites gelernt hast: ein Sprite anlegen, zeigen, setzen und versetzen und seine Position mit „x von Sprite" und „y von Sprite" abfragen.

Die Beispiele benutzen diese Bilder aus dem Material zum Buch: `fisch.png` (den Fisch kennst du aus Kapitel 12), `hai.png`, `pingpong.png`, `schlaeger.png` und `rennauto.png`. Lade sie über **Dateien → Bilder verwalten…** hoch. Du kannst auch eigene Bilder malen (**Dateien → Bild zeichnen…**).

Die drei Kollisions-Blöcke

Alle drei Blöcke findest du in der Kategorie **Sprites**. Sie haben spitze Enden: Sie sind *Frage-Blöcke*. Die Antwort ist immer **wahr** oder **falsch**. Deshalb steckst du sie in einen „falls"-Block oder in eine „solange"-Schleife.

Block	Fragt …
	Berührt das eine Sprite das andere?
	Ist das Sprite (ein Stück) aus dem Bildschirm gelaufen?
	Liegt das Sprite auf dieser Farbe des Hintergrunds?

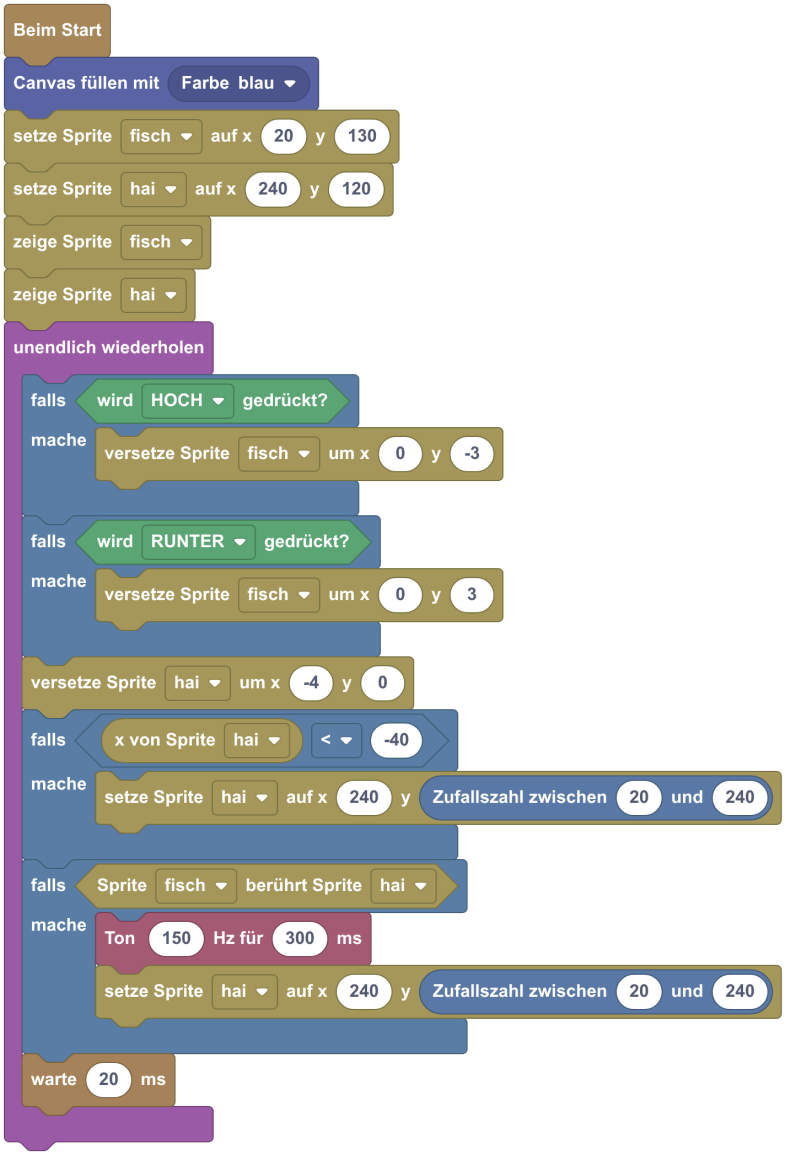
13.1 Sprite berührt Sprite

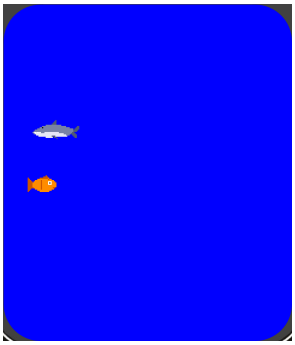
Der Fisch aus Kapitel 12 bekommt Besuch: Von rechts kommt ein Hai! Mit dem Joystick bewegst du den Fisch nach oben und unten und weichst aus.

Die Frage „Hat der Hai den Fisch erwischt?" stellst du so: Berührt der Fisch den Hai, ertönt ein tiefer Ton. Dann taucht der Hai wieder rechts auf, an einer zufälligen Höhe.



Dieser „falls“-Block muss **in der Schleife** stehen. Nur so fragt das Programm immer wieder nach. Stünde er vor der Schleife, würde das Programm nur ein einziges Mal fragen – ganz am Anfang, wenn der Hai noch weit weg ist. Das ganze Programm sieht so aus:





Fisch und Hai im Emulator

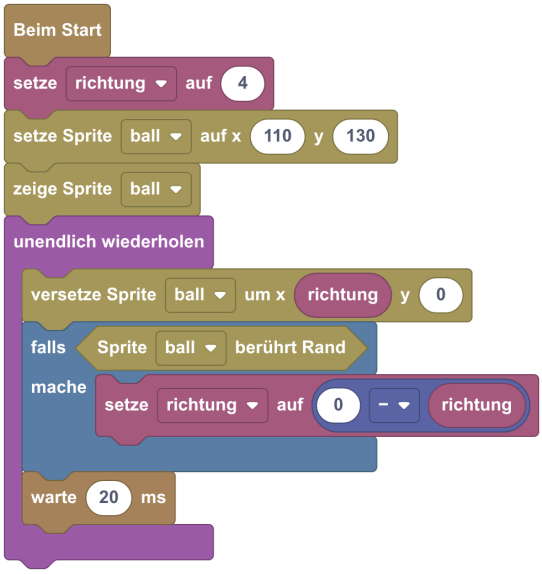
So funktioniert es:

- „Canvas füllen mit blau“ malt das Meer als Hintergrund (Kapitel 11).
- In jedem Durchlauf schwimmt der Hai 4 Pixel nach links.
- Der Hai ist 40 Pixel breit. Ist sein x kleiner als −40, ist er links ganz aus dem Bild verschwunden. Dann setzt ihn das Programm wieder nach rechts.

Gut zu wissen:

- Wähle im Block zwei **verschiedene** Sprites. „Sprite hai berührt Sprite hai“ ist immer wahr – ein Sprite liegt ja immer auf sich selbst.
- Auch ein **verstecktes** Sprite kann berühren! Der Block prüft nur die Position, nicht, ob du das Sprite siehst.

13.2 Sprite berührt Rand



Ein Ball soll hin und her fliegen. Er bewegt sich in jedem Schritt um `richtung` Pixel nach rechts. Berührt er den Rand, wird aus `richtung` die Gegenrichtung: aus 4 wird −4, aus −4 wird wieder 4. Das rechnet der Block „0 − richtung“ aus.

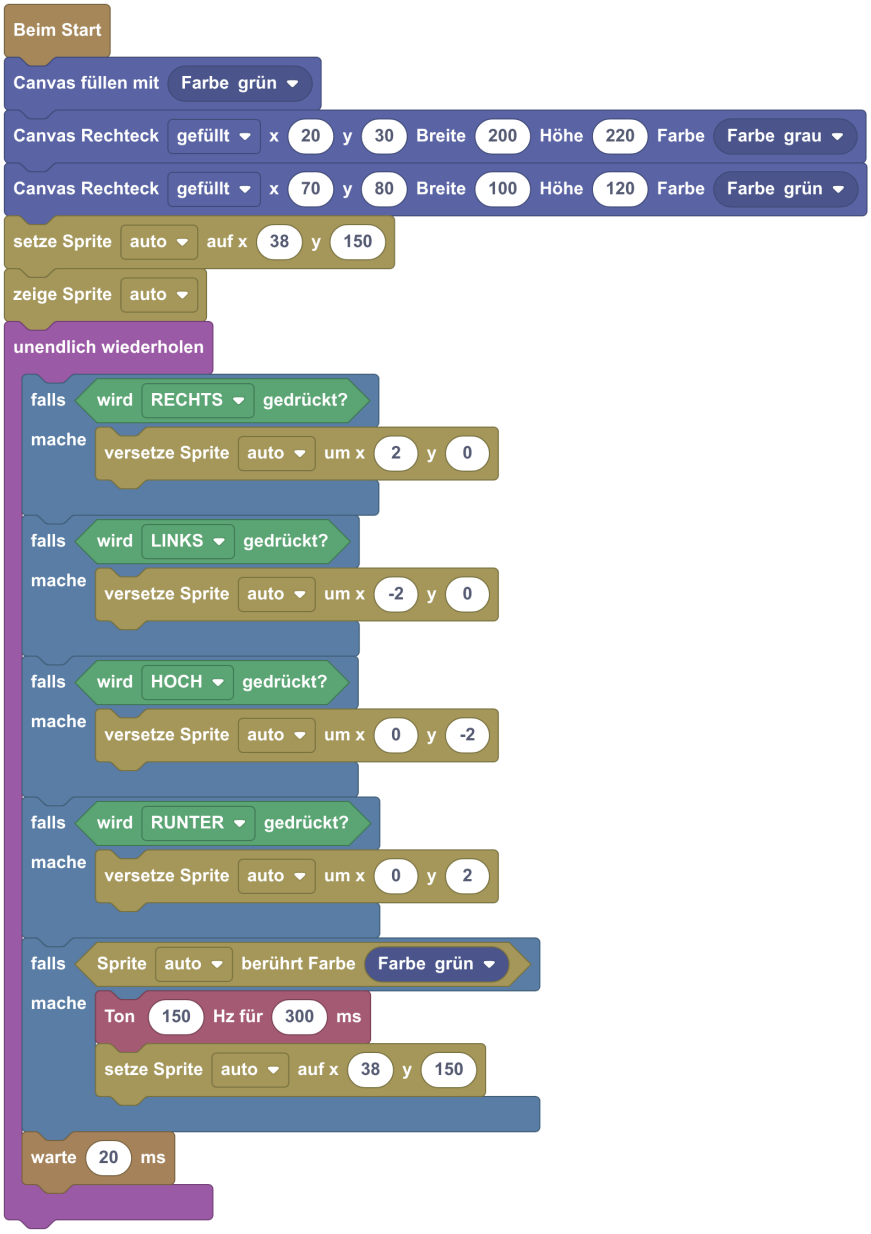
„berührt Rand“ ist wahr, sobald auch nur ein Pixel des Sprites außerhalb des Bildschirms liegt. Der Ball ist 12 Pixel breit. Steht er bei x = 228, liegt er ganz genau am rechten Rand (228 + 12 = 240): Die Antwort ist noch **falsch**. Bei x = 229 ragt ein Pixel hinaus: Die Antwort ist **wahr**.

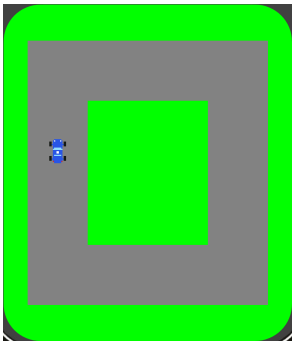
> **Tipp:** Der Block sagt dir nicht, *welchen* Rand das Sprite berührt. Das findest du mit „x von Sprite“ und „y von Sprite“ heraus: Ist x kleiner als 0, war es der linke Rand. Probier es in Aufgabe 6 aus.

13.3 Sprite berührt Farbe

Mit diesem Block kannst du Hindernisse einfach **malen**. Rote Lava, blaues Wasser, eine weiße Ziellinie – das Programm prüft, ob das Sprite auf einer bestimmten Farbe liegt.

Als Beispiel baust du eine Rennstrecke. Zuerst malst du sie mit den Canvas-Blöcken aus Kapitel 11 auf den Hintergrund: eine grüne Wiese, darauf ein graues Rechteck und in der Mitte wieder ein grünes. Übrig bleibt eine graue Bahn, 50 Pixel breit. Das sind die drei Canvas-Blöcke oben im Programm.





Bleib auf der grauen Bahn!

In der Schleife fragst du: Fährt das Auto auf der Wiese, also auf Grün? Dann ertönt ein Ton, und das Auto muss zurück zum Start.

Wichtig:

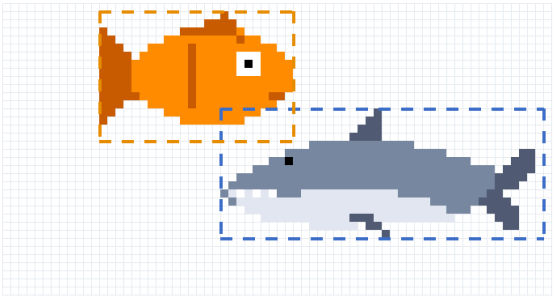
- Geprüft wird nur der **Hintergrund** – also alles, was auf dem Canvas liegt (gemalt oder mit „Bild auf Canvas“). Andere Sprites zählen nicht.
- Die Farbe muss **genau** gleich sein. Malst du die Wiese mit „Farbe grün“, prüfe auch auf „Farbe grün“. Ein Grün aus dem Farbwähler, das nur ein bisschen anders ist, wird nicht erkannt.
- Ohne Hintergrund ist die Antwort immer **falsch**.

13.4 Grenzen der Kollisionsabfrage

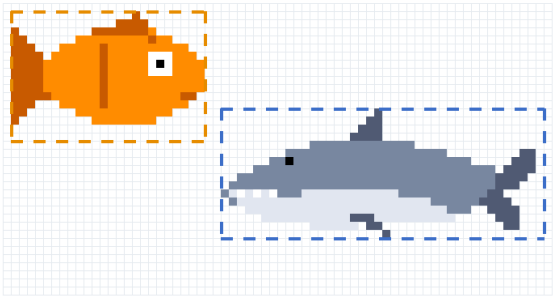
Sprites sind Rechtecke

Jedes Sprite-Bild ist ein Rechteck, auch wenn du nur einen Hai siehst. Alles um den Hai herum ist durchsichtig. „Sprite berührt Sprite“ prüft nur, ob sich die **Rechtecke** überlappen. Deshalb kann die Antwort schon wahr sein, wenn sich die Bilder noch gar nicht berühren:

Bilder berühren sich nicht – trotzdem: berührt = wahr



Rechtecke getrennt: berührt = falsch

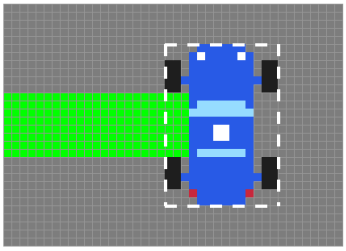


Bei kleinen, fast eckigen Sprites merkt man das beim Spielen kaum. Beim Hai schon: Er hat oben links eine große durchsichtige Ecke.

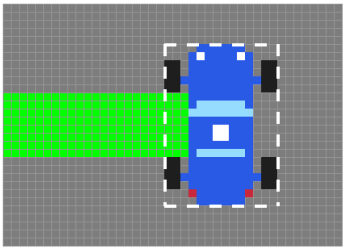
Bei „berührt Farbe“ zählen nur sichtbare Pixel

„Sprite berührt Farbe“ ist genauer: Hier zählen nur die sichtbaren Pixel des Sprites. Liegt nur ein durchsichtiges Stück auf der Wiese, passiert nichts.

Nur durchsichtige Pixel liegen auf Grün: falsch



Sichtbare Pixel liegen auf Grün: wahr



Die Frage ist eine Momentaufnahme

Ein Kollisions-Block schaut nur in dem Moment hin, in dem er an der Reihe ist. Was dazwischen passiert, sieht er nicht. Das hat zwei Folgen:

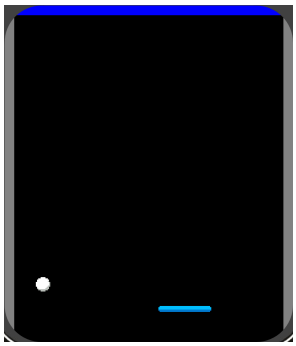
- **Große Sprünge:** Springt ein Sprite in einem Schritt weiter, als es selbst groß ist, kann es über eine dünne Wand hinwegspringen. Vorher liegt es davor, danach dahinter – die Wand hat es nie berührt. Bei einem Schritt von 3 Pixeln passiert das nicht. Aber mit „versetze um 0, 60“ kann eine 16 Pixel große Figur über einen 20 Pixel dicken Lavastreifen springen. Male Wände deshalb **dicker als der größte Sprung**.
- **Gleiten:** Während „gleite Sprite …“ läuft, wartet das Programm. Die Kollisions-Blöcke in deiner Schleife fragen erst wieder, wenn das Sprite angekommen ist. Für Spielfiguren, die mitten im Weg zusammenstoßen sollen, nimmst du deshalb „versetze“ in kleinen Schritten.

> Auch die Turtle kann Farben erkennen: Der Block „Turtle berührt Farbe“ prüft die Stelle direkt vor der Turtle (Kapitel 10.5).

Ausprobieren: Ball und Schläger

Jetzt kommen alle drei Blöcke zusammen. Du steuerst unten einen Schläger nach links und rechts. Der Ball fliegt schräg durch das Bild:

- Trifft der Ball eine **graue** Seitenwand, dreht sich `dx` um.
- Trifft er die **blaue** Decke, fliegt er nach unten (`dy` = 3).
- Trifft er den **Schläger**, fliegt er nach oben (`dy` = −3).
- Verpasst du ihn, fällt er unten aus dem Bild – er **berührt den Rand**. Dann ertönt ein tiefer Ton, und der Ball startet neu.



Ball und Schläger im Emulator

Das Programm findest du als Datei `k13_pong.pbcp` im Ordner `buch/programme`. Lade vor dem Öffnen die Bilder `pingpong.png` und `schlaeger.png` hoch (Dateien → Bilder verwalten…).

> Im Emulator läuft das Spiel langsamer als auf dem PicoBoy: Ein mit Blöcken bemalter Canvas braucht dort mehr Zeit.


Beim Start
Kommentar: Ball und Schläger: Wände, Schläger, Boden
Canvas Rechteck gefüllt x 0 y 0 Breite 8 Höhe 280 Farbe Farbe grau
Canvas Rechteck gefüllt x 232 y 0 Breite 8 Höhe 280 Farbe Farbe grau
Canvas Rechteck gefüllt x 0 y 0 Breite 240 Höhe 8 Farbe Farbe blau
setze dx auf 3
setze dy auf 3
setze Sprite ball auf x 110 y 100
setze Sprite schlaeger auf x 98 y 250
zeige Sprite ball
zeige Sprite schlaeger
unendlich wiederholen
falls wird RECHTS gedrückt?
mache versetze Sprite schlaeger um x 5 y 0
falls wird LINKS gedrückt?
mache versetze Sprite schlaeger um x -5 y 0
versetze Sprite ball um x dx y dy
falls Sprite ball berührt Farbe Farbe grau
mache setze dx auf 0 - dx
falls Sprite ball berührt Farbe Farbe blau
mache setze dy auf 3
falls Sprite ball berührt Sprite schlaeger
mache setze dy auf -3
Ton 880 Hz für 50 ms
falls Sprite ball berührt Rand
mache Ton 150 Hz für 300 ms
setze Sprite ball auf x 110 y 100
warte 20 ms

Übungsaufgaben

Aufgabe 1 ⭐ **Rotes Licht** Ändere das Hai-Programm aus 13.1: Statt eines Tons soll die rote LED 300 Millisekunden lang leuchten, wenn der Hai den Fisch erwischt.

Aufgabe 2 ⭐ **Wüste statt Wiese** Male in der Rennstrecke aus 13.3 die Wiese **gelb** (beide grünen Canvas-Blöcke). „berührt Farbe grün“ bleibt, wie es ist. Was passiert, wenn das Auto von der Bahn fährt? Warum? Repariere das Programm.

Aufgabe 3 ⭐ **Pling!** Der Ball aus 13.2 soll doppelt so schnell fliegen. Jedes Mal, wenn er am Rand abprallt, soll ein kurzer, hoher Ton erklingen.

Aufgabe 4 ⭐⭐ **Ziellinie** Male eine weiße Ziellinie in die Bahn (x = 20, y = 175, 50 breit, 4 hoch). Das Rennen läuft, **solange das Auto nicht auf Weiß steht**. Danach verschwindet das Auto, und „Ziel!“ erscheint mit der Zeit in Sekunden.

Aufgabe 5 ⭐⭐ **Ausgewichen!** Im Hai-Spiel bekommst du einen Punkt für jeden Hai, der links aus dem Bild schwimmt, ohne dich zu erwischen. Bei 10 Punkten verschwinden Fisch und Hai, und „Gewonnen!“ erscheint.

Aufgabe 6 ⭐⭐⭐ **Abprallen in alle Richtungen** Der Ball soll schräg fliegen (`dx` = 3, `dy` = 2) und an **allen vier** Rändern abprallen – nur mit „berührt Rand“ und „x/y von Sprite“.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Kollisions-Blöcke sind Frage-Blöcke: Sie antworten mit **wahr** oder **falsch**. Du steckst sie in „falls“ oder „solange“.
- Die Frage muss **in der Schleife** stehen, damit sie immer wieder gestellt wird.
- **berührt Sprite**: Überlappen sich die Rechtecke? **berührt Rand**: Ragt das Sprite aus dem Bildschirm? **berührt Farbe**: Liegt ein sichtbares Pixel auf genau dieser Farbe des Hintergrunds?
- Die Blöcke prüfen nur den Moment, in dem sie an der Reihe sind. Wände müssen dicker sein als der größte Sprung.

KAPITEL 14

Funktionen

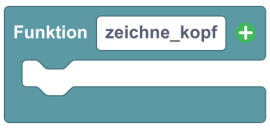
Worum geht's? Deine Programme werden länger. Irgendwann findest du dich unter „Beim Start“ kaum noch zurecht. Mit **Funktionen** teilst du ein Programm in kleine Stücke und gibst jedem Stück einen Namen. Eine Funktion kannst du so oft benutzen, wie du willst – mit anderen Werten oder mit einem Ergebnis, das sie dir zurückgibt.

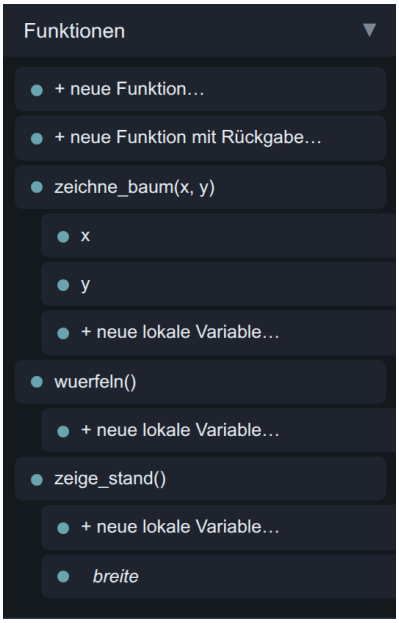
14.1 Eigene Funktionen: Programme aufteilen und benennen

Eine **Funktion** ist ein Stück Programm mit einem Namen. Du baust sie einmal und kannst sie dann überall mit ihrem Namen starten. Das nennt man **aufrufen**.

So legst du eine Funktion an:

1. Öffne die Kategorie **Funktionen** und klicke auf **„+ neue Funktion…“**.
2. Gib einen Namen ein, zum Beispiel `zeichne_kopf`, und bestätige.
3. Auf der Arbeitsfläche erscheint ein neuer Block:

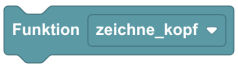




Die Kategorie Funktionen mit drei Funktionen

In diesen Block hängst du die Blöcke, die die Funktion ausführen soll. Der Funktions-Block hängt **nicht** unter „Beim Start“. Er liegt frei auf der Arbeitsfläche, genau wie „Beim Start“ selbst.

Sobald es die Funktion gibt, steht ihr Name in der Kategorie **Funktionen**, zum Beispiel `zeichne_kopf()`. Ein Klick darauf holt den **Aufruf-Block**:



Den Aufruf-Block hängst du dorthin, wo die Funktion laufen soll – zum Beispiel unter „Beim Start“. Erreicht das Programm diesen Block, springt es in die Funktion, führt ihre Blöcke aus und macht danach unter dem Aufruf weiter.

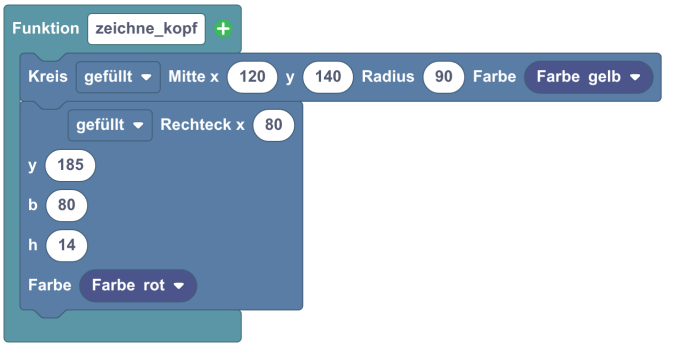
Wichtig: Eine Funktion, die nie aufgerufen wird, tut nichts. Der Funktions-Block allein ist nur der „Bauplan“.

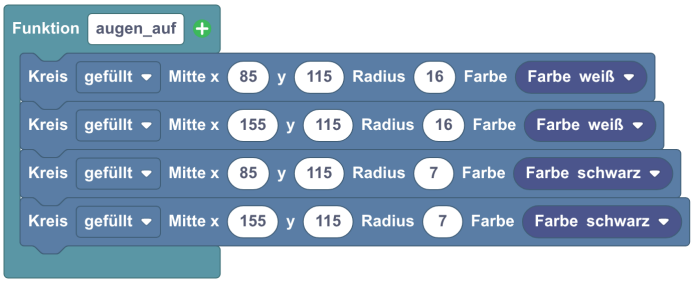
Beispiel: Ein Gesicht, das blinzelt. Das Gesicht besteht aus drei Funktionen: `zeichne_kopf` malt den gelben Kopf mit Mund, `augen_auf` malt offene Augen, `augen_zu` malt geschlossene Augen. Unter „Beim Start“ steht nur noch, **was** passiert – nicht mehr, **wie** es gemacht wird:

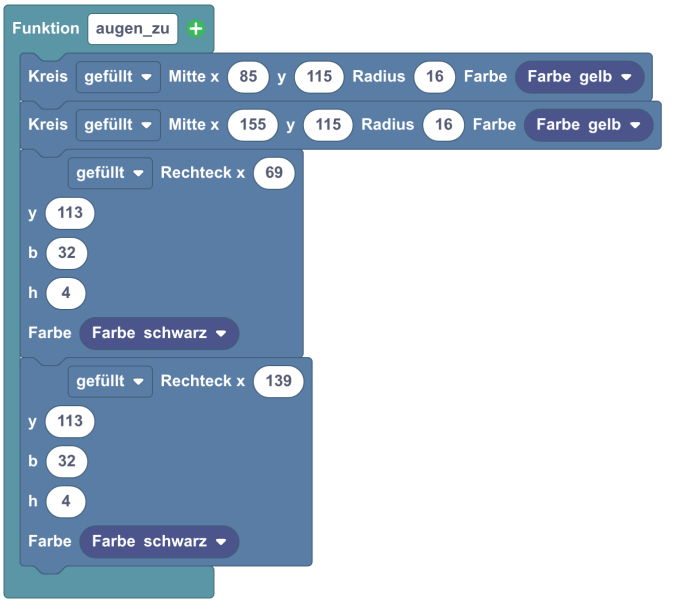


Zwei Sekunden offen, kurz zu

Das Hauptprogramm







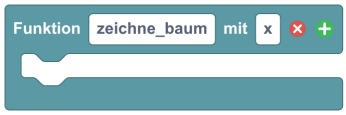
Das Hauptprogramm liest sich fast wie ein Satz: „Zeichne den Kopf. Dann immer wieder: Augen auf, warten, Augen zu, warten." Das ist der größte Vorteil von Funktionen: Ein gut gewählter Name erklärt, was die Blöcke darin tun.

> **Tipp: Gute Namen.** Nimm ein Tunwort, das sagt, was die Funktion macht: `zeichne_kopf`, `spiele_melodie`, `pruefe_treffer`. Namen dürfen nur Buchstaben, Ziffern und den Unterstrich `_` enthalten. Leerzeichen, Umlaute und andere Zeichen ersetzt PBC+ Studio durch `_` – aus „zeichne größe" wird `zeichne_gr__e`. Schreibe deshalb gleich `groesse`.

14.2 Funktionen mit Parametern

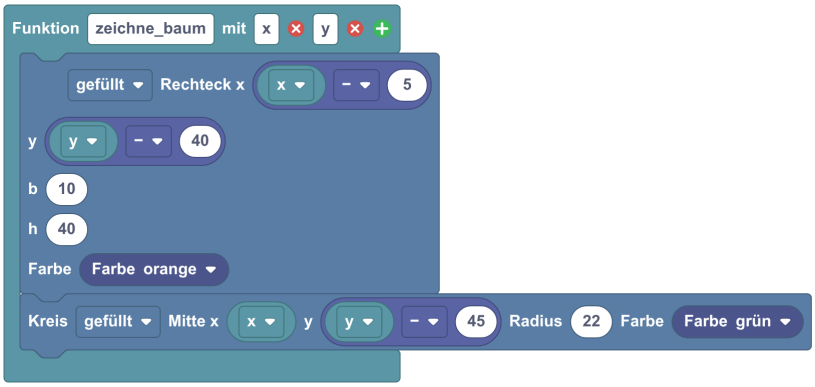
Du möchtest drei Bäume malen – an drei verschiedenen Stellen. Drei Funktionen `baum_links`, `baum_mitte`, `baum_rechts` wären umständlich. Besser: **eine** Funktion `zeichne_baum`, der du beim Aufruf sagst, **wo** der Baum stehen soll. Solche Werte heißen **Parameter**.

So bekommt eine Funktion einen Parameter:



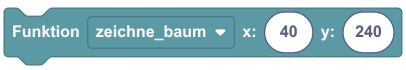
1. Klicke im Funktions-Block auf das grüne **+**. Ein Parameter mit dem Namen `x` erscheint.
2. Klicke in das Feld `x`, wenn du ihn umbenennen möchtest.
3. Ein Klick auf das rote **×** entfernt den Parameter wieder.

Jetzt stehen in der Kategorie **Funktionen** unter `zeichne_baum(x, y)` die Parameter `x` und `y` – etwas eingerückt. Ziehe sie in die Blöcke **innerhalb** der Funktion. Dort stehen sie für die Zahl, die beim Aufruf mitgegeben wird:

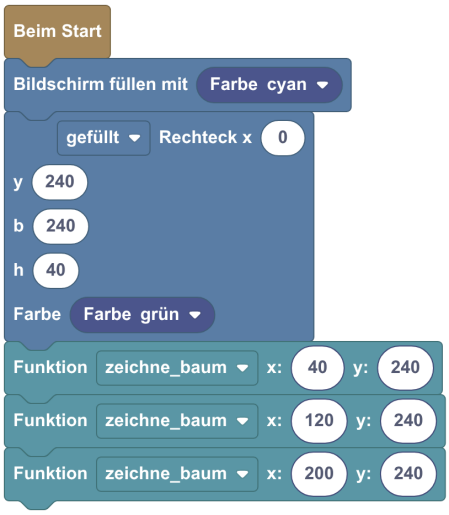


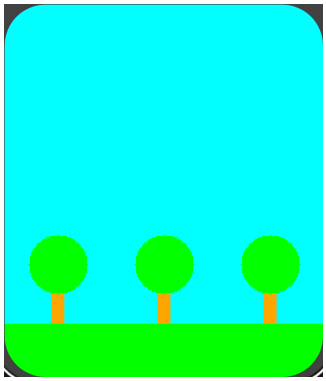
Der Stamm beginnt 40 Pixel über dem Boden, die Krone ist ein Kreis darüber. (verkleinert)

Der Aufruf-Block hat jetzt für jeden Parameter eine Lücke. Lege dort Zahlen hinein (Kategorie **Mathe**, Block „Zahl"):

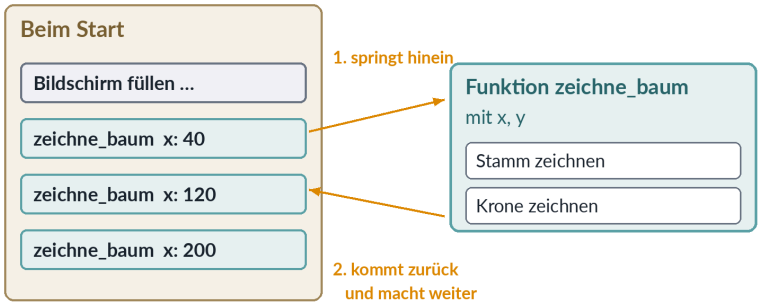


Beim Aufruf bekommt `x` den Wert 40 und `y` den Wert 240. Beim nächsten Aufruf können es ganz andere Werte sein:





Dreimal dieselbe Funktion



Bei jedem Aufruf springt das Programm in die Funktion und danach zurück.

> **Wichtig:** Ein Parameter gilt nur **innerhalb seiner Funktion**. Unter „Beim Start“ oder in einer anderen Funktion gibt es ihn nicht. Und jede Lücke im Aufruf muss gefüllt sein – sonst bekommt der Parameter keinen Wert, und meistens hält das Programm mit einer Fehlermeldung an.

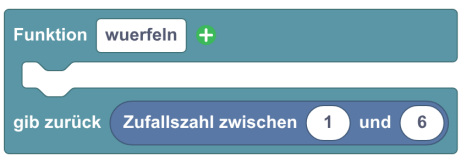
14.3 Funktionen mit Rückgabewert

Manche Funktionen sollen nicht etwas **tun**, sondern etwas **ausrechnen** und das Ergebnis zurückgeben. Dafür gibt es in der Kategorie **Funktionen** den Knopf **„+ neue Funktion mit Rückgabe…“**. Ihr Block hat unten eine zusätzliche Lücke **„gib zurück“**:

Was in dieser Lücke steht, ist das Ergebnis der Funktion. Die Funktion `wuerfeln` (rechts) gibt eine Zufallszahl von 1 bis 6 zurück:

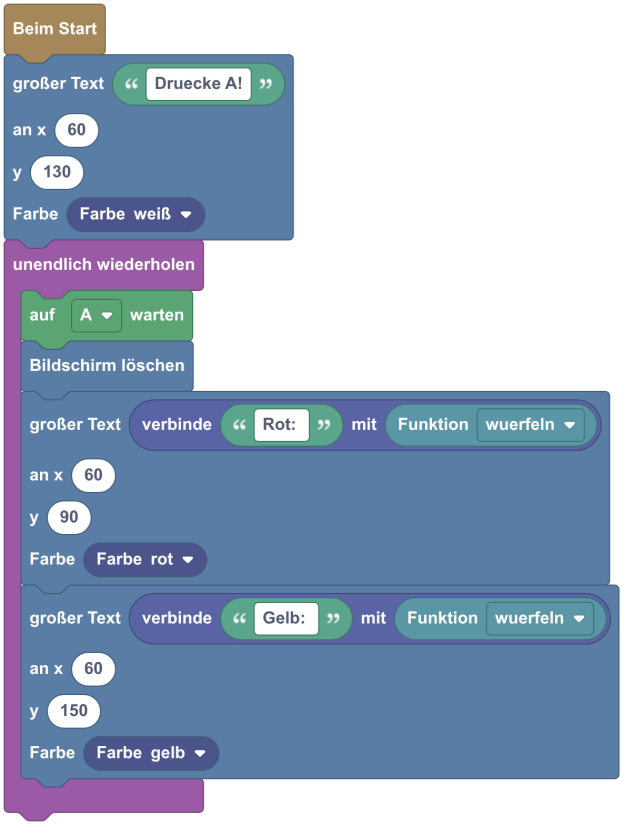


neu angelegt



fertig

Der Aufruf einer solchen Funktion ist ein **runder Wert-Block**. Du steckst ihn überall dorthin, wo ein Wert gebraucht wird – wie eine Variable oder eine Zahl. Hier zweimal in einen „verbinde“-Block:





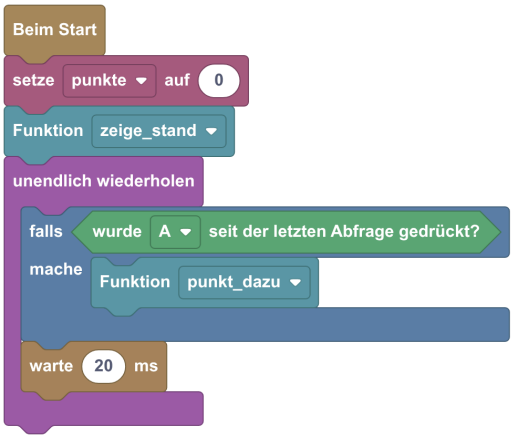
Nach einem Druck auf A

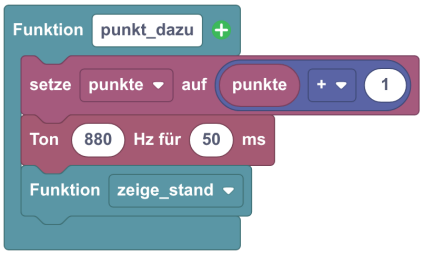
Jeder Aufruf führt die Funktion neu aus. Deshalb bekommen Rot und Gelb **eigene** Würfe – manchmal gleich, meistens verschieden.

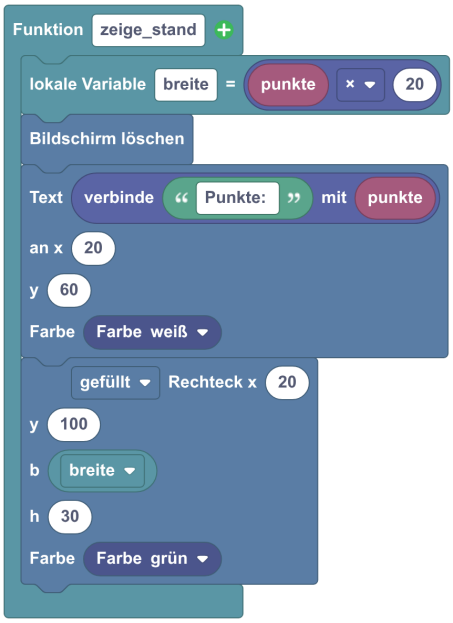
14.4 Lokale und gemeinsame Variablen

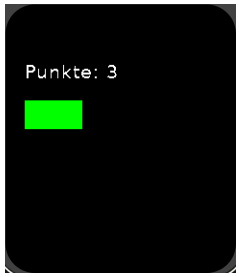
Gemeinsame Variablen

Variablen aus der Kategorie **Variablen** sind **gemeinsame** Variablen. Es gibt sie nur einmal im ganzen Programm. „Beim Start“ und alle Funktionen lesen und verändern dieselbe Variable. Im folgenden Programm zählt die Variable `punkte`, wie oft du A gedrückt hast:









Nach dreimal A

`punkt_dazu` erhöht `punkte` um 1. Danach zeigt `zeige_stand` den neuen Wert an. Beide Funktionen benutzen dieselbe Variable `punkte`, die „Beim Start“ auf 0 gesetzt hat. Dafür musst du nichts Besonderes tun – PBC+ Studio sorgt selbst dafür.

Lokale Variablen

Die Funktion `zeige_stand` braucht eine Hilfszahl: die Breite des Balkens (`punkte` × 20). Diese Zahl braucht sonst niemand. Dafür gibt es **lokale Variablen**. Unter jeder Funktion steht in der Kategorie **Funktionen** der Knopf **„+ neue lokale Variable…“**. Er legt diesen Block ganz oben in die Funktion:



Den Wert der lokalen Variable holst du mit ihrem Namen, der jetzt (schräg geschrieben) unter der Funktion in der Kategorie steht.

Eine lokale Variable

- gibt es **nur in ihrer Funktion** – andere Funktionen und „Beim Start“ sehen sie nicht,

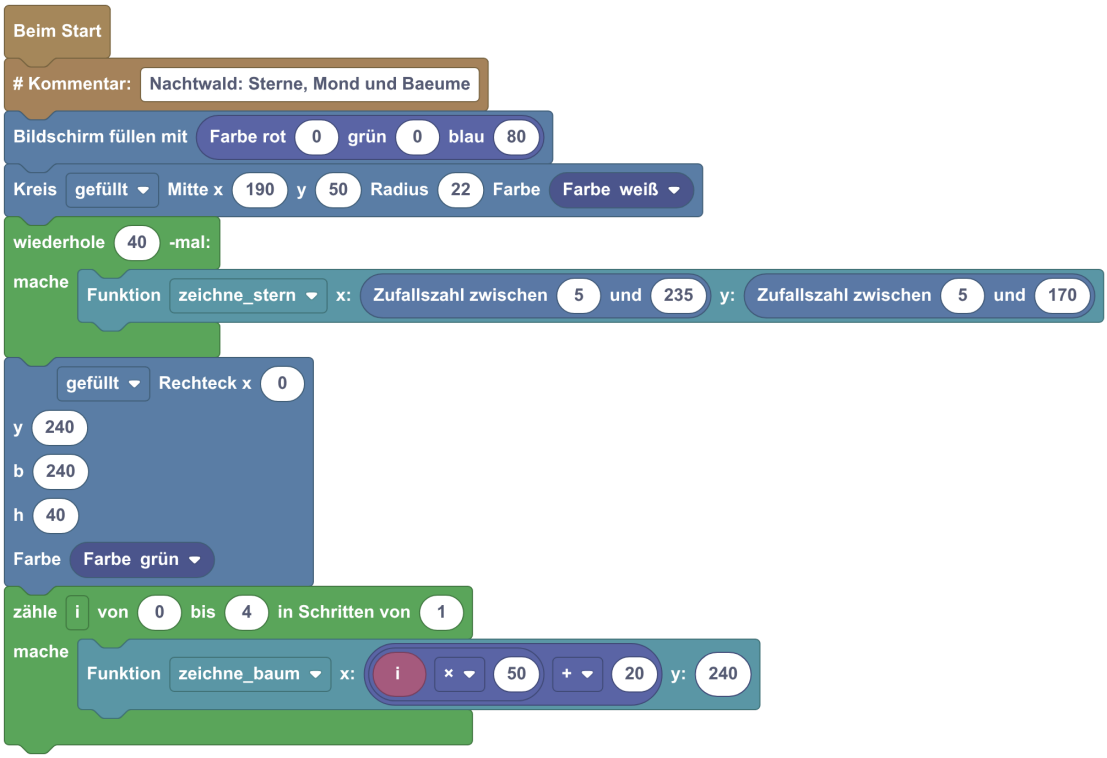
- entsteht bei **jedem Aufruf neu** und ist nach dem Ende der Funktion wieder weg.

Willst du den Wert einer lokalen Variable später in der Funktion ändern, nimm einen zweiten Block „lokale Variable … =“ mit demselben Namen (Rechtsklick auf den Block → **Kopieren**).

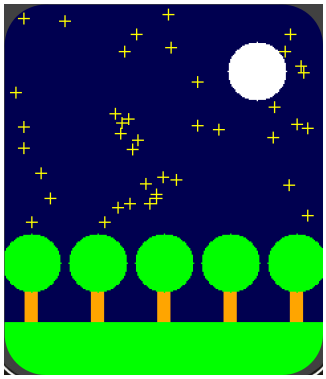
> **Tipp: Was nehme ich wann?**
>
> - Soll sich das ganze Programm den Wert merken, zum Beispiel Punkte, Leben oder eine Position? → **gemeinsame Variable** (Kategorie **Variablen**).
> - Braucht nur eine Funktion den Wert kurz zum Rechnen? → **lokale Variable**.
> - Gib Parametern und lokalen Variablen **andere Namen** als deinen gemeinsamen Variablen.

Ausprobieren: Der Nachtwald

Unter einem dunkelblauen Himmel funkeln 40 Sterne, darunter steht ein Wald aus fünf Bäumen. Die Funktion `zeichne_stern` malt ein kleines gelbes Kreuz. Sie wird 40-mal aufgerufen, jedes Mal mit zufälligen Werten für `x` und `y`. Die Bäume malt `zeichne_baum` aus 14.2 – in einer Zählschleife, die die Bäume im Abstand von 50 Pixeln verteilt.







Jedes Mal ein anderer Sternenhimmel

Dazu gehört noch die Funktion `zeichne_baum` aus 14.2. Das ganze Programm findest du als Datei `k14_nachtwald.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Zwinkern** Schreibe für das Gesicht aus 14.1 eine Funktion `zwinkern`, die nur das linke Auge schließt. Rufe sie in der Schleife statt `augen_zu` auf.

Aufgabe 2 ⭐ **Würfel mit vielen Seiten** Gib der Funktion `wuerfeln` aus 14.3 einen Parameter `seiten`. Die Zufallszahl soll von 1 bis `seiten` gehen. Würfle damit einmal mit einem 6er- und einmal mit einem 20er-Würfel.

Aufgabe 3 ⭐⭐ **Herbstwald** Gib `zeichne_baum` einen dritten Parameter `farbe` für die Farbe der Krone. Male drei Bäume in Orange, Rot und Gelb. *Tipp:* In die Lücke für `farbe` passt der Block „Farbe nach Namen“ aus der Kategorie **Display**.

Aufgabe 4 ⭐⭐ **Ein Wald auf Bestellung** Schreibe eine Funktion `zeichne_wald`, die mit einer Zählschleife fünf Bäume malt. Unter „Beim Start“ steht dann nur noch: Himmel und Wiese malen, `zeichne_wald`. Was passiert, wenn du den Aufruf von `zeichne_wald` weglässt?

Aufgabe 5 ⭐⭐ **Roter Balken** Ändere `zeige_stand` aus 14.4: Lege eine lokale Variable `farbe` an, die am Anfang Grün ist. Bei mehr als 5 Punkten bekommt sie den Wert Rot. Male den Balken in der Farbe `farbe`.

Aufgabe 6 ⭐⭐⭐ Gerade oder ungerade? Schreibe eine Funktion mit Rückgabe `ist_gerade` mit dem Parameter `zahl`. Sie gibt **wahr** zurück, wenn `zahl` gerade ist, sonst **falsch**. Zeige die Zahlen 1 bis 10 untereinander an: gerade Zahlen grün, ungerade rot. *Tipp:* Eine Zahl ist gerade, wenn beim Teilen durch 2 der Rest 0 bleibt (Block „Rechnen“ mit „Rest“).

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Eine **Funktion** ist ein Programmstück mit Namen. Du legst sie in der Kategorie **Funktionen** mit „+ neue Funktion…“ an.
- Eine Funktion läuft nur, wenn sie **aufgerufen** wird. Den Aufruf-Block holst du über ihren Namen in der Kategorie.
- **Parameter** (grünes +) sind Werte, die beim Aufruf mitgegeben werden. So arbeitet dieselbe Funktion mit verschiedenen Zahlen.
- Eine **Funktion mit Rückgabe** rechnet etwas aus. Ihr Aufruf ist ein Wert-Block, der in jede passende Lücke passt.
- **Gemeinsame Variablen** (Kategorie Variablen) gelten im ganzen Programm. **Lokale Variablen** und Parameter gelten nur in ihrer Funktion.
- Gute Namen machen ein Programm lesbar wie einen Satz.

KAPITEL 15

Listen

Worum geht's? Eine Variable merkt sich **einen** Wert. Für eine Bestenliste, eine Melodie oder die Positionen von zehn Sternen bräuchtest du sehr viele Variablen. Eine **Liste** merkt sich beliebig viele Werte unter einem einzigen Namen – in einer festen Reihenfolge.

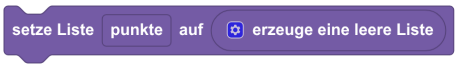
15.1 Eine Liste anlegen – leer oder mit Werten

Listen haben eine eigene Kategorie: **Listen**. Klicke auf **„+ neue Liste…"** und gib einen Namen ein, zum Beispiel `punkte`. Wie bei Variablen gilt: keine Umlaute, kein ß, keine Leerzeichen.



Die Kategorie Listen

Die neue Liste steht danach in der Kategorie, darunter alle Listen-Blöcke. Jede Liste ist am Anfang **leer**. Willst du sie im Programm wieder leer machen, nimmst du den Block „setze Liste … auf leer":



Soll die Liste gleich Werte enthalten, steckst du stattdessen den Block „Liste aus Werten" hinein. Er heißt auf der Arbeitsfläche „erzeuge Liste mit". Mit dem Zahnrad-Symbol stellst du ein, wie viele Lücken er hat:



In die Lücken passen Zahlen, Texte oder Farben – sogar Variablen.

> **Mehrere Listen:** Ein neuer Listen-Block kommt immer mit der **ersten** Liste auf die Arbeitsfläche. Brauchst du eine andere, klicke im Block auf den Namen der Liste (mit dem kleinen Pfeil ▾) und wähle die richtige aus.

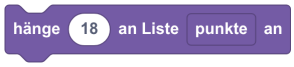
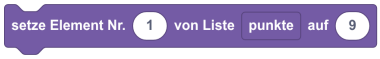
15.2 Anhängen, lesen, ändern – gezählt wird ab 0

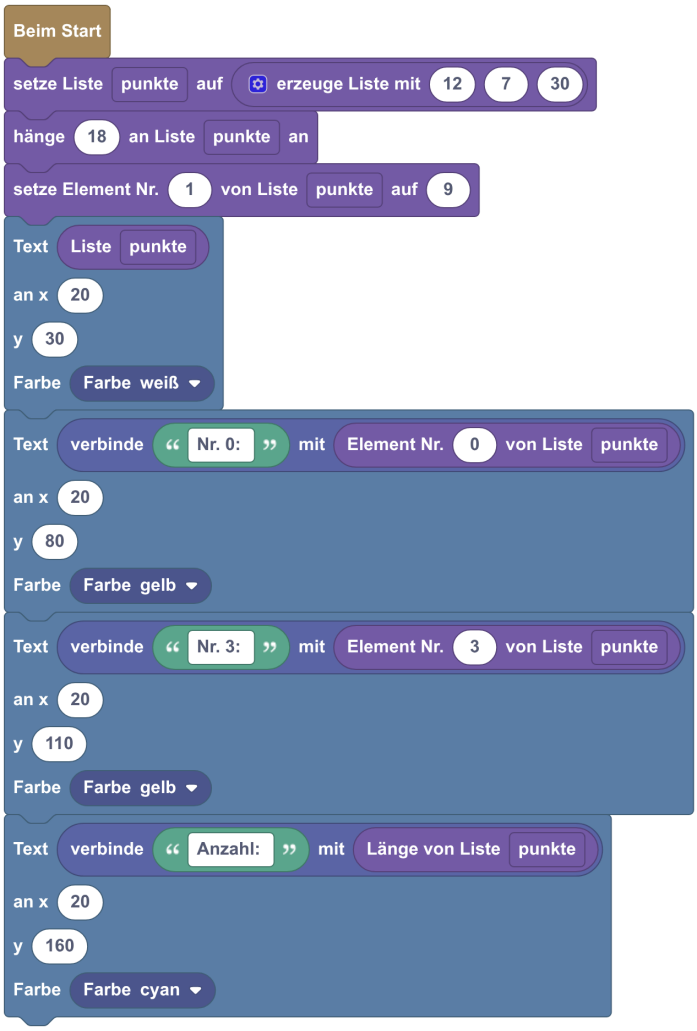
Stell dir eine Liste als Reihe von Fächern vor. Jedes Fach hat eine **Nummer**. Das erste Fach hat die Nummer **0**, nicht 1!

Liste punkte

12	9	30	18
Nr. 0	Nr. 1	Nr. 2	Nr. 3

Element Nr. 2 von Liste punkte → 30
Länge von Liste punkte → 4 (die letzte Nr. ist 3)

Block	Was passiert?
	Ein neues Fach kommt **hinten** dazu. Die Liste wird länger.
	„Element Nr. 0 von Liste punkte“ **liest** den Wert im Fach 0 – hier in einem Text-Block.
	Der Wert im Fach 1 wird **ersetzt**. Die Länge bleibt gleich.





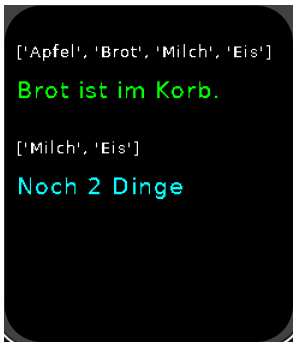
Die ganze Liste, Fach 0, Fach 3 und die Länge

Was passiert Schritt für Schritt? Am Anfang ist die Liste [12, 7, 30]. Die 18 kommt hinten dazu: [12, 7, 30, 18]. Dann wird Fach 1 (die 7) durch 9 ersetzt: [12, 9, 30, 18]. Der Block „Liste punkte“ (ganz unten in der Kategorie) zeigt die ganze Liste auf einmal an – mit eckigen Klammern.

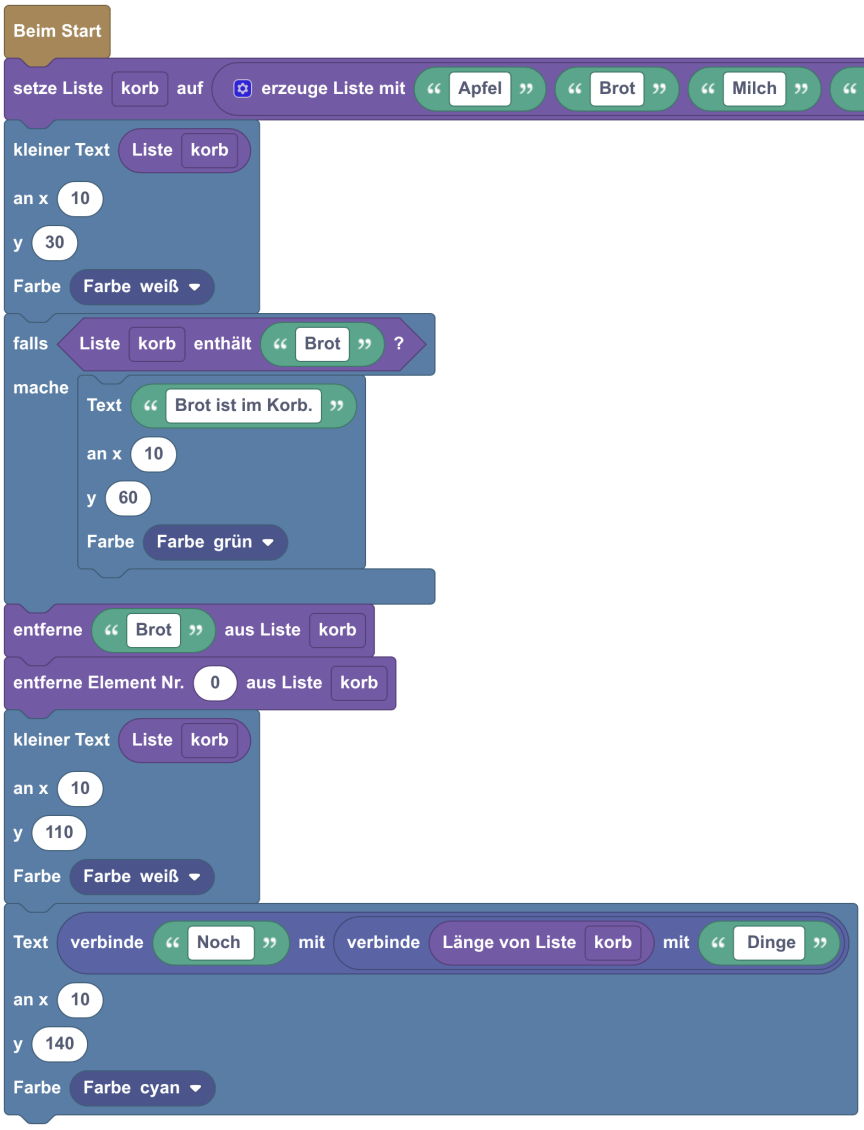
> **Wichtig: Es gibt kein Fach mit der Nummer „Länge“.** Eine Liste mit 4 Elementen hat die Fächer 0, 1, 2 und 3. Fragst du nach Element Nr. 4, hält das Programm an und meldet **„Listen-Position außerhalb“**.

15.3 Länge, enthält, entfernen

Block	Antwort / Wirkung
Länge von Liste korb	Wie viele Elemente hat die Liste?
Liste korb enthält " Brot " ?	Ist „Brot“ in der Liste? **wahr** oder **falsch**
entferne Element Nr. 0 aus Liste korb	entfernt das Fach mit dieser Nummer
entferne " Brot " aus Liste korb	sucht „Brot“ und entfernt es



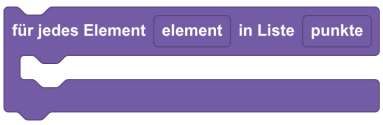
Ein Einkaufskorb als Liste von Texten:



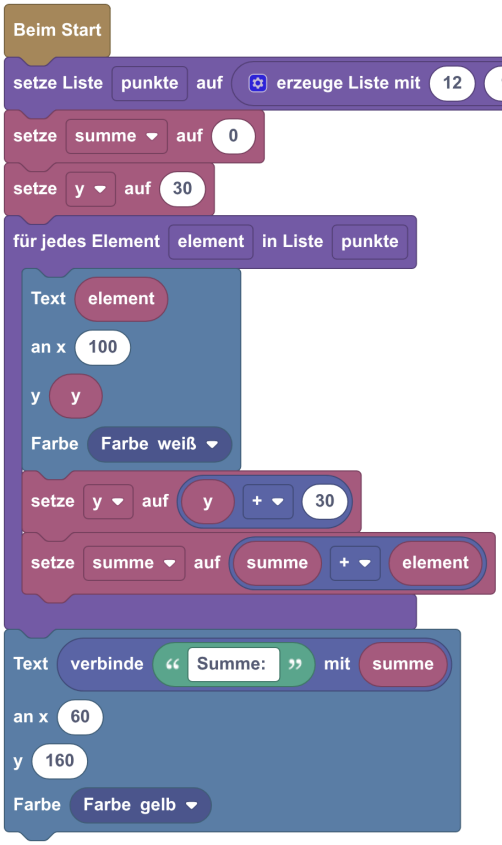
Nach dem Entfernen rücken die hinteren Elemente nach vorn. Erst fliegt „Brot“ raus: [Apfel, Milch, Eis]. Dann Fach 0, also „Apfel“: [Milch, Eis]. Steht der Wert gar nicht in der Liste, macht „entferne … aus Liste“ einfach nichts.

15.4 „für jedes Element in Liste“

Oft willst du mit **jedem** Wert einer Liste etwas tun: alle anzeigen, alle zusammenzählen. Dafür gibt es diese Schleife:



Sie läuft so oft, wie die Liste Elemente hat. Beim ersten Durchlauf steckt in der Variable `element` der Wert aus Fach 0, beim zweiten der aus Fach 1 und so weiter. Du musst dich um keine Nummern kümmern.

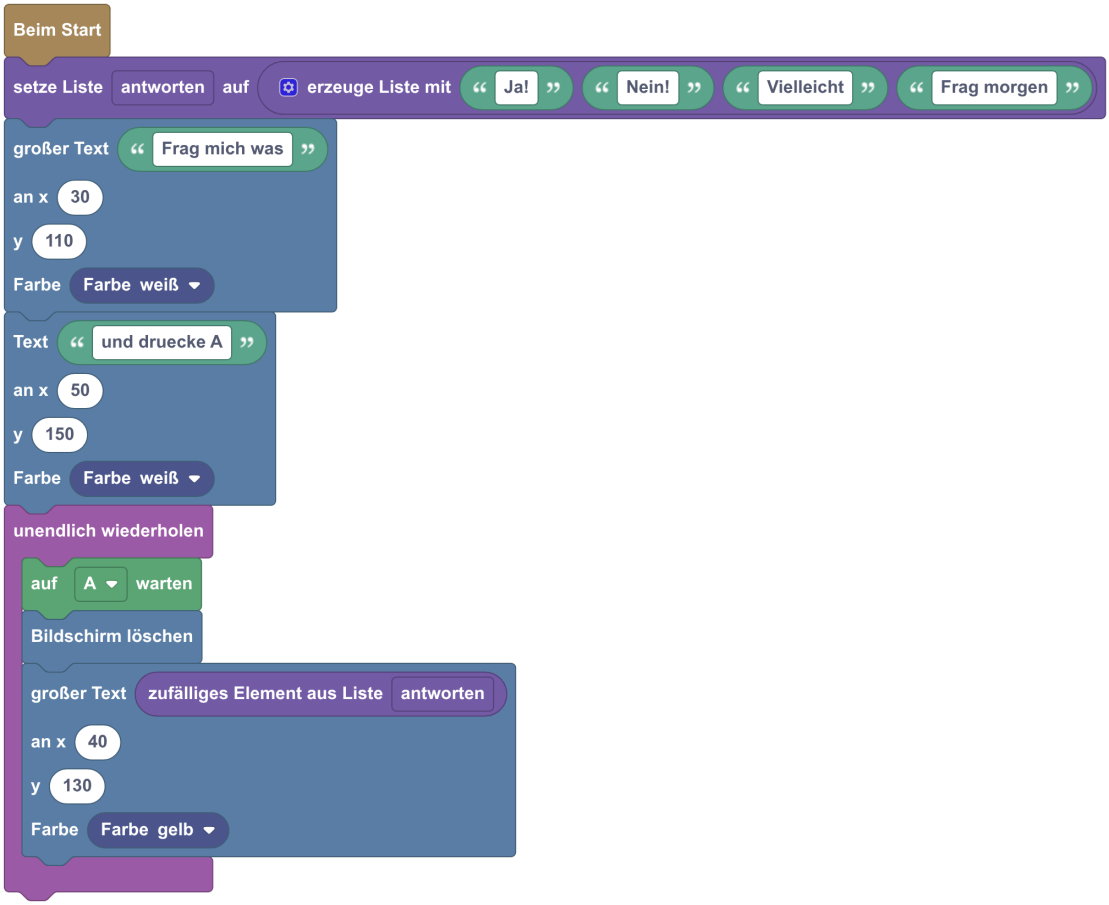




Jeder Wert eine Zeile tiefer

15.5 Zufälliges Element

„zufälliges Element aus Liste“ wählt per Zufall einen der Werte aus. Damit baust du ein Orakel, das auf jede Frage antwortet:





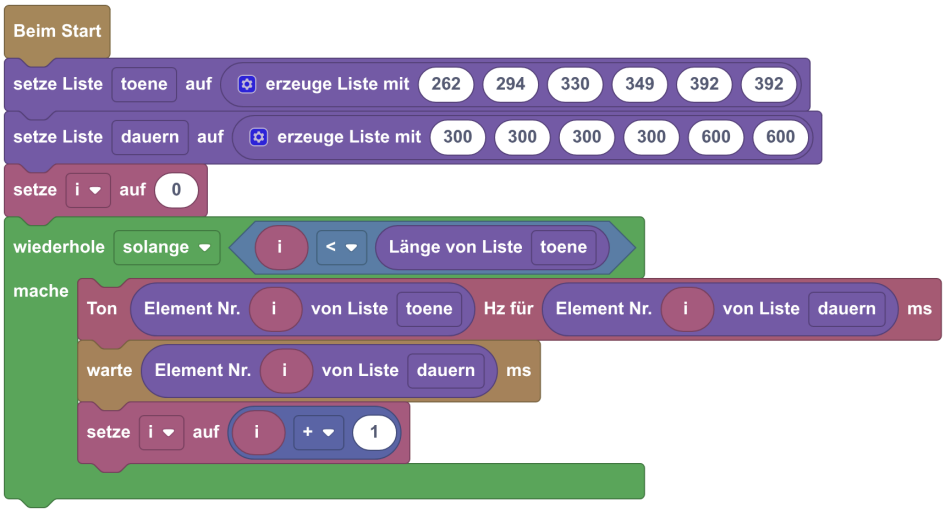
Nach einem Druck auf A

15.6 Listen in Spielen: Positionen, Punktestände, Abfolgen

Listen stecken in fast jedem Spiel:

- **Abfolgen:** eine Melodie, ein Level nach dem anderen, die Farbfolge bei „Simon sagt“.
- **Punktestände:** alle Ergebnisse für eine Bestenliste.
- **Positionen:** Für viele Dinge auf dem Bildschirm nimmst du **zwei** Listen – eine für alle x-Werte, eine für alle y-Werte. Fach 0 der beiden Listen gehört zum ersten Ding, Fach 1 zum zweiten …

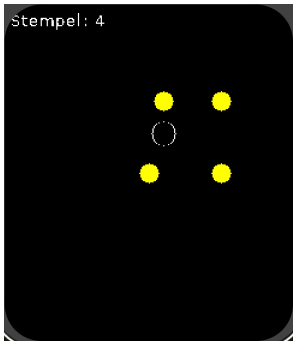
Zwei Listen mit gleichen Nummern hat auch diese Melodie: `toene` und `dauern`. Die Variable `i` zählt die Fachnummer von 0 hoch, **solange** sie kleiner als die Länge ist:



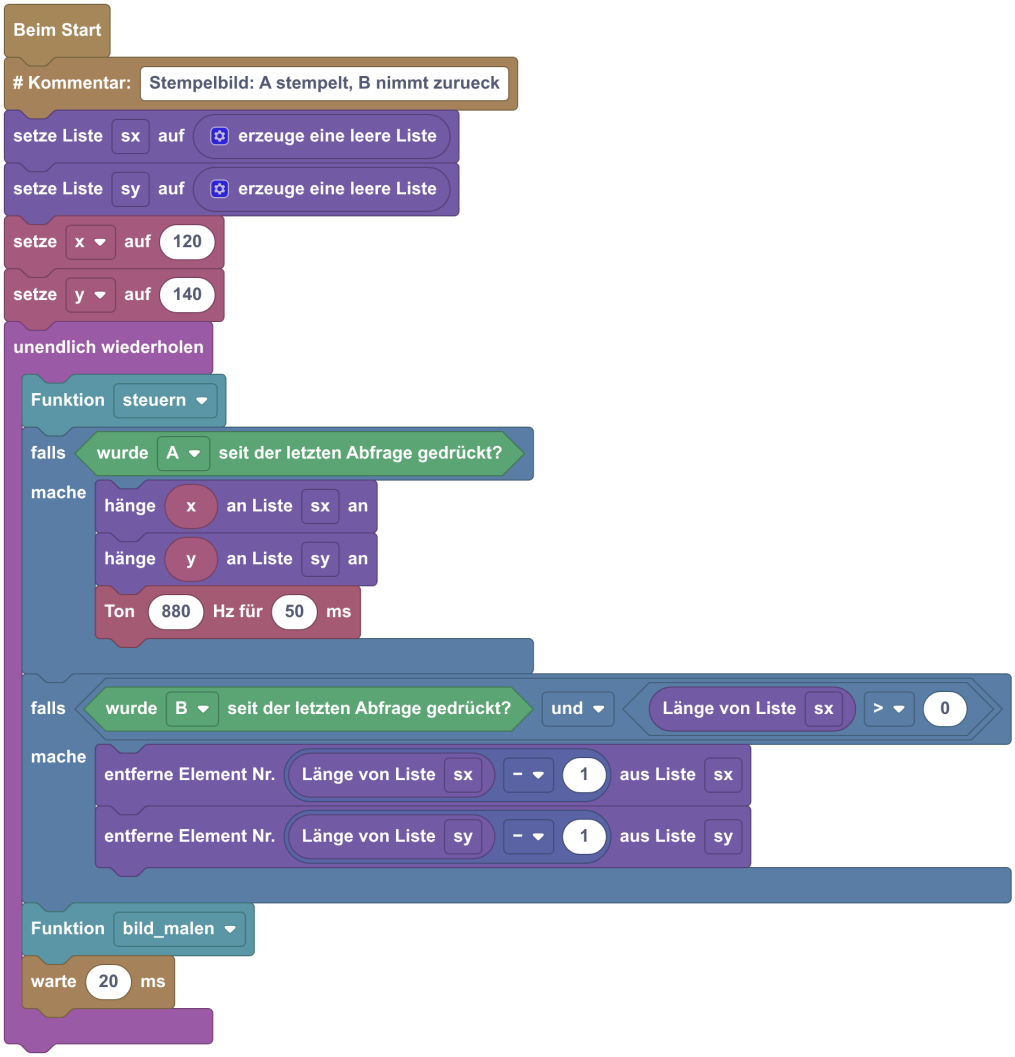
Der Ton-Block wartet nicht, bis der Ton fertig ist. Deshalb steht danach „warte“ mit derselben Dauer.

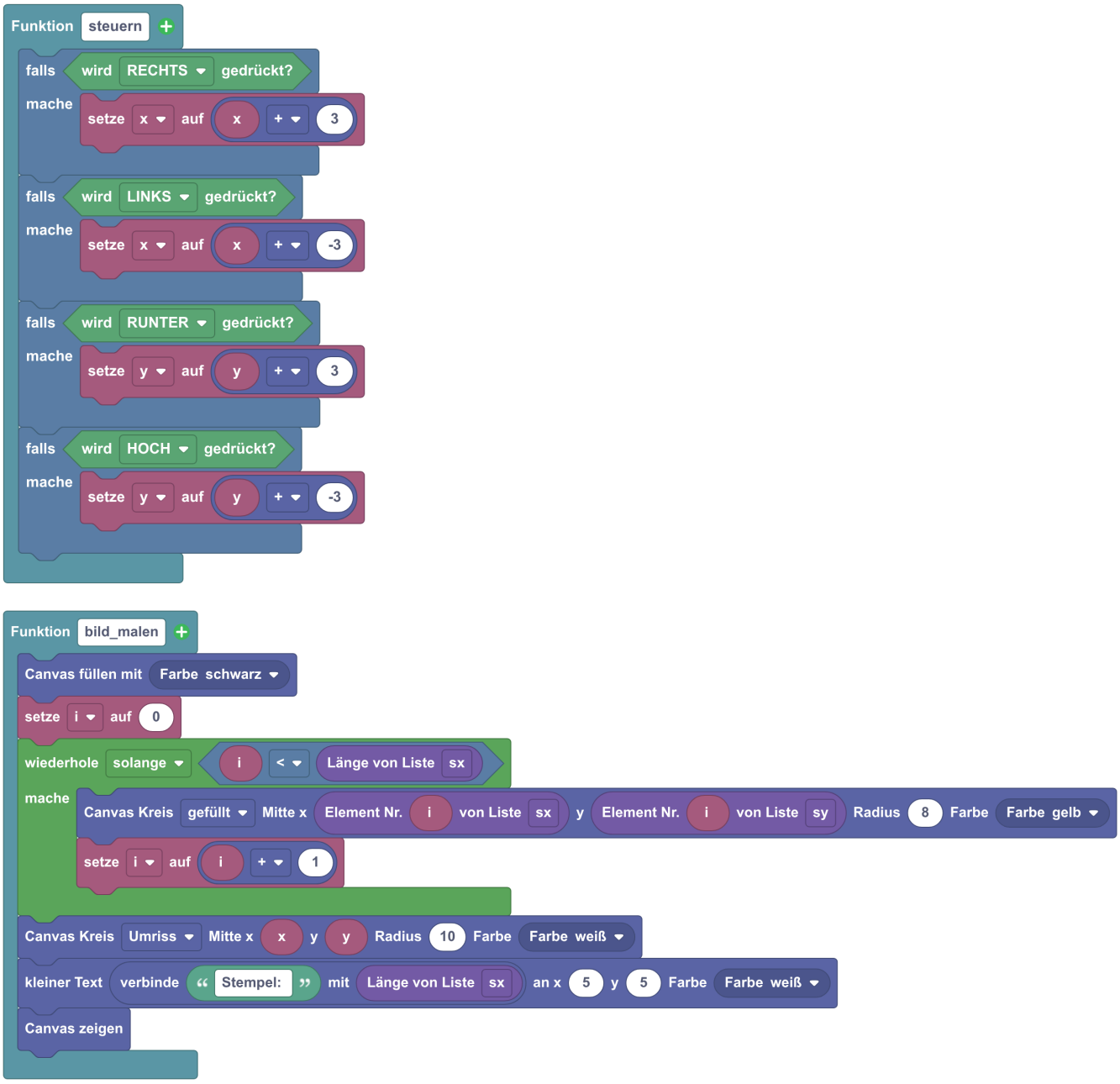
Ausprobieren: Das Stempelbild

Mit dem Joystick bewegst du einen weißen Ring. Taste **A** setzt an dieser Stelle einen gelben Stempel, Taste **B** nimmt den letzten Stempel wieder weg. Die Positionen der Stempel stehen in den Listen `sx` und `sy`. In jedem Schleifendurchlauf malt `bild_malen` alle Stempel neu auf den Canvas (Kapitel 11) – der letzte Stempel hat die Nummer Länge − 1.



Vier Stempel und der Cursor


Beim Start
Kommentar: Stempelbild: A stempelt, B nimmt zurueck
setze Liste sx auf erzeuge eine leere Liste
setze Liste sy auf erzeuge eine leere Liste
setze x auf 120
setze y auf 140
unendlich wiederholen
Funktion steuern
falls wurde A seit der letzten Abfrage gedrückt?
mache
hänge x an Liste sx an
hänge y an Liste sy an
Ton 880 Hz für 50 ms
falls wurde B seit der letzten Abfrage gedrückt? und Länge von Liste sx > 0
mache
entferne Element Nr. Länge von Liste sx − 1 aus Liste sx
entferne Element Nr. Länge von Liste sy − 1 aus Liste sy
Funktion bild_malen
warte 20 ms



Das Programm findest du als Datei `k15_stempel.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Wochentage** Lege eine Liste `tage` mit „Mo“, „Di“, „Mi“, „Do“, „Fr“, „Sa“, „So“ an. Überlege zuerst: Was zeigt „Element Nr. 2 von Liste tage“? Prüfe es. Welche Nummer hat „So“?

Aufgabe 2 ⭐ **Die Melodie geht weiter** Hänge an die Melodie aus 15.6 mit „hänge … an Liste an“ noch viermal den Ton 440 (Dauer 300) und einmal 392 (Dauer 900) an. Erkennst du das Lied?

Aufgabe 3 ⭐⭐ **Glücksfarbe** Lege eine Liste `farben` mit sechs Blöcken „Farbe nach Namen“ an. Bei jedem Druck auf A wird der ganze Bildschirm mit einer zufälligen Farbe aus der Liste gefüllt.

Aufgabe 4 ⭐⭐ **Der Durchschnitt** Erweitere das Programm aus 15.4: Zeige zusätzlich den Durchschnitt an – die Summe geteilt durch die Länge der Liste.

Aufgabe 5 ⭐⭐⭐ **Der beste Wert** Finde mit „für jedes Element“ den größten Wert in der Liste `punkte`. *Tipp:* Merke dir in einer Variable `beste` zuerst Element Nr. 0. Ist ein Element größer als `beste`, wird es der neue Bestwert.

Aufgabe 6 ⭐⭐⭐ **Bunte Stempel** Jeder Stempel im Stempelbild soll eine zufällige Farbe bekommen. Du brauchst eine Liste `farben` und eine dritte Liste `sf`, die sich zu jedem Stempel die Farbe merkt.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Eine **Liste** speichert viele Werte unter einem Namen. Du legst sie in der Kategorie **Listen** mit „+ neue Liste…“ an.
- Die Fächer sind **ab 0 nummeriert**. Das letzte Element hat die Nummer **Länge − 1**.
- **hänge … an** macht die Liste länger, **setze Element Nr.** ändert ein Fach, **entferne** macht sie kürzer.
- **Liste enthält …?** antwortet mit wahr oder falsch.
- **für jedes Element** geht alle Werte der Reihe nach durch.
- Für Positionen nimmst du zwei Listen: eine für x, eine für y.

KAPITEL 16

Texte und Konsole

Worum geht's? Texte bestehen aus einzelnen Zeichen – Buchstaben, Ziffern, Leerzeichen. In diesem Kapitel nimmst du Texte auseinander, setzt sie neu zusammen und durchsuchst sie. Außerdem lernst du die **Konsole** kennen: Dort kann dein Programm Nachrichten ausgeben und Eingaben von der Tastatur einlesen.

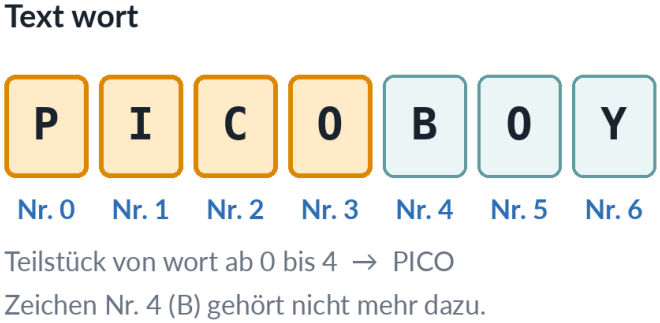
Texte merkst du dir in **Text-Variablen**: Kategorie **Variablen**, Knopf **„+ neuer Text…"**. Die Text-Blöcke dieses Kapitels findest du in der Kategorie **Text**.

> **Keine Umlaute auf dem Bildschirm!** Die Schrift des PicoBoy kennt kein ä, ö, ü und ß. Statt dieser Zeichen erscheint ein „?". Schreibe in Texten, die auf dem Bildschirm stehen sollen, „ae", „oe", „ue" und „ss": „Druecke A", „Laenge", „gross".

16.1 Texte verbinden, Länge, einzelne Zeichen, Teilstücke

Block	Ergebnis, wenn `wort` = „PICOBOY" ist
	hängt zwei Texte aneinander: „Hallo Mia" (wenn `name` „Mia" ist)
	Anzahl der Zeichen: 7
	ein einzelnes Zeichen: „P"
	ein Stück des Textes: „PICO"

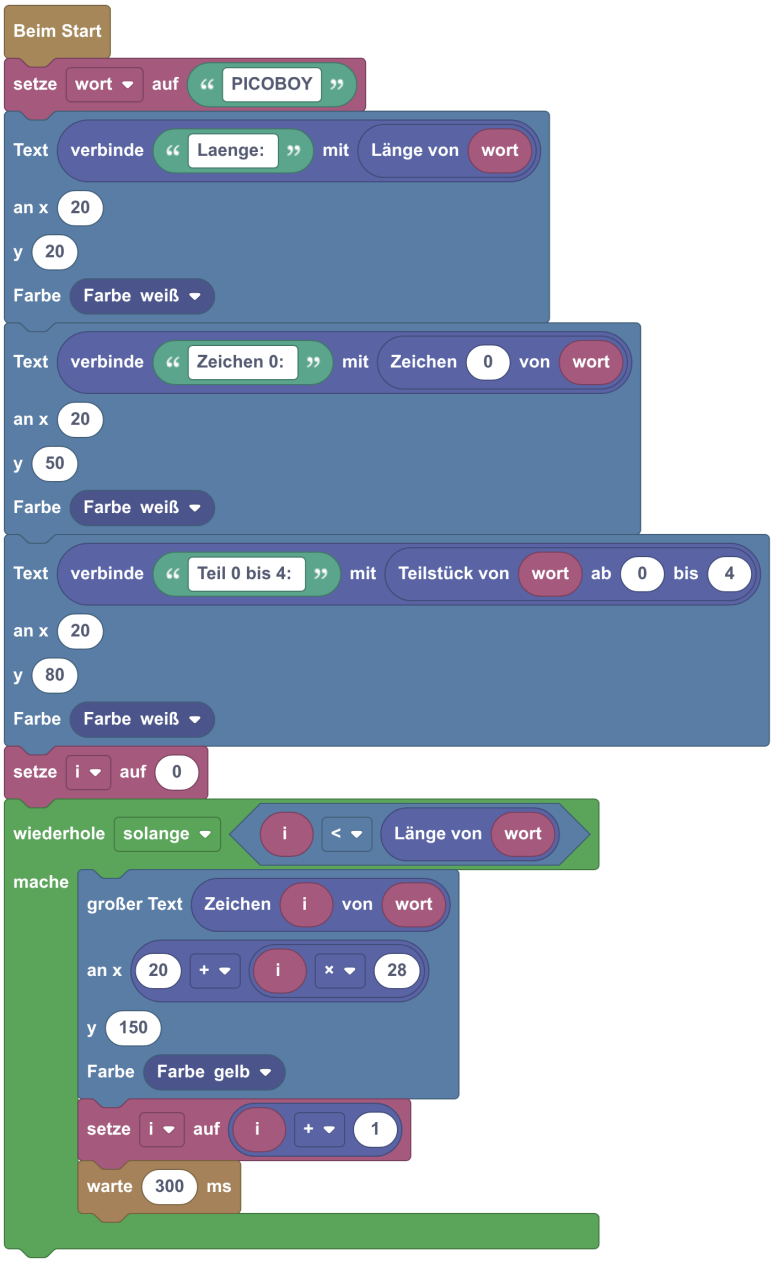
Wie bei Listen sind die Zeichen eines Textes **ab 0 nummeriert**. Beim Teilstück zählt das Zeichen bei „ab" mit, das Zeichen bei „bis" aber **nicht mehr**:

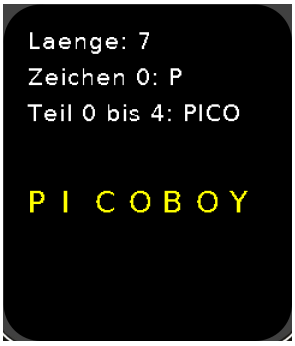


So bekommst du mit „ab 0 bis 4" genau die ersten 4 Zeichen.

> **Tipp:** In „verbinde“ darfst du auch Zahlen stecken. Aus „Punkte: “ und der Zahl 7 wird der Text „Punkte: 7“. Für mehr als zwei Teile steckst du einen „verbinde“-Block in den anderen.

Das folgende Programm zeigt oben Länge, erstes Zeichen und Teilstück an. Dann schreibt es das Wort Buchstabe für Buchstabe, jeden 28 Pixel weiter rechts. Die Variable `i` zählt dabei die Zeichen-Nummer hoch, **solange** sie kleiner als die Länge ist:





Das I ist schmal – deshalb die größere Lücke dahinter.

16.2 Groß- und Kleinschreibung, „enthält“

Block	Ergebnis bei „Mein PicoBoy“
satz in GROSSBUCHSTABEN	„MEIN PICOBOY“ – im Feld ▾ wählst du GROSSBUCHSTABEN oder kleinbuchstaben
satz enthält “ Pico ”	**wahr**, weil „Pico“ im Text vorkommt

„enthält“ achtet genau auf Groß- und Kleinschreibung: „Pico“ wird gefunden, „pico“ nicht. Willst du ohne Rücksicht auf Groß und klein suchen, verwandle den Text vorher in kleinbuchstaben und suche dann nach „pico“.



Ergebnis von `k16_gross.pbcp`

16.3 Ausgeben und Einlesen über die Konsole

Unter der Arbeitsfläche liegt das **Terminal** (Kapitel 1). Mit den Blöcken der Kategorie **Konsole** benutzt dein Programm es wie einen Chat:

Block	Was passiert?
ausgeben “ Hallo ” leere Zeile ausgeben	**ausgeben** schreibt einen Text als neue Zeile ins Terminal. **leere Zeile ausgeben** schreibt eine leere Zeile.
einlesen mit Aufforderung “ Wie heisst du? ”	schreibt die Aufforderung ins Terminal und **wartet**, bis du etwas eintippst und die **Eingabetaste** drückst. Das Ergebnis ist ein Text.
einlesen als Zahl mit Aufforderung “ Dein Tipp: ”	macht dasselbe, das Ergebnis ist aber eine Zahl. Damit kannst du rechnen.

Im Emulator klickst du dazu ins Terminal und tippst dort. Auf dem Gerät funktioniert die Konsole nur, solange der PBC+ mit PBC+ Studio verbunden ist und du das Programm mit ▶ **Testen** gestartet hast.



Tippst du „lena“ ein, steht im Terminal:

```
Wie heisst du? lena
Hallo lena
```

Auf dem Bildschirm erscheint „LENA“ in Großbuchstaben und darunter „4 Buchstaben“.

> **Wichtig:** Tippst du bei „einlesen als Zahl“ Buchstaben ein, kann das Programm daraus keine Zahl machen. Es hält an und meldet **„Eingabe ist keine Zahl“**.

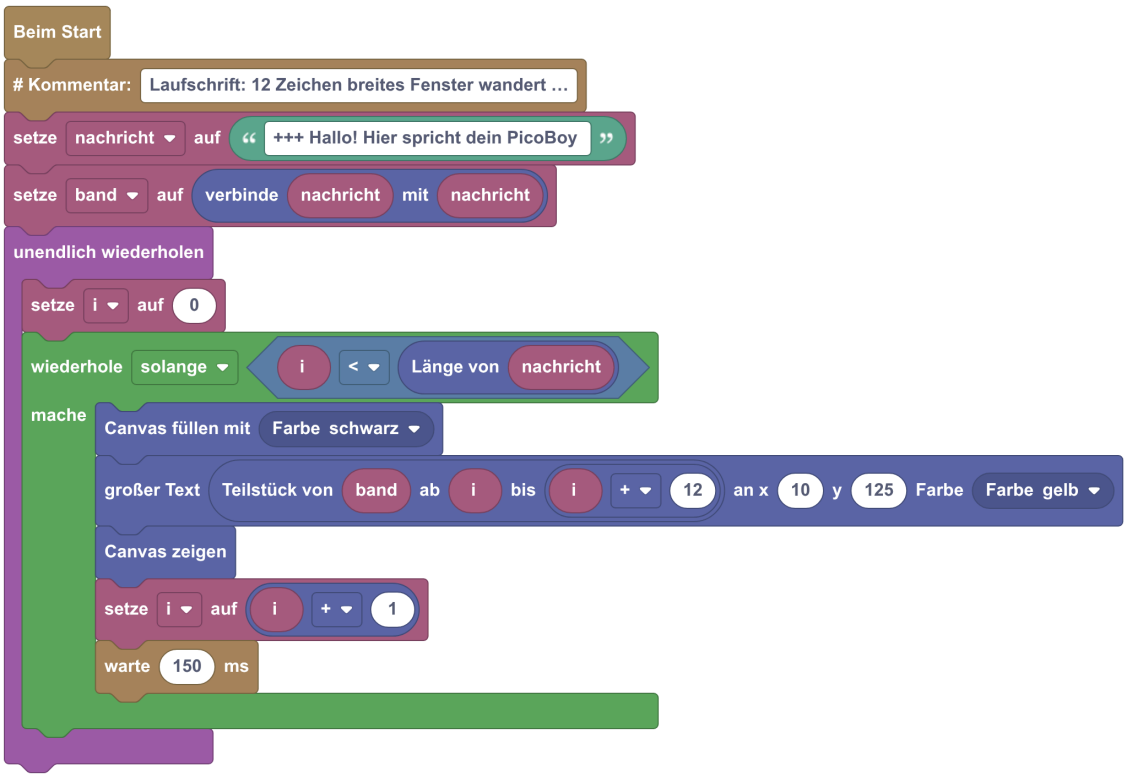
> Im Terminal funktionieren auch Umlaute, denn das Terminal ist ein Fenster deines Computers, nicht der Bildschirm des PicoBoy. Die Konsole ist außerdem praktisch, um Fehler zu suchen: Gib Variablen mit „ausgeben“ aus und sieh nach, welchen Wert sie gerade haben.

Ausprobieren: Die Laufschrift

Eine Laufschrift zeigt immer nur 12 Zeichen eines langen Textes: zuerst die Zeichen 0 bis 11, dann 1 bis 12, dann 2 bis 13 … So scheint der Text nach links zu wandern. Damit am Ende kein Loch entsteht, wird die Nachricht mit sich selbst verbunden (`band`). Gemalt wird auf dem Canvas (Kapitel 11), damit nichts flackert.



Die Schrift läuft von rechts nach links.



Das Programm findest du als Datei `k16_laufschrift.pbcp` im Ordner `buch/programme`. Schreib deine eigene Nachricht hinein!

Übungsaufgaben

Aufgabe 1 ⭐ **Dein Name** Setze im Programm aus 16.1 `wort` auf deinen Namen in Großbuchstaben. Was zeigt jetzt „Laenge“ an? Was zeigt „Teil 0 bis 4“, wenn dein Name kürzer als 4 Buchstaben ist?

Aufgabe 2 ⭐ **Initialen** Lege die Texte `vorname` und `nachname` an. Zeige groß die Anfangsbuchstaben mit Punkt an, zum Beispiel „L. S.“.

Aufgabe 3 ⭐⭐ **Rückwärts** Schreibe „PICOBOY“ rückwärts auf den Bildschirm: „YOBOCIP“. *Tipp:* Beginne mit einem leeren Text `rueckwaerts` und `i` = Länge − 1. Hänge Zeichen `i` an und zähle `i` herunter, solange `i` ≥ 0 ist.

Aufgabe 4 ⭐⭐ **Zahlenraten** Der PicoBoy denkt sich eine Zufallszahl von 1 bis 100. Du rätst über die Konsole. Nach jedem Tipp antwortet er „Zu klein!“ oder „Zu gross!“, bis du richtig liegst.

Aufgabe 5 ⭐⭐⭐ **Geheimes Passwort** Lies über die Konsole ein Passwort ein. Ist es „sesam“ – egal ob groß oder klein geschrieben –, leuchtet die grüne LED, sonst die rote.

Aufgabe 6 ⭐⭐⭐ **Vokale zählen** Zähle, wie viele Vokale (a, e, i, o, u) im Wort „Programmieren“ stecken. *Tipp:* Der Text „aeiouAEIOU“ enthält alle Vokale. Prüfe für jedes Zeichen des Wortes, ob dieser Text es enthält.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Die Zeichen eines Textes sind **ab 0 nummeriert** – wie bei Listen.
- **verbinde** hängt Texte (und Zahlen) aneinander, **Länge von** zählt die Zeichen.
- **Teilstück ab a bis b** nimmt die Zeichen a bis b − 1.
- **enthält** unterscheidet Groß- und Kleinschreibung. Mit **in GROSS / klein** machst du vorher alles gleich.
- Auf dem Bildschirm keine Umlaute – im Terminal schon.
- **ausgeben** schreibt ins Terminal, **einlesen** wartet auf eine Eingabe mit Eingabetaste.

KAPITEL 17

Mathe-Werkzeuge

Worum geht's? Plus, minus, mal und geteilt kennst du schon. In der Kategorie **Mathe** liegen noch mehr Werkzeuge: Runden, Abstände ohne Minus, das Größere von zwei Zahlen, Werte in einen Bereich zwingen oder von einem Bereich in einen anderen umrechnen, Wurzeln und Potenzen. Damit werden Spiele genauer und Programme kürzer.

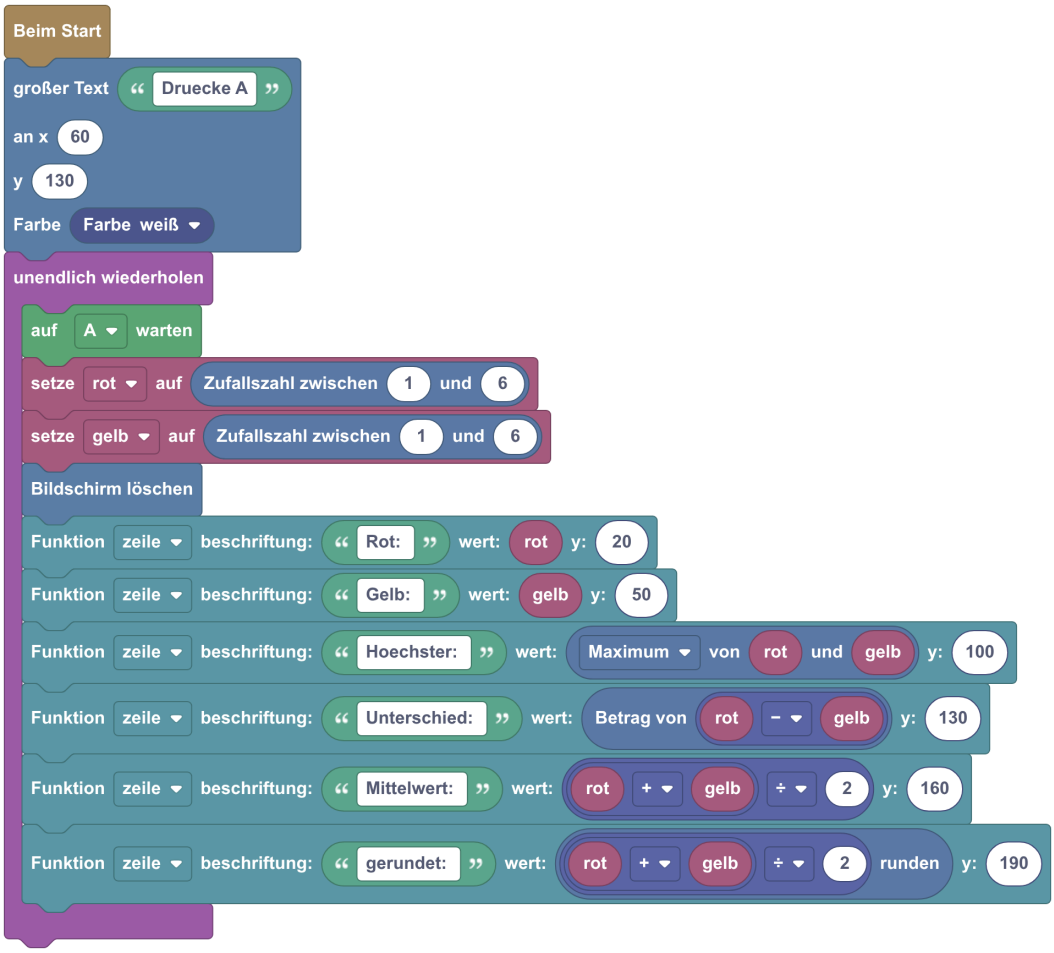
17.1 Runden, Betrag, Minimum und Maximum

Block	Was kommt heraus?	Beispiel
	die nächste ganze Zahl	7 ÷ 2 = 3.5 → **4**
	die Zahl ohne Minus	Betrag von −3 → **3**
	die größere (Maximum) oder kleinere (Minimum) von zwei Zahlen – im Feld ▾ umschalten	Maximum von 3 und 5 → **5**

Beim Teilen entstehen oft **Kommazahlen**. PBC+ schreibt sie mit einem **Punkt** statt mit einem Komma: 3.5 bedeutet dreieinhalb. Auch 4 ÷ 2 ergibt eine Kommazahl: 2.0.

> **Achtung beim Runden von ,5:** Liegt eine Zahl genau in der Mitte, rundet PBC+ zur **geraden** Zahl: 2.5 wird zu **2**, 3.5 wird zu **4**, 4.5 wird zu **4**. Das ist anders, als du es in Mathe gelernt hast.

Beispiel: Das Würfelduell. Rot und Gelb würfeln. Das Programm zeigt den höchsten Wurf, den Unterschied (immer ohne Minus!), den Mittelwert und den gerundeten Mittelwert. Damit nicht sechsmal fast derselbe Text-Block dasteht, schreibt die Funktion `zeile` (Kapitel 14) jeweils eine Zeile:



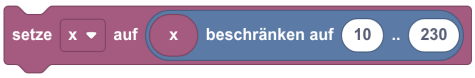


Nach einem Druck auf A

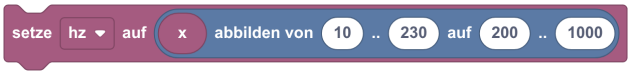


17.2 Wert beschränken und Wert abbilden

Wert beschränken hält eine Zahl in einem Bereich fest. Ist sie zu klein, kommt die untere Grenze heraus, ist sie zu groß, die obere. Sonst bleibt sie unverändert. So läuft ein Punkt nie aus dem Bildschirm:

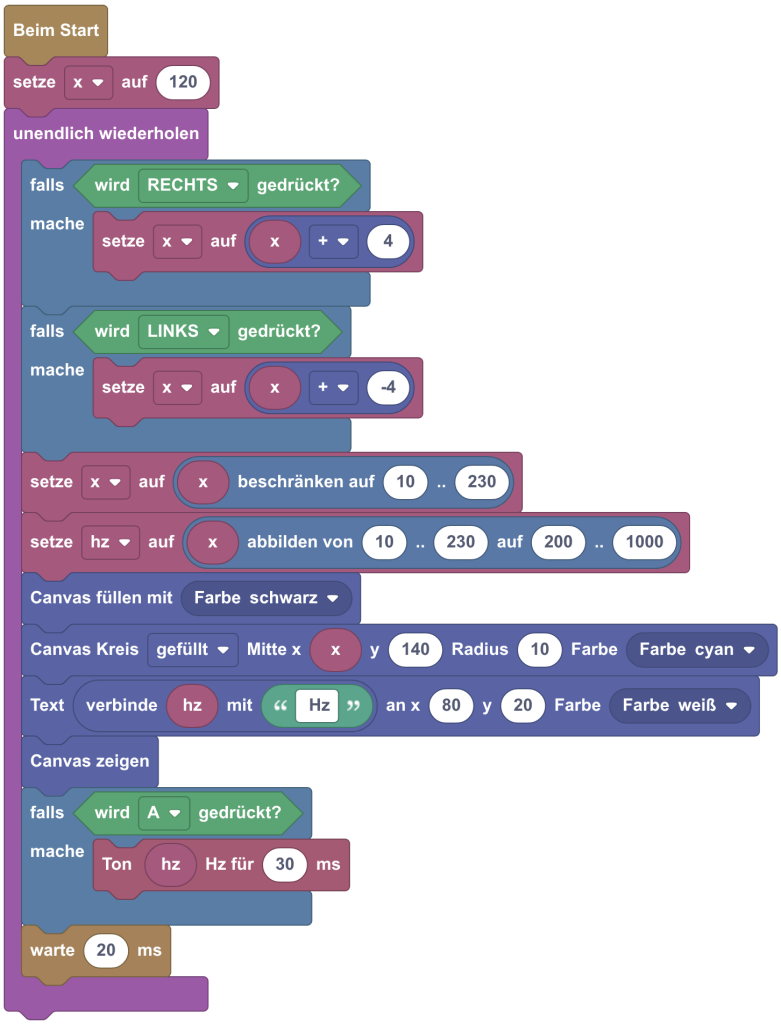


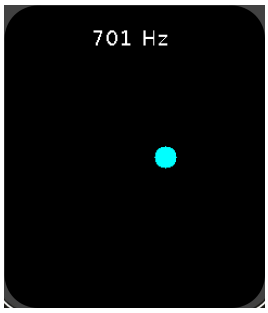
Wert abbilden rechnet eine Zahl von einem Bereich in einen anderen um. Stell dir zwei Lineale nebeneinander vor: Das eine geht von 10 bis 230 (die x-Position), das andere von 200 bis 1000 (die Tonhöhe in Hz). Liegt `x` in der Mitte des ersten Lineals, liegt das Ergebnis in der Mitte des zweiten:



Das Ergebnis ist immer eine **ganze Zahl** – die Stellen nach dem Punkt werden abgeschnitten. Liegt `x` außerhalb von 10 bis 230, liegt auch das Ergebnis außerhalb von 200 bis 1000. Willst du das verhindern, beschränke den Wert vorher oder nachher.

Beispiel: Das Theremin. Mit dem Joystick schiebst du einen Punkt nach links und rechts. Je weiter rechts, desto höher der Ton, solange du A drückst. Gemalt wird auf dem Canvas (Kapitel 11).





Rechts von der Mitte: höherer Ton

17.3 Wurzel und Potenz



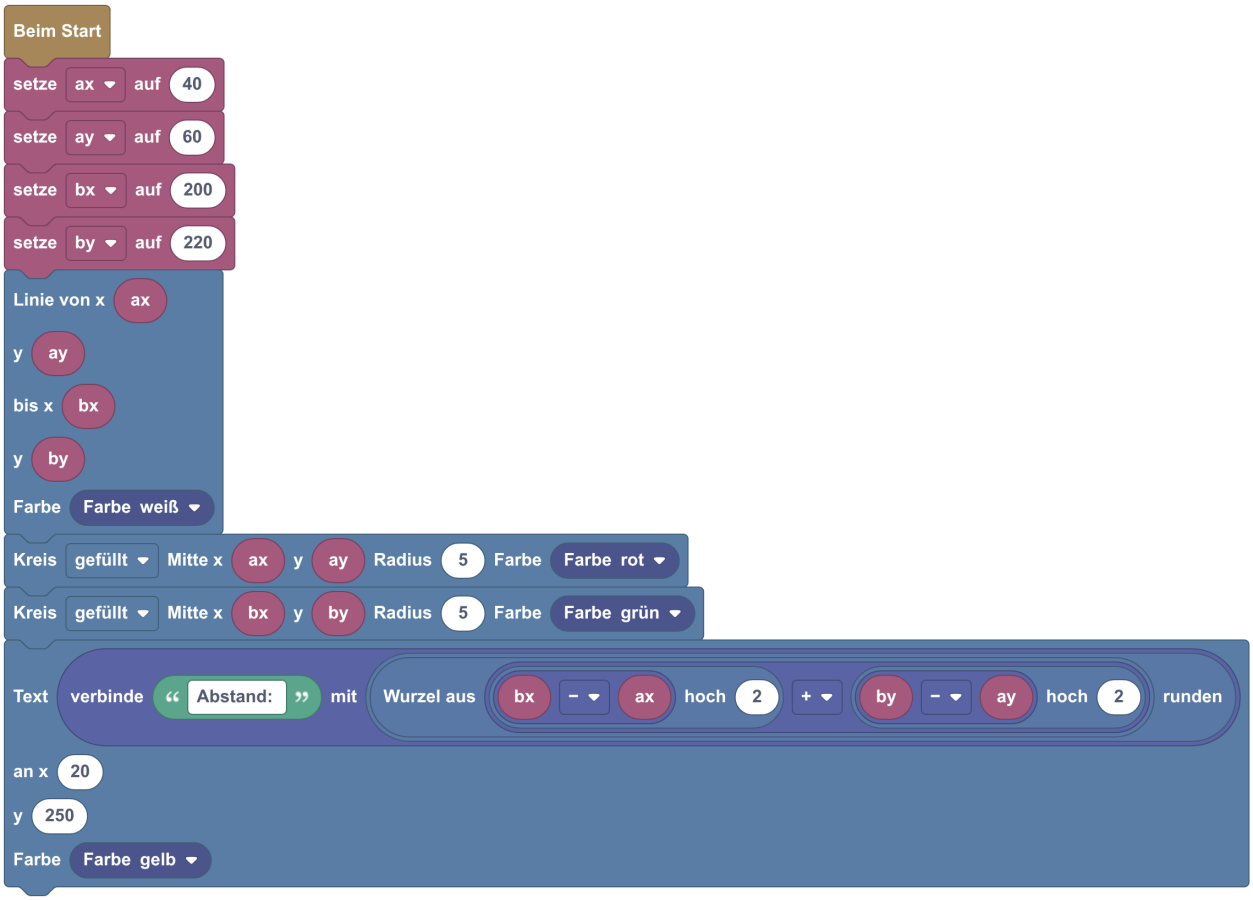
„2 hoch 10“ bedeutet 2 · 2 · 2 · … · 2, zehnmal: **1024**. „5 hoch 2“ ist 5 · 5 = 25 – das Quadrat von 5.

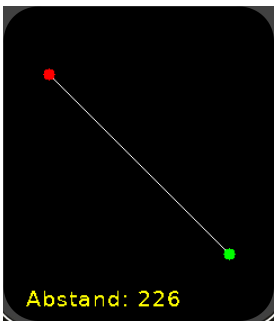


Die **Wurzel** ist die Umkehrung des Quadrats: Wurzel aus 25 ist 5, denn 5 · 5 = 25. Meist kommt eine Kommazahl heraus: Wurzel aus 50 ist 7.0710678…

Beide zusammen brauchst du für den **Abstand zwischen zwei Punkten**. Gehe von Punkt A zu Punkt B erst waagerecht, dann senkrecht. Die beiden Wege heißen `dx` und `dy`. Dann gilt (Satz des Pythagoras):

Abstand = Wurzel aus (dx hoch 2 + dy hoch 2)





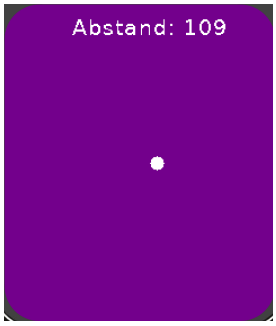
Von (40, 60) nach (200, 220)

Hier ist dx = 160 und dy = 160. Wurzel aus 51200 ist 226.27..., gerundet 226.

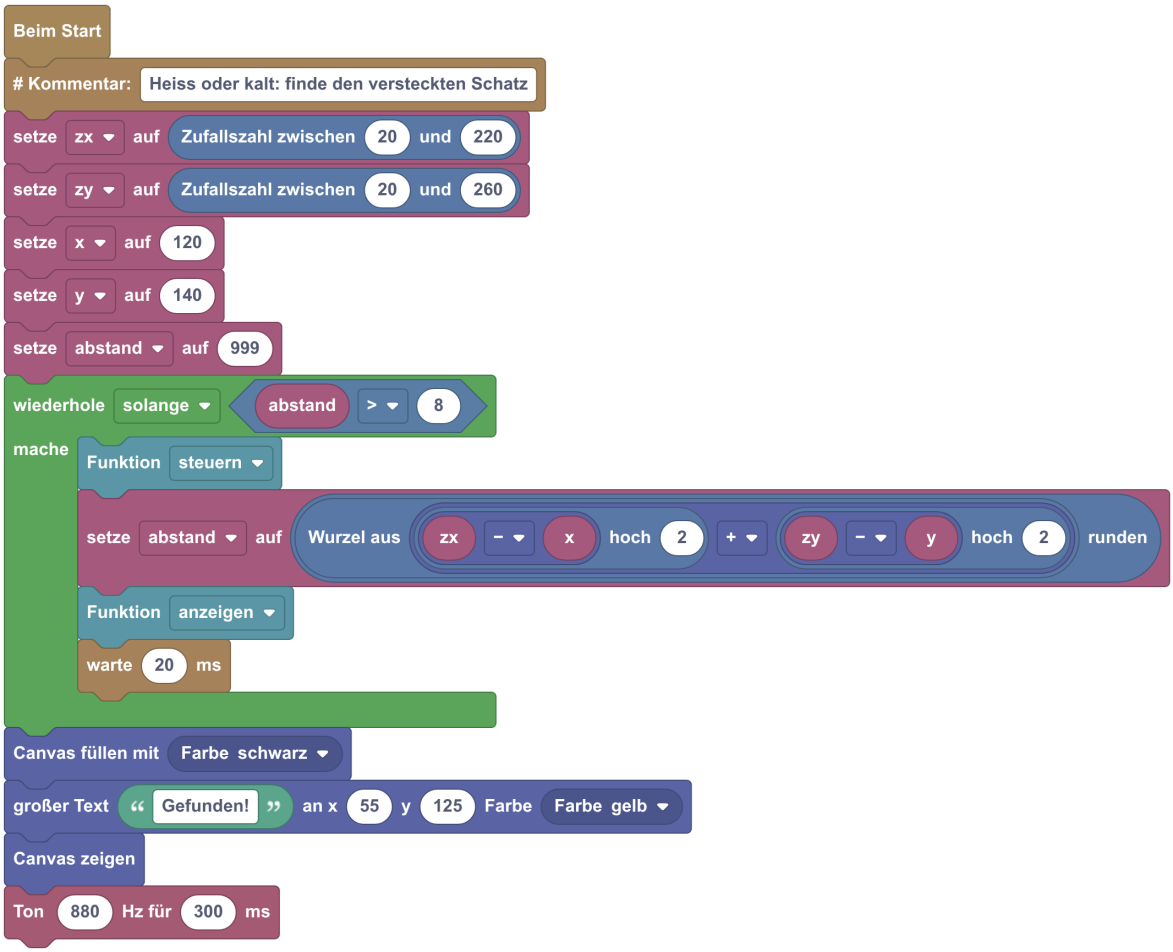
> **Tipp:** Kommazahlen wie 226.27 eignen sich nicht als Position oder Größe beim Zeichnen. Auf dem Gerät führen sie zu einem Fehler, auch wenn der Emulator sie annimmt. Runde deshalb, bevor du mit dem Ergebnis zeichnest. „Wert abbilden“ liefert schon ganze Zahlen.

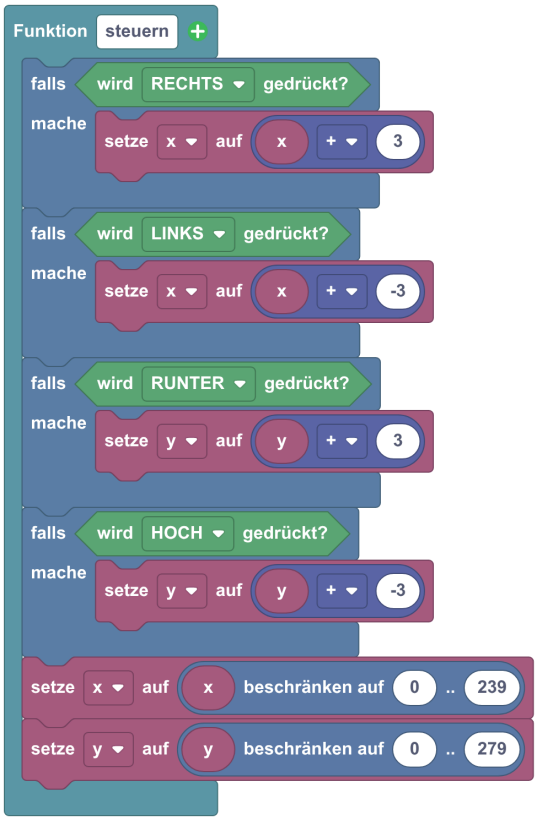
Ausprobieren: Heiß oder kalt?

Irgendwo auf dem Bildschirm ist ein Schatz versteckt. Du steuerst den weißen Punkt. Der Hintergrund zeigt, wie nah du bist: Blau heißt kalt, Rot heißt heiß. Das Programm rechnet in jedem Schritt den Abstand zum Schatz aus (Wurzel und hoch), bildet ihn auf einen Rot-Wert von 255 bis 0 ab und beschränkt ihn auf 0 bis 255. Ist der Abstand höchstens 8, hast du den Schatz gefunden.



Lila: schon ziemlich nah







Das Programm findest du als Datei `k17_heisskalt.pbcp` im Ordner `buch/programme`.

Übungsaufgaben

Aufgabe 1 ⭐ **Runden vorhersagen** Schreibe zuerst auf, was du erwartest: 2.4 runden, 2.6 runden, 3.5 runden, 4.5 runden. Lass dann ein Programm die vier Ergebnisse anzeigen.

Aufgabe 2 ⭐ **Der kleinste Wurf** Ergänze das Würfelduell um eine Zeile „Niedrigster:“ mit dem kleineren der beiden Würfe.

Aufgabe 3 ⭐⭐ Zeitbalken Ein grüner Balken soll in 10 Sekunden von 0 auf 240 Pixel wachsen und dann stehen bleiben. Bilde dazu „Zeit seit Programmstart (ms)“ von 0 bis 10000 auf 0 bis 240 ab und beschränke das Ergebnis.

Aufgabe 4 ⭐⭐ Hoch und runter Erweitere das Theremin: Der Punkt soll auch nach oben und unten fahren können, beschränkt auf 10 bis 270.

Aufgabe 5 ⭐⭐⭐ Abstand als Funktion Schreibe eine Funktion mit Rückgabe `entfernung` mit den Parametern `x1`, `y1`, `x2`, `y2`. Sie gibt den gerundeten Abstand zurück. Benutze sie in „Heiß oder kalt?“. *Achtung:* Die Funktion darf nicht so heißen wie die Variable `abstand`.

Aufgabe 6 ⭐⭐⭐ Verdoppeln Wie oft musst du 2 mit sich selbst malnehmen, bis mehr als 1000 herauskommt? Lass das Programm `n` hochzählen, solange 2 hoch `n` kleiner als 1000 ist.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Kommazahlen schreibt PBC+ mit **Punkt**: 3.5. Teilen ergibt immer eine Kommazahl.
- **runden** macht ganze Zahlen; genau ,5 wird zur geraden Zahl gerundet.
- **Betrag** entfernt das Minus, **Minimum/Maximum** wählt die kleinere oder größere Zahl.
- **Wert beschränken** hält eine Zahl zwischen zwei Grenzen.
- **Wert abbilden** rechnet von einem Bereich in einen anderen um und liefert ganze Zahlen.
- **Abstand** = Wurzel aus (dx hoch 2 + dy hoch 2).

KAPITEL 18

Der Beschleunigungssensor

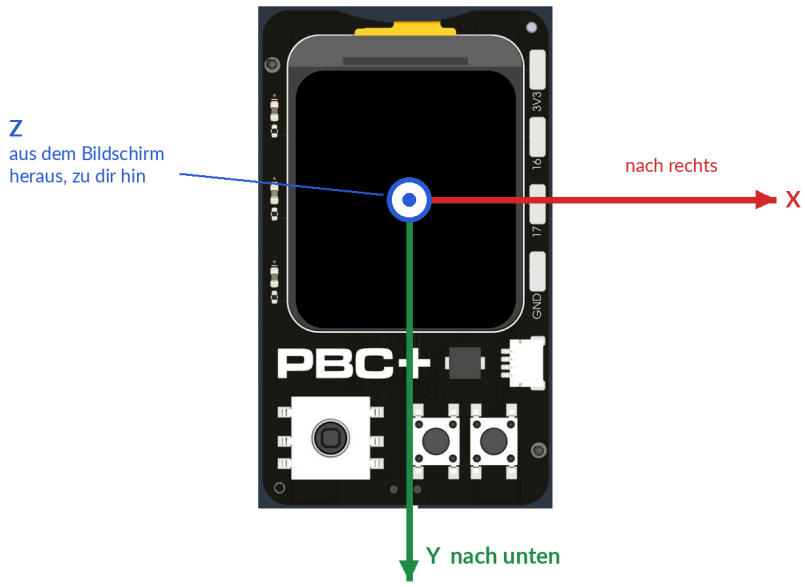
Worum geht's? Im PicoBoy steckt ein kleiner Sensor, der spürt, wie du das Gerät hältst. Damit kannst du Spiele bauen, die man durch **Kippen** steuert – wie eine Kugel in einem Holz-Labyrinth. In diesem Kapitel liest du den Sensor aus und lässt eine Kugel über den Bildschirm rollen.

> **Nur auf dem echten Gerät!** Der Emulator hat keinen Sensor. Er tut immer so, als läge der PBC+ flach auf dem Tisch: X = 0, Y = 0, Z = 1 g. Deine Programme laufen im Emulator ohne Fehler – aber kippen kannst du ihn nicht. Teste die Programme dieses Kapitels deshalb auf dem PBC+.

Was misst der Sensor?

Auf alles auf der Erde wirkt die **Erdanziehung**. Sie zieht nach unten. Der Sensor im PBC+ spürt, wie stark und in welche Richtung sie auf das Gerät wirkt. Daraus kann dein Programm ablesen, wie schief du das Gerät hältst.

Der Sensor misst in drei Richtungen. Man nennt sie **Achsen**:



Achse	zeigt …
X	nach rechts – wie x auf dem Bildschirm
Y	nach unten – wie y auf dem Bildschirm
Z	aus dem Bildschirm heraus, zu dir hin

Der Block

Den Block findest du in der Kategorie **Sensor**:



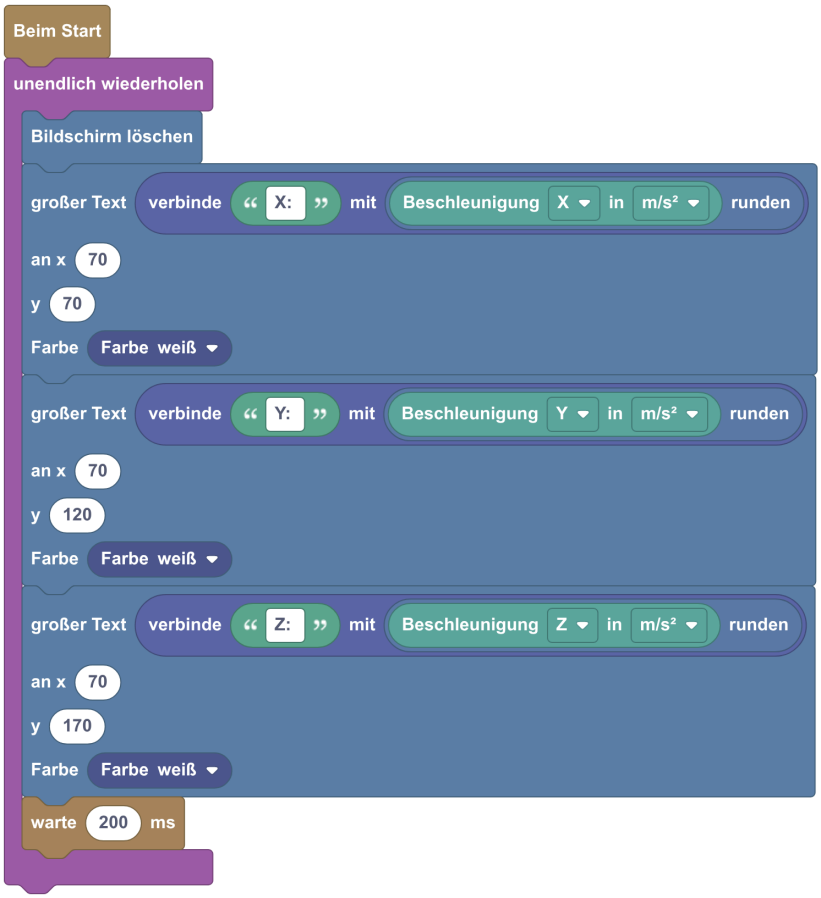
Er ist ein *Wert-Block* mit runden Enden – wie eine Zahl. Im ersten Auswahlfeld wählst du die Achse **X**, **Y** oder **Z**. Im zweiten wählst du die Einheit:

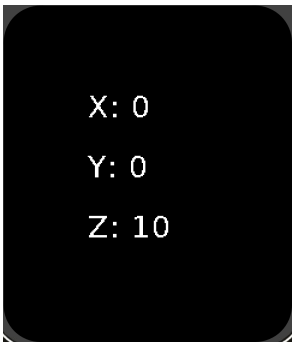
- **g** zählt in „Erdanziehungen“. Liegt der PBC+ flach, zeigt Z genau **1 g**.
- **m/s²** (gesprochen: Meter pro Sekunde zum Quadrat) ist die Einheit der Physiker. 1 g sind etwa **9,81 m/s²**.

Die Zahlen haben meistens viele Stellen hinter dem Komma, zum Beispiel 0,2734375. Das liegt daran, dass der Sensor sehr fein misst.

18.1 Neigung messen

Bevor du mit dem Sensor steuerst, schaust du dir die Zahlen an. Dieses Programm zeigt die drei Werte in m/s² an. Mit „runden“ (Kategorie **Mathe**) werden daraus ganze Zahlen, die man gut lesen kann.





Im Emulator stehen die Werte immer still.

Lade das Programm auf den PBC+ und probiere aus:

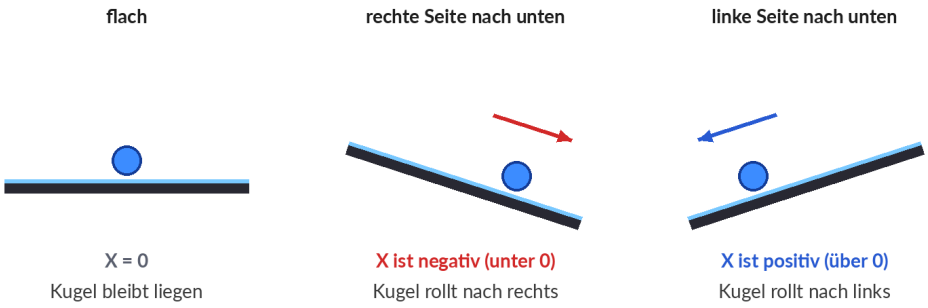
- **Flach auf dem Tisch:** X und Y sind 0. Z ist 10. Die ganze Erdanziehung steckt in der Z-Achse.
- **Rechte Seite nach unten kippen:** X wird **negativ** (kleiner als 0). Je schiefer, desto kleiner – bis etwa −10, wenn das Gerät auf der rechten Kante steht.
- **Linke Seite nach unten kippen:** X wird **positiv** (größer als 0).
- **Obere Kante nach unten kippen** (von dir weg): Y wird positiv.
- **Untere Kante nach unten kippen** (zu dir hin): Y wird negativ.

> **So kannst du es dir merken:** Der Sensor zeigt an, wo **oben** ist. Die Achse, die nach oben zeigt, bekommt eine positive Zahl. Kippst du die rechte Seite nach unten, zeigt die linke Seite – also die *Gegenrichtung* von X – nach oben. Deshalb wird X negativ.

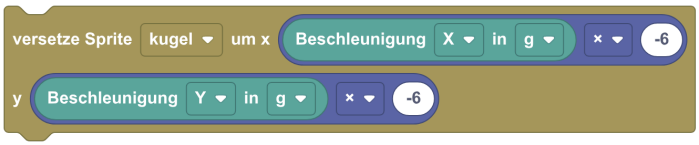
> **Tipp:** Halte das Gerät beim Spielen wie ein Tablett, also ungefähr flach. Hältst du es schräg vor dich wie ein Buch, ist Y schon ohne Kippen weit im Minus.

18.2 Steuern durch Kippen

Jetzt soll eine Kugel über den Bildschirm rollen – immer nach unten, wie eine echte Kugel auf einem schiefen Brett.

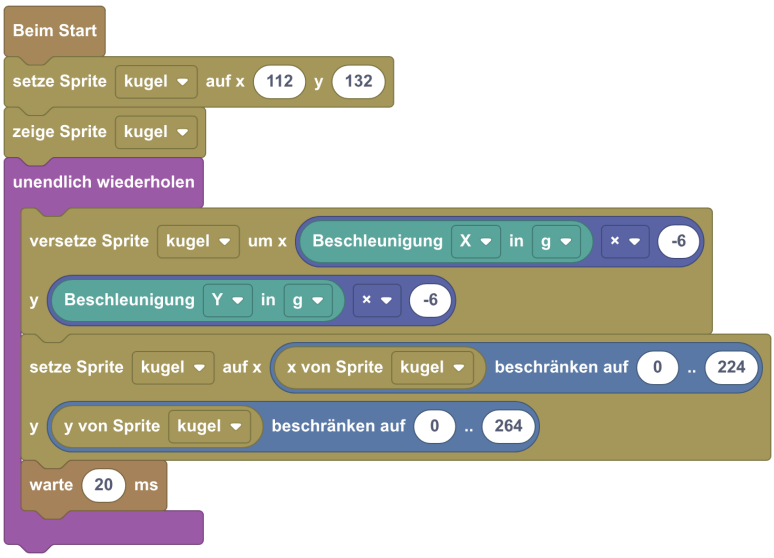


Kippst du die rechte Seite nach unten, ist X negativ. Die Kugel soll aber nach **rechts** rollen, also muss ihr x **größer** werden. Die Kugel bewegt sich immer **gegen** das Vorzeichen des Messwerts. Darum nimmst du den Wert mal **−6**:



Ein Beispiel: Bei X = −0,5 g wird daraus −0,5 × −6 = **3**. Die Kugel rollt 3 Pixel nach rechts. Je schiefer du kippst, desto schneller rollt sie. Mit der Zahl 6 stellst du die Geschwindigkeit ein.

Für dieses Programm brauchst du ein Sprite `kugel`. Lade dafür das Bild `kugel.png` hoch (**Dateien → Bilder verwalten…**) und lege in der Kategorie **Sprites** mit „+ neues Sprite…“ das Sprite `kugel` an.



Damit die Kugel nicht aus dem Bildschirm rollt, steht in der Schleife ein zweiter Block. Er setzt die Kugel wieder an ihre Stelle – aber mit „Wert beschränken“ aus Kapitel 17: x bleibt zwischen 0 und 224, y zwischen 0 und 264. Die Kugel ist 16 Pixel groß, so bleibt sie immer ganz zu sehen (240 − 16 = 224 und 280 − 16 = 264).



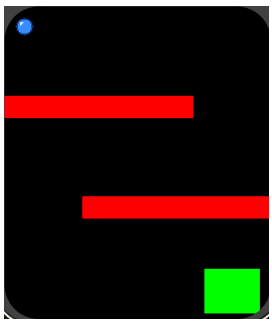
> **Warum bleibt die Kugel bei wenig Neigung liegen?** Ein Sprite kann nur um ganze Pixel versetzt werden. Bei 0,9 Pixeln bewegt es sich gar nicht – die Zahl hinter dem Komma fällt weg. Erst ab etwa 10 Grad Neigung (0,17 g × 6 = 1 Pixel) rollt die Kugel los. Das ist praktisch: So zittert sie nicht, wenn du das Gerät ruhig hältst.

> **Ohne Gerät testen:** Im Emulator bleibt die Kugel liegen. Willst du dein Spiel trotzdem dort ausprobieren, baue zusätzlich die Joystick-Steuerung aus Kapitel 12 ein. Auf dem Gerät funktionieren dann beide Steuerungen.

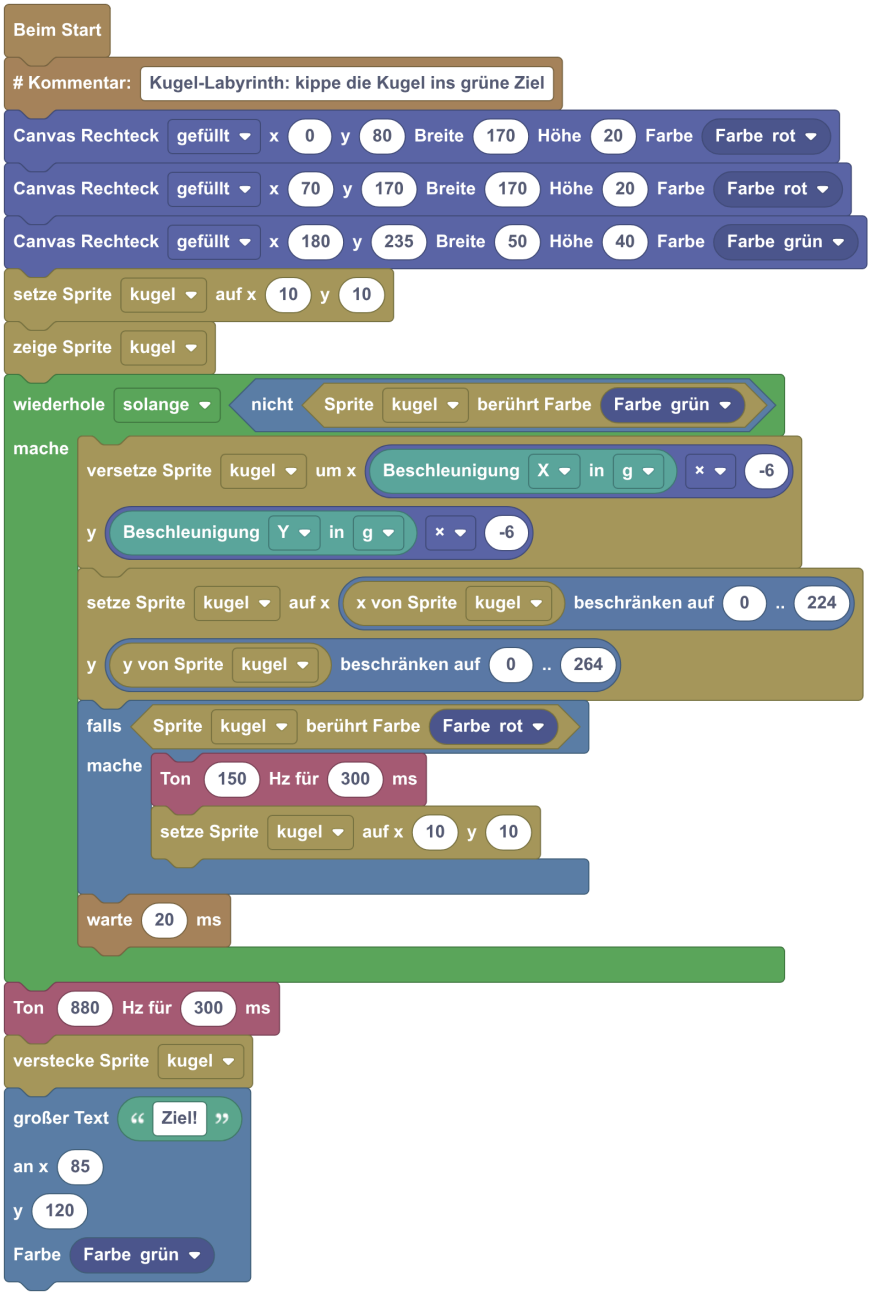
Ausprobieren: Das Kugel-Labyrinth

Kippe die Kugel von oben links in das grüne Ziel unten rechts. Die roten Wände sind Lava: Berührt die Kugel Rot, gibt es einen tiefen Ton und sie muss zurück zum Start. Die Wände und das Ziel werden auf den Canvas gemalt, damit „berührt Farbe“ sie findet (Kapitel 13).

Die Schleife läuft, **solange die Kugel nicht auf Grün** liegt. Danach ertönt ein hoher Ton und „Ziel!“ erscheint.



Start im Labyrinth



Das Programm findest du als Datei `k18_labyrinth.pbcp` im Ordner `buch/programme`. Lade vorher das Bild `kugel.png` hoch.

Übungsaufgaben

> **Aufgabe 1** ⭐ **Forschungsauftrag** Lade das Messprogramm aus 18.1 auf den PBC+. Halte das Gerät auf fünf verschiedene Arten: flach, auf der rechten Kante, auf der linken Kante, auf der unteren Kante (Bildschirm zu dir), mit dem Bildschirm nach unten. Schreibe jedes Mal X, Y und Z in eine Tabelle. Was fällt dir auf?

Aufgabe 2 ⭐ **Wasserwaage** Baue eine Wasserwaage: Liegt der PBC+ gerade, wird der Bildschirm grün und zeigt „gerade“. Sonst wird er rot und zeigt „schief“. *Tipp:* „Gerade“ heißt: Der **Betrag** von X ist kleiner als 0,05 g **und** der Betrag von Y auch. Den Block „Betrag von…“ findest du unter **Mathe**.

Aufgabe 3 ⭐⭐ **Schneller, langsamer, falsch herum** Ändere im Kugel-Programm aus 18.2 die Zahl −6. Was passiert mit −2, mit −12 und mit +6? Erkläre.

Aufgabe 4 ⭐⭐ **Kipp-Anzeige** Die Status-LEDs zeigen, wohin du kippst: rechte Seite nach unten → grüne LED, linke Seite nach unten → rote LED, ungefähr flach → gelbe LED. Als „ungefähr flach“ gilt: X liegt zwischen −0,3 und 0,3.

Aufgabe 5 ⭐⭐⭐ **Schüttel-Würfel** Wenn du den PBC+ kräftig schüttelst, zeigt er eine Zufallszahl von 1 bis 6 und macht einen kurzen Ton. Beim Schütteln wird der Betrag von X oder Y größer als 1,5 g. Damit du den Würfel auch im Emulator testen kannst, soll er zusätzlich bei Taste A würfeln. *Achtung:* Auf dem Bildschirm kann der PBC+ keine Umlaute schreiben. Schreibe also „Schuettel mich!“.

Aufgabe 6 ⭐⭐⭐ **Münzen kippen** Baue ein Münz-Spiel mit Kippsteuerung: Die Kugel sammelt Münzen. Nach 10 Münzen ist Schluss, und die Spielzeit in Sekunden erscheint.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Der Beschleunigungssensor spürt die Erdanziehung in drei Richtungen: X nach rechts, Y nach unten, Z aus dem Bildschirm heraus.
- Der Block „Beschleunigung … in …“ steht in der Kategorie **Sensor**. Er liefert eine Zahl in **g** (1 g = Erdanziehung) oder in **m/s²** (1 g ≈ 9,81 m/s²).
- Flach auf dem Tisch: X = 0, Y = 0, Z = 1 g. Die Achse, die nach oben zeigt, bekommt eine positive Zahl.
- Zum Steuern nimmst du den Messwert mal eine Zahl, z. B. −6. Das Minus sorgt dafür, dass die Kugel bergab rollt; die Zahl bestimmt die Geschwindigkeit.
- Der Emulator hat keinen Sensor. Kippen funktioniert **nur auf dem echten Gerät**.

KAPITEL 19

Projekt: Reaktionstest

Worum geht's? Wie schnell bist du? Der PBC+ zeigt nach einer zufälligen Wartezeit ein grünes Signal – und misst, wie viele Millisekunden du bis zum Drücken von Taste A brauchst. Wer schummelt und zu früh drückt, wird erwischt. Du baust das Programm in vier Schritten und brauchst dafür vieles, was du schon kannst: Zeit messen, Zufall, Tasten, Entscheidungen und Funktionen.

19.1 Der Plan

Bevor du Blöcke ziehst, überlegst du, was nacheinander passieren soll:

1. Der Bildschirm zeigt „Achtung …".
2. Der PBC+ wartet – mal kürzer, mal länger.
3. Der Bildschirm wird grün und zeigt „JETZT!". Ab jetzt läuft die Stoppuhr.
4. Du drückst Taste A. Die Stoppuhr hält an.
5. Der PBC+ zeigt die gemessene Zeit in Millisekunden.

Eine **Millisekunde** (ms) ist der tausendste Teil einer Sekunde. Die meisten Menschen brauchen für so einen Test 200 bis 400 ms.

19.2 Schritt 1: Signal und Stoppuhr

Der PBC+ hat keine Stoppuhr-Taste. Aber er kennt die „Zeit seit Programmstart (ms)" (Kategorie **Zeit**, Kapitel 4). Die Idee:

- Wenn das Signal kommt, merkst du dir die Uhrzeit in der Variable `start`.
- Wenn die Taste gedrückt ist, fragst du: Wie viele Millisekunden sind **seit** `start` vergangen? Das ist deine Reaktionszeit. Dafür gibt es den Block „vergangene ms seit…".



Das Signal: Jetzt drücken!

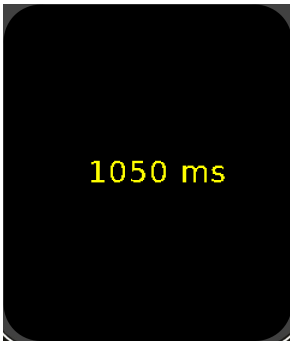
Dazwischen steht eine Schleife „wiederhole solange nicht wird A gedrückt?". Sie ist **leer**: Sie tut nichts, außer immer wieder zu fragen, ob Taste A unten ist. Sobald du drückst, geht das Programm unter der Schleife weiter.



Teste das Programm ein paar Mal. Im Emulator drückst du die Taste **A** auf der Tastatur.

Merkst du etwas? Das Signal kommt **immer nach genau 3 Sekunden**. Nach ein paar Versuchen zählst du einfach mit „21, 22, 23“ – und drückst schon, bevor du das Grün überhaupt siehst. Das ist kein fairer Test.

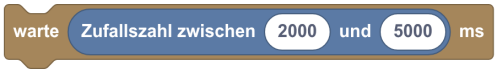
> **Warum nicht „auf A warten“?** Dieser Block wartet, bis du die Taste drückst **und wieder loslässt**. Dann würde die Zeit fürs Loslassen mitgezählt. Die leere Schleife hört schon beim Drücken auf.



Das Ergebnis

19.3 Schritt 2: Zufall

Die Wartezeit soll jedes Mal anders sein. Dafür ersetzt du „warte 3 Sekunden“ durch diesen Block:

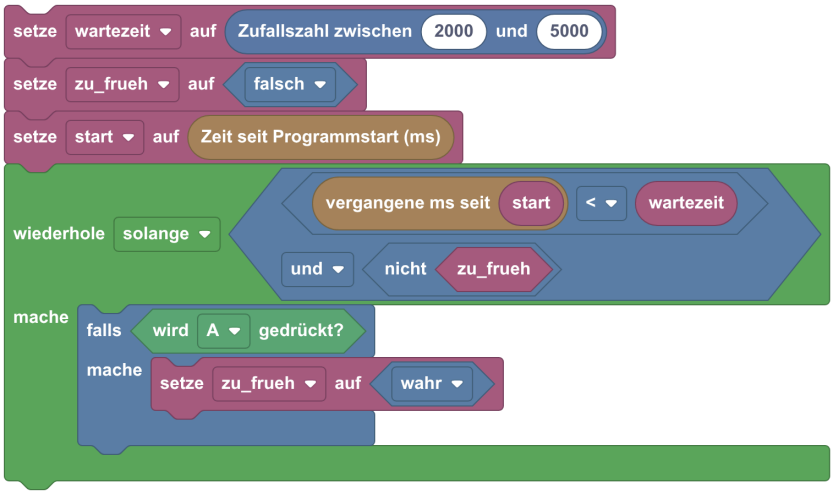


Jetzt wartet der PBC+ irgendwo zwischen 2000 und 5000 ms, also zwischen 2 und 5 Sekunden. Mitzählen klappt nicht mehr. Die Datei mit diesem Schritt heißt `k19_schritt2.pbcp`.

19.4 Schritt 3: Der faire Start

Es gibt noch einen Trick zum Schummeln: Man hält A einfach die ganze Zeit gedrückt. Dann ist die leere Schleife sofort fertig, wenn das Signal kommt – und die Zeit ist fast 0 ms.

Ein fairer Test muss also merken, ob du **zu früh** drückst. Mit „warte“ geht das nicht: Während das Programm wartet, kann es nichts anderes tun. Darum baust du das Warten selbst – als Schleife, die bei jedem Durchlauf nach der Taste schaut:



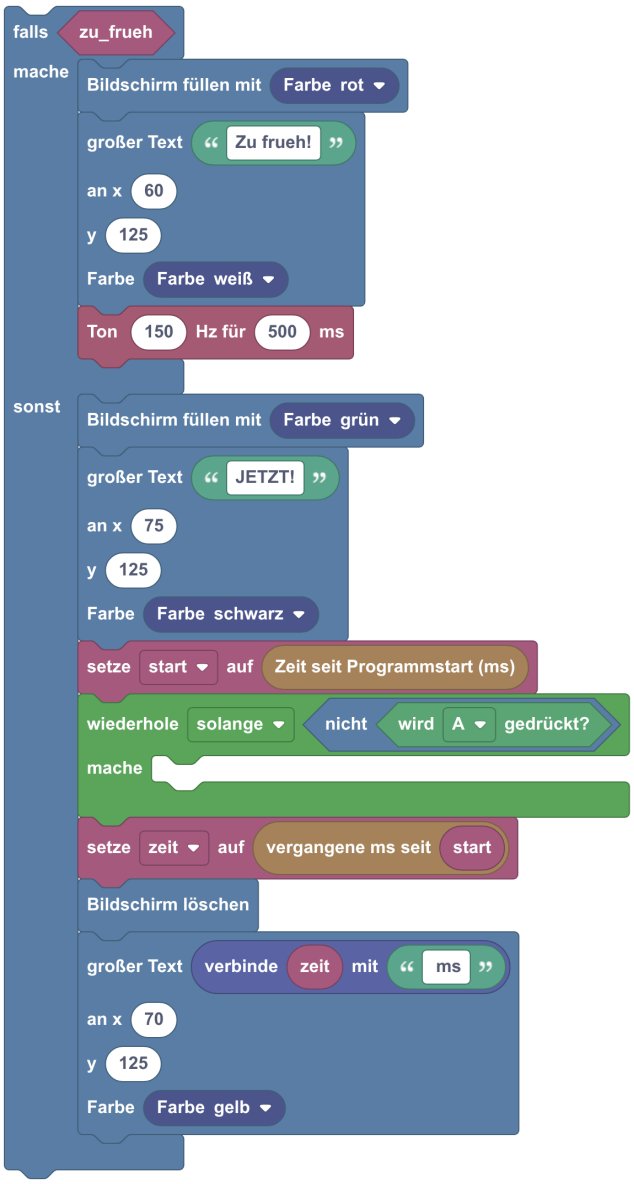
So funktioniert es:

- `wartezeit` ist eine Zufallszahl zwischen 2000 und 5000.
- `zu_frueh` ist am Anfang **falsch** (Block „wahr / falsch“ aus **Mathe**).
- `start` merkt sich die Uhrzeit, zu der das Warten beginnt.
- Die Schleife läuft, **solange** noch nicht genug Zeit vergangen ist **und** noch niemand zu früh gedrückt hat.
- Wird in dieser Zeit A gedrückt, wird `zu_frueh` **wahr** – und die Schleife hört sofort auf.

Nach der Schleife entscheidet ein „falls … sonst“: War es zu früh, wird der Bildschirm rot und ein tiefer Ton erklingt. Sonst kommt das Signal und die Zeit wird gemessen wie in Schritt 1.



Erwischt!



Der Block „falls `zu_frueh`“ braucht keinen Vergleich: `zu_frueh` ist schon selbst **wahr** oder **falsch**. Die Datei heißt `k19_schritt3.pbcp`.

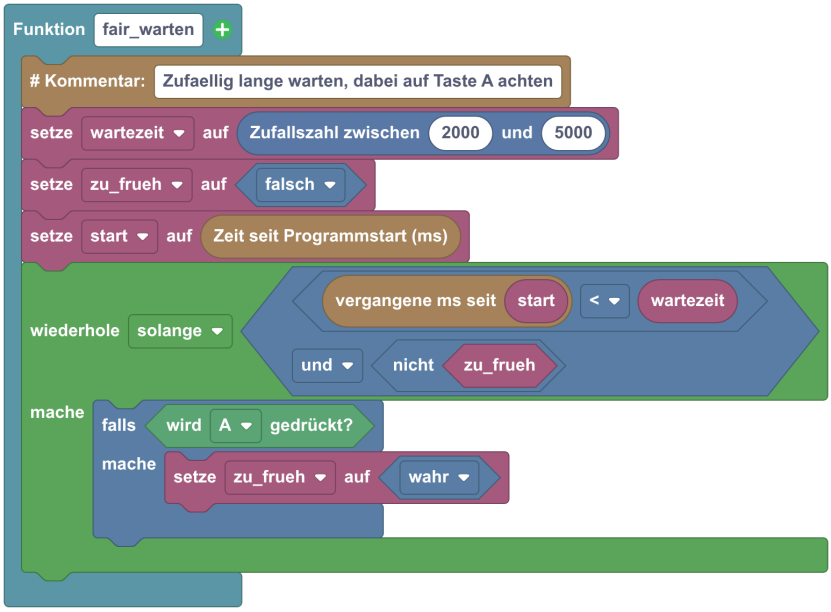
> **Keine Umlaute auf dem Bildschirm!** Die Schrift des PBC+ kennt kein ä, ö, ü und ß. Statt „Zu früh!“ erscheint „Zu fr?h!“. Schreibe deshalb in Text-Blöcken „Zu frueh!“. Das gilt im Emulator und auf dem Gerät.

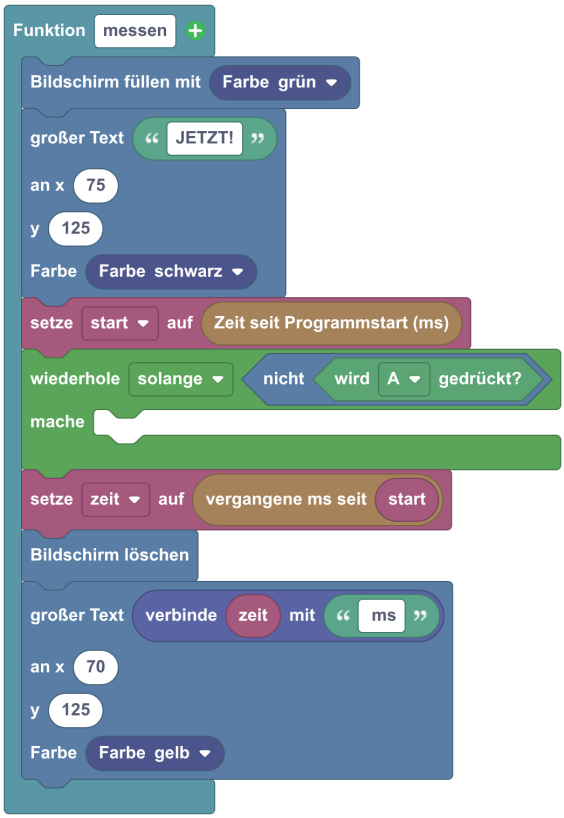
19.5 Schritt 4: Fünf Runden und eine Bestzeit

Ein einzelner Versuch ist oft Glückssache. Besser sind fünf Runden – und am Ende zählt die **Bestzeit**, also die kleinste Zeit.

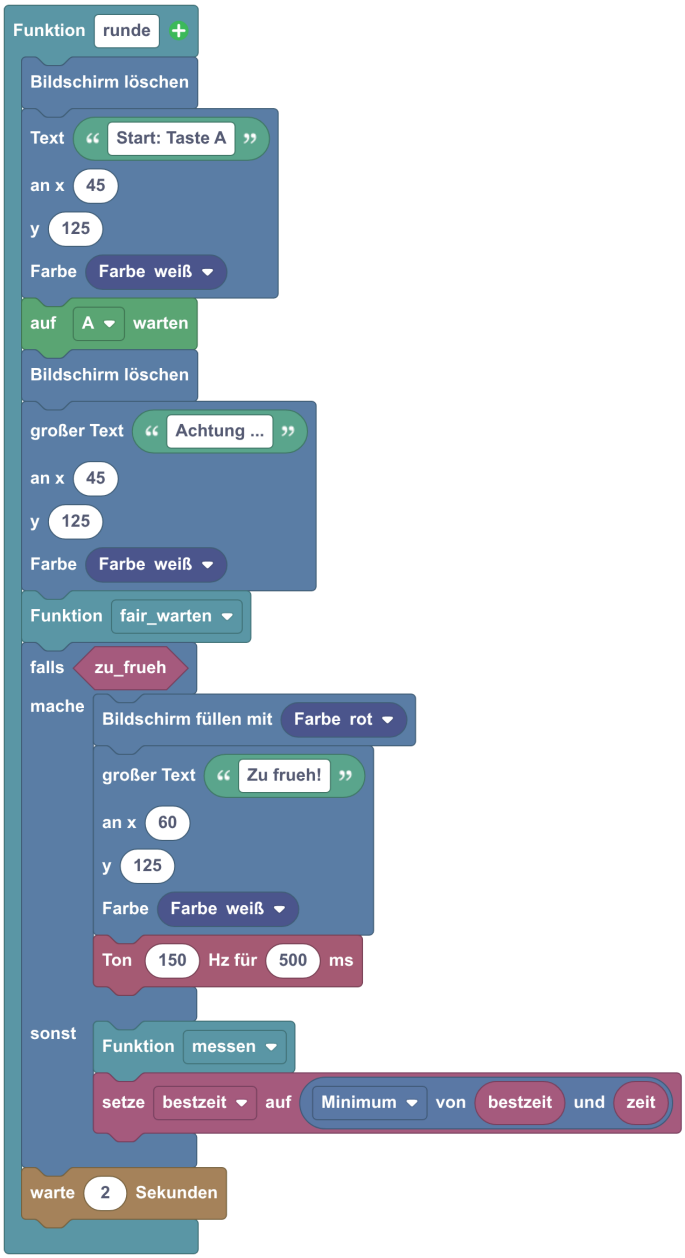
Das Programm wird jetzt länger. Damit es übersichtlich bleibt, teilst du es in **Funktionen** auf (Kapitel 14). Du legst sie in der Kategorie **Funktionen** mit „+ neue Funktion…“ an:

Funktion	Was sie tut
`fair_warten`	zufällig lange warten und dabei auf zu frühes Drücken achten (aus Schritt 3)
`messen`	grünes Signal zeigen, Zeit messen und anzeigen (aus Schritt 1)
`runde`	eine ganze Runde: auf Taste A zum Start warten, `fair_warten`, dann `messen` oder „Zu frueh!“





Die Funktion `runde` ruft die beiden anderen auf. Neu ist der Anfang: Jede Runde beginnt erst, wenn du A drückst. So kannst du in Ruhe Luft holen. Hier passt „auf A warten“ (Kategorie **Eingabe**) genau: Es geht erst weiter, wenn du A wieder losgelassen hast. Sonst würde schon der Start-Druck als „zu früh“ zählen.



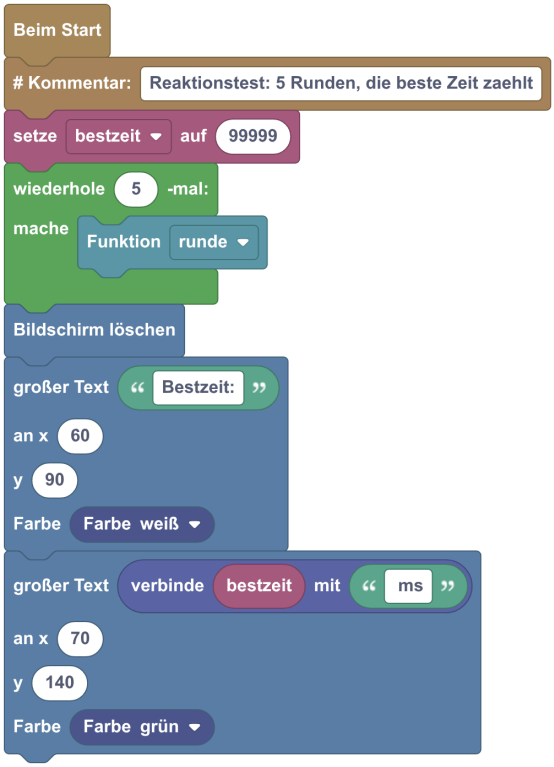
Die Bestzeit merken: Die Variable `bestzeit` startet mit einer riesigen Zahl (99999). Nach jeder gültigen Runde fragt das Programm mit dem Block „Minimum von … und …“ (Kategorie **Mathe**, Kapitel 17): Was ist kleiner – die alte Bestzeit oder die neue Zeit? Der kleinere Wert wird die neue Bestzeit. Weil 99999 so groß ist, wird die erste gültige Zeit auf jeden Fall die Bestzeit.

Ausprobieren: Der fertige Reaktionstest

Unter „Beim Start“ steht jetzt nur noch wenig: die Bestzeit vorbereiten, fünfmal `runde` aufrufen und am Ende die Bestzeit zeigen. Man kann es lesen wie einen Satz.



Nach fünf Runden



Das ganze Programm findest du als Datei `k19_reaktionstest.pbcp` im Ordner `buch/programme`. Fordere jemanden heraus: Wer hat die beste Bestzeit?

> **Tipp:** Auf dem echten Gerät macht der Test mehr Spaß – und er ist auch etwas genauer, weil keine Tastatur und kein Browser dazwischen sitzen. Vergleiche deine Zeiten im Emulator und auf dem PBC+.

Übungsaufgaben

> **Aufgabe 1** ⭐ **Mehr Signal** Ändere Schritt 3: Wenn das Signal kommt, soll zusätzlich die RGB-LED grün leuchten und ein kurzer hoher Ton (880 Hz, 50 ms) erklingen. Nach dem Drücken geht die LED wieder aus.

> **Aufgabe 2** ⭐ **Schummel-Test** Halte bei Schritt 2 (`k19_schritt2.pbcp`) die Taste A schon gedrückt, bevor das Signal kommt, und lass sie erst danach los. Welche Zeit erscheint? Probiere dasselbe bei Schritt 3. Erkläre den Unterschied.

Aufgabe 3 ⭐⭐ Wie gut war das? Unter der Zeit soll eine Bewertung erscheinen: unter 250 ms „Blitzschnell!“, unter 400 ms „Gut!“, sonst „Schlafmuetze?“. *Tipp:* Du brauchst ein „falls … sonst“ in einem „sonst“.

Aufgabe 4 ⭐⭐ Durchschnitt Statt der Bestzeit soll am Ende der **Durchschnitt** aller gültigen Runden erscheinen. Zähle dafür in `summe` alle Zeiten zusammen und in `gueltig`, wie viele Runden gültig waren. Der Durchschnitt ist `summe` ÷ `gueltig`, gerundet. Was passiert, wenn alle Runden zu früh waren?

Aufgabe 5 ⭐⭐⭐ Das Duell Zwei Spieler halten den PBC+ gemeinsam. Einer drückt den Joystick nach **links**, die andere Taste **A**. Wer nach dem Signal zuerst drückt, gewinnt. Wer vor dem Signal drückt, verliert sofort. *Tipp:* Eine Variable `sieger` (Text) ist am Anfang leer. Die Schleifen laufen, solange `sieger` leer ist.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- **Zeit messen:** Uhrzeit in einer Variable merken („Zeit seit Programmstart (ms)“), später „vergangene ms seit…“ abfragen.
- **Zufall** macht einen Test fair: Mit „warte Zufallszahl … ms“ kann niemand mitzählen.
- **Zu früh erkennen:** Statt „warte“ eine Schleife bauen, die bei jedem Durchlauf nach der Taste schaut.
- Eine Variable kann **wahr** oder **falsch** speichern. „falls“ kann sie direkt prüfen.
- Mit „Minimum von … und …“ findest du die kleinste Zahl – praktisch für Bestzeiten.
- Größere Programme teilst du in Funktionen auf. „Beim Start“ liest sich dann wie ein Plan.
- Die Bildschirm-Schrift kennt keine Umlaute: „frueh“ statt „früh“.

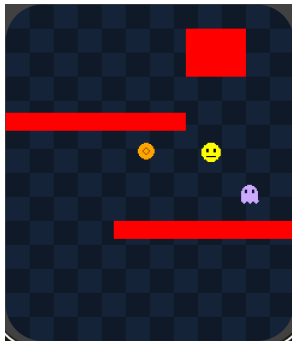
KAPITEL 20

Projekt: Schatzjagd

Worum geht's? Zum Abschluss baust du ein richtiges Spiel: Ein Held sammelt fünf Schätze ein. Dabei muss er der Lava ausweichen und einem Geist, der hin und her schwebt. Am Ende zeigt das Spiel, wie lange du für jeden Schatz gebraucht hast. Du brauchst fast alles, was du in diesem Buch gelernt hast: Sprites, Kollisionen, Variablen, **Listen** und **Funktionen**.

20.1 Das Spiel und der Plan

Die Regeln:



Mitten in der Schatzjagd

- Du steuerst den Helden (gelb) mit dem Joystick. Er startet oben links.
- Ein Schatz (orange) liegt auf dem Spielfeld. Hast du ihn, erscheint der nächste – insgesamt fünf.
- Berührst du die rote Lava oder den Geist, verlierst du ein Leben und musst zurück zum Start. Du hast drei Leben.
- Hast du alle fünf Schätze: gewonnen! Sind alle Leben weg: Game Over.

Ein Spiel wie dieses wird schnell lang. Damit du nicht den Überblick verlierst, teilst du es von Anfang an in **Funktionen** auf (Kapitel 14). Jede Funktion hat eine Aufgabe und einen Namen, der sagt, was sie tut:

Funktion	Aufgabe
`vorbereiten`	Spielfeld zeigen, Startwerte setzen, Figuren aufstellen
`held_steuern`	den Helden mit dem Joystick bewegen
`geist_bewegen`	den Geist hin und her schweben lassen
`gefahr_pruefen`	Lava oder Geist berührt? Dann ein Leben weniger
`schatz_platzieren`	den aktuellen Schatz an seine Stelle setzen
`schatz_pruefen`	Schatz gefunden? Dann Zeit merken und nächster Schatz
`ende_anzeigen`	„Gewonnen!“ mit allen Zeiten oder „Game Over“

Du baust das Spiel in vier Schritten. Nach jedem Schritt kannst du es starten und testen. So findest du Fehler sofort – und nicht erst, wenn alles fertig ist.

Vorbereitung: Lade die Bilder `hintergrund_jagd.png`, `held.png`, `muenze.png` und `geist.png` hoch (**Dateien → Bilder verwalten…**). Lege in der Kategorie **Sprites** mit „+ neues Sprite…“ drei Sprites an: `held` (Bild `held.png`), `schatz` (Bild `muenze.png`) und `geist` (Bild `geist.png`).

20.2 Schritt 1: Spielfeld und Held

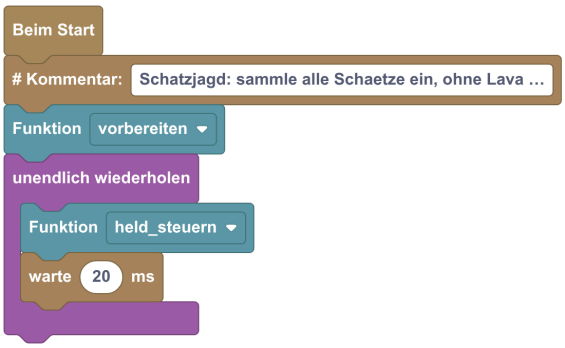
Lege in der Kategorie **Funktionen** mit „+ neue Funktion…“ die Funktion `vorbereiten` an. Sie legt das Hintergrundbild auf den Canvas (Kapitel 11), stellt den Helden auf den Start und zeigt ihn:

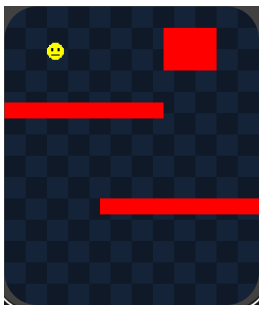


Die zweite Funktion, `held_steuern`, kennst du fast schon aus Kapitel 12: vier „falls … gedrückt?“-Blöcke, je 3 Pixel pro Schritt.



Eine Funktion tut nichts, solange niemand sie **aufruft**. Sobald es eine Funktion gibt, steht ihr Name in der Kategorie **Funktionen**. Ein Klick darauf gibt dir den Aufruf-Block. Unter „Beim Start“ kommt der Plan des Spiels:





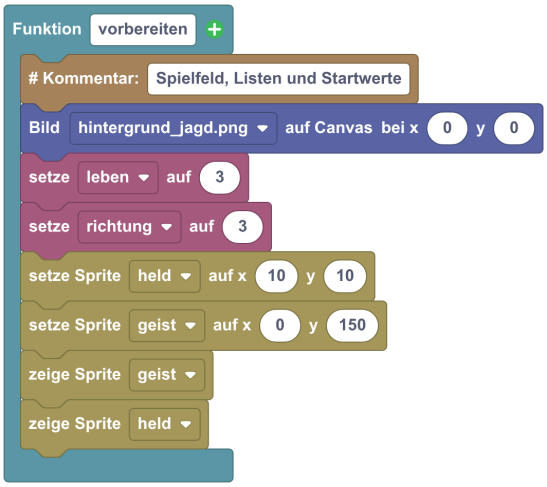
Schritt 1: Der Held läuft.

Starte das Programm: Der Held läuft über das Spielfeld – auch durch die Lava, denn die prüft noch niemand. Datei: `k20_schritt1.pbcp`.

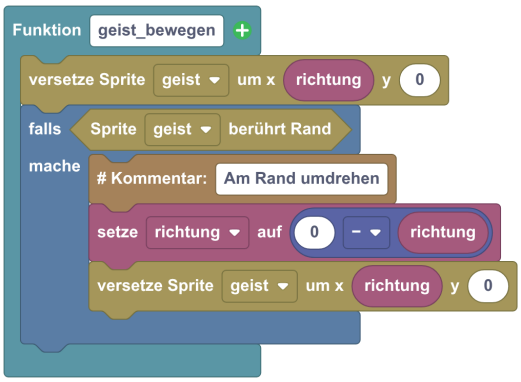
20.3 Schritt 2: Lava, Geist und Leben

Jetzt wird es gefährlich. In `vorbereiten` kommen drei neue Startwerte und der Geist dazu:

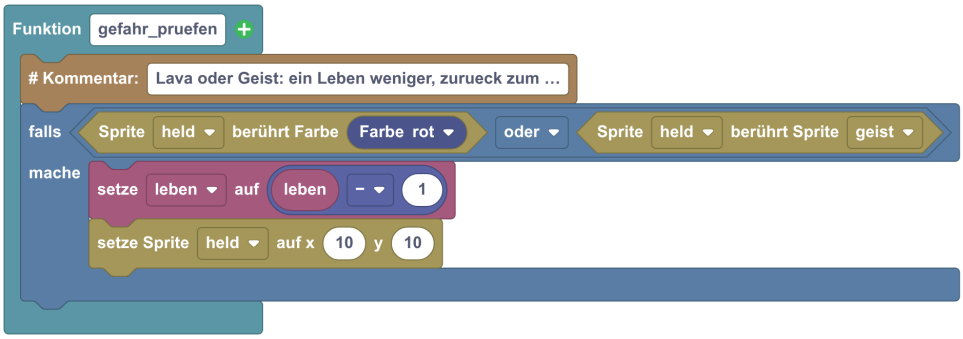
- `leben` startet bei 3.
- `richtung` ist die Geschwindigkeit des Geists: 3 Pixel pro Schritt nach rechts.
- Der Geist startet am linken Rand bei y = 150 und wird gezeigt.



Die Funktion `geist_bewegen` lässt den Geist hin und her schweben wie in Kapitel 13. Neu: Nach dem Umdrehen wird er gleich einen Schritt zurück versetzt. So steckt er nie im Rand fest.

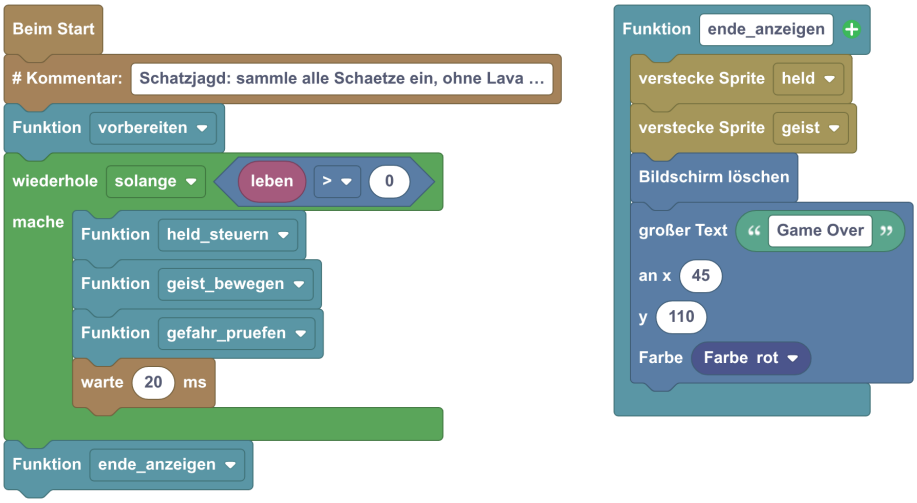


Die Funktion `gefahr_pruefen` stellt **eine** Frage mit „oder“: Berührt der Held die Farbe Rot **oder** den Geist? Dann kostet es ein Leben, und der Held muss zurück zum Start.



> Die Lava ist im Hintergrundbild `hintergrund_jagd.png` in genau dem Rot gemalt, das auch „Farbe rot“ hat (Rot 255, Grün 0, Blau 0). Nur deshalb erkennt „berührt Farbe rot“ sie. Malst du ein eigenes Spielfeld, achte auf genau diese Farbe (Kapitel 13).

Die Schleife unter „Beim Start“ läuft jetzt nicht mehr unendlich, sondern **solange `leben` > 0** ist. Danach wird `ende_anzeigen` aufgerufen. Fürs Erste zeigt diese Funktion nur „Game Over“:



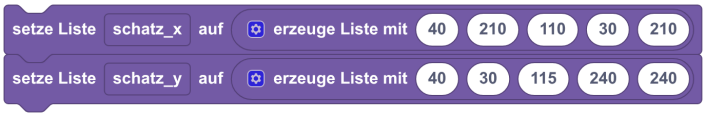


Alle Leben weg

Datei: `k20_schritt2.pbcp`.

20.4 Schritt 3: Schätze aus Listen

Die fünf Schätze sollen an festen Stellen liegen, die du dir ausgedacht hast. Jede Stelle hat eine x- und eine y-Koordinate. Statt zehn Variablen nimmst du **zwei Listen** (Kapitel 15): `schatz_x` für alle x-Werte und `schatz_y` für alle y-Werte. Lege sie in der Kategorie **Listen** mit „+ neue Liste…“ an.



Den Block „erzeuge Liste mit“ bekommst du über „Liste aus Werten“. Er hat zuerst drei Plätze; mit dem Zahnrad ⚙ fügst du weitere hinzu.

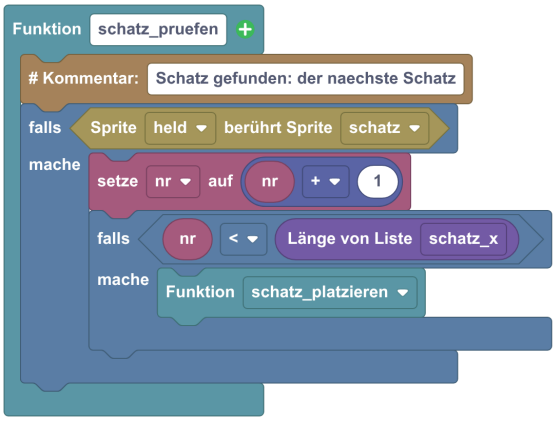
Die Listen gehören zusammen: Schatz Nr. 0 liegt bei x = 40 und y = 40, Schatz Nr. 1 bei x = 210 und y = 30 und so weiter. Gezählt wird ab **0**.

Nr.	0	1	2	3	4
`schatz_x`	40	210	110	30	210
`schatz_y`	40	30	115	240	240

Welcher Schatz gerade dran ist, merkt sich die Variable `nr`. Sie startet bei 0. Die Funktion `schatz_platzieren` holt die passenden Koordinaten aus den beiden Listen:

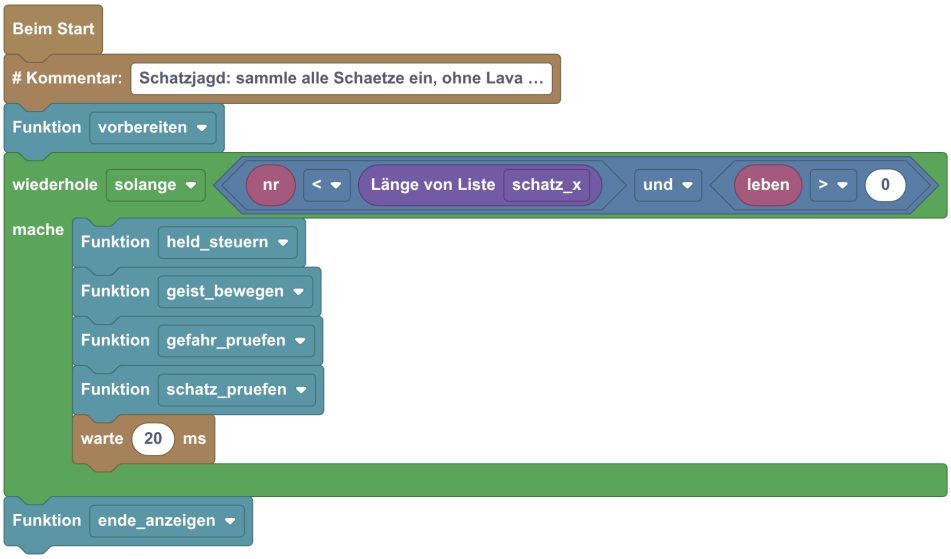


Die Funktion `schatz_pruefen` fragt: Hat der Held den Schatz berührt? Dann wird `nr` um 1 größer, und der nächste Schatz wird platziert.

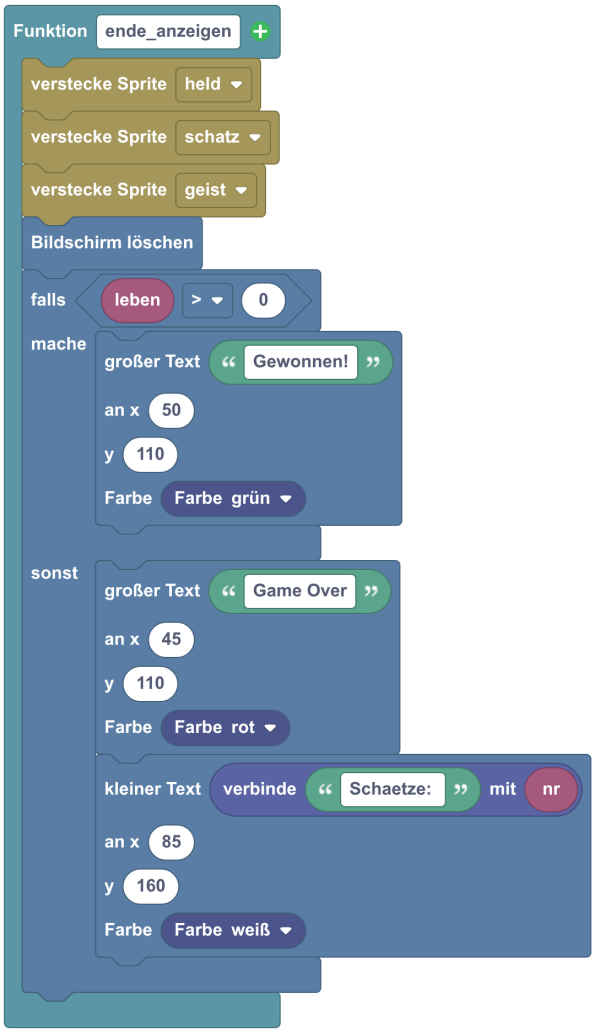


> **Warum das innere „falls“?** Nach dem fünften Schatz ist `nr` gleich 5. Einen Schatz Nr. 5 gibt es aber nicht – die Listen haben nur die Plätze 0 bis 4. Ohne die Frage „nr < Länge von Liste schatz_x“ würde `schatz_platzieren` nach Element Nr. 5 fragen, und das Programm bräche mit einem Fehler ab.

In `vorbereiten` kommen die beiden Listen, `nr` auf 0, der Aufruf von `schatz_platzieren` und „zeige Sprite schatz“ dazu (siehe das vollständige Bild in 20.5). Die Schleife unter „Beim Start“ bekommt eine Bedingung mit „und“: Sie läuft, solange noch Schätze übrig sind **und** der Held noch Leben hat.



Weil die Schleife jetzt auf zwei Arten enden kann, fragt `ende_anzeigen` nach, welche es war: Hat der Held noch Leben, hat er gewonnen.

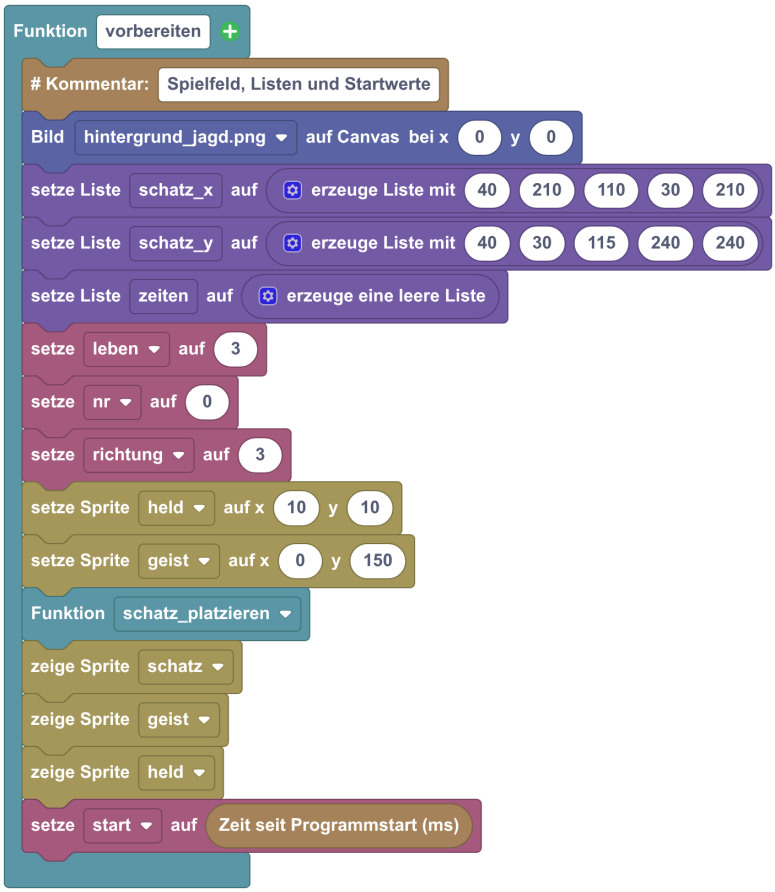


Datei: `k20_schritt3.pbcp`. Das Spiel ist jetzt schon spielbar!

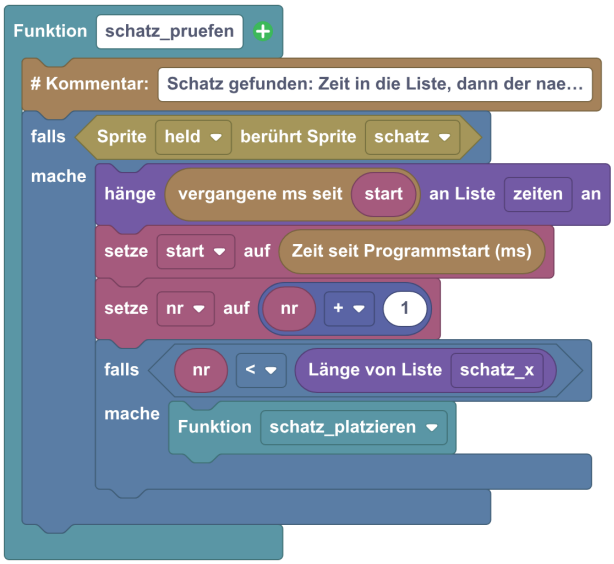
20.5 Schritt 4: Zeiten stoppen

Zum Schluss soll das Spiel für jeden Schatz die Zeit messen. Die Zeiten entstehen erst während des Spiels. Deshalb startet die Liste `zeiten` leer, und jede neue Zeit wird hinten **angehängt**.

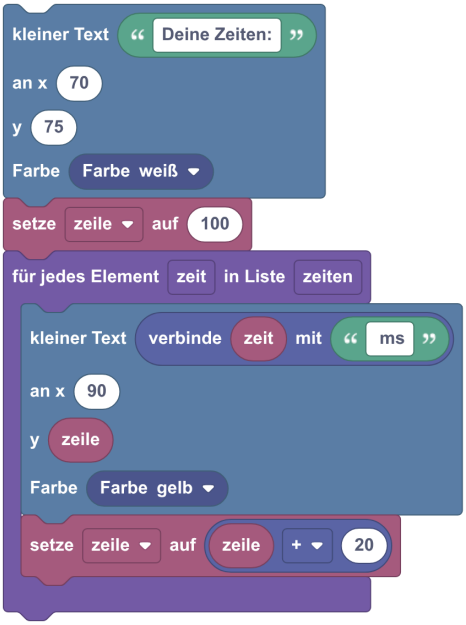
Die Zeit misst du wie beim Reaktionstest (Kapitel 19): Die Variable `start` merkt sich die Uhrzeit. So sieht `vorbereiten` jetzt vollständig aus:



In `schatz_pruefen` kommen zwei Blöcke dazu: Die vergangene Zeit wird an die Liste `zeiten` angehängt, und `start` beginnt von vorn – für den nächsten Schatz.



Am Ende zeigt `ende_anzeigen` alle Zeiten untereinander. Dafür schiebst du „Gewonnen!“ nach oben (y = 30) und hängst darunter diese Blöcke an:





Gewonnen – mit allen Zeiten

„für jedes Element `zeit` in Liste `zeiten`“ (Kapitel 15) geht die Liste von vorn bis hinten durch. Bei jedem Durchlauf steht in `zeit` ein anderer Wert. Die Variable `zeile` sorgt dafür, dass jede Zeit 20 Pixel tiefer steht als die vorige.

Ausprobieren: Das ganze Spiel

Das fertige Spiel liegt als `k20_schatzjagd.pbcp` im Ordner `buch/programme`. Unter „Beim Start“ steht nur der Plan – genau wie in Schritt 3. Die ganze Arbeit erledigen die sieben Funktionen. Spiele ein paar Runden und notiere deine Zeiten: Bei welchem Schatz brauchst du am längsten? Warum? Ordne die acht Blockstapel auf der Arbeitsfläche nebeneinander an, mit „Beim Start“ ganz links – dann findest du jede Funktion schnell wieder.

20.6 Erweiterungsideen

Jetzt ist es **dein** Spiel. Ein paar Ideen – die ersten stehen auch in den Übungsaufgaben:

- **Neue Schatzkarte:** andere Stellen, mehr Schätze, ein eigenes Spielfeld mit **Dateien → Bild zeichnen…**
- **Leben anzeigen** mit den drei Status-LEDs. (Ein Text direkt auf dem Bildschirm hilft hier nicht: Er verschwindet, sobald sich ein Sprite bewegt.)
- **Mehr Gegner:** ein zweiter Geist, der auf und ab schwebt.
- **Schwieriger werden:** Der Geist wird nach jedem Schatz schneller.
- **Statistik:** Gesamtzeit oder schnellster Schatz am Ende.
- **Kippsteuerung:** Steuere den Helden durch Kippen (Kapitel 18). Du musst nur die Funktion `held_steuern` austauschen – das ist der große Vorteil von Funktionen.

- **Sound:** ein Ton beim Schatz, ein tiefer Ton bei der Lava, eine kleine Melodie beim Sieg (Kapitel 9).

Übungsaufgaben

Aufgabe 1 ⭐ **Ein sechster Schatz** Füge einen sechsten Schatz bei x = 120 und y = 205 hinzu. Was musst du ändern? Muss die Schleife unter „Beim Start“ auch geändert werden?

Aufgabe 2 ⭐ **Leben-Anzeige** Am Anfang leuchten alle drei Status-LEDs. Bei jedem verlorenen Leben geht eine aus: zuerst die grüne, dann die gelbe, dann die rote.

Aufgabe 3 ⭐⭐ **Gesamtzeit** Unter den fünf Zeiten soll „Gesamt: … ms“ stehen – die Summe aller Zeiten. *Tipp:* Noch eine „für jedes Element“-Schleife und eine Variable `summe`.

Aufgabe 4 ⭐⭐ **Der Geist wird schneller** Nach jedem gefundenen Schatz soll der Geist um 1 Pixel pro Schritt schneller werden. Achtung: `richtung` kann positiv oder negativ sein!

Aufgabe 5 ⭐⭐⭐ **Ein zweiter Geist** Ein zweiter Geist `geist2` (Bild `geist.png`) startet bei x = 130 oben am Rand und schwebt auf und ab. Auch er kostet ein Leben.

Aufgabe 6 ⭐⭐⭐ **Der schnellste Schatz** Zeige unter den Zeiten an, wie schnell dein schnellster Schatz war. *Tipp:* „Minimum von … und …“ aus Kapitel 17 hilft.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Ein großes Spiel planst du zuerst: Welche Aufgaben gibt es? Jede Aufgabe wird eine **Funktion** mit einem sprechenden Namen.
- „Beim Start“ enthält nur noch den Plan: vorbereiten, Spielschleife, Ende.
- Baue in **Schritten** und teste nach jedem Schritt.
- Zwei Listen, die zusammengehören (`schatz_x` und `schatz_y`), speichern Positionen. Eine Variable (`nr`) zeigt auf den aktuellen Platz – gezählt wird ab 0.
- Mit „hänge … an Liste an“ sammelst du Werte, deren Anzahl du vorher nicht kennen musst, z. B. Zeiten.
- Mit „und“ / „oder“ verbindest du Bedingungen: Die Schleife läuft, solange Schätze übrig sind **und** Leben da sind.
- Prüfe vor dem Lesen aus einer Liste, ob es den Platz überhaupt gibt.

KAPITEL 21

Ein eigenes Spiel entwickeln

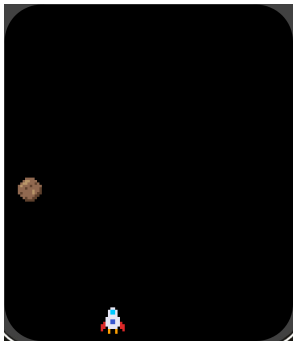
Worum geht's? Du kennst jetzt fast alle Blöcke. In diesem Kapitel lernst du, wie aus einer Idee ein fertiges Spiel wird: erst planen, dann das Spiel in Funktionen zerlegen, Stück für Stück bauen und testen – und Fehler finden, ohne zu verzweifeln. Als Beispiel entsteht das Spiel „Meteoritenregen".

21.1 Von der Idee zum Plan

Viele fangen sofort an, Blöcke zu ziehen. Nach einer Stunde ist das Programm riesig, nichts klappt, und keiner weiß mehr, welcher Block wofür war. Besser: Du nimmst dir zehn Minuten Zeit und **planst**. Dafür beantwortest du ein paar Fragen auf einem **Planungsbogen**.

Die Idee: Ein Raumschiff fliegt unten am Bildschirm. Von oben fallen Meteoriten. Das Schiff muss ausweichen. Jeder Meteorit, der unten ankommt, bringt einen Punkt. Wird das Schiff getroffen, ist ein Leben weg. Die drei Leben zeigen die drei Status-LEDs an.

So sieht der ausgefüllte Planungsbogen aus:



Meteoritenregen im Emulator

Frage	Meteoritenregen
Worum geht es?	Ein Raumschiff weicht Meteoriten aus, die von oben fallen.
Figuren (Sprites)	`schiff` (Bild `raumschiff.png`), `meteor` (Bild `meteor.png`)
Steuerung	Joystick links und rechts bewegen das Schiff.
Wie endet das Spiel?	Wenn alle 3 Leben weg sind. Dann erscheinen „Game Over" und die Punkte.
Variablen	`leben` (Start: 3), `punkte` (Start: 0), `tempo` (Start: 4 – so viele Pixel fällt der Meteor pro Runde)
Muss (das baue ich zuerst)	Schiff steuern, Meteor fällt, Treffer kostet ein Leben, Spielende mit Punkten
Kann (wenn noch Zeit ist)	Meteor wird schneller, Pause-Taste, Bonus-Stern

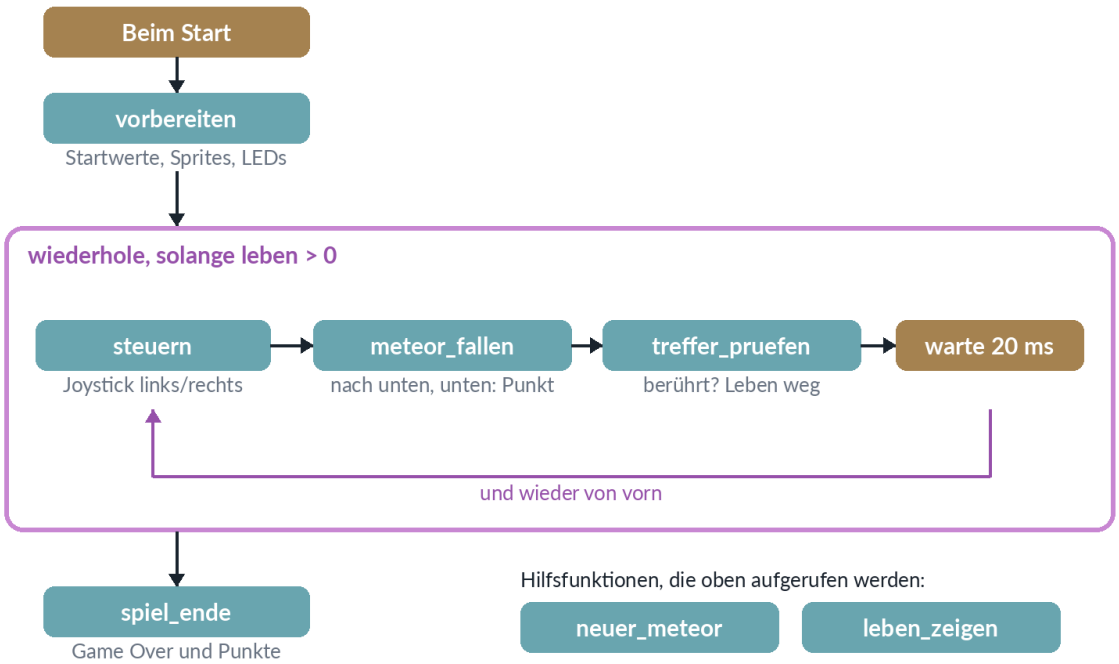
Das Wichtigste sind die **Regeln**. Jede Regel hat die Form „Wenn …, dann …". Später wird aus jeder Regel ein „falls"-Block.

Wenn …	dann …
der Joystick nach links gedrückt ist und das Schiff nicht am linken Rand steht,	fliegt das Schiff 5 Pixel nach links.
der Joystick nach rechts gedrückt ist und das Schiff nicht am rechten Rand steht,	fliegt das Schiff 5 Pixel nach rechts.
der Meteor unten aus dem Bildschirm gefallen ist (y größer als 280),	gibt es einen Punkt, und ein neuer Meteor startet oben an einer zufälligen Stelle.
das Schiff den Meteor berührt,	ist ein Leben weg, ein tiefer Ton erklingt, eine LED geht aus, ein neuer Meteor startet.
keine Leben mehr übrig sind,	ist das Spiel vorbei.

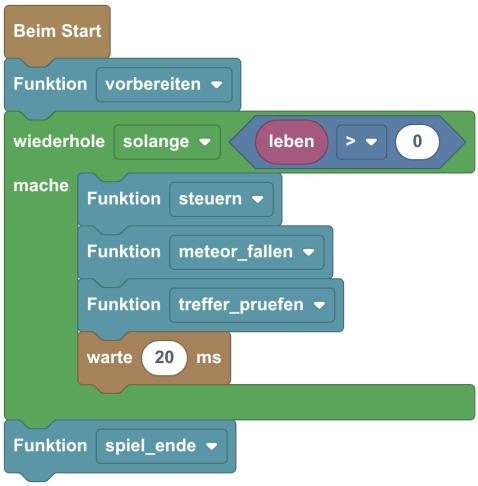
> **Tipp: Klein anfangen.** Schreib in die Zeile **Muss** nur das, ohne das dein Spiel gar nicht funktioniert. Alle schönen Extras kommen unter **Kann**. Ein kleines Spiel, das läuft, ist viel besser als ein großes, das nie fertig wird.

21.2 Das Spiel in Funktionen zerlegen

Jetzt überlegst du, aus welchen **Teilen** das Programm besteht. Jeder Teil wird eine eigene Funktion (Kapitel 14) mit einem Namen, der sagt, was sie tut. Ein Spiel hat fast immer denselben Aufbau: vorbereiten – dann immer wieder steuern und Regeln prüfen – und am Schluss das Ende zeigen.

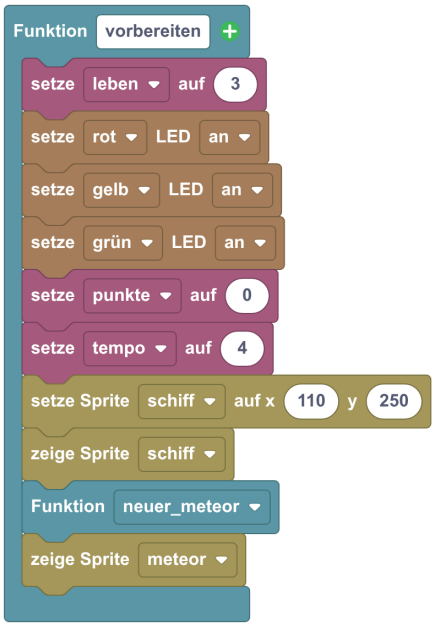


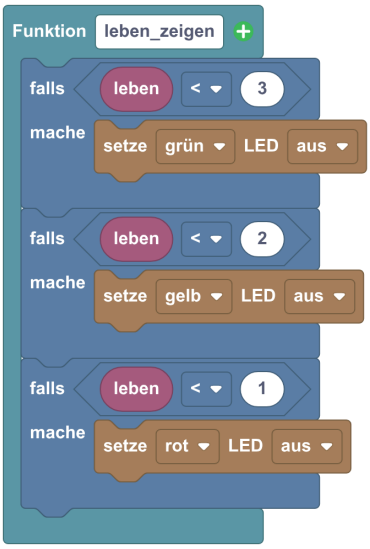
Das Hauptprogramm unter „Beim Start“ wird dadurch ganz kurz. Es liest sich wie ein Inhaltsverzeichnis:



Jeder Block „Funktion …“ ist ein **Aufruf**: Hier laufen die Blöcke der Funktion, danach geht es mit dem nächsten Block weiter. Die Schleife läuft, solange `leben` größer als 0 ist. Dann geht es unten mit `spiel_ende` weiter.

Und so sehen die Funktionen aus. **vorbereiten** setzt alle Startwerte. Die drei LEDs gehen an – sie zeigen die Leben.





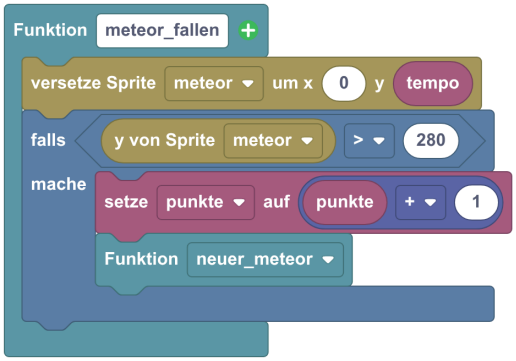
leben_zeigen (rechts) schaltet die LEDs passend zu `leben` aus: Bei 2 Leben geht Grün aus, bei 1 Leben auch Gelb, bei 0 Leben auch Rot.

steuern bewegt das Schiff. Mit „und“ (Kapitel 7) prüft jeder „falls“-Block zwei Dinge: Ist die Taste gedrückt? Und ist das Schiff noch nicht am Rand? Das Schiff ist 20 Pixel breit, deshalb ist bei x = 220 rechts Schluss.

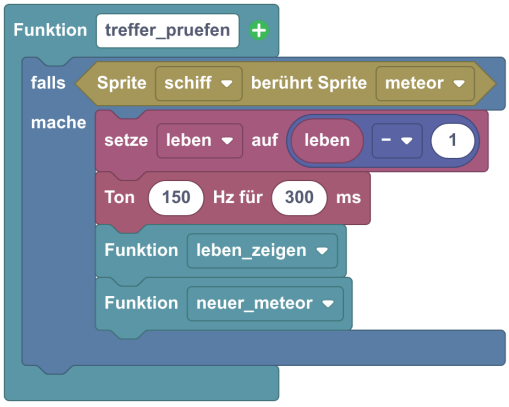


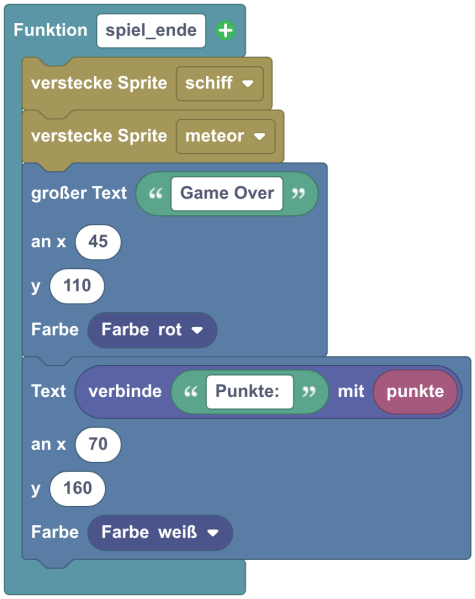
neuer_meteor setzt den Meteor oben an eine zufällige Stelle. **meteor_fallen** schiebt ihn um `tempo` Pixel nach unten. Ist er unten herausgefallen, gibt es einen Punkt, und ein neuer Meteor startet.





treffer_pruefen fragt, ob sich Schiff und Meteor berühren (Kapitel 13). **spiel_ende** versteckt beide Sprites und zeigt das Ergebnis.





> **Warum werden die Leben mit LEDs angezeigt und nicht als Text?** Sobald sich ein Sprite bewegt, malt PBC+ Studio den ganzen Bildschirm neu: erst den Hintergrund, dann alle Sprites. Ein Text, den du mit „Text … an x … y …“ direkt auf den Bildschirm geschrieben hast, ist dabei sofort wieder weg. Text während des Spiels gehört deshalb auf den **Canvas** (Kapitel 11) – der wird beim Neumalen mitgezeichnet. Am Ende sind die Sprites versteckt, dann bleibt auch normaler Text stehen.

21.3 Schrittweise bauen und testen

Du baust nicht das ganze Spiel auf einmal, sondern in **Stufen**. Nach jeder Stufe testest du. Läuft alles, speicherst du die Stufe als eigene Datei (z. B. `meteor_stufe1.pbcp`). Geht später etwas kaputt, hast du einen funktionierenden Stand, zu dem du zurückkannst.

Stufe 1 – Das Schiff fliegt. Nur `vorbereiten` (mit dem Schiff) und `steuern`. Noch reicht „unendlich wiederholen“.



Stufe 2 – Der Meteor fällt. Dazu kommen `neuer_meteor` und `meteor_fallen` (noch ohne Punkte). In `vorbereiten` kommen `tempo`, „Funktion neuer_meteor“ und „zeige Sprite meteor“ dazu.

Stufe 3 – Treffer kosten Leben. Jetzt kommen `leben`, die LEDs, `treffer_pruefen` und `leben_zeigen` dazu. „unendlich wiederholen“ wird zu „wiederhole solange leben > 0“.

Stufe 4 – Punkte und Spielende. `punkte` zählen und `spiel_ende` aufrufen – fertig ist das Spiel aus 21.2.

Die Stufen liegen als `k21_stufe1.pbcp` bis `k21_stufe3.pbcp` im Ordner `buch/programme`.

Testen mit Plan

Testen heißt nicht nur „mal spielen“. Überlege vorher: **Was soll passieren?** Dann probierst du es aus und schreibst auf, was **wirklich** passiert. Teste besonders die Ränder und die Sonderfälle. So kann ein **Testprotokoll** für Stufe 3 aussehen:

Was teste ich?	Was soll passieren?	Was passiert?	ok?
Joystick lange nach links halten	Schiff bleibt am linken Rand stehen	Schiff bleibt stehen	✓
Joystick lange nach rechts halten	Schiff bleibt am rechten Rand stehen	Schiff bleibt stehen	✓
Meteor fällt unten heraus	Neuer Meteor oben, an anderer Stelle	Neuer Meteor oben	✓
Schiff wird einmal getroffen	1 Leben weg, grüne LED aus	**alle LEDs aus, Spiel vorbei**	✗

Der letzte Test ist schiefgegangen. Gut so – jetzt weißt du, wo du suchen musst. Wie man so einen Fehler findet, zeigt der nächste Abschnitt.

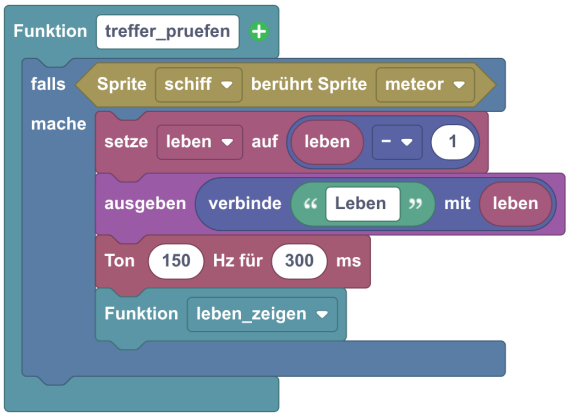
> **Tipp:** Lass jemand anderen dein Spiel testen und schau nur zu, ohne etwas zu erklären. Andere drücken Tasten, an die du nie gedacht hättest.

21.4 Fehler finden und beheben

Fehler gehören zum Programmieren dazu – auch Profis machen ständig welche. Wichtig ist, dass du sie geschickt suchst. Das geht in vier Schritten:

1. **Genau beschreiben:** Was passiert? Wann passiert es? Kann ich es wiederholen?
2. **Vermuten:** Welche Funktion ist dafür zuständig? Was könnte dort falsch sein?
3. **Nachsehen:** Lass dir Werte anzeigen, statt zu raten.
4. **Eine Sache ändern** und wieder testen. Hat es nicht geholfen, mach die Änderung rückgängig.

Ein Beispiel: Im Testprotokoll oben sind bei einem einzigen Treffer alle Leben auf einmal weg. Zuständig ist `treffer_pruefen`. In dieser fehlerhaften Fassung fehlt etwas. Damit du siehst, was mit `leben` passiert, steckst du einen Block „ausgeben“ (Kategorie **Konsole**) hinein. Er schreibt den Wert ins Terminal:



Beim Test erscheint im Terminal „Leben 2“, „Leben 1“, „Leben 0“ – alle drei in weniger als einer Sekunde. Das heißt: Der „falls“-Block war mehrmals hintereinander wahr. Der Grund: Nach dem Treffer fällt der Meteor einfach weiter und berührt das Schiff auch in der nächsten und übernächsten Runde der Schleife. Die Lösung: Nach einem Treffer muss sofort ein **neuer Meteor** oben starten. Dafür fehlt „Funktion neuer_meteor“ am Ende des „falls“-Blocks – in der richtigen Fassung in 21.2 steht er da. Danach kannst du den Block „ausgeben“ wieder löschen.

Weitere Spuren, wenn du nicht weiterkommst:

Das passiert	Mögliche Ursache	Das hilft
Ein Sprite ist nicht zu sehen.	„zeige Sprite“ fehlt, oder das Sprite steht außerhalb des Bildschirms.	In `vorbereiten` nachsehen; x und y prüfen.
Eine Funktion tut gar nichts.	Sie wird nirgends aufgerufen.	Block „Funktion …“ an der richtigen Stelle einbauen.
Ein Wert ändert sich nie.	Er wird in der Schleife jedes Mal neu auf den Startwert gesetzt.	Startwerte gehören nach `vorbereiten`, nicht in die Schleife.
Text verschwindet sofort wieder.	Die Sprites malen den Bildschirm neu.	Text auf den Canvas schreiben oder erst nach dem Verstecken der Sprites.
Statt „ä“ steht „?“ auf dem Bildschirm.	Die Schrift auf dem PicoBoy kennt keine Umlaute.	„ae“, „oe“, „ue“, „ss“ schreiben.

Was die Meldungen von PBC+ Studio bedeuten, steht in **Anhang B**.

Ausprobieren: Meteoritenregen

Das ganze Spiel besteht aus dem Hauptprogramm aus 21.2 und den sieben Funktionen `vorbereiten`, `steuern`, `neuer_meteor`, `meteor_fallen`, `treffer_pruefen`, `leben_zeigen` und `spiel_ende`. Du findest es als `k21_meteoritenregen.pbcp` im Ordner `buch/programme`.

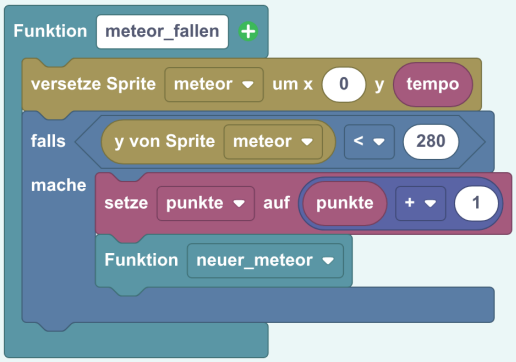
So bekommst du es zum Laufen:

1. Lade die Bilder `raumschiff.png` und `meteor.png` hoch: **Dateien → Bilder verwalten…**
2. Öffne die Datei. Die Sprites `schiff` und `meteor` sind darin schon angelegt.
3. Klicke auf ▶ **Testen** und weiche mit den Pfeiltasten aus. Achte auf die drei LEDs links neben dem Bildschirm.

Probiere danach: Was ändert sich, wenn du `tempo` in `vorbereiten` auf 8 setzt? Was passiert, wenn das Schiff in `steuern` 10 statt 5 Pixel weit fliegt?

Übungsaufgaben

Aufgabe 1 ⭐ **Finde den Fehler** In dieser Fassung von `meteor_fallen` steckt ein Fehler. Was passiert beim Spielen? Erkläre, warum, und repariere die Funktion.



Aufgabe 2 ⭐ **Plane „Fang den Apfel“** Äpfel fallen von einem Baum, ein Korb unten fängt sie. Nach 10 Äpfeln hast du gewonnen, nach 3 verpassten Äpfeln verloren. Fülle für dieses Spiel einen Planungsbogen aus: Figuren, Steuerung, Variablen, Ende und mindestens vier „Wenn …, dann …“-Regeln. Welche Funktionen brauchst du?

Aufgabe 3 ⭐⭐ **Immer schneller** Nach je 5 Punkten soll der Meteor 1 Pixel pro Runde schneller fallen. *Tipp:* Mit „Rest“ (Kapitel 6) findest du heraus, ob `punkte` durch 5 teilbar ist.

Aufgabe 4 ⭐⭐ **Pause!** Mit Taste A hält das Spiel an, und „Pause“ erscheint. Mit Taste B geht es weiter. *Tipp:* „wurde … seit der letzten Abfrage gedrückt?“ und „auf … warten“ (Kapitel 8).

Aufgabe 5 ⭐⭐⭐ **Der Bonus-Stern** Ab und zu fällt langsam ein Stern (`stern.png`). Fängt das Schiff ihn, bekommt es ein Leben zurück – aber höchstens bis 3 – und die passende LED geht wieder an. Schreibe dafür eine neue Funktion `stern_pruefen` und ändere `leben_zeigen`.

Aufgabe 6 ⭐⭐⭐ **Dein eigenes Spiel** Erfinde ein eigenes Spiel. Fülle zuerst den Planungsbogen aus. Baue es dann in Stufen und schreibe zu jeder Stufe ein kleines Testprotokoll. Stelle dein Spiel zum Schluss jemandem vor.

Die Lösungen stehen in Anhang C.

Das Wichtigste in Kürze

- Erst **planen**, dann bauen: Figuren, Steuerung, Regeln, Ende, Variablen.
- Jede Regel hat die Form **„Wenn …, dann …“** – daraus wird ein „falls“-Block.
- Zerlege das Spiel in **Funktionen** mit sprechenden Namen. Das Hauptprogramm liest sich dann wie ein Inhaltsverzeichnis.
- Baue in **Stufen** und teste nach jeder Stufe. Speichere funktionierende Stände.
- Teste mit Plan: Was **soll** passieren, was passiert **wirklich**?
- Bei Fehlern: beschreiben, vermuten, mit „ausgeben“ nachsehen, eine Sache ändern.

ANHANG A

Blockreferenz

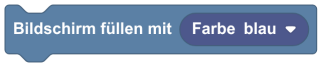
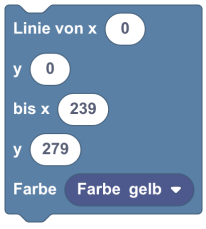
Hier findest du alle Blöcke von PBC+ Studio – in derselben Reihenfolge wie in der Blockleiste. Zu jedem Block steht in einem Satz, was er macht, und in welchem Kapitel er erklärt wird (letzte Spalte). Die Zahlen in den Blöcken sind nur Beispiele.

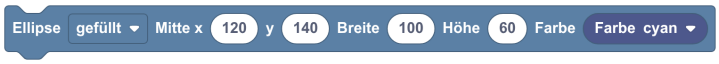
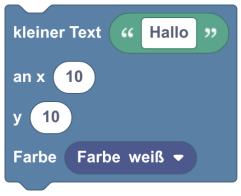
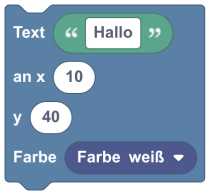
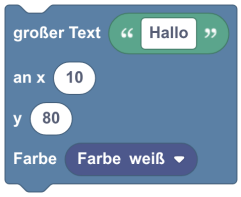
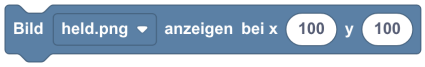
> **So liest du die Tabelle:** Blöcke mit Nase oben und Kerbe unten sind *Befehle* – du hängst sie untereinander. Blöcke mit einem Stecker links sind *Werte* (eine Zahl, ein Text oder eine Farbe), Blöcke mit spitzen Enden sind *Fragen* (Antwort wahr oder falsch). Werte und Fragen steckst du in die Lücken anderer Blöcke.

Die Kategorie **Externe Sensoren** erscheint nur, wenn in den Einstellungen ein zusätzlicher Sensor eingeschaltet ist. Dieses Buch braucht sie nicht.

Display

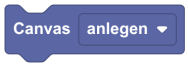
Alles, was direkt auf den Bildschirm zeichnet.

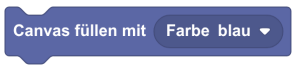
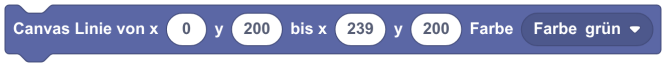
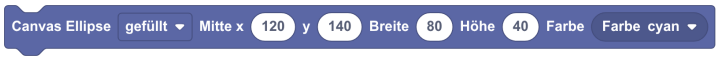
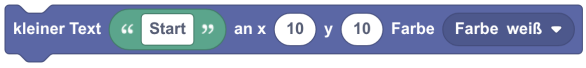
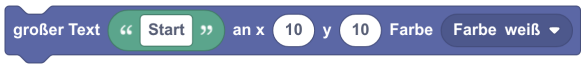
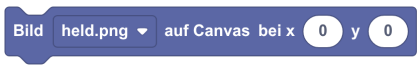
	Macht den ganzen Bildschirm schwarz.	2
	Füllt den ganzen Bildschirm mit einer Farbe.	3
	Eine Farbe zum Einstecken: weiß, schwarz, rot, grün, blau, gelb, cyan, magenta, grau, orange oder lila.	3
	Färbt einen einzigen Bildpunkt an der Stelle x, y.	3
	Zeichnet eine gerade Linie von einem Punkt zu einem anderen.	3
	Zeichnet ein Rechteck – als Umriss oder gefüllt. x, y ist die Ecke oben links, b und h sind Breite und Höhe.	3

	Zeichnet eine Ellipse (ein „Ei“) um den Mittelpunkt x, y mit der angegebenen Breite und Höhe.	3
	Zeichnet einen Kreis um den Mittelpunkt x, y mit dem angegebenen Radius.	3
	Schreibt einen Text in kleiner Schrift an die Stelle x, y. Zahlen und Variablen gehen auch.	2
	Wie „kleiner Text“, aber in mittlerer Schriftgröße.	2
	Wie „kleiner Text“, aber in großer Schrift.	2
	Die Breite (240) oder Höhe (280) des Bildschirms als Zahl.	3
	Zeigt ein hochgeladenes Bild; x, y ist seine Ecke oben links.	3
	Mischt eine Farbe aus Rot, Grün und Blau (je 0 bis 255). Steht auch unter Canvas, Turtle und LEDs.	3
	Eine Farbe aus dem Farbwähler: Klicke auf das Farbfeld und wähle. Steht auch unter Canvas, Turtle und LEDs.	3

Canvas

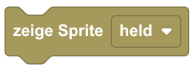
Der Canvas ist ein Hintergrundbild im Speicher. Man sieht ihn erst nach „Canvas zeigen“; Sprites laufen davor.

	„anlegen“ macht einen neuen, schwarzen Canvas, „leeren“ malt den vorhandenen schwarz an.	11
	Bringt den Canvas auf den Bildschirm.	11

	Füllt den ganzen Canvas mit einer Farbe.	11
	Färbt einen Bildpunkt auf dem Canvas.	11
	Zeichnet eine Linie auf den Canvas.	11
	Zeichnet ein Rechteck auf den Canvas – gefüllt oder als Umriss.	11
	Zeichnet eine Ellipse auf den Canvas.	11
	Zeichnet einen Kreis auf den Canvas.	11
	Schreibt kleinen Text auf den Canvas.	11
	Schreibt mittelgroßen Text auf den Canvas.	11
	Schreibt großen Text auf den Canvas.	11
	Legt ein hochgeladenes Bild auf den Canvas, z. B. ein ganzes Hintergrundbild. Durchsichtige Stellen bleiben durchsichtig.	11

Sprites

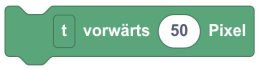
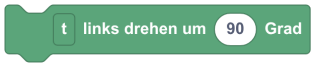
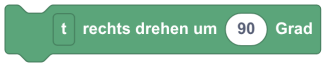
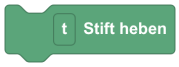
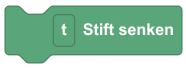
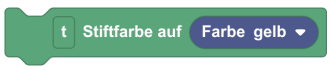
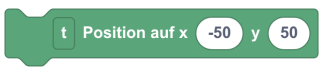
Sprites sind Bilder, die sich bewegen. Lege sie zuerst mit „+ neues Sprite …“ an. x, y ist immer die Ecke oben links des Bildes.

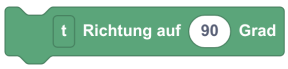
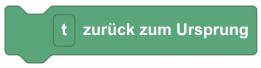
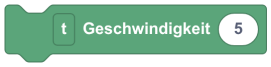
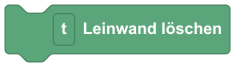
	Macht das Sprite sichtbar.	12
	Macht das Sprite unsichtbar.	12
	Setzt das Sprite sofort an die Stelle x, y.	12
	Schiebt das Sprite um so viele Pixel weiter (minus = nach links bzw. oben).	12
	Lässt das Sprite in der angegebenen Zeit zur Stelle x, y gleiten. Das Programm wartet, bis es angekommen ist.	12

	Lässt das Sprite um so viele Pixel weitergleiten. Auch hier wartet das Programm.	12
	Die x-Position des Sprites als Zahl.	12
	Die y-Position des Sprites als Zahl.	12
	Wahr, wenn sich die Rechtecke der beiden Sprite-Bilder überlappen. Achtung: Neu geholt steht in beiden Feldern dasselbe Sprite.	13
	Wahr, wenn ein Teil des Sprites außerhalb des Bildschirms liegt.	13
	Wahr, wenn ein sichtbarer Pixel des Sprites auf genau dieser Farbe des Canvas liegt.	13

Turtle

Die Turtle ist eine Schildkröte mit Stift. (0, 0) ist die Bildschirmmitte, Richtung 0 zeigt nach oben. Zuerst brauchst du „neue Turtle“.

	Erzeugt eine Turtle mit Namen (hier `t`). „neuer Hintergrund“ macht vorher alles schwarz.	10
	Die Turtle läuft vorwärts und zeichnet dabei, wenn der Stift unten ist.	10
	Die Turtle läuft rückwärts.	10
	Die Turtle dreht sich nach links.	10
	Die Turtle dreht sich nach rechts.	10
	Hebt den Stift: Die Turtle läuft, ohne zu zeichnen.	10
	Senkt den Stift: Die Turtle zeichnet wieder.	10
	Wählt die Farbe des Stifts.	10
	Stellt die Turtle an die Stelle x, y (Mitte = 0, 0; y zählt nach oben).	10

	Dreht die Turtle in eine feste Richtung: 0 = oben, 90 = rechts.	10
	Schickt die Turtle zurück in die Bildschirmmitte.	10
	Wie schnell die Turtle zeichnet: 0 = sofort, 1 = sehr langsam, 10 = schnell.	10
	Versteckt das Dreieck, das die Turtle zeigt.	10
	Zeigt das Turtle-Dreieck wieder.	10
	Löscht alles, was gezeichnet wurde (der Hintergrund wird schwarz).	10
	Wahr, wenn die Stelle direkt vor der Turtle genau diese Farbe hat.	10

LEDs

Die drei Status-LEDs und die bunte RGB-LED.

	Schaltet die rote, gelbe oder grüne LED an oder aus.	9
	Schaltet alle drei Status-LEDs aus.	9
	Lässt die RGB-LED in einer gemischten Farbe leuchten (je 0 bis 255).	9
	Lässt die RGB-LED in einer Farbe aus „Farbe …“ oder dem Farbwähler leuchten.	9
	Schaltet die RGB-LED aus.	9

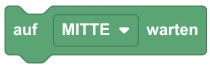
Sound

Töne aus dem Lautsprecher. Die Zahl in Hz ist die Tonhöhe: je größer, desto höher.

	Spielt einen Ton für so viele Millisekunden. Das Programm läuft dabei sofort weiter – für eine Melodie brauchst du „warte“.	9
	Startet einen Ton, der so lange klingt, bis „Ton stoppen“ kommt.	9
	Beendet den Ton.	9

Eingabe

Joystick und Tasten. Im Emulator: Pfeiltasten, Leertaste (MITTE), Taste A und Taste B.

	Wahr, solange die Taste gerade gedrückt ist.	8
	Wahr, wenn die Taste seit der letzten Frage gedrückt wurde – genau einmal pro Druck.	8
	Das Programm hält an, bis eine beliebige Taste gedrückt und wieder losgelassen wurde.	8
	Das Programm hält an, bis genau diese Taste gedrückt und wieder losgelassen wurde.	8

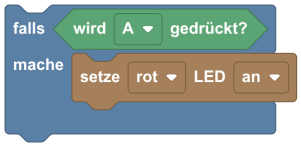
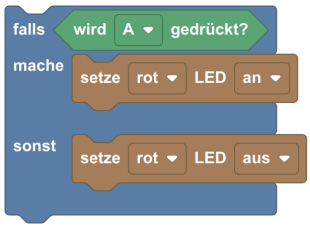
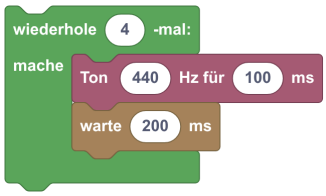
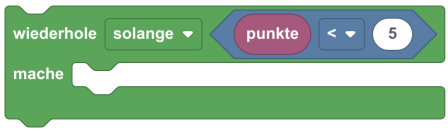
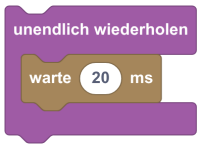
Sensor

Der Beschleunigungssensor spürt, wie du das Gerät hältst.

	Wie stark das Gerät in Richtung X, Y oder Z gekippt bzw. beschleunigt wird. Im Emulator liegt das Gerät immer flach (X = 0, Y = 0, Z = 1 g).	18

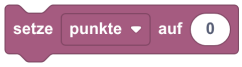
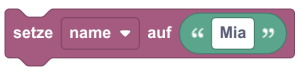
Kontrollstrukturen

Blöcke, die entscheiden und wiederholen.

	Führt die Blöcke darin nur aus, wenn die Bedingung wahr ist.	7
	Wenn die Bedingung wahr ist, laufen die Blöcke bei „mache“, sonst die bei „sonst“.	7
	Wiederholt die Blöcke darin so oft, wie die Zahl sagt.	5
	Wiederholt, solange die Bedingung wahr ist – oder („bis“) bis sie wahr wird.	7
	Zählt mit der Variable `i` von der ersten bis zur zweiten Zahl (die zweite gehört dazu) und wiederholt dabei.	5
	Wiederholt die Blöcke darin immer wieder, bis du Stop drückst. Gehört fast immer ein „warte“ hinein.	5

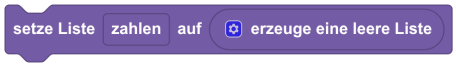
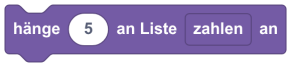
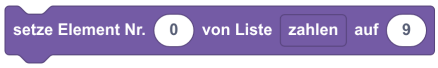
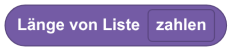
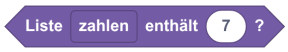
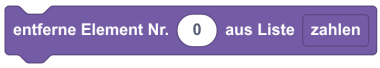
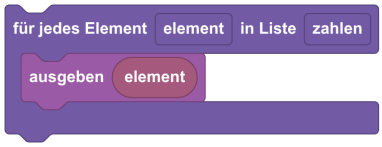
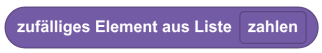
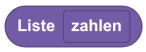
Variablen

Erst mit „+ neue Variable …“ (für Zahlen) oder „+ neuer Text …“ anlegen, dann erscheinen die Blöcke.

	Speichert einen Wert in der Variable. Der alte Wert ist danach weg.	6
	Der Wert, der gerade in der Variable steht. Klicke in der Leiste auf den Namen der Variable.	6
	„setze Text … auf …“: dasselbe für Text-Variablen.	6

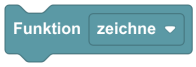
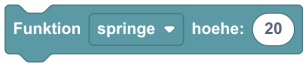
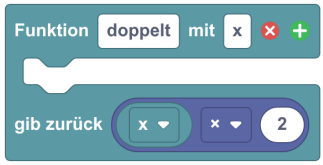
Listen

Erst mit „+ neue Liste …“ anlegen. Die Plätze einer Liste werden ab **0** gezählt.

	Macht die Liste leer – oder füllt sie mit einer „Liste aus Werten“.	15
	Eine fertige Liste mit Werten zum Einstecken in „setze Liste … auf“.	15
	Hängt einen Wert hinten an die Liste an.	15
	Der Wert auf diesem Platz der Liste (der erste Platz hat die Nummer 0).	15
	Ersetzt den Wert auf diesem Platz.	15
	Wie viele Werte in der Liste stehen.	15
	Wahr, wenn der Wert irgendwo in der Liste steht.	15
	Nimmt den Wert auf diesem Platz heraus; die Werte dahinter rücken auf.	15
	Nimmt den ersten passenden Wert heraus. Steht er nicht in der Liste, passiert nichts.	15
	Wiederholt die Blöcke darin für jeden Wert der Liste; `element` ist dabei der aktuelle Wert.	15
	Ein zufällig gewählter Wert aus der Liste.	15
	Die ganze Liste als Wert, z. B. zum Ausgeben im Terminal. Klicke in der Leiste auf den Namen der Liste.	15

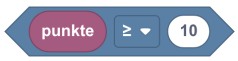
Funktionen

Eigene Blöcke: mit „+ neue Funktion …“ oder „+ neue Funktion mit Rückgabe …“ anlegen. Das grüne Plus fügt Parameter hinzu.

	Legt fest, was die Funktion tut. Sie läuft erst, wenn sie aufgerufen wird.	14
	Ruft die Funktion auf: Ihre Blöcke laufen jetzt, danach geht es hier weiter.	14
	Eine Funktion mit Parameter: Beim Aufruf gibst du einen Wert mit, den die Funktion benutzt.	14
	Der Aufruf mit dem Wert für den Parameter.	14
	Eine Funktion, die ein Ergebnis zurückgibt („gib zurück“). `x` ist ihr Parameter.	14
	Der Aufruf einer Funktion mit Rückgabe ist ein Wert-Block: Hier wird `punkte` verdoppelt.	14
	„lokale Variable“: eine Variable, die nur in dieser Funktion gilt.	14

Mathe

Zahlen, Rechnen und Vergleichen.

	Eine Zahl. Klicke hinein, um sie zu ändern.	6
	Rechnet +, −, × oder ÷. „Rest“ ist der Rest beim Teilen (7 Rest 2 ergibt 1). 7 ÷ 2 ergibt 3.5.	6
	Vergleicht zwei Werte: =, ≠, <, ≤, >, ≥. Die Antwort ist wahr oder falsch.	7
	„und“: Beide Bedingungen müssen wahr sein. „oder“: Eine reicht.	7

	Dreht die Antwort um: aus wahr wird falsch und umgekehrt.	7
	Der Wert wahr oder falsch.	7
	Eine zufällige ganze Zahl; beide Grenzen können vorkommen.	6
	Hält einen Wert zwischen zwei Grenzen: Ist er zu klein, kommt die untere Grenze heraus, ist er zu groß, die obere.	17
	Rechnet einen Wert von einem Bereich in einen anderen um, z. B. die Neigung −1 … 1 in x = 0 … 240. Das Ergebnis ist eine ganze Zahl.	17
	Die Quadratwurzel (aus 16 wird 4.0).	17
	Die Potenz: 2 hoch 3 ist 2 · 2 · 2 = 8.	17
	Der Betrag: die Zahl ohne Minuszeichen.	17
	Rundet auf eine ganze Zahl.	17
	Die kleinere (Minimum) oder größere (Maximum) von zwei Zahlen.	17

Text

Mit Texten arbeiten. Die Zeichen eines Textes werden ab 0 gezählt.

	Ein Text. Klicke hinein, um ihn zu ändern.	6
	Hängt zwei Texte aneinander. Zahlen werden dabei automatisch zu Text.	16
	Wie viele Zeichen der Text hat (bei „Hallo“: 5).	16
	Ein einzelnes Zeichen: Nummer 0 ist das erste („H“).	16

	Ein Stück des Textes: ab der ersten Nummer bis vor die zweite (aus „Hallo“ wird „all“).	16
	Der Text in GROSSBUCHSTABEN oder in kleinbuchstaben.	16
	Wahr, wenn der zweite Text im ersten vorkommt.	16

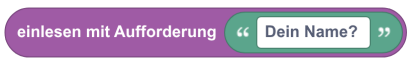
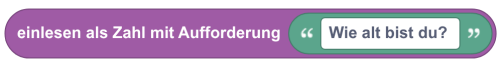
Zeit

Warten und Zeit messen. 1000 Millisekunden (ms) sind 1 Sekunde.

	Das Programm wartet so viele Millisekunden.	4
	Das Programm wartet so viele Sekunden.	4
	Wie viele Millisekunden seit dem Programmstart vergangen sind.	4
	Wie viele Sekunden seit dem Programmstart vergangen sind – mit Nachkommastellen.	4
	Wie viele Millisekunden vergangen sind, seit du die Zeit in einer Variable (hier `start`) gemerkt hast.	4

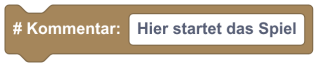
Konsole

Texte im Terminal unter dem Emulator ausgeben und eintippen.

	Schreibt einen Wert als neue Zeile ins Terminal – praktisch, um bei der Fehlersuche Werte anzusehen.	16
	Schreibt eine leere Zeile ins Terminal.	16
	Zeigt die Frage im Terminal und wartet, bis du etwas eintippst und Enter drückst. Ergebnis: der getippte Text.	16
	Wie „einlesen“, aber das Ergebnis ist eine Zahl. Tippt man Buchstaben, stoppt das Programm mit einem Fehler.	16

Kommentar

Der Kommentar steht ganz unten in der Blockleiste.

	Eine Notiz für Menschen. Der PicoBoy überspringt sie.	2

ANHANG B

Fehlermeldungen verstehen und Fehler suchen

Jedes Programm hat am Anfang Fehler – das ist ganz normal. PBC+ Studio hilft dir dabei, sie zu finden: Es prüft dein Programm vor dem Start, erklärt Fehler in einem Fenster und zeigt Meldungen im Terminal. Hier steht, was die Meldungen bedeuten und was du tun kannst. Am Schluss findest du Tipps, wie du Fehler ohne Meldung aufspürst.

Wo erscheinen Meldungen?

- **Ein Fenster in der Mitte** mit einer Überschrift (z. B. „Division durch Null“) und einer Erklärung. Du schließt es mit **OK**.
- **Das Terminal** unter dem Emulator. Dort steht bei jedem Start „-- Testlauf --“. Endet das Programm von selbst, folgt „-- fertig --“, nach einem Klick auf ■ **Stop** steht dort „-- gestoppt --“. Bei einem Fehler erscheint **roter Text**.
- **Eine kleine Meldung unten** am Bildschirmrand, die nach kurzer Zeit verschwindet, z. B. „Erst verbinden.“

Der rote Text im Terminal

Stoppt dein Programm wegen eines Fehlers, sieht das im Terminal zum Beispiel so aus:

```
Traceback (most recent call last):
  File "<stdin>", line 87, in __m
ZeroDivisionError: divide by zero

Hinweis: Die Zeilennummer oben bezieht sich auf den intern umgewandelten
Code, nicht auf deine geschriebene Zeile. Nutze die Fehlerart und die
Erklärung als Wegweiser.
```

Das ist Englisch und sieht kompliziert aus. Wichtig ist nur die **letzte rote Zeile**: Sie nennt die *Fehlerart* (hier `ZeroDivisionError`, Teilen durch null). Die **Zeilennummer** (`line 87`) hilft dir nicht – deine Blöcke haben keine Zeilennummern. Das sagt auch der gelbe Hinweis darunter. Meist öffnet PBC+ Studio zusätzlich ein Fenster mit einer deutschen Erklärung.

Meldungen beim Bauen

Diese Meldungen erscheinen, während du Blöcke holst oder etwas anlegst.

Meldung	Was bedeutet das?	Was tun?
Sprite fehlt – „Erstelle erst ein Sprite über „+ neues Sprite…“ in der Sprite-Sidebar.“	Du hast einen Sprite-Block geholt, aber es gibt noch kein Sprite.	In der Kategorie **Sprites** auf „+ neues Sprite…“ klicken. Vorher ein Bild hochladen (**Dateien → Bilder verwalten…**).
Liste fehlt – „Erstelle erst eine Liste unter „+ neue Liste…““.	Du hast einen Listen-Block geholt, aber es gibt noch keine Liste.	In der Kategorie **Listen** auf „+ neue Liste…“ klicken.
„Erstelle erst eine Variable unter „+ neue Variable…““.	Du wolltest „setze … auf…“ holen, es gibt aber noch keine Variable.	Erst eine Variable anlegen.
Turtle fehlt – „Erstelle erst eine Turtle mit „neue Turtle erstellen““.	Turtle-Befehle brauchen eine Turtle.	Zuerst „neue Turtle erstellen“ holen.
„Ungültiger Name. Erlaubt sind Buchstaben, Ziffern und Unterstriche. Nicht mit einer Ziffer beginnen.“	Beim Anlegen eines Sprites steht im Namen ein Umlaut, ein Leerzeichen oder ein Sonderzeichen.	ae, oe, ue, ss statt ä, ö, ü, ß; `_` statt Leerzeichen: `mein_held`.
„Der Name „…“ wird schon als Name einer Variable verwendet.“ (oder: eines Sprites, einer Funktion, einer Liste)	Zwei Dinge dürfen nicht gleich heißen.	Einen anderen Namen wählen, z. B. `punkte` und `punkte_anzeigen`.
„Der Name „…“ ist in Python reserviert. Bitte einen anderen Namen wählen.“	Wörter wie `if`, `print` oder `list` braucht die Programmiersprache selbst.	Einen deutschen Namen wählen: `liste`, `wert`, `zahl`.

> **Tipp zu Namen:** Bei Funktionsnamen ersetzt PBC+ Studio Umlaute und Leerzeichen stillschweigend durch `_` – aus `zähle` wird `z_hle`. Schreibe deshalb bei **allen** Namen gleich ae, oe, ue und ss.

Meldungen beim Starten

Wenn du auf ▶ **Testen** klickst, prüft PBC+ Studio dein Programm, bevor es startet. Findet es ein Problem, startet das Programm nicht.

Meldung	Was bedeutet das?	Was tun?
Endlosschleife ohne Pause – „Eine „unendlich wiederholen"-Schleife enthält keinen „warte … ms"-Block … Trotzdem starten?"	In der Schleife fehlt ein „warte"-Block. Das Gerät arbeitet dann ohne Pause und reagiert vielleicht nicht mehr auf Stop.	**Abbrechen** und „warte 20 ms" in die Schleife legen. Steht das „warte" in einer Funktion, die in der Schleife aufgerufen wird, kommt die Frage trotzdem – dann kannst du **OK** wählen.
Funktionsaufruf ohne Funktion – „Diese Funktionen werden aufgerufen, aber nicht (mehr) definiert: …"	Ein Block „Funktion …" ruft eine Funktion auf, die es nicht (mehr) gibt – vielleicht umbenannt oder gelöscht. Im Aufruf steht dann ein „(?)" hinter dem Namen.	Die Funktion wieder anlegen oder im Aufruf eine andere Funktion auswählen.
Sprite-Befehl ohne Sprite – „In einem oder mehreren Sprite-Befehlen ist kein Sprite ausgewählt."	Ein Sprite wurde gelöscht, Blöcke benutzen es aber noch.	Im Block ein anderes Sprite auswählen oder den Block löschen.
Sprite-Bild fehlt – „Diese Sprites haben kein Bild ausgewählt: …"	Ein Sprite hat kein Bild.	Das Sprite neu anlegen und dabei ein Bild auswählen.
Turtle fehlt – „Es gibt Turtle-Befehle für: t … Bevor du sie benutzt, musst du sie mit einem „neue Turtle"-Block erstellen."	Der Block „neue Turtle" wurde gelöscht oder hängt nicht im Programm.	„neue Turtle" wieder einbauen – vor dem ersten Turtle-Befehl.
Bild nicht ausgewählt – „In einem oder mehreren Bild-Blöcken ist noch kein Bild ausgewählt."	In „Bild anzeigen" oder „Bild auf Canvas" ist kein Bild eingestellt.	Im Block ein Bild wählen. Ist die Liste leer: erst ein Bild hochladen.
Bilder fehlen auf dem PBC+ – „Diese Bild-Dateien sind nicht auf dem Gerät: …"	Ein Sprite braucht ein Bild, das nicht hochgeladen ist. Das kommt auch im Emulator, z. B. wenn du eine Datei von jemand anderem öffnest.	**Dateien → Bilder verwalten…** öffnen und die genannten Bilder hochladen – am Gerät bei verbundenem PicoBoy.
„Erst verbinden."	Als Ziel ist **PBC+** gewählt, aber kein Gerät verbunden.	Oben rechts **Emulator** wählen – oder das Gerät verbinden.

Meldungen, wenn das Programm stoppt

Diese Fenster erscheinen, wenn dein Programm schon läuft und dann wegen eines Fehlers anhält. In Klammern steht die Fehlerart aus der letzten roten Zeile.

Meldung (Fehlerart)	Was bedeutet das?	Was tun?
Division durch Null (`ZeroDivisionError`)	Dein Programm hat durch 0 geteilt – oder „Rest“ mit 0 gerechnet. Oft steht in einer Variable unerwartet 0.	Mit „falls“ vorher prüfen, dass der Teiler nicht 0 ist.
Listen-Position außerhalb (`IndexError`)	Du fragst nach einem Platz, den es in der Liste nicht gibt. Beispiel: Die Liste hat 3 Werte (Plätze 0, 1, 2), du willst Platz 3.	Daran denken: Gezählt wird ab 0. Mit „Länge von Liste“ prüfen, wie lang die Liste wirklich ist.
Unbekannter Name (`NameError`) – „Das Programm kennt den Namen „…“ nicht.“	Ein Name ist an dieser Stelle unbekannt. Beispiel: Ein Parameter-Block einer Funktion steht außerhalb dieser Funktion.	Den Block in die richtige Funktion ziehen; prüfen, ob Variable, Sprite oder Funktion noch existieren.
Falscher Typ (`TypeError`) – „Du rechnest oder verbindest vermutlich Text mit Zahlen …“	Du rechnest mit einem Text, z. B. „+“ mit einer Text-Variable.	Zum Rechnen Zahl-Variablen nehmen. Text und Zahl zusammenfügen geht mit „verbinde … mit…“. (Die Blöcke „Text(…)“ und „Zahl(…)“, die das Fenster nennt, gibt es nur beim Programmieren mit Text.)
Unbekannte Funktion oder Eigenschaft (`AttributeError`) – „Ein Objekt vom Typ „int“ hat kein „append“.“	Etwas ist nicht das, was du denkst. Beispiel: In der Liste steckt gar keine Liste, sondern eine Zahl, weil du „setze Liste … auf“ eine Zahl-Variable gegeben hast – dann geht „hänge … an“ nicht.	Nachsehen, was du der Variable oder Liste zuletzt gegeben hast. Für Listen „Liste aus Werten“ oder „leer“ verwenden.
Eingabe ist keine Zahl (`ValueError`) – „Das Programm hat eine Zahl erwartet, aber etwas anderes bekommen …“	Bei „einlesen als Zahl“ wurden Buchstaben eingetippt.	Nur Ziffern eintippen.
Endlose Selbst-Aufrufe – „Eine Funktion ruft sich selbst immer wieder auf.“	In einer Funktion steht ein Aufruf derselben Funktion, ohne dass es je aufhört.	Den Aufruf entfernen – oder mit „falls“ dafür sorgen, dass er irgendwann nicht mehr passiert.
Schreibfehler im Code (`SyntaxError`)	Das Programm ist für den PicoBoy nicht lesbar. Mit Blöcken passiert das fast nie – eher, wenn ein Programm im Textmodus verändert wurde.	Programm speichern, im Blockmodus neu öffnen und testen; sonst die Lehrkraft fragen.

Meldung (Fehlerart)	Was bedeutet das?	Was tun?
Nicht genug Speicher (`MemoryError`)	Der Speicher des Geräts ist voll: zu viele Bilder oder Sprites gleichzeitig, sehr lange Listen.	Weniger und kleinere Bilder verwenden.
Zahl zu groß (`OverflowError`)	Eine Zahl ist zu groß geworden, z. B. durch „hoch“ mit großen Zahlen.	Rechnung prüfen.
Modul nicht gefunden	Ein Teil der Programmiersprache fehlt. Mit Blöcken kommt das praktisch nicht vor.	Der Lehrkraft Bescheid sagen.

Fehler ohne Meldung

Die schwierigsten Fehler sind die ohne Meldung: Das Programm läuft, tut aber nicht, was es soll. Hier die häufigsten:

Das passiert	Mögliche Ursache	Das hilft
Ein Block tut nichts.	Er hängt nicht unter „Beim Start“ (oder in einer Funktion), sondern liegt frei auf der Arbeitsfläche.	Block anhängen, bis er einschnappt.
Eine Frage („berührt …?“, „wird … gedrückt?“) wirkt nur einmal oder nie.	Der „falls“-Block steht nicht in der Schleife.	Ihn in die Schleife ziehen.
„Sprite … berührt Sprite …“ ist immer wahr.	In beiden Feldern steht dasselbe Sprite – so ist der Block eingestellt, wenn du ihn neu holst.	Im zweiten Feld das andere Sprite wählen.
„berührt Farbe“ reagiert nie.	Die Farbe ist nicht **genau** gleich, oder sie ist nur direkt auf den Bildschirm gemalt statt auf den Canvas.	Mit derselben „Farbe …“ malen und prüfen, auf den Canvas malen.
Text verschwindet sofort wieder.	Sobald sich ein Sprite bewegt, wird der ganze Bildschirm neu gemalt.	Text auf den Canvas schreiben (Kapitel 11).
Statt „ä“, „ö“, „ü“ oder „ß“ steht „?“ auf dem Bildschirm.	Die Schrift des PicoBoy kennt diese Zeichen nicht – auch im Emulator.	In Texten für den Bildschirm ae, oe, ue und ss schreiben.
Ein Wert ist immer gleich.	Er wird in der Schleife jedes Mal wieder auf den Startwert gesetzt.	Startwerte vor die Schleife setzen.
Das Programm macht nach dem Start gar nichts.	Es wartet – auf eine Taste („auf … warten“) oder auf eine Eingabe im Terminal („einlesen“).	Taste drücken bzw. ins Terminal klicken und tippen.
Ein Ton ist zu kurz, eine Melodie klingt verwischt.	„Ton … für … ms“ wartet nicht, bis der Ton fertig ist.	Nach jedem Ton ein „warte“ mit derselben Zeit.

So findest du jeden Fehler

Fehlersuche in fünf Schritten

1. **Ruhig bleiben.** Ein Fehler heißt nicht, dass du schlecht programmierst.
2. **Genau hinsehen:** Was passiert? Wann passiert es? Passiert es jedes Mal? Lies die Meldung ganz durch.
3. **Eingrenzen:** Welcher Teil des Programms ist zuständig? Bei einem Spiel in Funktionen ist das meist schnell klar (Kapitel 21).
4. **Nachsehen statt raten:** Lass dir Werte mit „ausgeben“ im Terminal anzeigen. Oder lass an der verdächtigen Stelle eine LED leuchten oder einen Ton spielen – dann weißt du, ob das Programm dort überhaupt ankommt. Schreibe mit „verbinde“ ein kurzes Wort vor jeden Wert, z. B. „x=“ – dann weißt du, welche Zahl was bedeutet.
5. **Eine Sache ändern**, wieder testen. Hat es nicht geholfen: rückgängig machen (**Strg+Z**) und die nächste Vermutung prüfen.

Noch mehr Tricks:

- **Laut vorlesen:** Erkläre dein Programm Block für Block jemandem – oder einem Kuscheltier. Beim Erklären merkst du oft selbst, wo es hakt.
- **Verkleinern:** Löse Blöcke testweise vom Programm (auf die Arbeitsfläche ziehen, sie laufen dann nicht mit). Tritt der Fehler immer noch auf? So findest du heraus, welcher Teil schuld ist.
- **Mit kleinen Zahlen testen:** Statt 10 Leben erst 1 Leben, statt 100 Punkte erst 3 – dann siehst du das Spielende schneller.
- **Speichern, bevor du umbaust:** Eine funktionierende Fassung als eigene Datei ist dein Rettungsring.
- **Langsamer machen:** Ein größeres „warte“ in der Schleife zeigt dir, was Schritt für Schritt passiert. Und manchmal hilft einfach eine Pause.

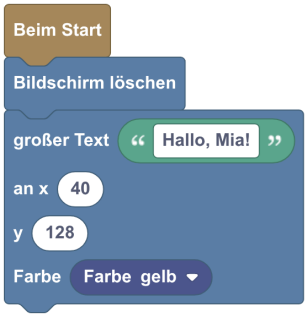
Lösungen zu Kapitel 1: Der PicoBoy Color Plus und PBC+ Studio

Aufgabe 1 – Teile finden

Der Joystick-Druck (MITTE) ist im Emulator die **Leertaste**. Ein Programm hältst du mit ■ **Stop** an.

Aufgabe 2 – Dein Name (`k01_loesung2.pbcp`)

Im Block „großer Text“ steht jetzt der neue Text, die Farbe ist „gelb“. Weil der Text länger ist als „Bereit?“, beginnt er weiter links (x = 40).



Aufgabe 3 – Zwei Zeilen (`k01_loesung3.pbcp`)

Der zweite Block „großer Text“ hat einen größeren y-Wert. Je größer y, desto weiter unten steht der Text.





Aufgabe 4 – Speichern und wieder öffnen

Nach **Dateien → Vom Computer…** ist das Programm genau so wieder da, wie du es gespeichert hast.

Aufgabe 5 – Falsche Reihenfolge (`k01_loesung5.pbcp`)

Der Bildschirm bleibt schwarz. Die Blöcke laufen von oben nach unten: Zuerst wird „Bereit?“ geschrieben – und gleich danach löscht „Bildschirm löschen“ alles wieder. Das geht so schnell, dass du den Text kaum oder gar nicht siehst.

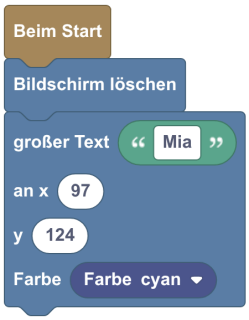


Lösungen zu Kapitel 2: Das erste Programm

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Dein Name in der Mitte (`k02_loesung1.pbcp`)

Für den Namen „Mia“ passt x = 97 und y = 124. Ein längerer Name braucht ein kleineres x. Der große Text ist 32 Pixel hoch. Damit er senkrecht in der Mitte steht, ist y = 140 − 16 = 124.

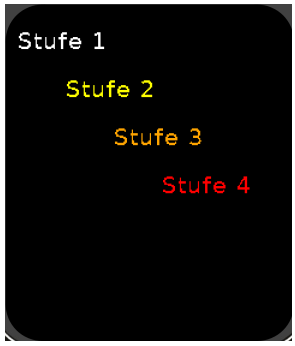


Aufgabe 2 – Wo landet der Text?

Der Text bei (180, 10) steht **oben rechts**: x = 180 ist weit rechts, y = 10 ganz oben. Ist der Text länger als etwa 60 Pixel, wird er am rechten Rand abgeschnitten. Der Text bei (10, 250) steht **unten links**: y = 250 ist weit unten.

Aufgabe 3 – Die Text-Treppe (`k02_loesung3.pbcp`)

Von Stufe zu Stufe werden x und y um 40 größer. Die Treppe läuft dadurch nach rechts und nach unten. Der Kommentar steht ganz oben im Programm.





Aufgabe 4 – Grüße mit Umlauten (`k02_loesung4.pbcp`)

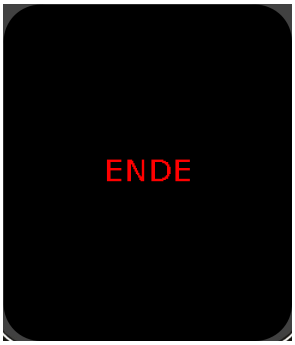
Auf dem Bildschirm steht „Gr???e“. Die Schrift des PBC+ kennt kein ü und kein ß. Für jeden dieser Buchstaben erscheinen sogar zwei Fragezeichen. Richtig lesbar wird es mit „Gruesse“.

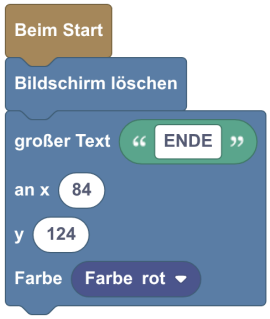




Aufgabe 5 – Genau in der Mitte (`k02_loesung5.pbcp`)

„ENDE“ ist im großen Text 73 Pixel breit und 32 Pixel hoch. Links und rechts bleiben dann je (240 − 73) : 2 ≈ 84 Pixel frei, oben und unten je (280 − 32) : 2 = 124 Pixel. Also: x = 84, y = 124. Wenn du durch Ausprobieren auf 83 oder 85 gekommen bist, ist das auch richtig – der Unterschied ist kaum zu sehen.





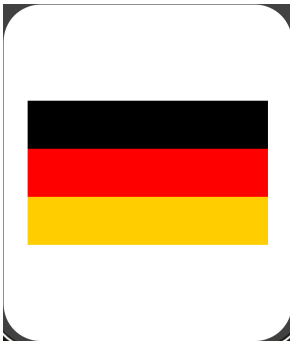
ANHANG C – LÖSUNGEN

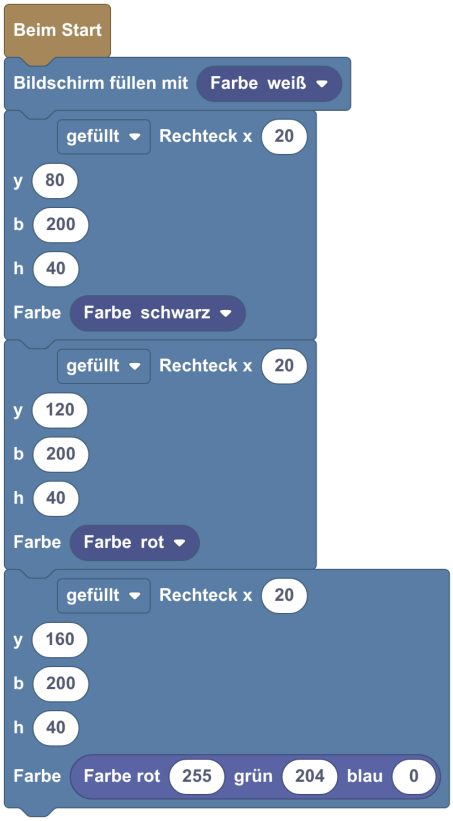
Lösungen zu Kapitel 3: Zeichnen und Farben

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Bild so ähnlich aussieht wie verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

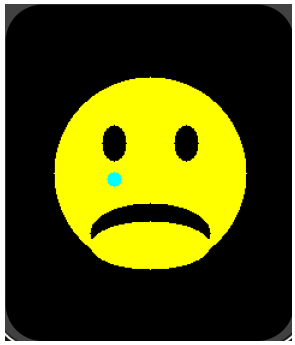
Aufgabe 1 – Die Deutschlandfahne (`k03_loesung1.pbcp`)

„Bildschirm füllen“ mit Weiß kommt zuerst. Die drei Streifen sind gleich breit (200) und gleich hoch (40). Jeder beginnt genau dort, wo der vorige aufhört: bei y = 80, 120 und 160.

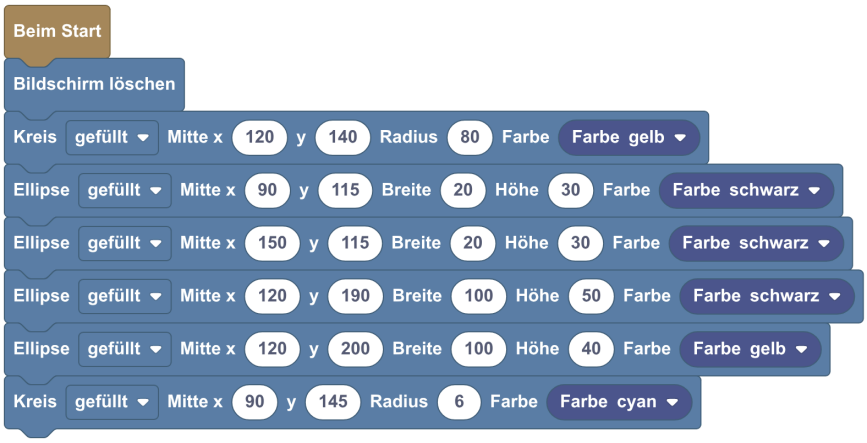




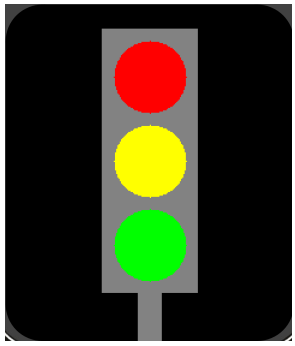
Aufgabe 2 – Der traurige Smiley (`k03_loesung2.pbcp`)



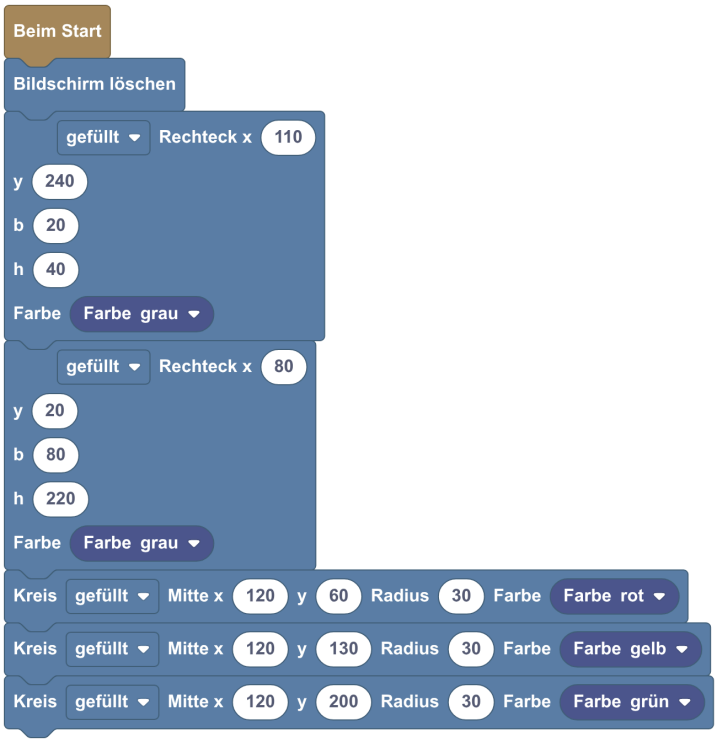
Für einen traurigen Mund wird der Trick umgedreht: Die gelbe Ellipse liegt jetzt **unter** der schwarzen (größeres y). Sie malt den unteren Teil zu, und der schwarze Bogen zeigt nach oben. Die Träne ist ein kleiner Kreis in Cyan.



Aufgabe 3 – Eine Ampel (`k03_loesung3.pbcp`)

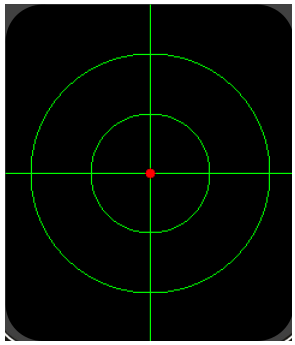


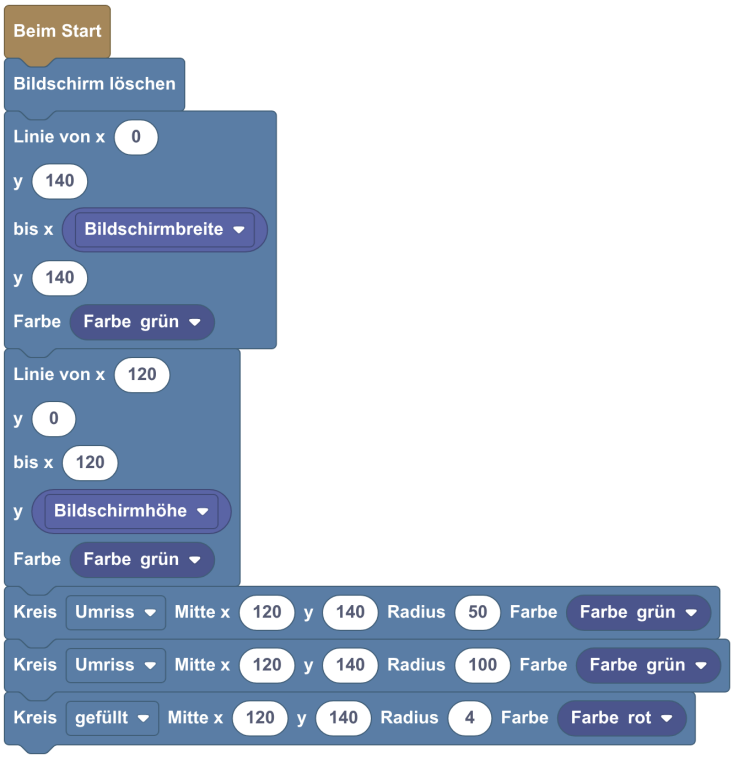
Zuerst kommen Mast und Kasten, danach die drei Lichter – sonst würde der graue Kasten sie wieder übermalen. Die Lichter haben alle x = 120, nur y ist verschieden.



Aufgabe 4 – Das Zielfernrohr (`k03_loesung4.pbcp`)

Die waagerechte Linie geht von x = 0 bis zur Bildschirmbreite, die senkrechte von y = 0 bis zur Bildschirmhöhe. Die Mitte liegt bei (120, 140).

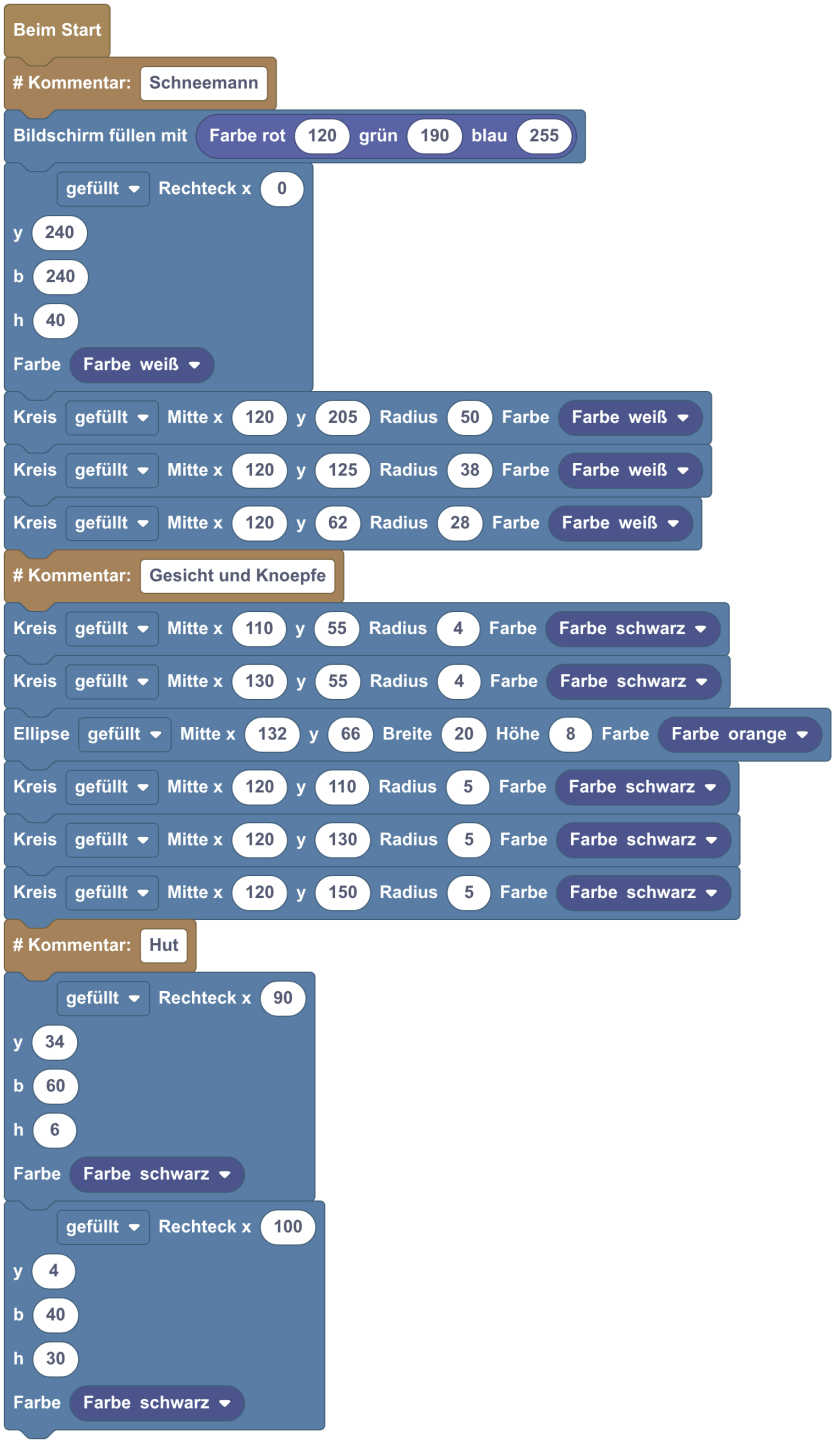




Aufgabe 5 – Der Schneemann (`k03_loesung5.pbcp`)

Erst der Himmel und der Schnee, dann die drei Kugeln von unten nach oben, zum Schluss Gesicht, Knöpfe und Hut. Die Nase ist eine kleine orangefarbene Ellipse. Kommentare gliedern das lange Programm.



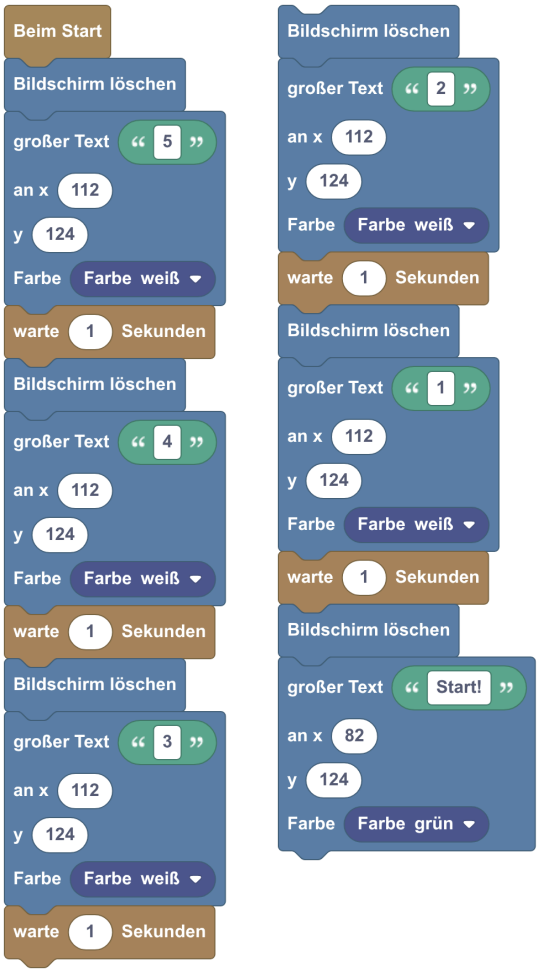

Beim Start
Kommentar: Schneemann
Bildschirm füllen mit Farbe rot 120 grün 190 blau 255
gefüllt Rechteck x 0
y 240
b 240
h 40
Farbe Farbe weiß
Kreis gefüllt Mitte x 120 y 205 Radius 50 Farbe Farbe weiß
Kreis gefüllt Mitte x 120 y 125 Radius 38 Farbe Farbe weiß
Kreis gefüllt Mitte x 120 y 62 Radius 28 Farbe Farbe weiß
Kommentar: Gesicht und Knoepfe
Kreis gefüllt Mitte x 110 y 55 Radius 4 Farbe Farbe schwarz
Kreis gefüllt Mitte x 130 y 55 Radius 4 Farbe Farbe schwarz
Ellipse gefüllt Mitte x 132 y 66 Breite 20 Höhe 8 Farbe Farbe orange
Kreis gefüllt Mitte x 120 y 110 Radius 5 Farbe Farbe schwarz
Kreis gefüllt Mitte x 120 y 130 Radius 5 Farbe Farbe schwarz
Kreis gefüllt Mitte x 120 y 150 Radius 5 Farbe Farbe schwarz
Kommentar: Hut
gefüllt Rechteck x 90
y 34
b 60
h 6
Farbe Farbe schwarz
gefüllt Rechteck x 100
y 4
b 40
h 30
Farbe Farbe schwarz

Lösungen zu Kapitel 4: Schritt für Schritt: Reihenfolge und Zeit

Oft gibt es mehrere richtige Lösungen. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Countdown ab 5 (`k04_loesung1.pbcp`)

Vor die „3“ kommen zwei weitere Dreiergruppen für „5“ und „4“: Bildschirm löschen, Zahl schreiben, 1 Sekunde warten. Am schnellsten geht das mit **Kopieren**. Zum Schluss steht „Start!“ bei x = 82, weil das Wort breiter ist als „Los!“. Der zweite Teil hängt direkt unter dem ersten.



Im nächsten Kapitel lernst du, wie das mit viel weniger Blöcken geht.

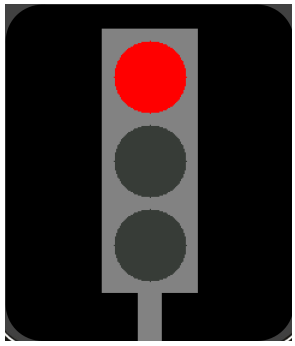
Aufgabe 2 – Wo ist der Smiley? (`k04_loesung2.pbcp`)

Der Smiley wird gemalt – aber gleich danach malt „Bildschirm füllen“ den ganzen Bildschirm blau an. Das geht so schnell, dass du den Smiley gar nicht siehst. „Bildschirm füllen“ muss deshalb **ganz nach oben**, direkt unter „Beim Start“.

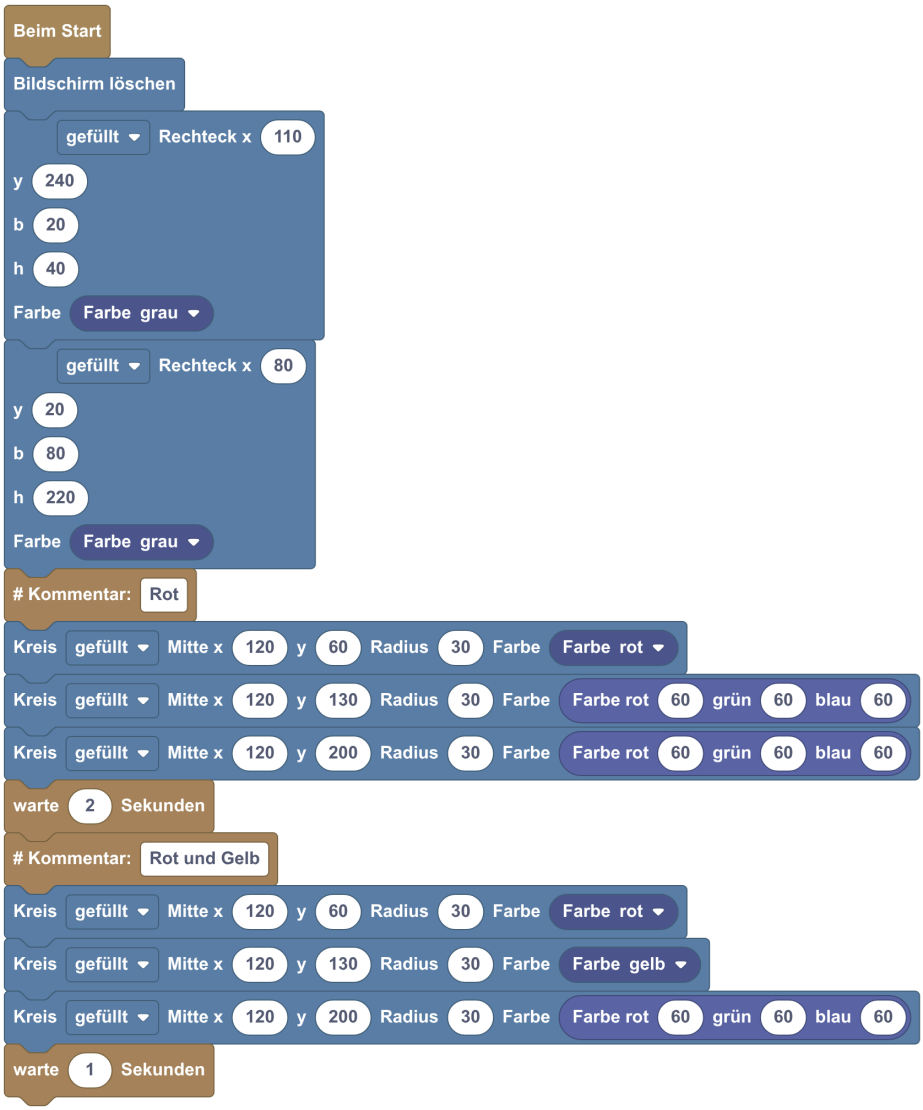


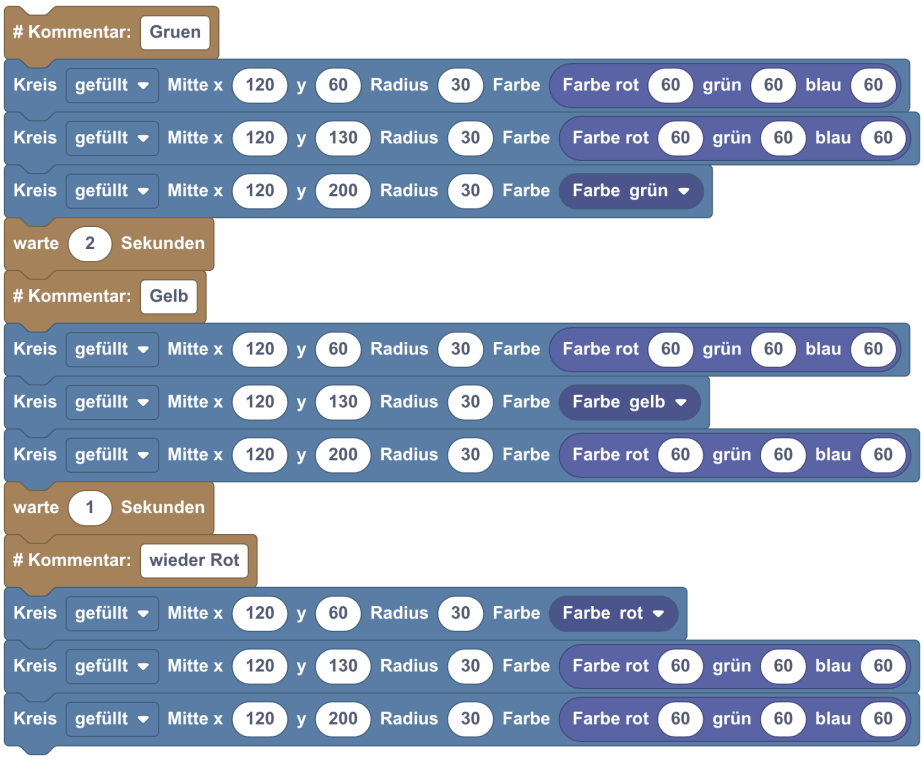
Aufgabe 3 – Die Ampel schaltet (`k04_loesung3.pbcp`)

Zuerst werden Mast und Kasten einmal gemalt. Danach malt jede Phase alle drei Lichter neu: Ein Licht, das leuchtet, bekommt seine Farbe, ein Licht, das aus ist, wird dunkelgrau. Dann wird gewartet. Kommentare zeigen, wo eine Phase beginnt. Die letzte Phase „wieder Rot“ braucht keine Wartezeit mehr.



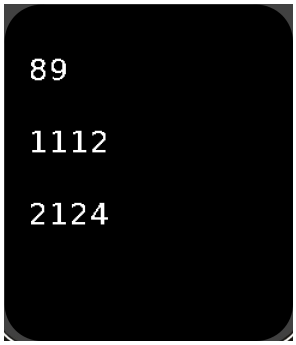
Die erste Phase: Rot

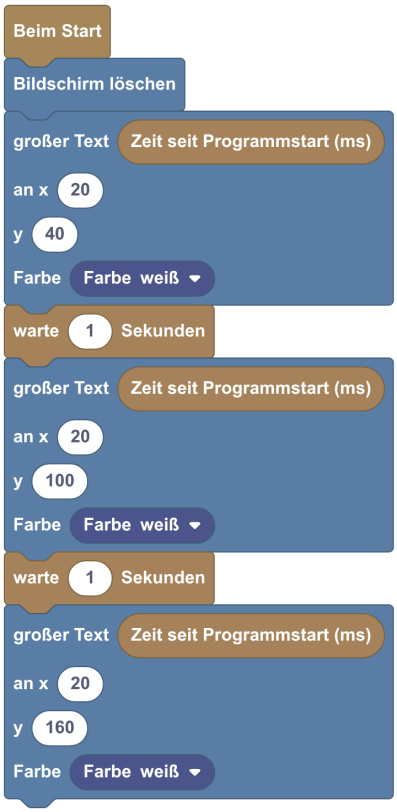




Aufgabe 4 – Die Zeitansage (`k04_loesung4.pbcp`)

Die Zahlen wachsen jedes Mal um etwas **mehr als 1000**, im Bild um etwa 1010 bis 1020. Zu der einen Sekunde Warten kommt die Zeit dazu, die der PBC+ zum Schreiben der Zahl braucht. Auch Schreiben und Zeichnen dauern ein bisschen.





Aufgabe 5 – Achtung, Achtung! (`k04_loesung5.pbcp`)

Die vier Blöcke Text – warte – Rechteck – warte kommen dreimal hintereinander. Das schwarze Rechteck liegt genau über dem Text: bei x = 59 und y = 124, so breit (122) und so hoch (32) wie der Text „Achtung!“. Es darf auch etwas größer sein.

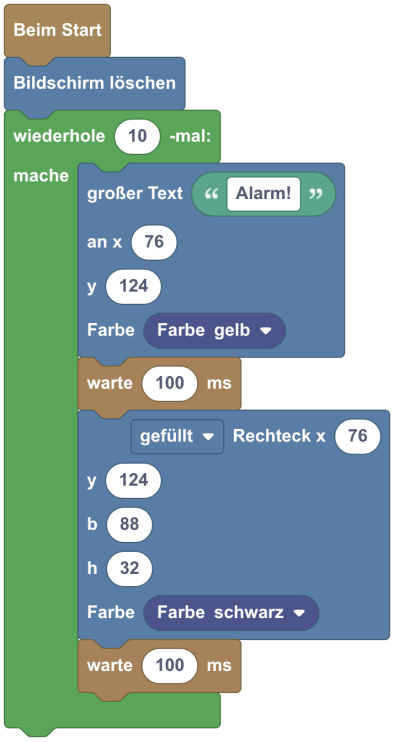

Beim Start
Bildschirm löschen
großer Text “ Achtung! ”
an x 59
y 124
Farbe Farbe rot
warte 500 ms
gefüllt Rechteck x 59
y 124
b 122
h 32
Farbe Farbe schwarz
warte 500 ms
großer Text “ Achtung! ”
an x 59
y 124
Farbe Farbe rot
warte 500 ms
gefüllt Rechteck x 59
y 124
b 122
h 32
Farbe Farbe schwarz
warte 500 ms
großer Text “ Achtung! ”
an x 59
y 124
Farbe Farbe rot
warte 500 ms
gefüllt Rechteck x 59
y 124
b 122
h 32
Farbe Farbe schwarz
warte 500 ms

Lösungen zu Kapitel 5: Wiederholungen

Oft gibt es mehrere richtige Lösungen. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

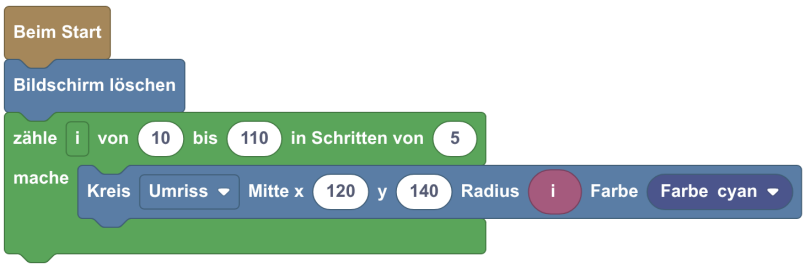
Aufgabe 1 – Schneller blinken (`k05_loesung1.pbcp`)

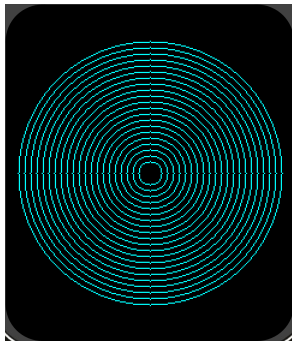
Drei Änderungen: 10 statt 5 Wiederholungen, die Farbe „gelb“ und kürzere Wartezeiten (100 statt 300 ms).



Aufgabe 2 – Enge Ringe (`k05_loesung2.pbcp`)

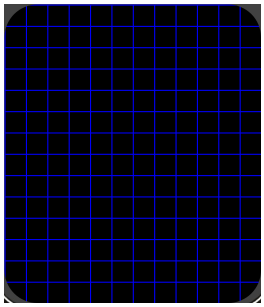
Die Ringe liegen jetzt ganz dicht beieinander. `i` ist nacheinander 10, 15, 20 … 110 – das sind **21 Ringe** statt 6. Hier sind sie zusätzlich in Cyan.

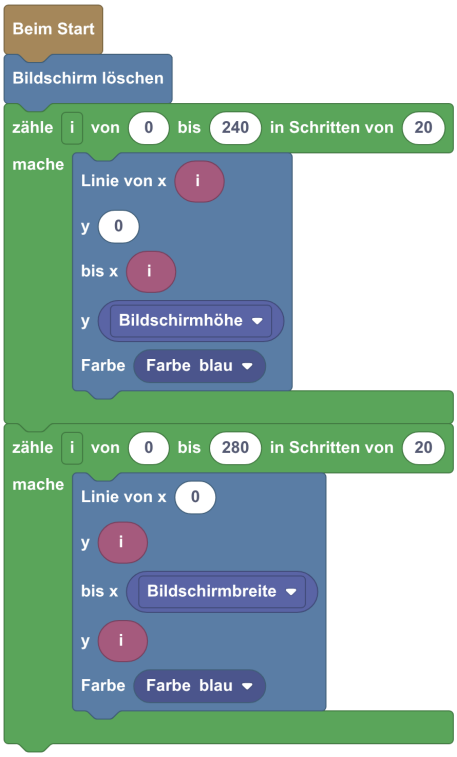




Aufgabe 3 – Karopapier (`k05_loesung3.pbcp`)

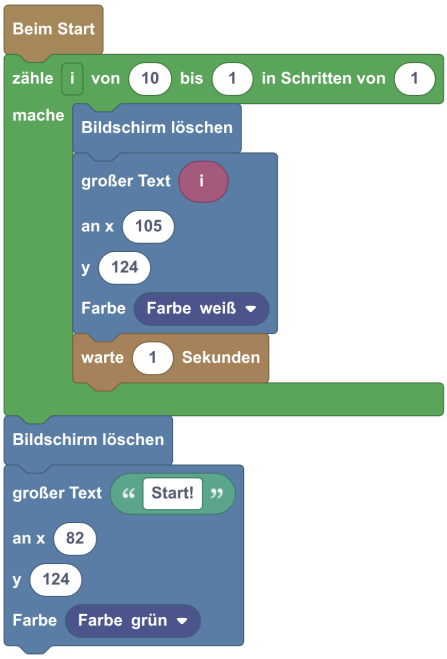
Die erste Schleife zeichnet senkrechte Linien: x ist `i`, y geht von 0 bis zur Bildschirmhöhe. Die zweite Schleife zeichnet waagerechte Linien: y ist `i`, x geht von 0 bis zur Bildschirmbreite. Beide Schleifen dürfen den Zähler `i` benutzen, weil sie nacheinander laufen.





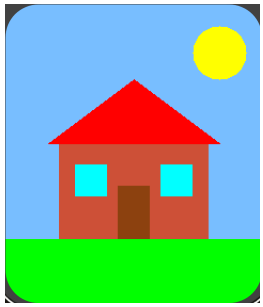
Aufgabe 4 – Countdown mit Schleife (`k05_loesung4.pbcp`)

Die Schleife zählt von 10 bis 1 – weil 10 größer ist als 1, zählt sie rückwärts. In der Schleife stehen nur drei Blöcke: Bildschirm löschen, `i` groß schreiben, 1 Sekunde warten. „Start!“ kommt **unter** die Schleife, denn es soll nur einmal am Ende erscheinen.

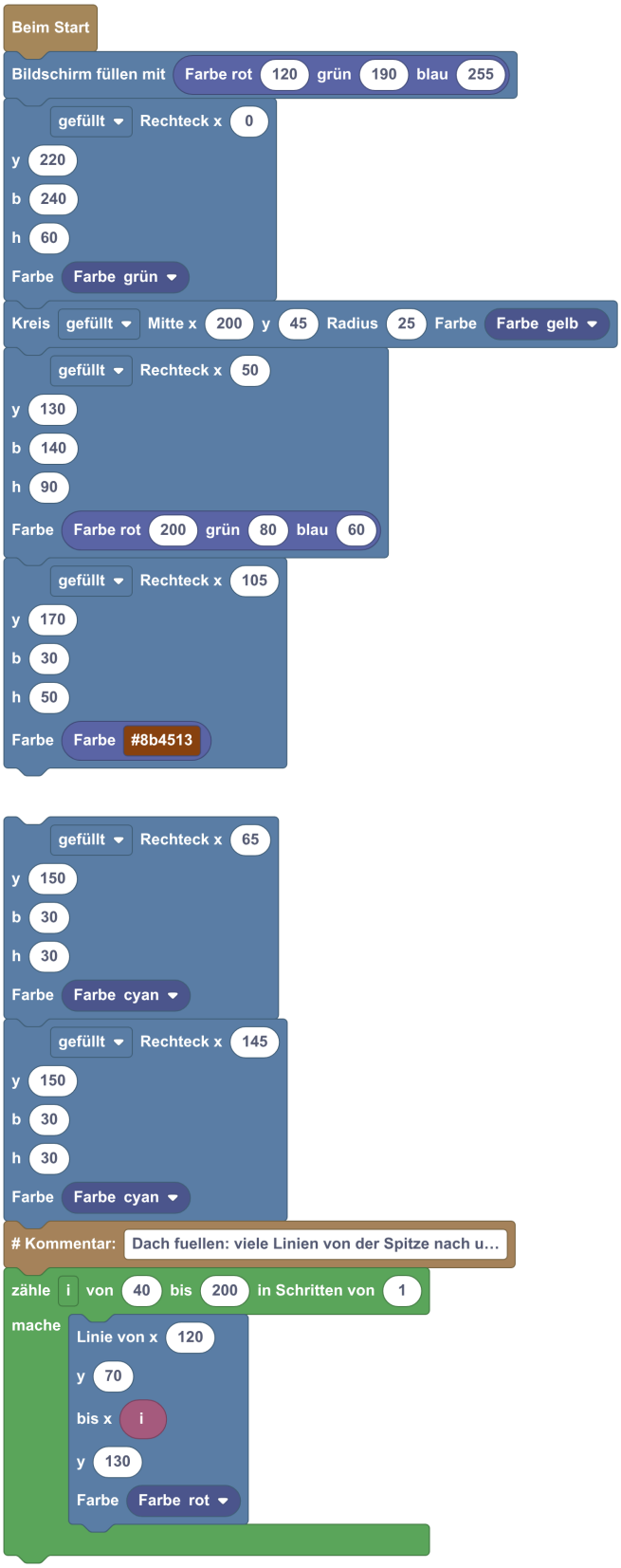


Vergleiche mit Aufgabe 1 aus Kapitel 4: Dort brauchte schon ein Countdown ab 5 siebzehn Blöcke.

Aufgabe 5 – Ein volles Dach (`k05_loesung5.pbcp`)

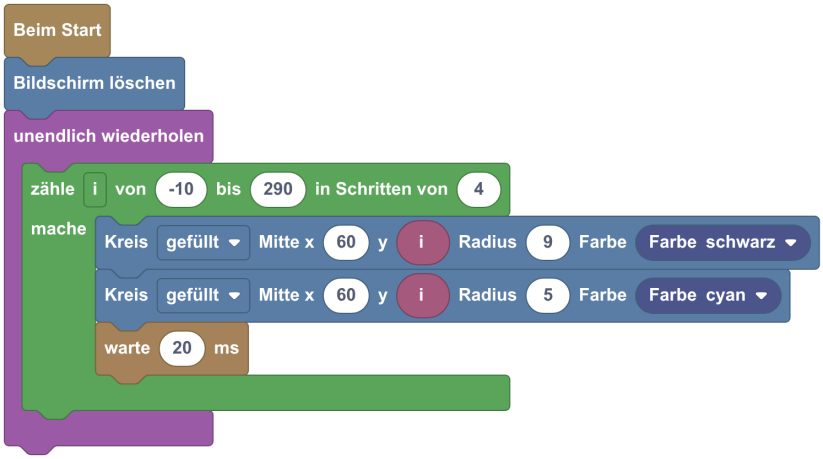


Das Haus wird gezeichnet wie in Kapitel 3, nur ohne die drei Dachlinien. Am Ende zeichnet eine Schleife 161 rote Linien von der Spitze (120, 70) nach unten zu (40, 130), (41, 130), (42, 130) … (200, 130). Zusammen füllen sie das ganze Dreieck. Der zweite Teil hängt direkt unter dem ersten.



Aufgabe 6 – Regentropfen (`k05_loesung6.pbcp`)

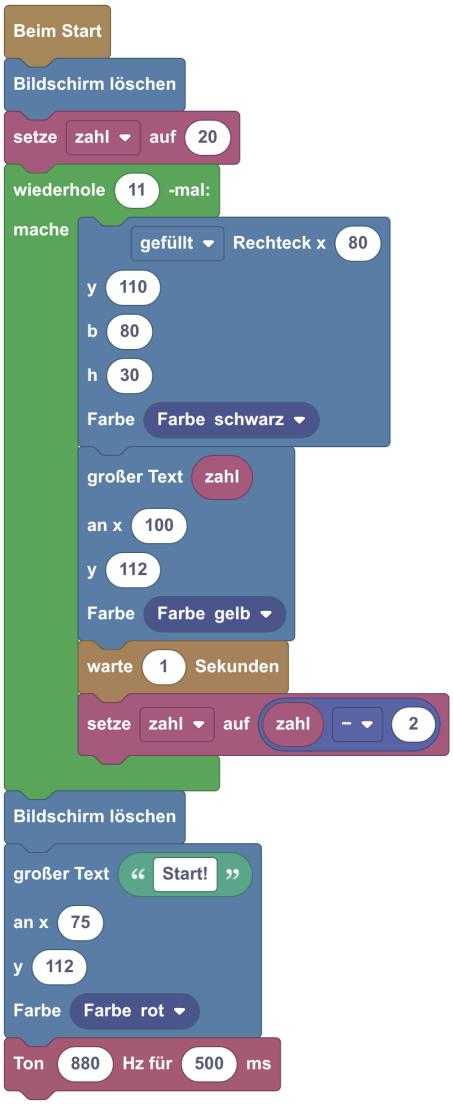
Eine Zählschleife lässt den Tropfen fallen: `i` ist seine y-Koordinate. Vor jedem neuen Tropfen malt ein schwarzer Kreis (Radius 9) die alte Stelle zu. Der Tropfen hat Radius 5 und fällt 4 Pixel pro Schritt – 5 + 4 = 9, also deckt der schwarze Kreis ihn ganz zu. „unendlich wiederholen“ lässt die Zählschleife immer wieder von vorn beginnen. Weil der Tropfen bei y = 290 ganz unter dem Bildschirm verschwunden ist, bleibt unten nichts von ihm übrig.



Lösungen zu Kapitel 6: Variablen

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Countdown in Zweierschritten (`k06_loesung1.pbcp`)



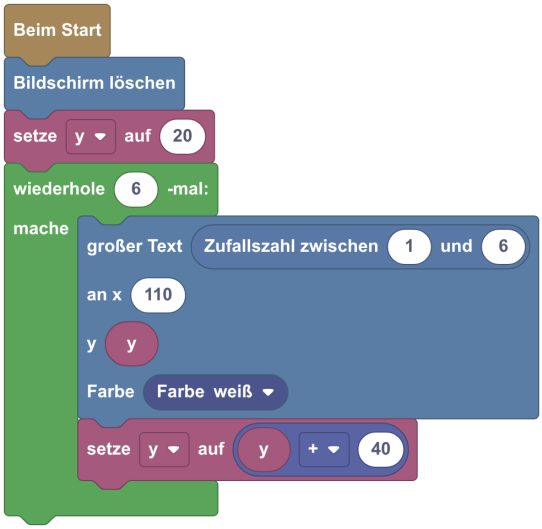
`zahl` startet bei 20 und wird in jedem Durchlauf um 2 kleiner. Von 20 bis 0 sind es 11 Zahlen (20, 18, 16, 14, 12, 10, 8, 6, 4, 2, 0), also muss die Schleife **11-mal** laufen.

Aufgabe 2 – Was kommt heraus?

`x` ist zuerst 4. Dann wird `x` auf 4 × 3 = 12 gesetzt und danach auf 12 − 2 = **10**.

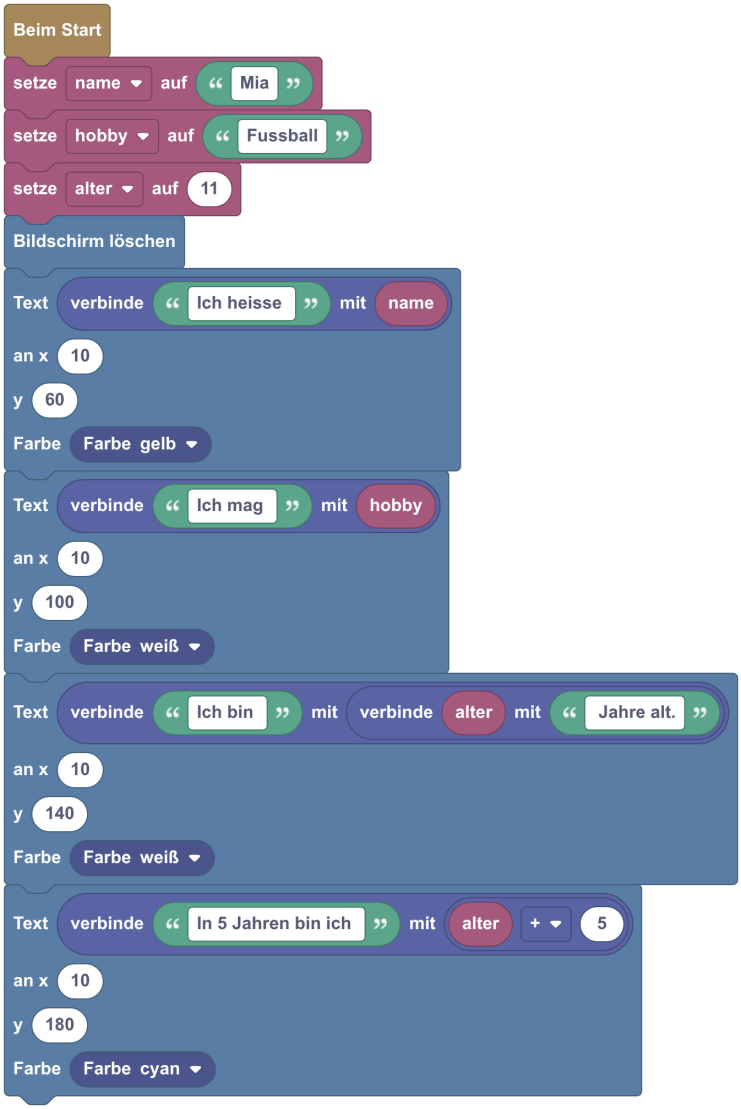
„verbinde 3 mit 4“ hängt die Zeichen aneinander und zeigt **34**. Der Rechen-Block „3 + 4“ rechnet und zeigt **7**.

Aufgabe 3 – Sechs Würfe (`k06_loesung3.pbcp`)



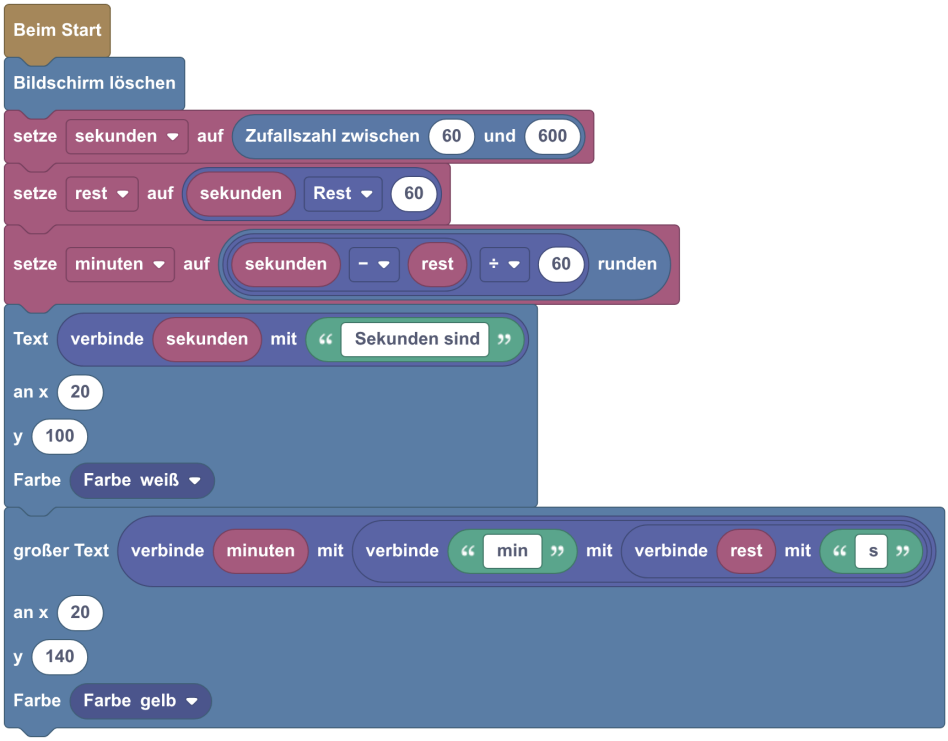
Die Zufallszahl steckt direkt im Text-Block. Nach jedem Wurf rutscht `y` um 40 Pixel nach unten.

Aufgabe 4 – Mein Steckbrief (`k06_loesung4.pbcp`)



`name` und `hobby` sind Text-Variablen („+ neuer Text…“), `alter` ist eine Zahl-Variable. Die letzte Zeile rechnet `alter + 5` aus. „heisse“ ist ohne ß geschrieben, weil der Bildschirm kein ß kennt.

Aufgabe 5 – Zeit umrechnen (`k06_loesung5.pbcp`)



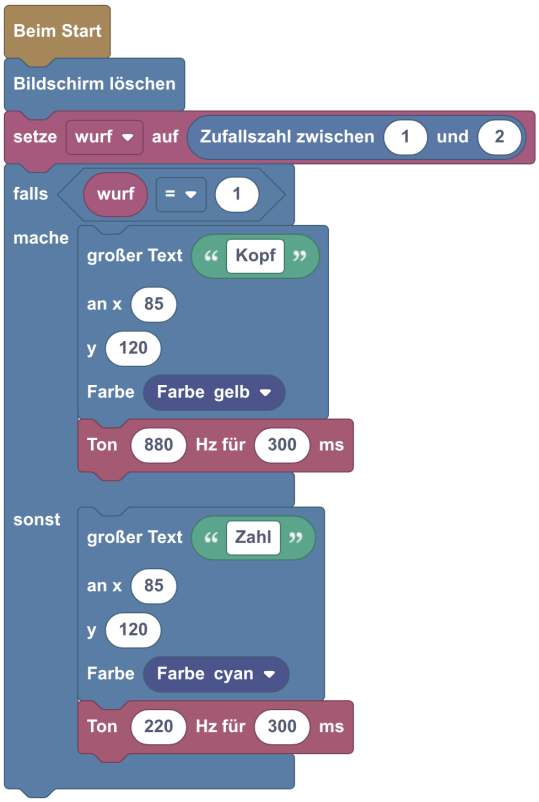
`rest` = `sekunden` Rest 60 sind die Sekunden, die übrig bleiben. Zieht man sie ab, bleibt eine Zahl, die genau durch 60 teilbar ist. Geteilt durch 60 ergibt das die Minuten – allerdings als Kommazahl wie `5.0`. Der Block „… runden“ macht daraus `5`.

Beispiel: 347 Rest 60 = 47. (347 − 47) ÷ 60 = 300 ÷ 60 = 5. Also 5 min 47 s.

Lösungen zu Kapitel 7: Entscheidungen

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Münzwurf mit Ton (`k07_loesung1.pbcp`)

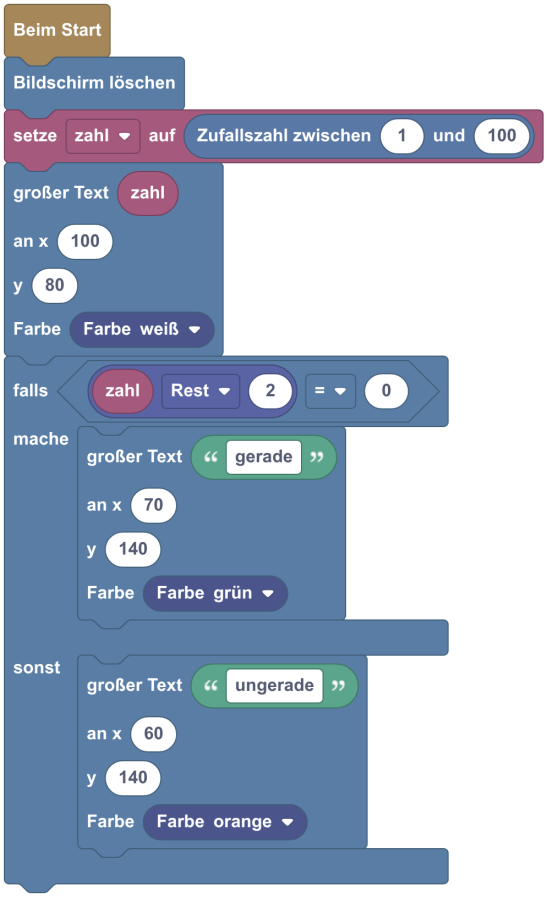


In jeden der beiden Wege kommt ein Ton-Block: 880 Hz bei „Kopf“, 220 Hz bei „Zahl“.

Aufgabe 2 – Wahr oder falsch?

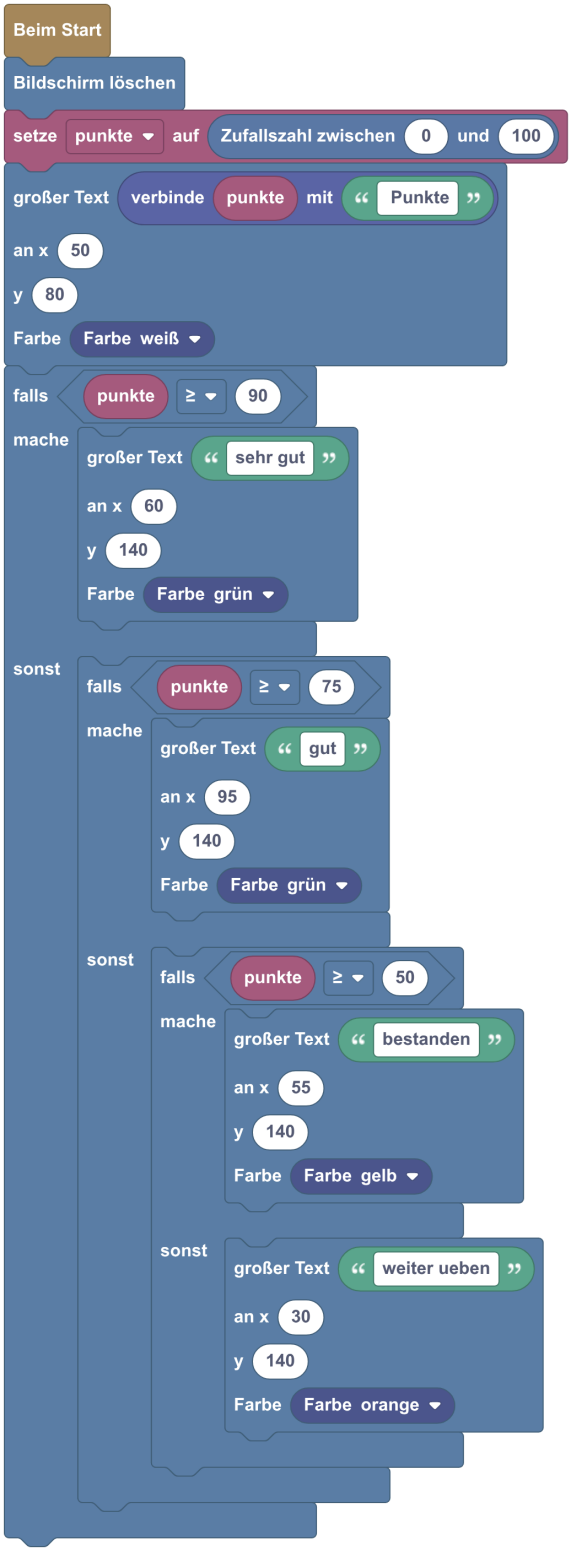
	Bedingung	Antwort	Warum?
a)	7 > 3	wahr	7 ist größer als 3.
b)	5 = 6	falsch	5 und 6 sind verschieden.
c)	4 < 4	falsch	4 ist nicht kleiner als 4 (4 ≤ 4 wäre wahr).
d)	nicht (2 > 5)	wahr	2 > 5 ist falsch, „nicht“ dreht es um.
e)	3 < 5 und 8 < 6	falsch	8 < 6 ist falsch – bei „und“ müssen beide stimmen.
f)	3 < 5 oder 8 < 6	wahr	3 < 5 ist wahr – bei „oder“ reicht eins.

Aufgabe 3 – Gerade oder ungerade? (`k07_loesung3.pbcp`)



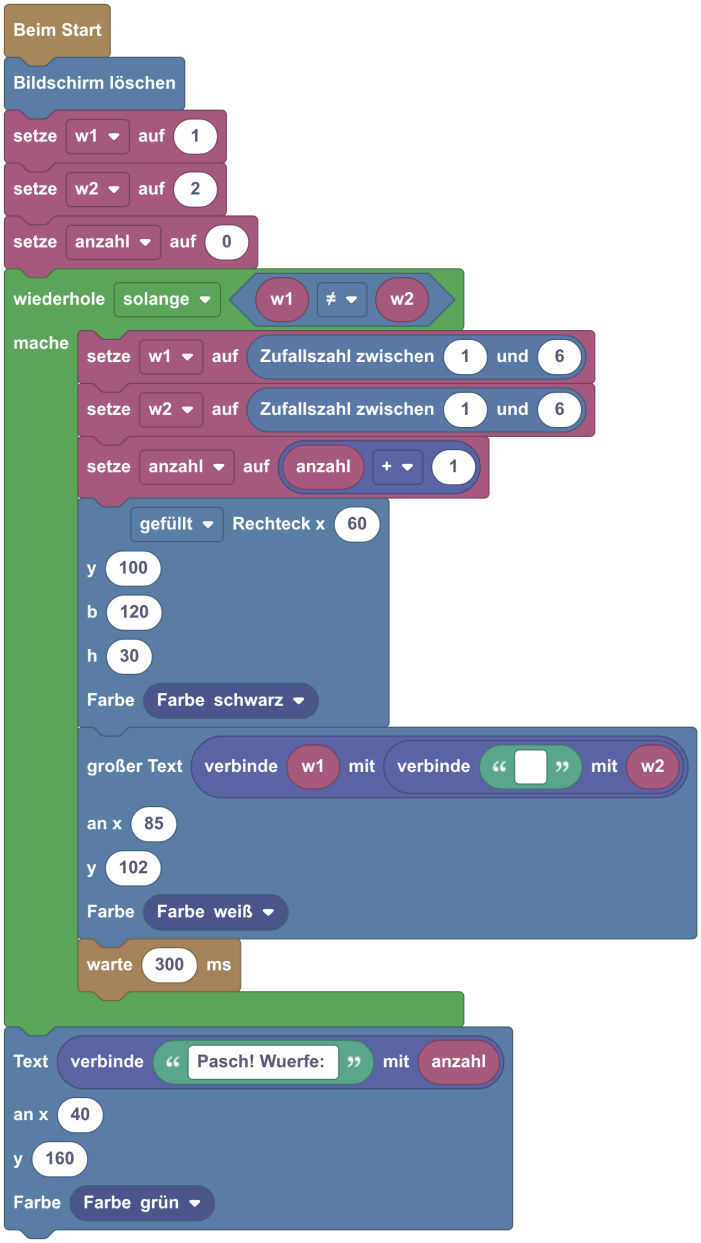
Eine Zahl ist gerade, wenn beim Teilen durch 2 kein Rest bleibt: „zahl Rest 2 = 0“.

Aufgabe 4 – Punkte und Noten (`k07_loesung4.pbcp`)



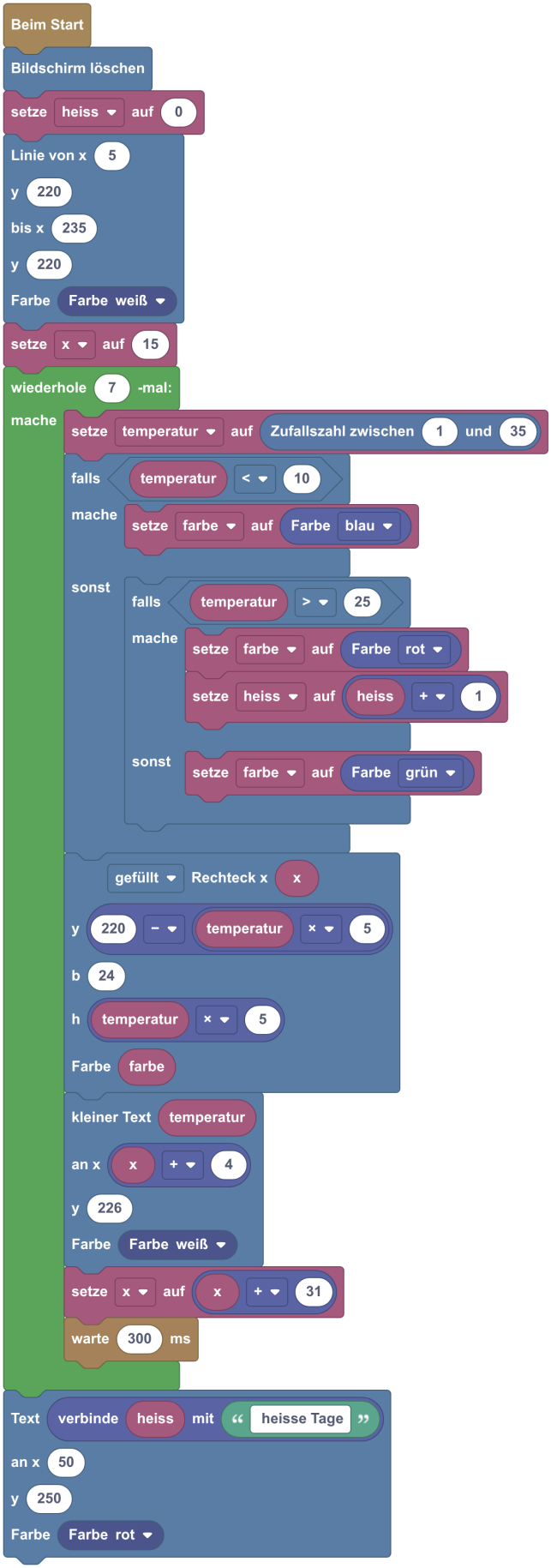
Drei „falls / sonst“ stecken ineinander. Die Reihenfolge ist wichtig: Zuerst wird die höchste Grenze geprüft. Würde man zuerst „≥ 50“ fragen, bekämen auch 95 Punkte nur „bestanden“.

Aufgabe 5 – Pasch! (`k07_loesung5.pbcp`)



`w1` und `w2` müssen am Anfang **verschieden** sein (hier 1 und 2), sonst ist „w1 ≠ w2“ gleich falsch, und die Schleife läuft kein einziges Mal.

Aufgabe 6 – Hitzewelle (`k07_loesung6.pbcp`)

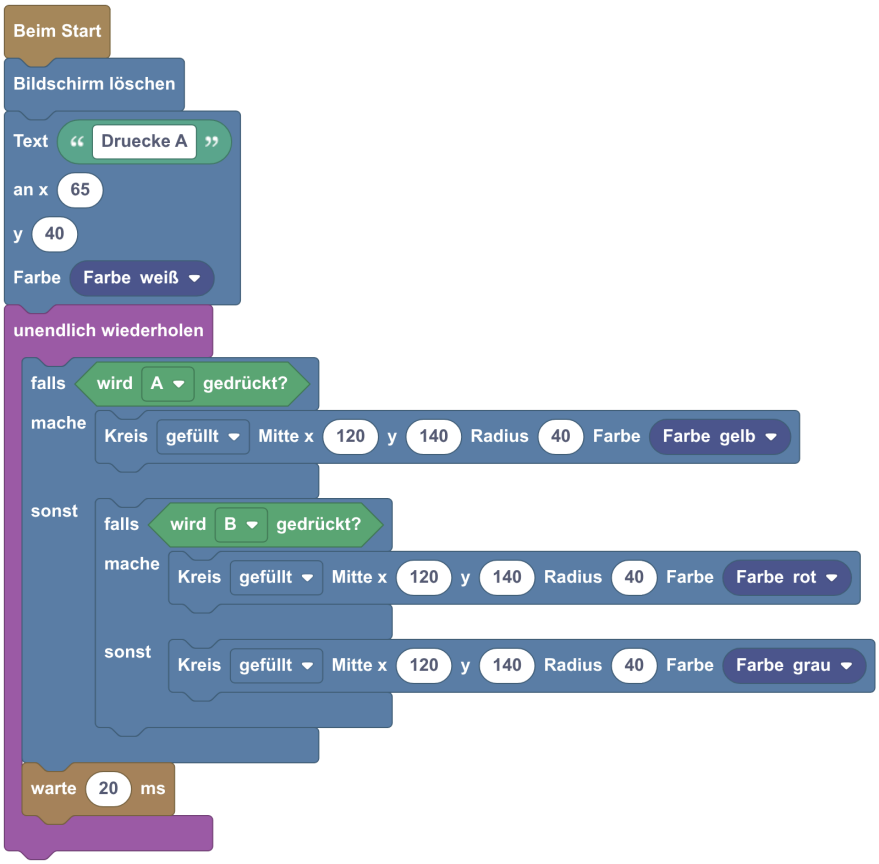


`heiss` beginnt bei 0. Im Zweig für die roten Tage wird `heiss` um 1 erhöht. Nach der Schleife erscheint das Ergebnis.

Lösungen zu Kapitel 8: Eingabe: Tasten und Joystick

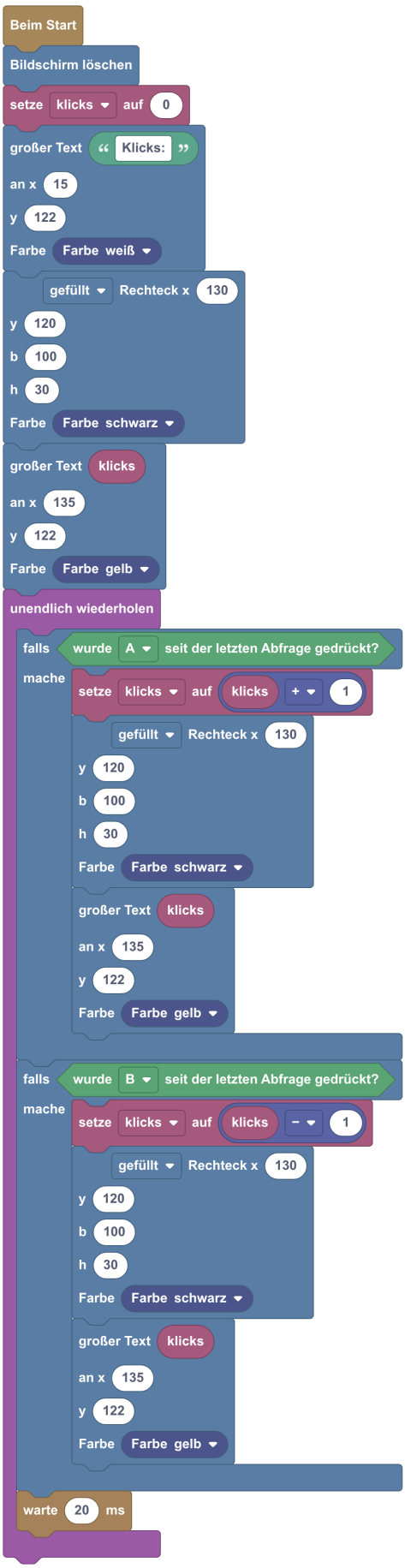
Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Zwei Leuchtknöpfe (`k08_loesung1.pbcp`)



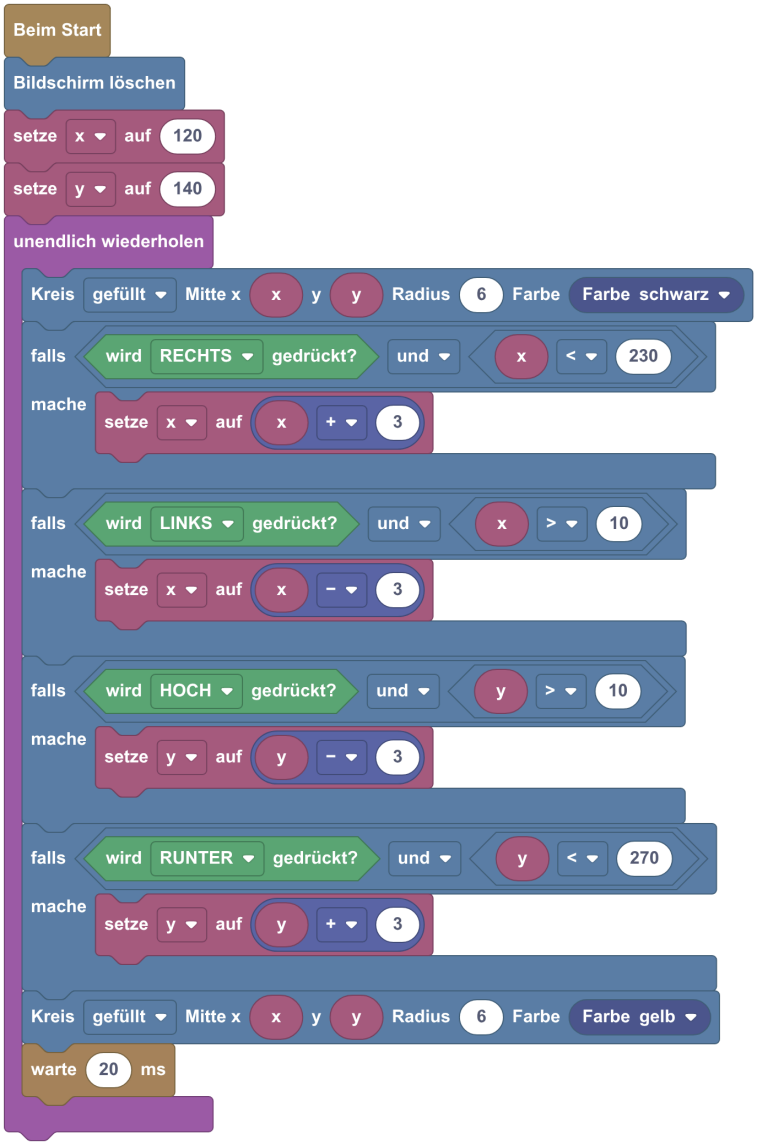
Ist A gedrückt, wird der Kreis gelb. Sonst fragt das zweite „falls / sonst“ nach B: rot oder grau. Drückst du beide Tasten, gewinnt A, weil A zuerst gefragt wird.

Aufgabe 2 – Hoch und runter zählen (`k08_loesung2.pbcp`)



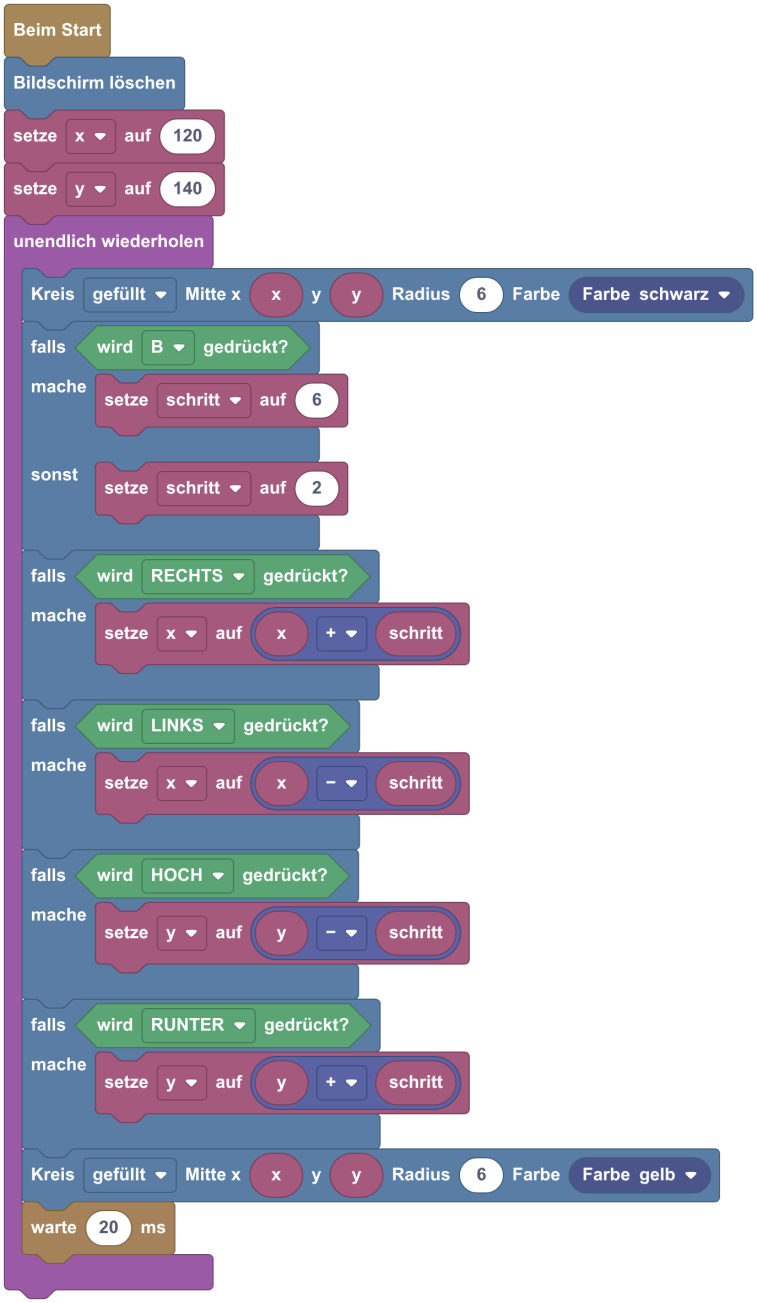
Ein zweites „falls“ fragt „wurde B … gedrückt?“, macht `klicks` um 1 kleiner und malt die Zahl neu.

Aufgabe 3 – Nicht aus dem Bild (`k08_loesung3.pbcp`)



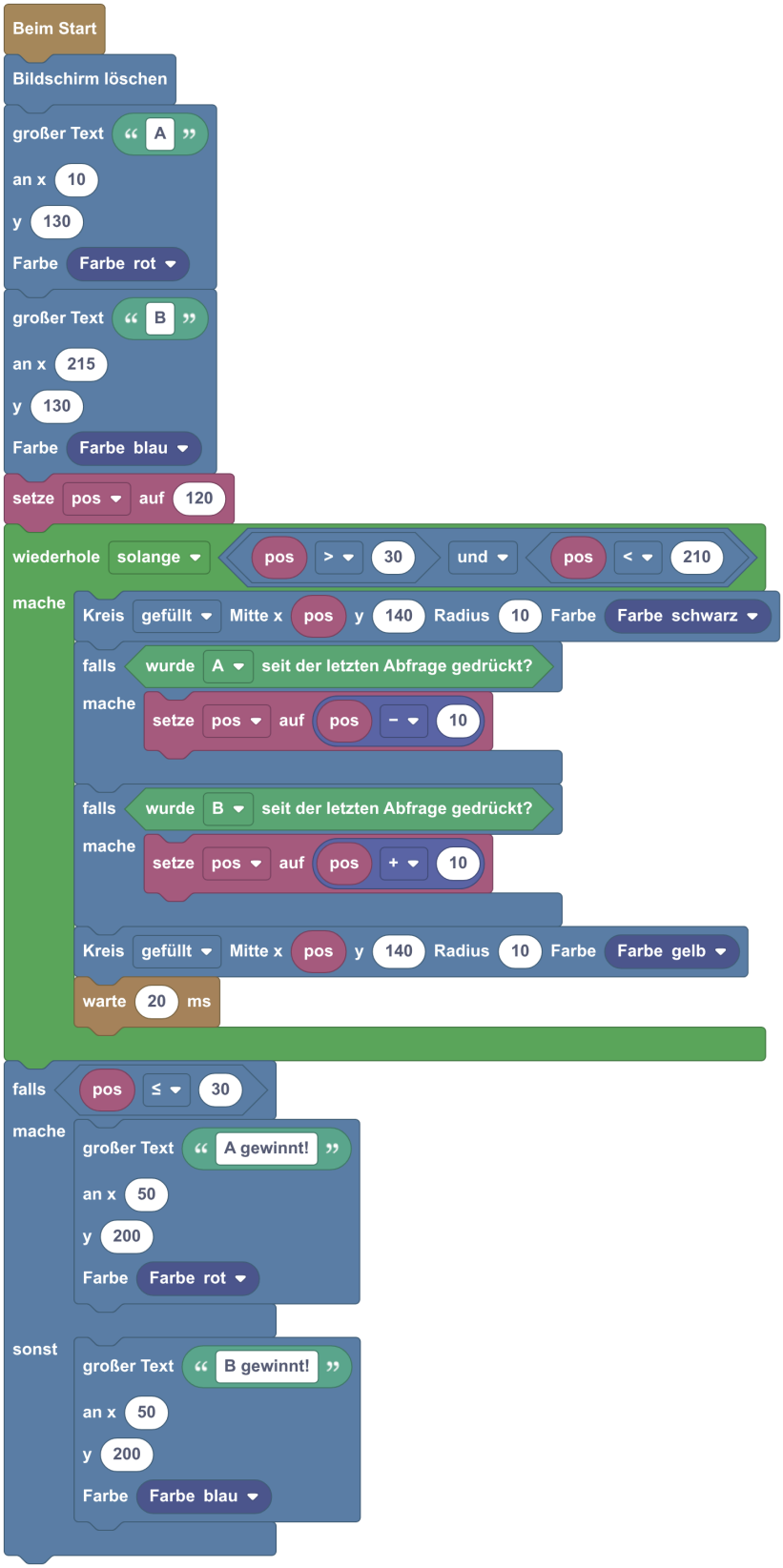
Jede Richtung bekommt eine „und"-Bedingung. Der Punkt hat den Radius 6. Bei x = 230 reicht er bis 236 – noch ein Schritt um 3 Pixel, und er wäre am Rand (240). Für oben und links gilt dasselbe mit 10, für unten mit 270.

Aufgabe 4 – Turbo (`k08_loesung4.pbcp`)



Zu Beginn jeder Runde legt ein „falls / sonst“ fest, wie groß `schritt` ist: 6 mit B, sonst 2. Die vier Bewegungs-Blöcke rechnen mit `schritt`.

Aufgabe 5 – Tauziehen für zwei (`k08_loesung5.pbcp`)



Die Schleife läuft, solange der Kreis noch zwischen 30 und 210 steht. Danach entscheidet ein „falls / sonst“, wer gewonnen hat. „wurde … gedrückt?“ sorgt dafür, dass Festhalten nicht hilft – nur schnelles Drücken.

Lösungen zu Kapitel 9: Ausgabe: LEDs und Töne

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Lauflicht rückwärts (`k09_loesung1.pbcp`)



Die Reihenfolge ist grün, gelb, rot. Doppelt so schnell heißt: halbe Wartezeit, also 100 ms.

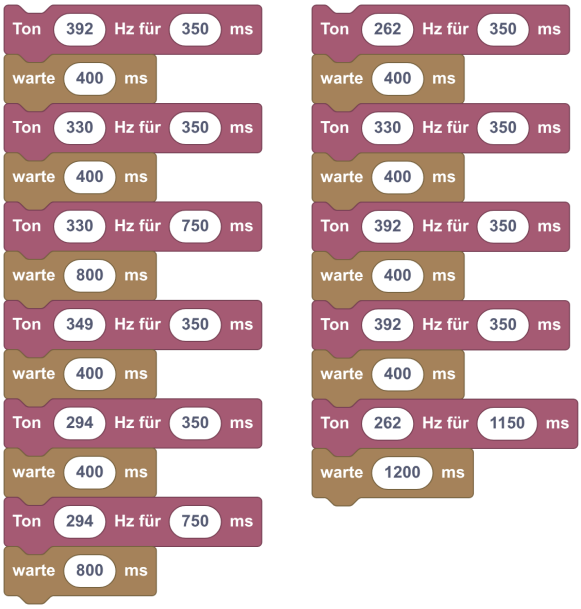
Aufgabe 2 – Mischfarben (`k09_loesung2.pbcp`)



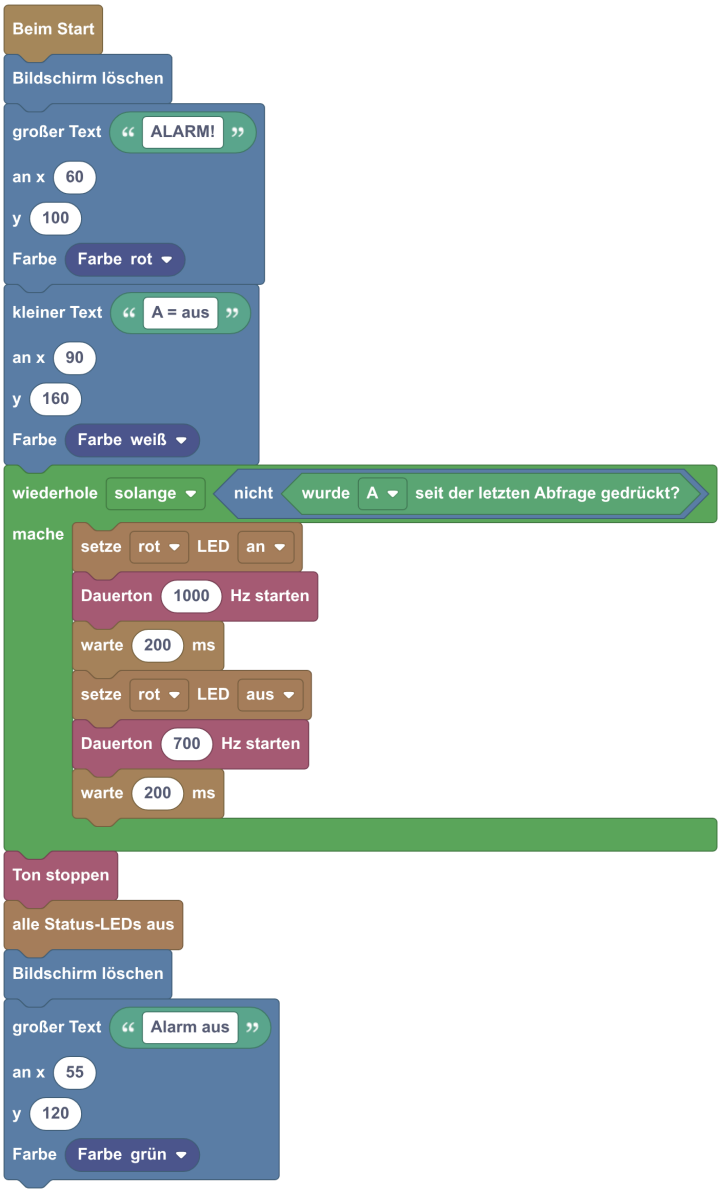
Farbe	rot	grün	blau
gelb	255	255	0
türkis	0	255	255
lila	255	0	255
weiß	255	255	255

Aufgabe 3 – Hänschen klein, zweite Zeile (`k09_loesung3.pbcp`)

Die Datei enthält das ganze Lied: die erste Zeile aus 9.4 und darunter die zweite. Hier ist die zweite Zeile, in zwei Teilen nebeneinander:

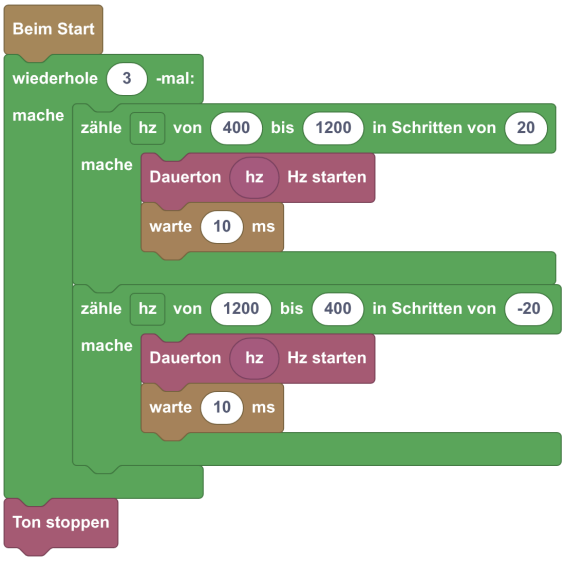


Aufgabe 4 – Alarmanlage (`k09_loesung4.pbcp`)



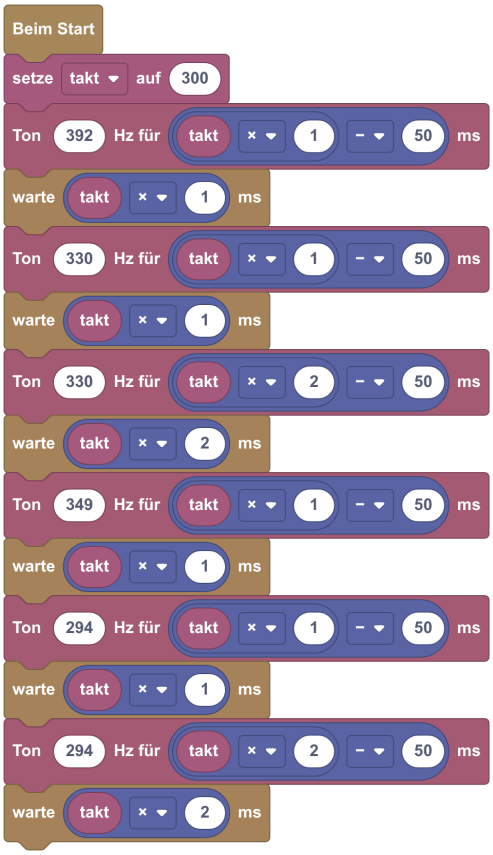
Die Schleife läuft, solange A **nicht** gedrückt wurde. „wurde … gedrückt?“ merkt sich den Druck auch dann, wenn das Programm gerade bei einem „warte“ steht. Nach der Schleife werden Ton und LEDs ausgeschaltet.

Aufgabe 5 – Heulende Sirene (`k09_loesung5.pbcp`)



Zwei Zählschleifen: Die erste zählt `hz` in 20er-Schritten von 400 auf 1200, die zweite mit der Schrittweite −20 wieder hinunter. Jeder Wert klingt 10 ms lang.

Aufgabe 6 – Das Tempo in einer Variable (`k09_loesung6.pbcp`)



Eine Viertelnote wartet `takt` ms, eine halbe Note `takt` × 2 ms. Der Ton ist jeweils 50 ms kürzer. Mit `takt` = 300 klingt das Lied schneller als mit den 400 ms aus 9.4.

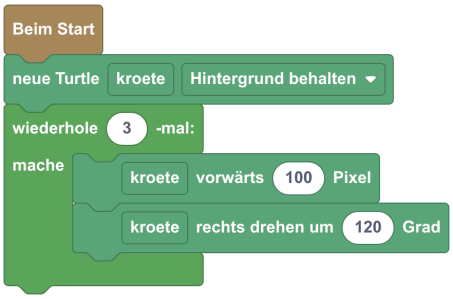
ANHANG C – LÖSUNGEN

Lösungen zu Kapitel 10: Turtle-Grafik

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

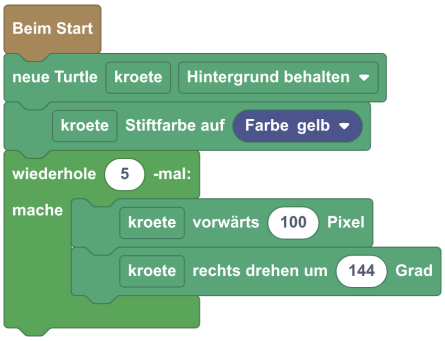
Aufgabe 1 – Dreieck (`k10_loesung1.pbcp`)

Ein Dreieck hat 3 Ecken. Die Turtle dreht sich an jeder Ecke um 360 ÷ 3 = 120 Grad.



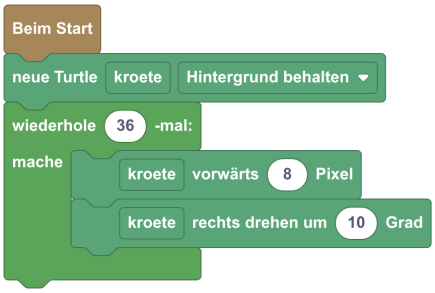
Aufgabe 2 – Ein Stern (`k10_loesung2.pbcp`)

Es entsteht ein fünfzackiger Stern. Die Turtle dreht sich insgesamt 5 × 144° = 720° – das sind **zwei** ganze Umdrehungen. Deshalb kreuzen sich die Linien.



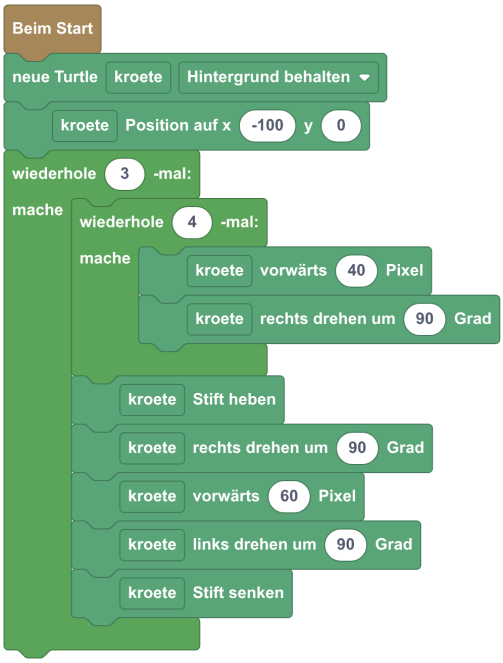
Aufgabe 3 – Ein Kreis (`k10_loesung3.pbcp`)

Die Turtle malt ein Vieleck mit 36 Ecken (36 × 10° = 360°). Die Seiten sind nur 8 Pixel lang und die Ecken sehr flach. Deshalb sieht das Vieleck fast wie ein Kreis aus.



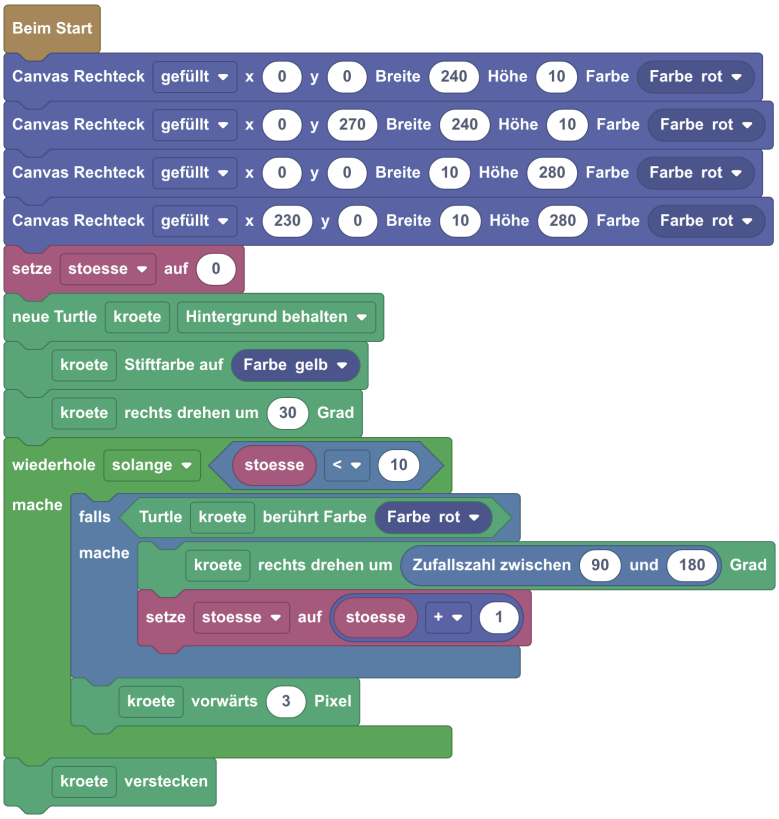
Aufgabe 4 – Drei Quadrate (`k10_loesung4.pbcp`)

Die innere Schleife malt ein Quadrat. Danach läuft die Turtle mit gehobenem Stift 60 Pixel nach rechts: 40 Pixel Quadratbreite plus 20 Pixel Abstand.



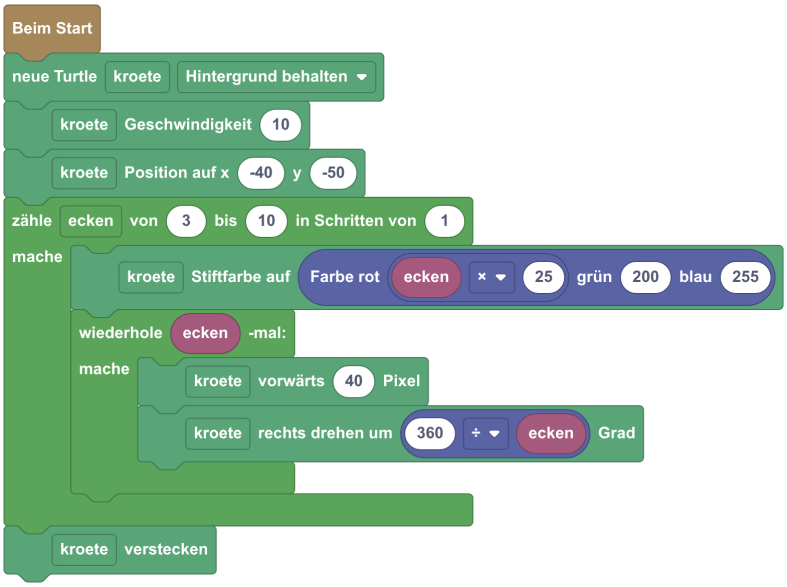
Aufgabe 5 – Zehn Stöße (`k10_loesung5.pbcp`)

`stoesse` startet bei 0. Bei jeder Wandberührung wird die Variable im „falls“-Block um 1 größer. Nach der zehnten Berührung endet die Schleife, und die Turtle versteckt sich.



Aufgabe 6 – Die Vieleck-Maschine (`k10_loesung6.pbcp`)

Die Zählschleife setzt `ecken` nacheinander auf 3, 4, 5 … 10. Die innere Schleife läuft `ecken`-mal, und der Drehwinkel ist jedes Mal 360 ÷ `ecken`. Der Rot-Anteil der Farbe ist `ecken` × 25: Jedes Vieleck bekommt so eine etwas andere Farbe. Die Turtle startet etwas links unterhalb der Mitte (x = −40, y = −50), damit auch das große Zehneck auf den Bildschirm passt.

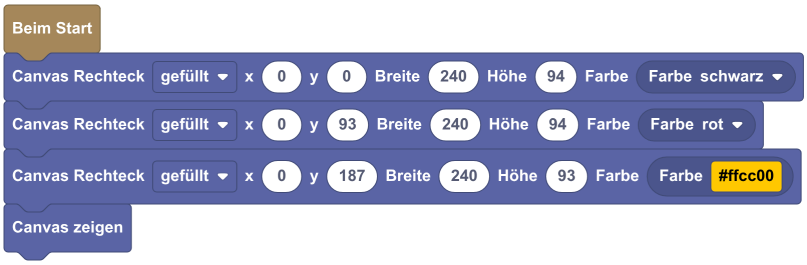


Lösungen zu Kapitel 11: Der Canvas

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Eine Flagge (`k11_loesung1.pbcp`)

Drei Rechtecke, je 240 Pixel breit und etwa 93 Pixel hoch. Der zweite Streifen beginnt bei y = 93, der dritte bei y = 187. Der erste Block (schwarz) ist eigentlich nicht nötig, weil der Canvas am Anfang schwarz ist.

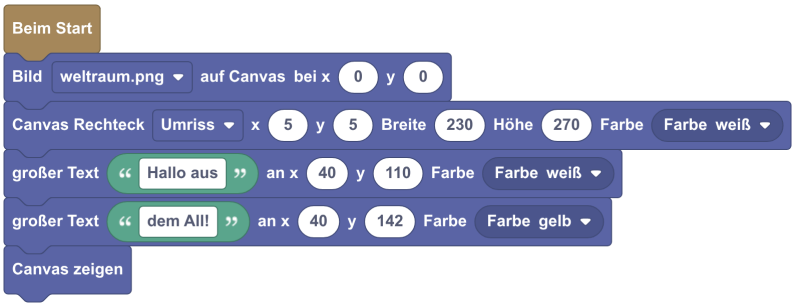


Aufgabe 2 – Wo ist das Bild?

Der Bildschirm bleibt schwarz. Das Programm malt die Landschaft zwar auf den Canvas, aber der Canvas liegt nur im Speicher. Ohne „Canvas zeigen“ kommt das Bild nie auf den Bildschirm.

Aufgabe 3 – Postkarte aus dem All (`k11_loesung3.pbcp`)

Der Rahmen ist ein Rechteck mit der Einstellung „Umriss“. Er beginnt 5 Pixel vom Rand entfernt.



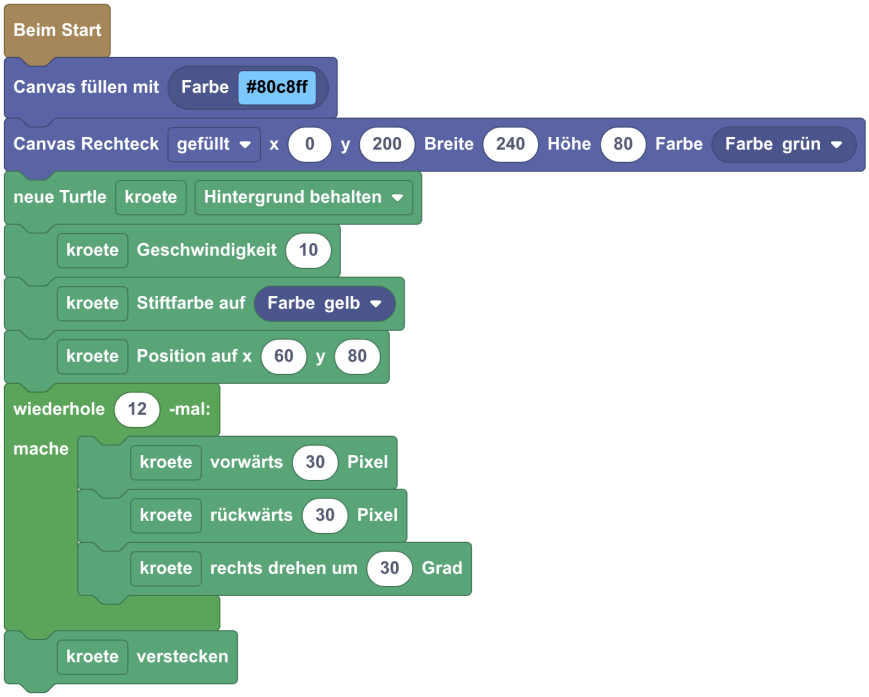
Aufgabe 4 – Sternenhimmel (`k11_loesung4.pbcp`)

Jeder Durchlauf der Schleife setzt einen Pixel an eine neue zufällige Stelle. Die Zufallszahlen gehen bis 239 und 279, denn der Canvas zählt ab 0.



Aufgabe 5 – Die Turtle malt eine Sonne (`k11_loesung5.pbcp`)

Zuerst kommt der Hintergrund auf den Canvas, danach die Turtle mit „Hintergrund behalten“. Oben rechts liegt für die Turtle bei x = 60, y = 80. Jeder Strahl: 30 Pixel vorwärts, 30 Pixel zurück, 30 Grad drehen. 12 × 30° = 360°.



Aufgabe 6 – Fischschwarm (`k11_loesung6.pbcp`)

Der Text „Fischschwarm“ steht zwischen Hintergrund und Fischen. So bleibt der Hintergrund auch im Emulator erhalten. Die Fische sind 24 Pixel breit; deshalb geht die Zufallszahl für x nur bis 216.

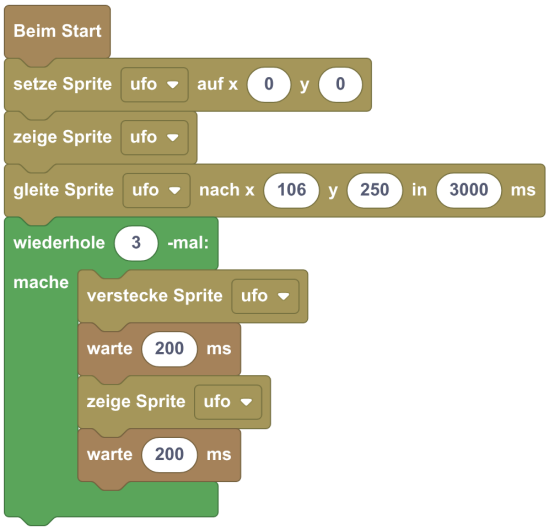


Lösungen zu Kapitel 12: Sprites

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern). Lade vorher die Bilder hoch und lege die Sprites an (Namen siehe Aufgabe).

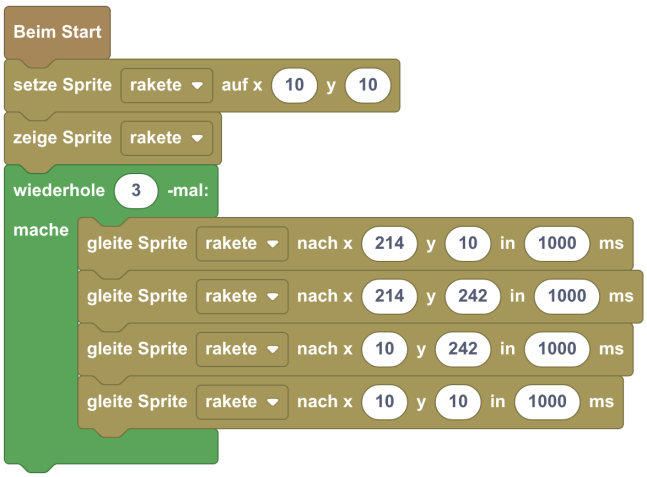
Aufgabe 1 – Ufo-Landung (`k12_loesung1.pbcp`)

Das Ufo startet bei (0 | 0). „gleite“ bringt es in 3000 ms zur Stelle 106, 250. Erst danach beginnt die Blink-Schleife – das Programm wartet ja, bis das Ufo angekommen ist.



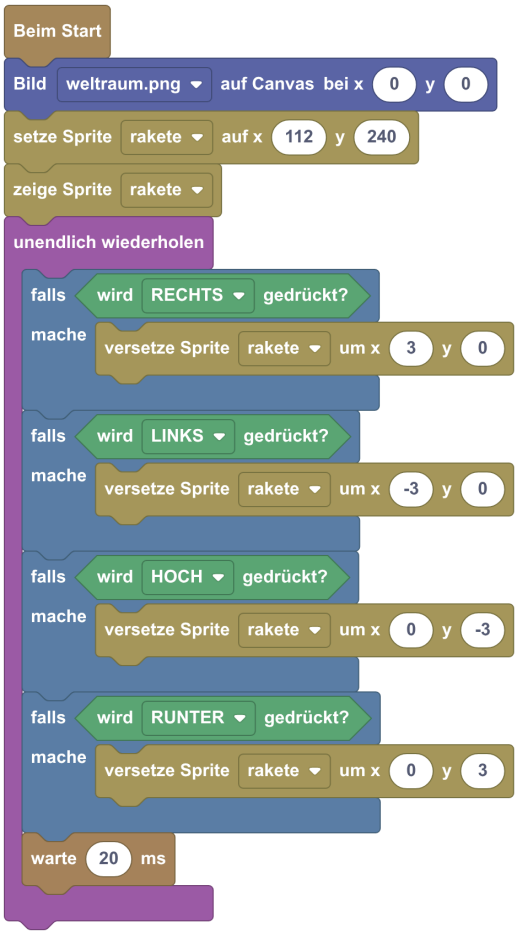
Aufgabe 2 – Rundflug (`k12_loesung2.pbcp`)

Die Rakete ist 16 Pixel breit und 28 Pixel hoch. Damit sie ganz zu sehen ist, darf x höchstens 240 − 16 = 224 und y höchstens 280 − 28 = 252 sein. Die Lösung hält 10 Pixel Abstand zum Rand.



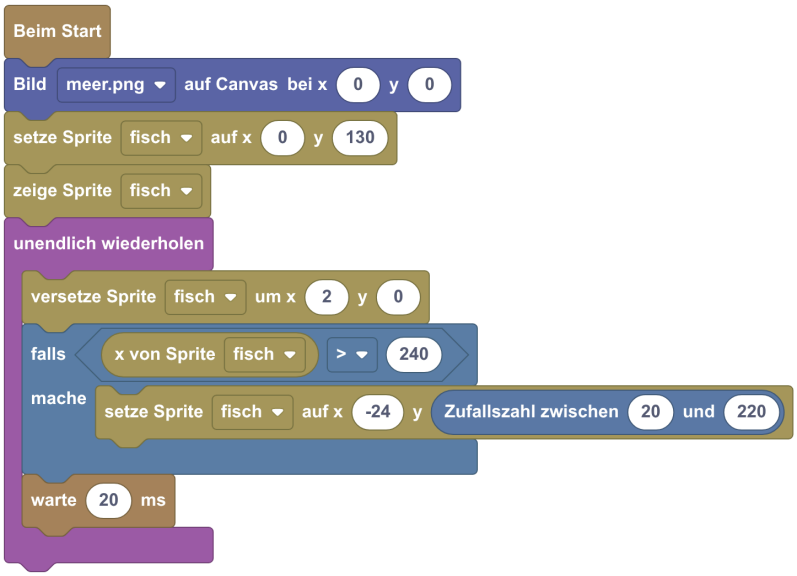
Aufgabe 3 – Rakete im Weltall (`k12_loesung3.pbcp`)

Vier „falls … gedrückt?“-Blöcke in der Schleife. Nach oben heißt: y wird kleiner. Das Bild auf dem Canvas ist automatisch der Hintergrund.



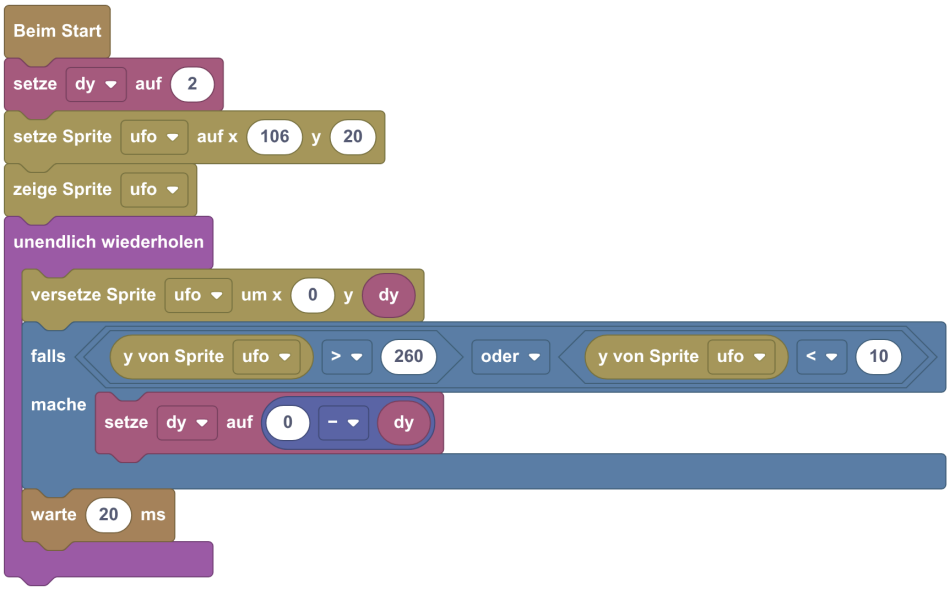
Aufgabe 4 – Neue Höhe (`k12_loesung4.pbcp`)

Statt „y von Sprite fisch“ steckt eine Zufallszahl im Block „setze Sprite“.



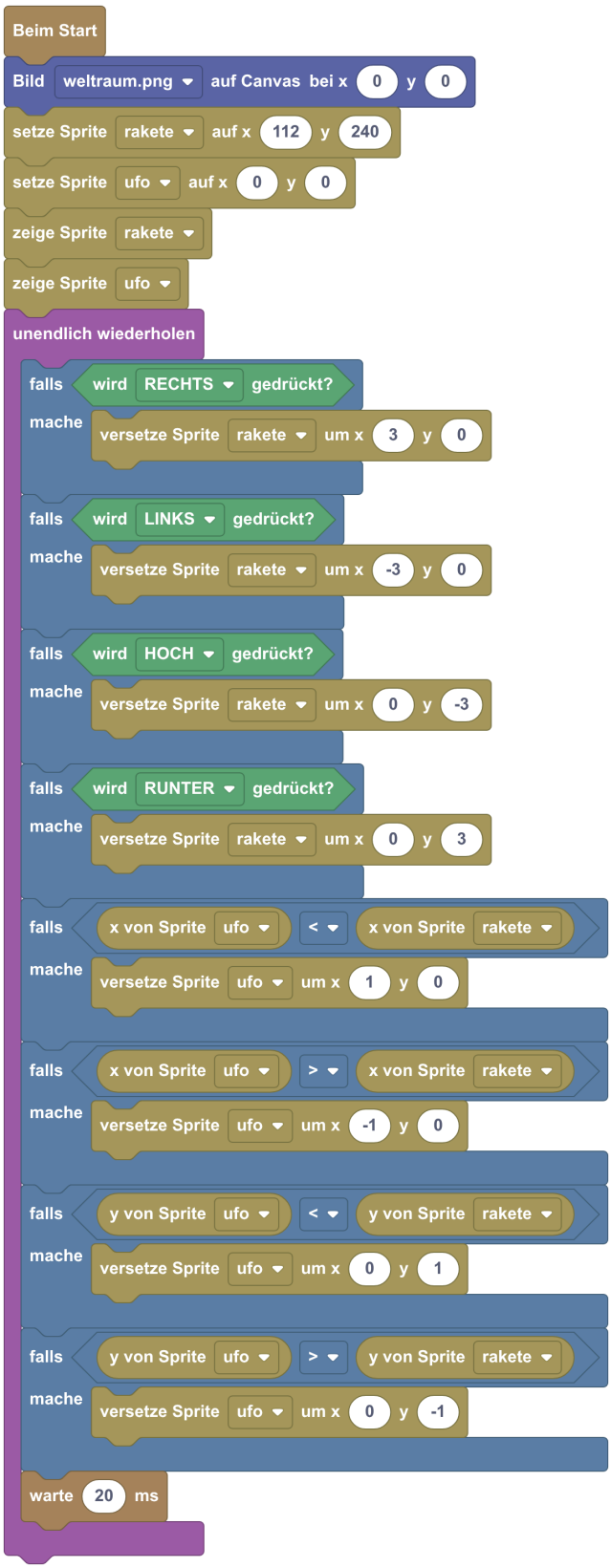
Aufgabe 5 – Auf und ab (`k12_loesung5.pbcp`)

`dy` startet bei 2. Die Rechnung 0 − `dy` dreht das Vorzeichen um: aus 2 wird −2 und aus −2 wieder 2.



Aufgabe 6 – Das Ufo verfolgt dich (`k12_loesung6.pbcp`)

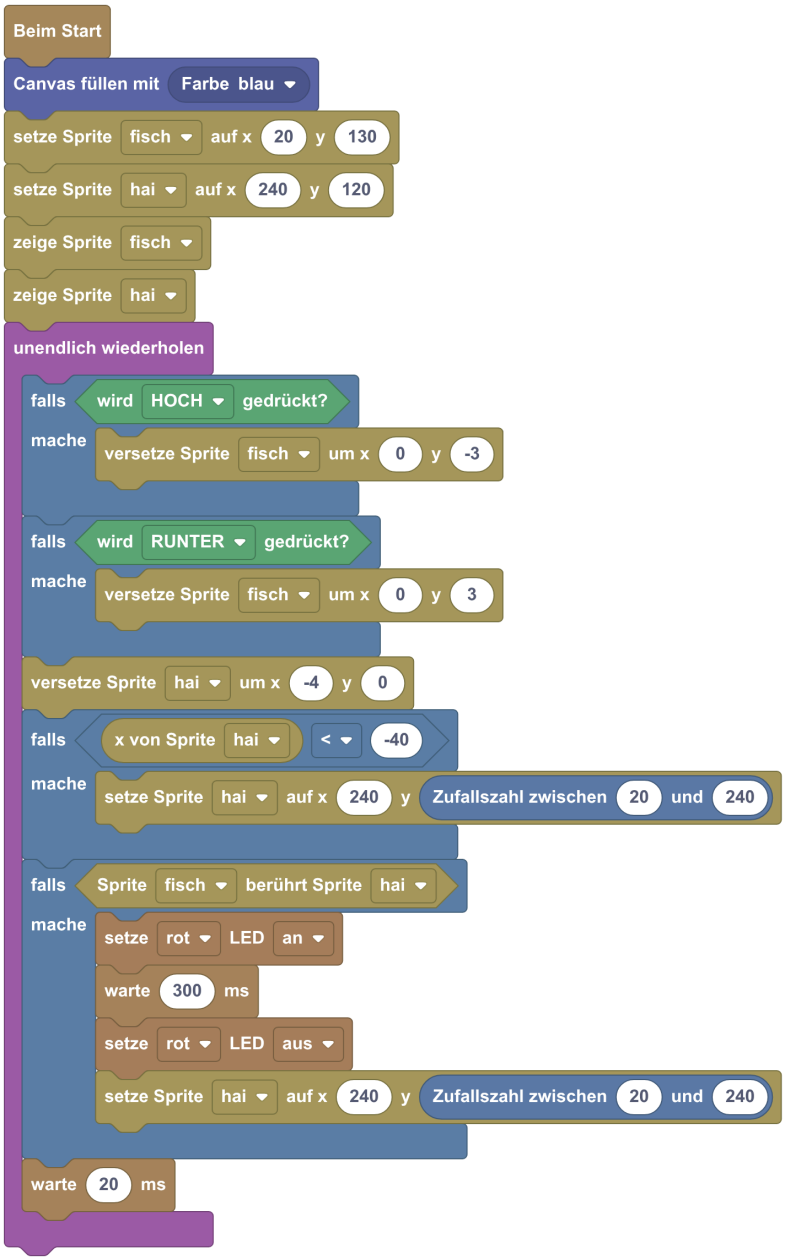
Vier „falls“-Blöcke vergleichen die Positionen von Ufo und Rakete und rücken das Ufo jeweils 1 Pixel näher. Die Rakete ist mit 3 Pixeln pro Schritt schneller – so kannst du entkommen.


Beim Start
Bild weltraum.png auf Canvas bei x 0 y 0
setze Sprite rakete auf x 112 y 240
setze Sprite ufo auf x 0 y 0
zeige Sprite rakete
zeige Sprite ufo
unendlich wiederholen
falls wird RECHTS gedrückt?
mache versetze Sprite rakete um x 3 y 0
falls wird LINKS gedrückt?
mache versetze Sprite rakete um x -3 y 0
falls wird HOCH gedrückt?
mache versetze Sprite rakete um x 0 y -3
falls wird RUNTER gedrückt?
mache versetze Sprite rakete um x 0 y 3
falls x von Sprite ufo < x von Sprite rakete
mache versetze Sprite ufo um x 1 y 0
falls x von Sprite ufo > x von Sprite rakete
mache versetze Sprite ufo um x -1 y 0
falls y von Sprite ufo < y von Sprite rakete
mache versetze Sprite ufo um x 0 y 1
falls y von Sprite ufo > y von Sprite rakete
mache versetze Sprite ufo um x 0 y -1
warte 20 ms

Lösungen zu Kapitel 13: Kollisionen

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Rotes Licht (`k13_loesung1.pbcp`)

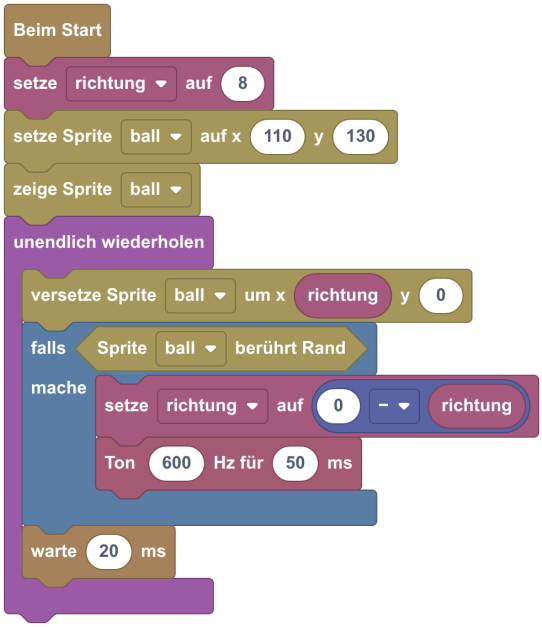


Im „falls … berührt Sprite …“-Block wird der Ton durch drei Blöcke ersetzt: rote LED an, 300 ms warten, rote LED aus. Während des Wartens steht das Spiel kurz still – das zeigt auch, dass du erwischt wurdest.

Aufgabe 2 – Wüste statt Wiese

Das Auto fährt einfach über die gelbe Wüste, es passiert nichts. Der Block „berührt Farbe grün“ sucht nach Grün – und auf dem Hintergrund gibt es kein Grün mehr. Zum Reparieren stellst du im Kollisions-Block ebenfalls „gelb“ ein (oder malst die Wiese wieder grün). Die Farbe im Kollisions-Block und die gemalte Farbe müssen genau gleich sein.

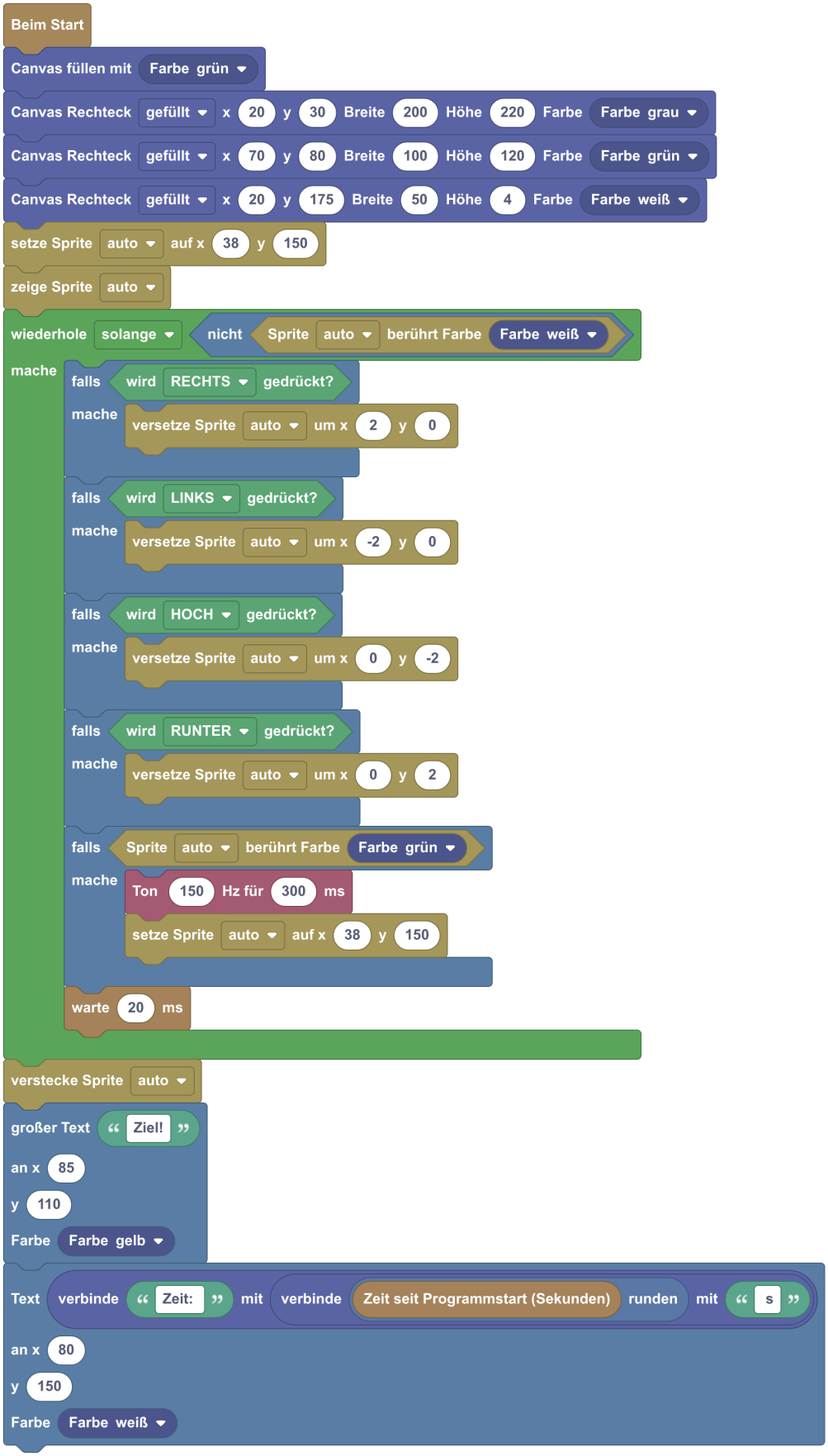
Aufgabe 3 – Pling! (`k13_loesung3.pbcp`)



`richtung` startet bei 8 statt 4. Im „falls … berührt Rand“ kommt ein kurzer, hoher Ton dazu.

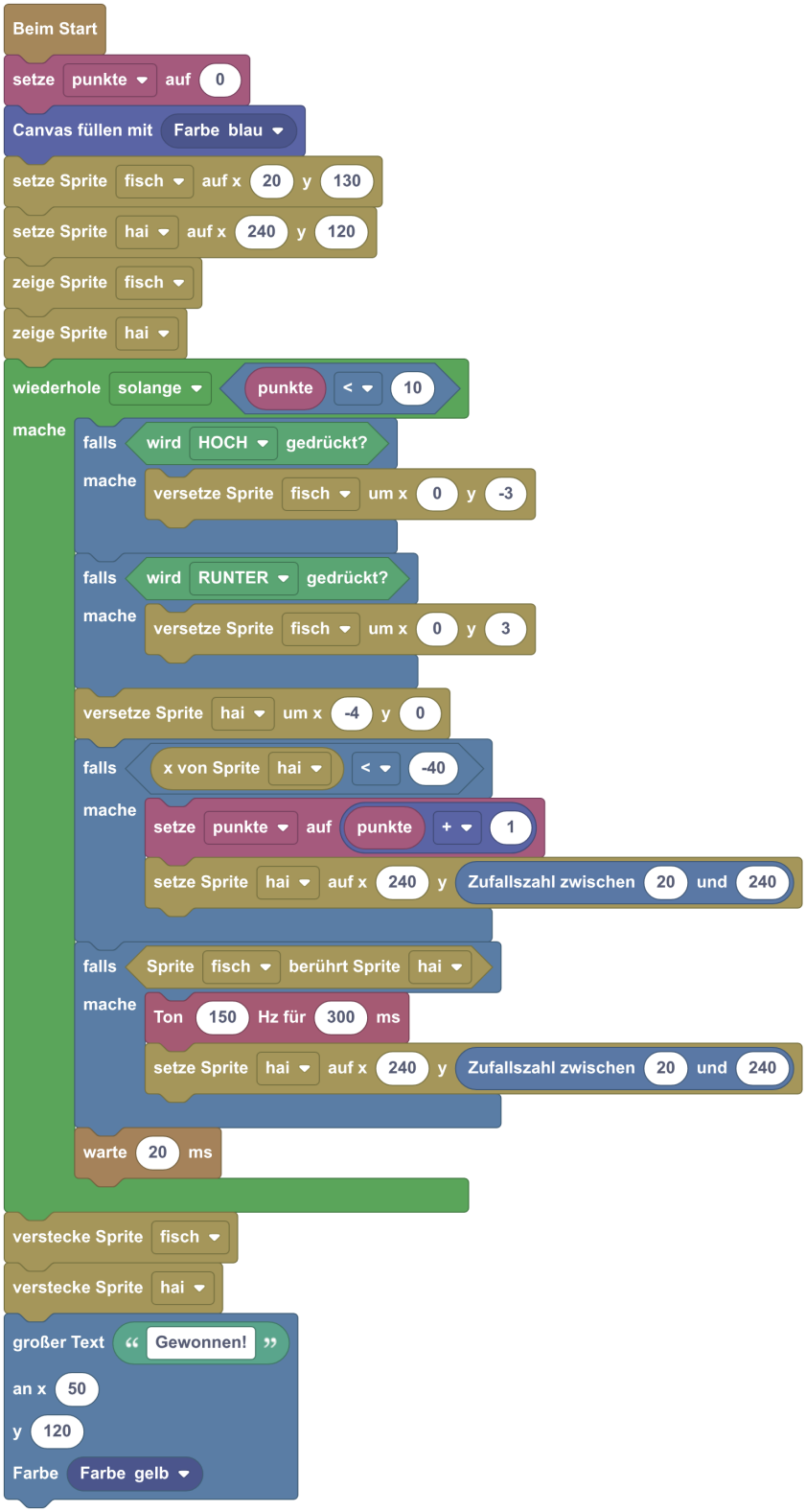
Aufgabe 4 – Ziellinie (`k13_loesung4.pbcp`)

Die Ziellinie wird wie die Bahn auf den Canvas gemalt – als letztes, damit sie oben liegt. Die Schleife fragt mit „nicht“: Solange das Auto **nicht** auf Weiß steht, geht das Rennen weiter. Danach läuft das Programm unter der Schleife weiter.



Hast du gemerkt, dass man schummeln kann? Fährt das Auto am Start ein Stück rückwärts, ist es sofort im Ziel. Für Profis: Male oben in die Bahn ein gelbes Kontrollfeld. Merke dir in einer Variable, ob das Auto schon darüber gefahren ist, und lass die Ziellinie erst danach zählen.

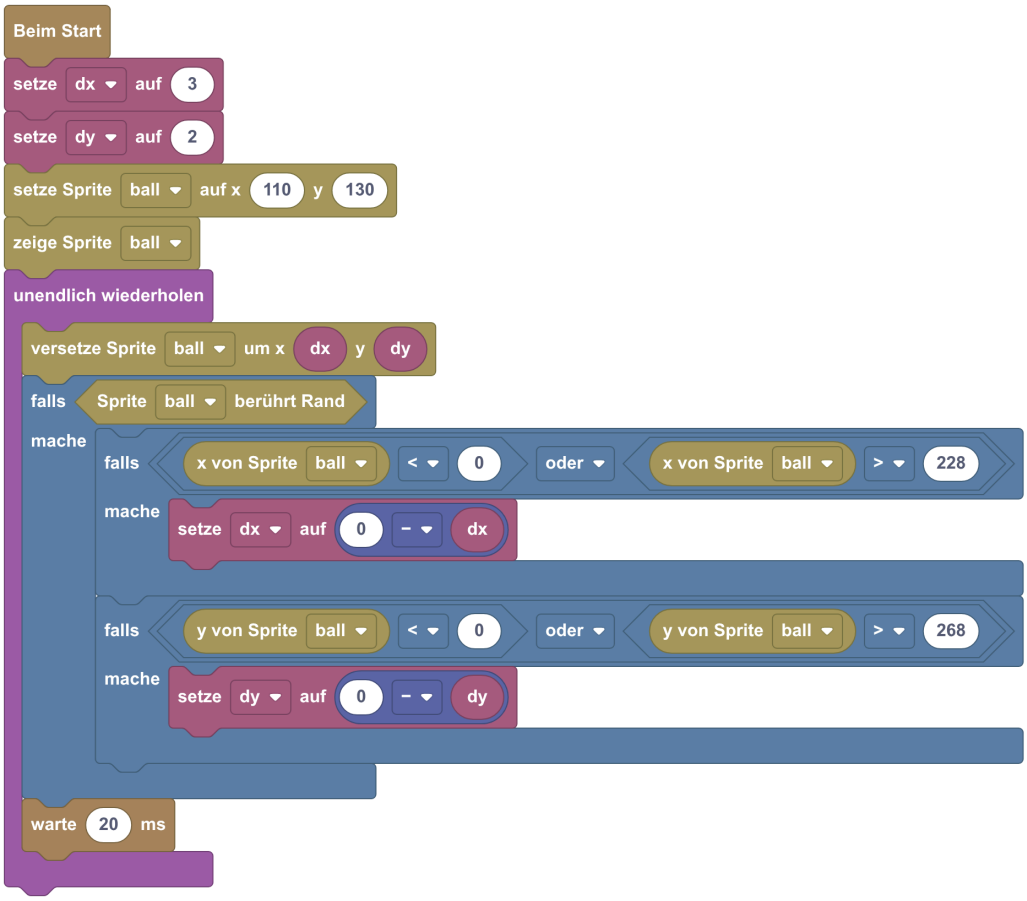
Aufgabe 5 – Ausgewichen! (`k13_loesung5.pbcp`)



Die Variable `punkte` startet bei 0. Der Punkt kommt in den „falls“-Block, der den Hai wieder nach rechts setzt: Dort ist der Hai an dir vorbei. Erwischt er dich, wird er ebenfalls nach rechts gesetzt, aber ohne Punkt. Die Schleife läuft, solange `punkte` kleiner als 10 ist.

Aufgabe 6 – Abprallen in alle Richtungen (`k13_loesung6.pbcp`)

Erst wenn der Ball den Rand berührt, schaut das Programm genauer hin: Ist x kleiner als 0 oder größer als 228 (240 minus 12 Pixel Ballbreite), war es der linke oder rechte Rand – dann dreht sich `dx` um. Ist y kleiner als 0 oder größer als 268 (280 − 12), war es oben oder unten – dann dreht sich `dy` um.



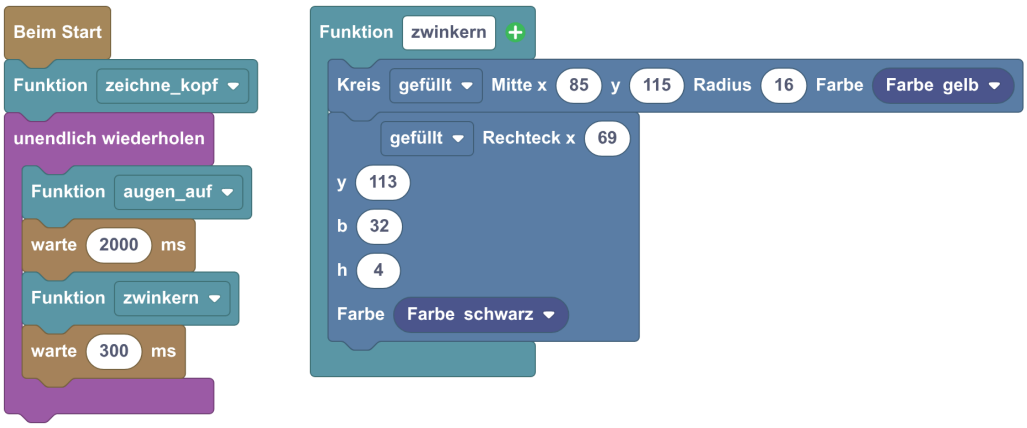
ANHANG C – LÖSUNGEN

Lösungen zu Kapitel 14: Funktionen

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

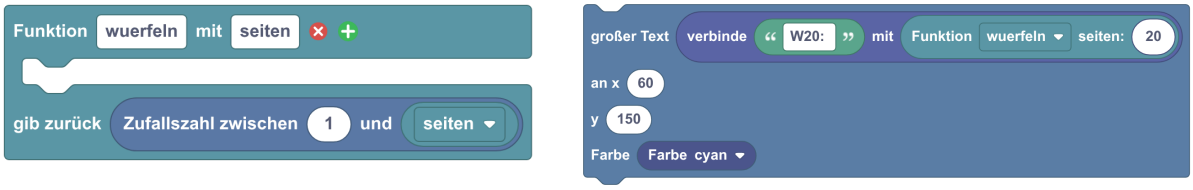
Aufgabe 1 – Zwinkern (`k14_loesung1.pbcp`)

Die neue Funktion `zwinkern` malt nur über das **linke** Auge einen gelben Kreis und einen schwarzen Strich. Das rechte Auge bleibt offen. Die Funktion `augen_zu` wird nicht mehr gebraucht; `zeichne_kopf` und `augen_auf` bleiben wie in 14.1.



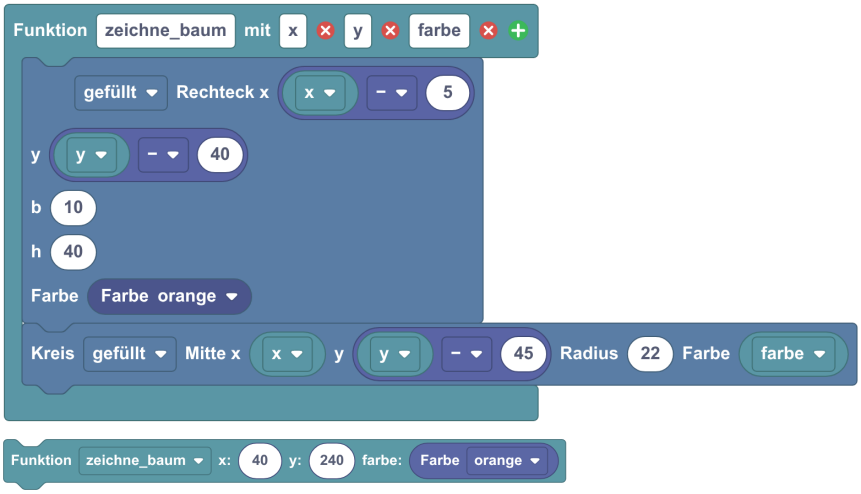
Aufgabe 2 – Würfel mit vielen Seiten (`k14_loesung2.pbcp`)

`wuerfeln` hat jetzt den Parameter `seiten`. Er steckt als obere Grenze in der Zufallszahl. Beim Aufruf trägst du 6 oder 20 ein (rechts der Aufruf für den 20er-Würfel).



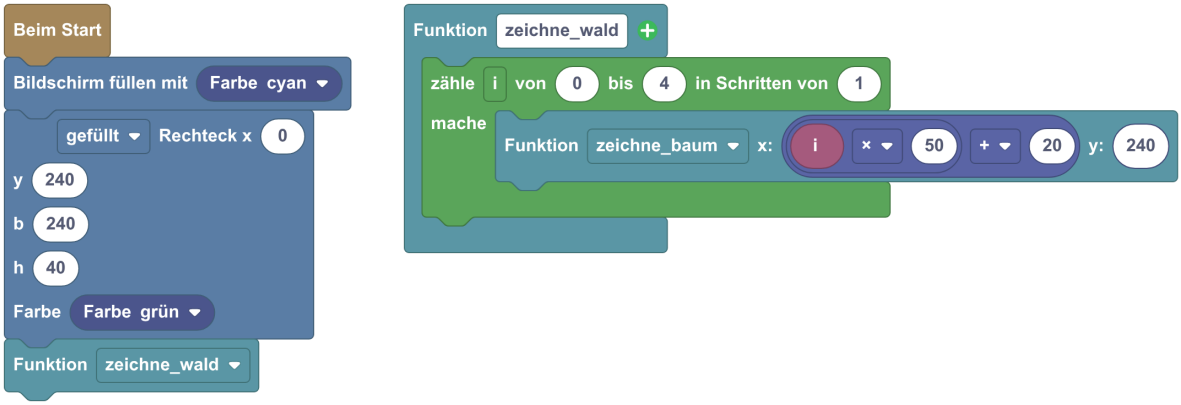
Aufgabe 3 – Herbstwald (`k14_loesung3.pbcp`)

Mit dem grünen + bekommt `zeichne_baum` den dritten Parameter `farbe`. Er ersetzt im Kreis-Block die Farbe Grün. In jeden Aufruf kommt ein Block „Farbe nach Namen“ (unten der erste Aufruf).



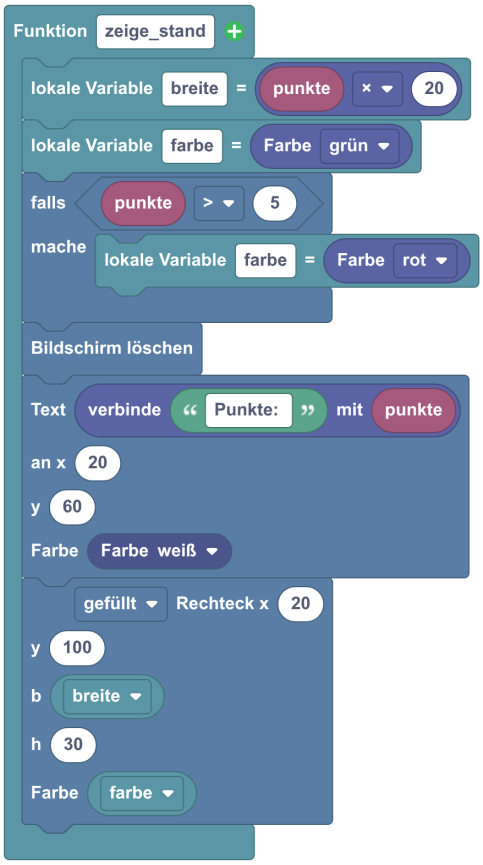
Aufgabe 4 – Ein Wald auf Bestellung (`k14_loesung4.pbcp`)

`zeichne_wald` ruft selbst wieder eine Funktion auf: `zeichne_baum`. Das ist erlaubt. Fehlt der Aufruf von `zeichne_wald` unter „Beim Start“, siehst du nur Himmel und Wiese – die Funktion wird nie ausgeführt. `zeichne_baum` bleibt wie in 14.2.



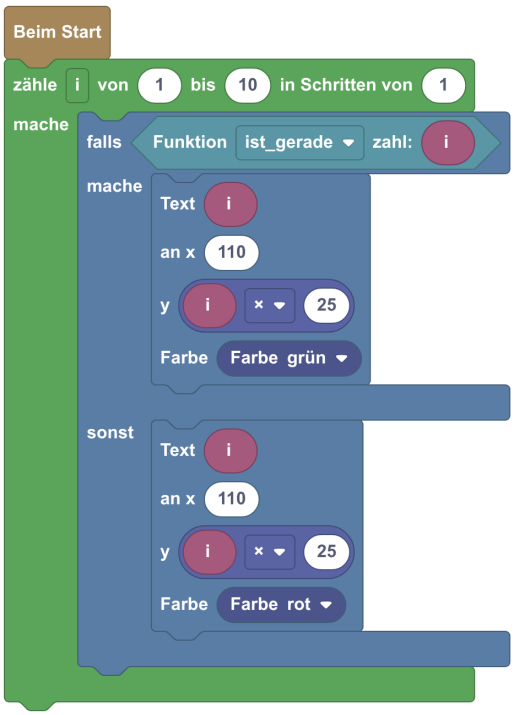
Aufgabe 5 – Roter Balken (`k14_loesung5.pbcp`)

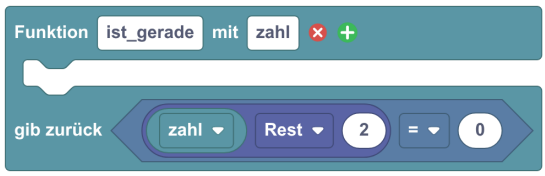
Die lokale Variable `farbe` bekommt zuerst Grün. Ein zweiter Block „lokale Variable farbe =“ im „falls“ setzt sie auf Rot. Der Rechteck-Block liest `farbe`. „Beim Start“ und `punkt_dazu` bleiben wie in 14.4.



Aufgabe 6 – Gerade oder ungerade? (`k14_loesung6.pbcp`)

In der „gib zurück“-Lücke steht ein Vergleich: Ist `zahl` Rest 2 gleich 0? Der Vergleich ist **wahr** oder **falsch** – genau das gibt die Funktion zurück. Deshalb kann ihr Aufruf direkt in den „falls“-Block.





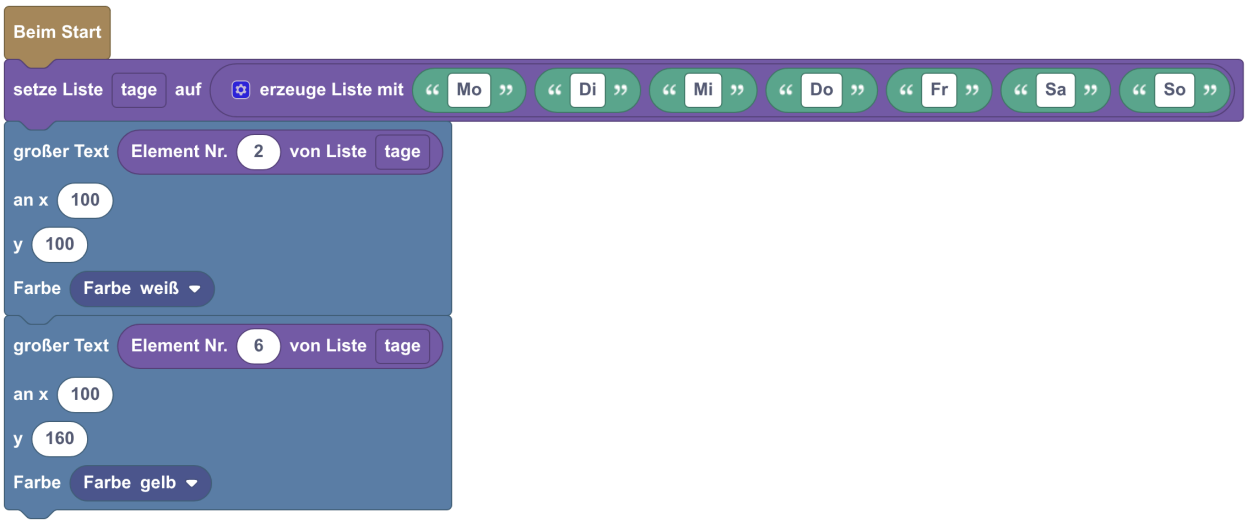
ANHANG C – LÖSUNGEN

Lösungen zu Kapitel 15: Listen

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

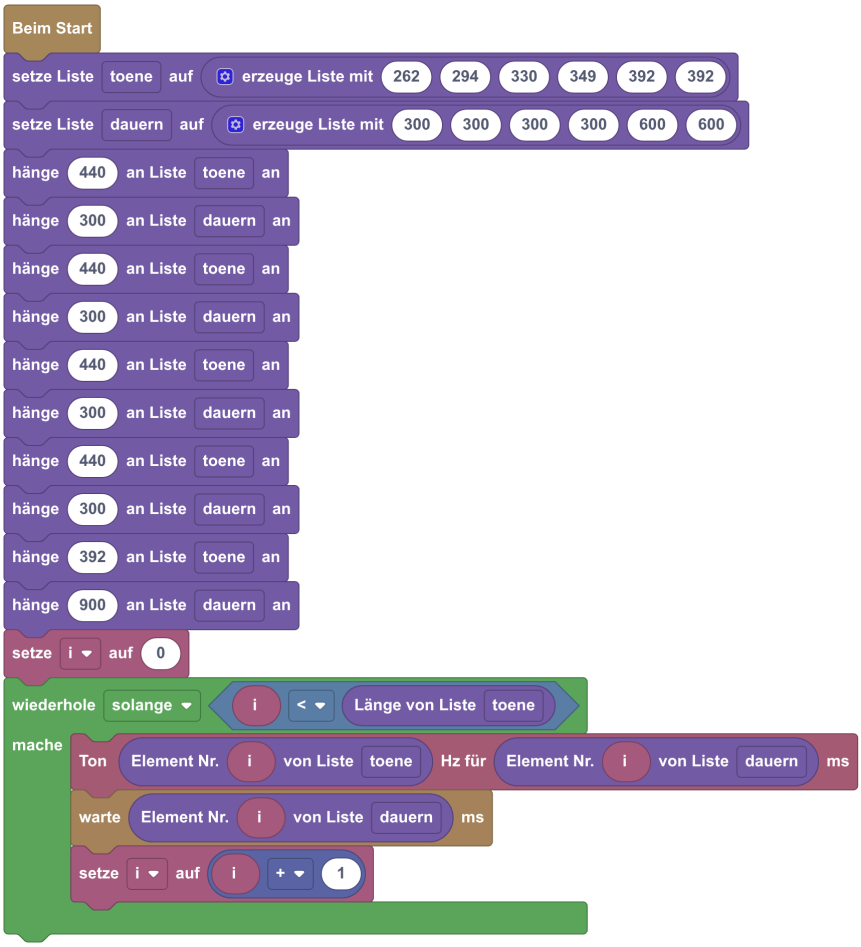
Aufgabe 1 – Wochentage (`k15_loesung1.pbcp`)

Element Nr. 2 ist „Mi“ – gezählt wird ab 0: Mo = 0, Di = 1, Mi = 2. „So“ hat die Nummer 6. Die Liste hat 7 Elemente, die letzte Nummer ist also 7 − 1 = 6.



Aufgabe 2 – Die Melodie geht weiter (`k15_loesung2.pbcp`)

Zu jedem Ton gehört eine Dauer. Deshalb wird immer an **beide** Listen angehängt. Es ist der Anfang von „Alle meine Entchen“. Unter den „hänge an“-Blöcken folgt die Schleife aus 15.6 unverändert.



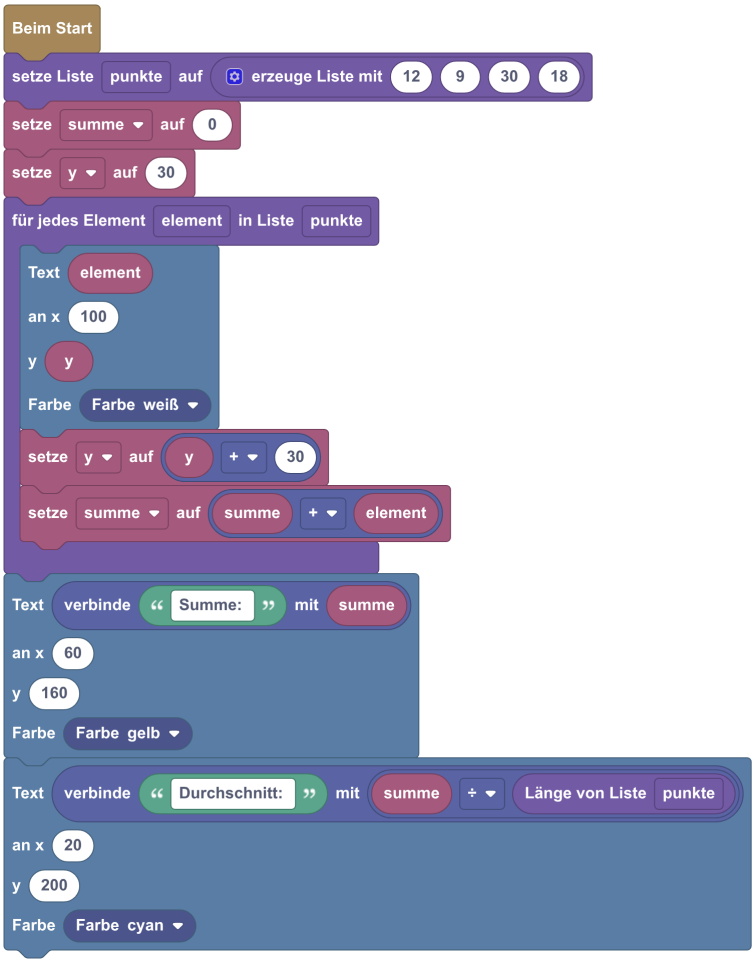
Aufgabe 3 – Glücksfarbe (`k15_loesung3.pbcp`)

„zufälliges Element aus Liste farben“ steckt direkt in „Bildschirm füllen mit“.



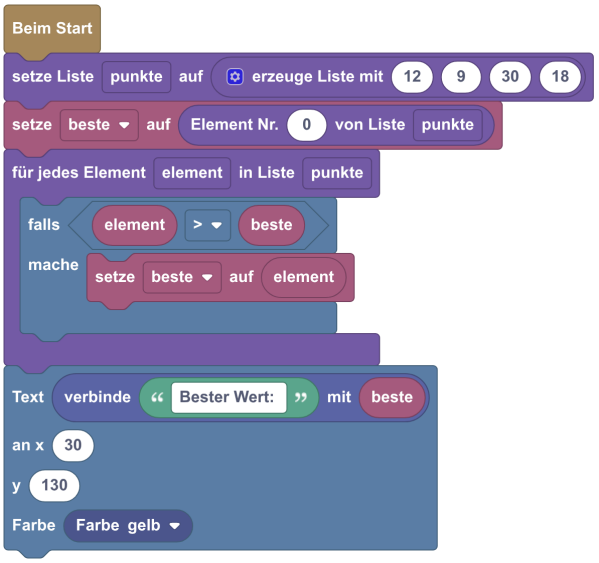
Aufgabe 4 – Der Durchschnitt (`k15_loesung4.pbcp`)

Unten kommt ein Text-Block dazu: `summe` ÷ Länge von Liste `punkte`. Das Ergebnis ist 69 ÷ 4 = 17.25. PBC+ schreibt Kommazahlen mit Punkt.



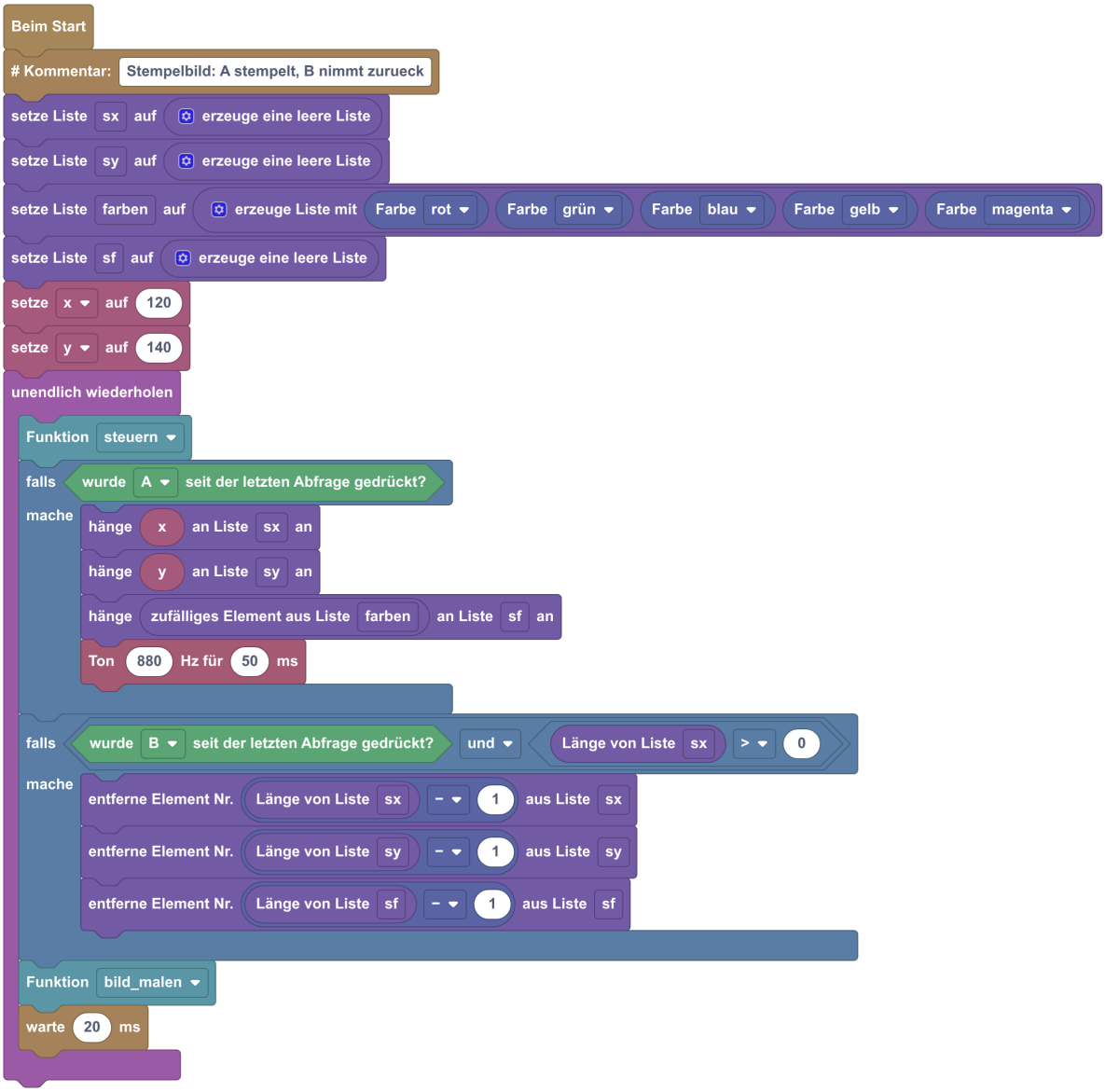
Aufgabe 5 – Der beste Wert (`k15_loesung5.pbcp`)

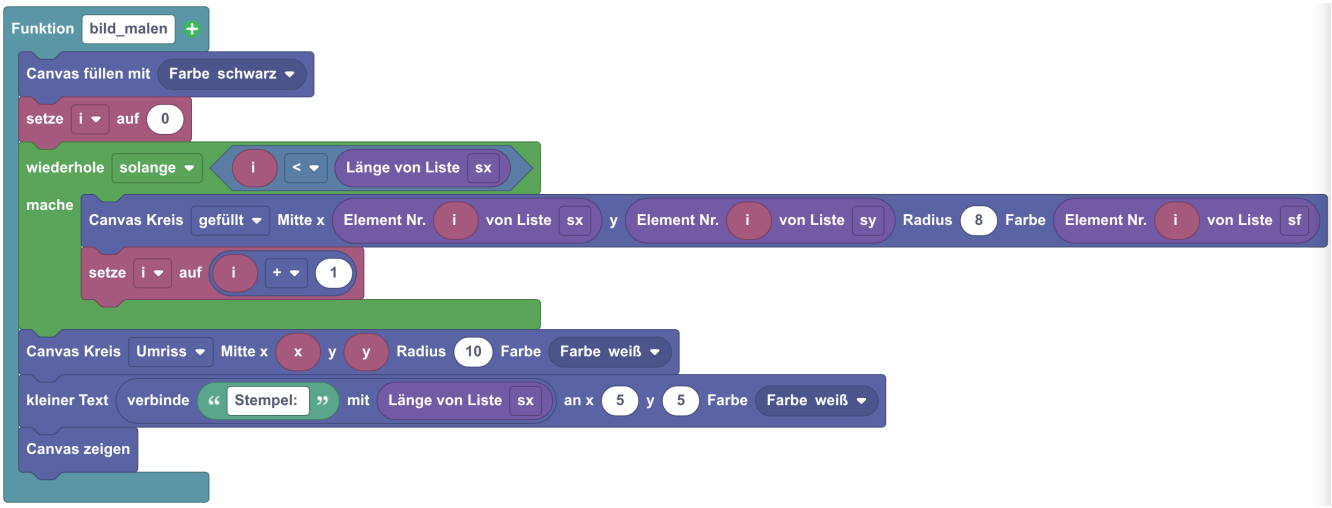
`beste` startet mit dem ersten Element (12). Die Schleife vergleicht jedes Element mit `beste`. Am Ende steht dort 30.



Aufgabe 6 – Bunte Stempel (`k15_loesung6.pbcp`)

Neu sind die Listen `farben` und `sf`. Beim Stempeln wird eine zufällige Farbe an `sf` angehängt, beim Zurücknehmen auch dort das letzte Element entfernt. `bild_malen` nimmt als Farbe Element Nr. `i` von `sf`. Die Funktion `steuern` bleibt gleich.





Lösungen zu Kapitel 16: Texte und Konsole

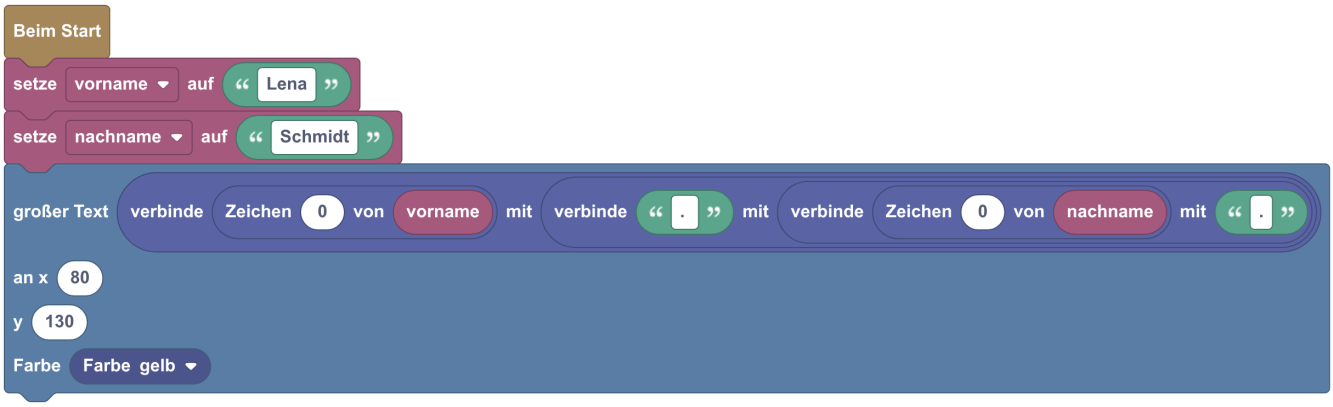
Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

Aufgabe 1 – Dein Name (`k16_loesung1.pbcp`)

Nur der Text im Block „setze wort auf“ ändert sich, zum Beispiel auf „LENA“. „Laenge“ zeigt dann 4 – die Anzahl der Buchstaben. Ist der Name kürzer als 4 Buchstaben, zeigt „Teil 0 bis 4“ einfach den ganzen Namen. Es gibt keinen Fehler, denn ein Teilstück nimmt nur die Zeichen, die es gibt.

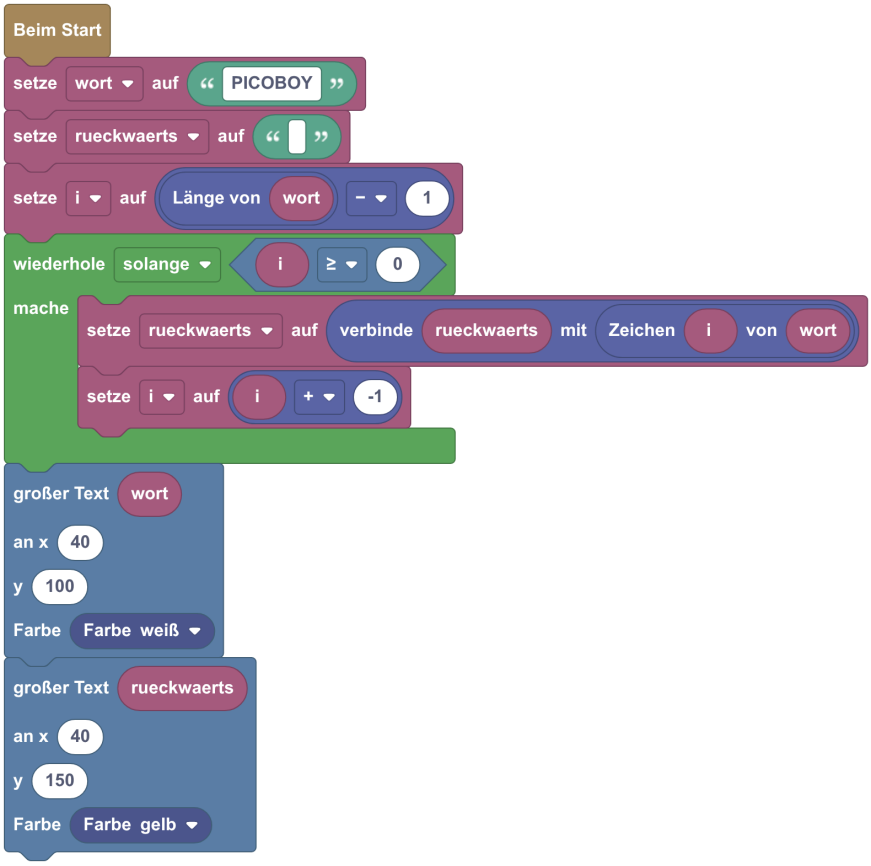
Aufgabe 2 – Initialen (`k16_loesung2.pbcp`)

Drei „verbinde“-Blöcke ineinander: erstes Zeichen von `vorname`, „. “, erstes Zeichen von `nachname`, „.“.



Aufgabe 3 – Rückwärts (`k16_loesung3.pbcp`)

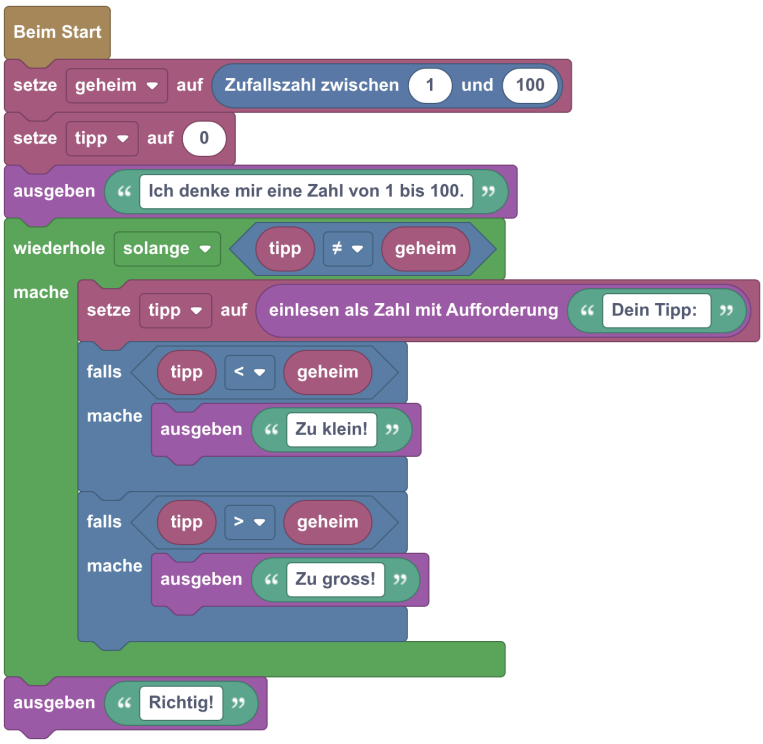
`i` startet beim letzten Zeichen (Nummer Länge − 1 = 6) und wird in jedem Durchlauf um 1 kleiner. Jedes Zeichen wird **hinten** an `rueckwaerts` angehängt – so kommt das letzte Zeichen zuerst.



Aufgabe 4 – Zahlenraten (`k16_loesung4.pbcp`)

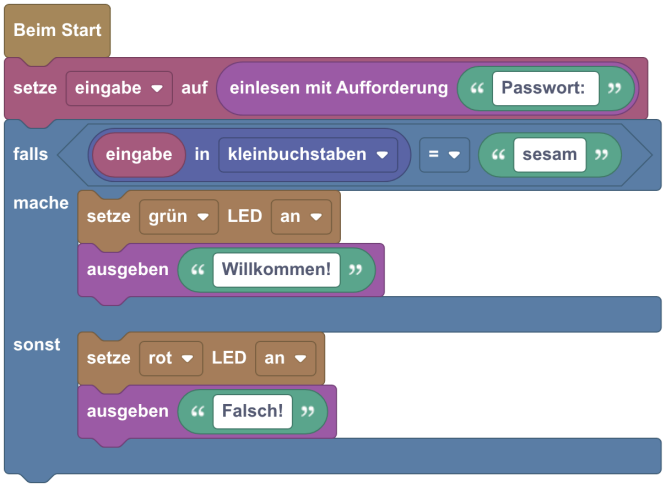
`tipp` startet mit 0, damit die Schleife sicher einmal läuft. Sie läuft, solange `tipp` ≠ `geheim` ist. Danach kommt „Richtig!“. Ein Spiel im Terminal kann so aussehen:

```
Ich denke mir eine Zahl von 1 bis 100.
Dein Tipp: 50
Zu gross!
Dein Tipp: 25
Zu klein!
Dein Tipp: 37
Richtig!
```



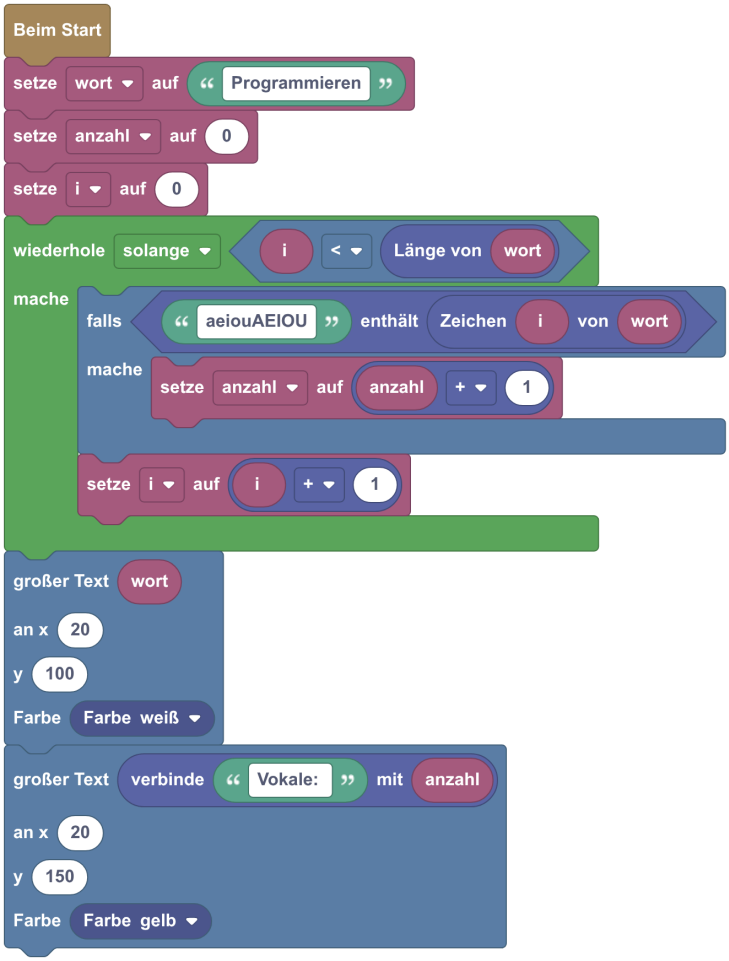
Aufgabe 5 – Geheimes Passwort (`k16_loesung5.pbcp`)

Die Eingabe wird in kleinbuchstaben verwandelt und dann mit „sesam“ verglichen. So passen „Sesam“, „SESAM“ und „sesam“.



Aufgabe 6 – Vokale zählen (`k16_loesung6.pbcp`)

„Programmieren“ hat 5 Vokale: o, a, i, e, e. Der Trick: Nicht das Wort wird durchsucht, sondern der Text „aeiouAEIOU“ – enthält er das aktuelle Zeichen, ist es ein Vokal.


Beim Start
setze wort auf “ Programmieren ”
setze anzahl auf 0
setze i auf 0
wiederhole solange i < Länge von wort
mache
falls “ aeiouAEIOU ” enthält Zeichen i von wort
mache
setze anzahl auf anzahl + 1
setze i auf i + 1
großer Text wort
an x 20
y 100
Farbe Farbe weiß
großer Text verbinde “ Vokale: ” mit anzahl
an x 20
y 150
Farbe Farbe gelb

Lösungen zu Kapitel 17: Mathe-Werkzeuge

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern).

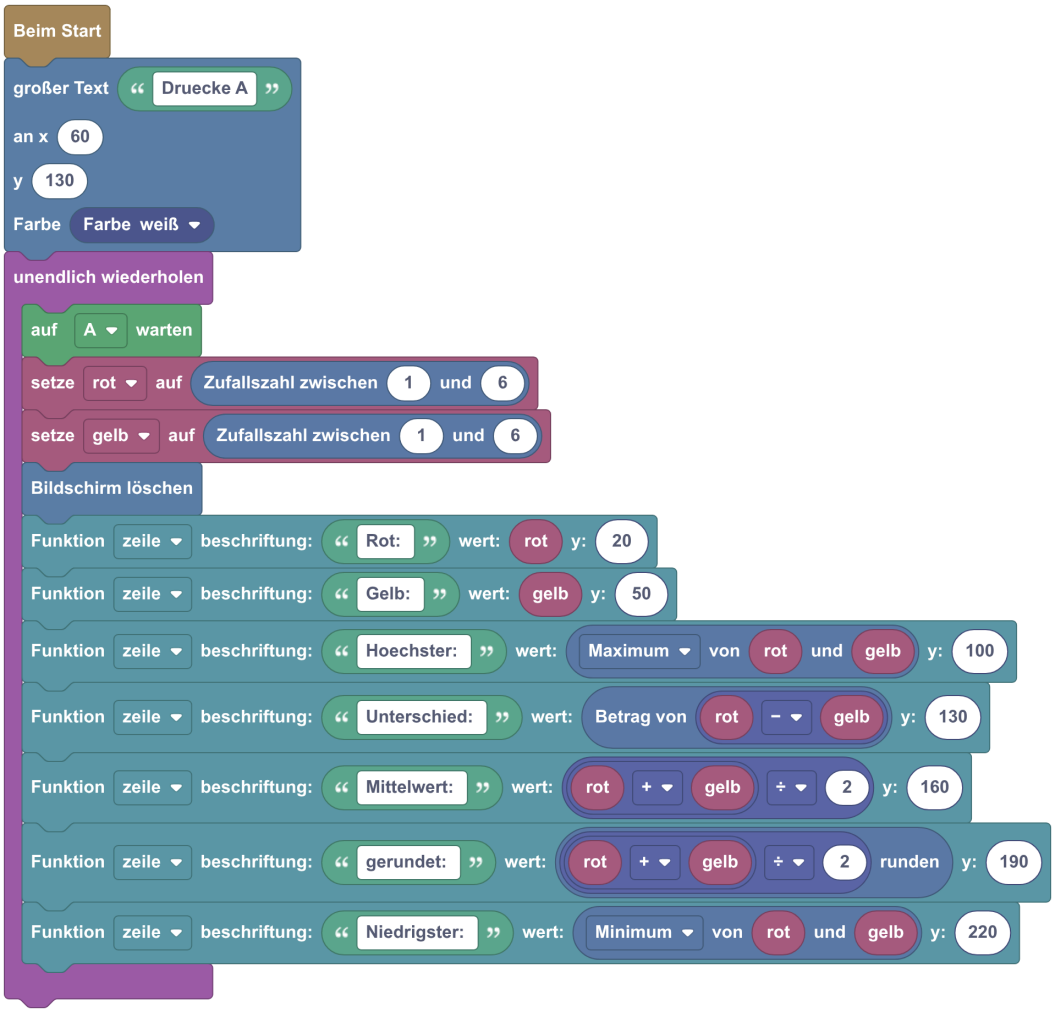
Aufgabe 1 – Runden vorhersagen (`k17_loesung1.pbcp`)

2.4 → 2, 2.6 → 3, 3.5 → 4 und 4.5 → **4**. Die letzte Zahl überrascht: Liegt eine Zahl genau in der Mitte, rundet PBC+ zur geraden Zahl.



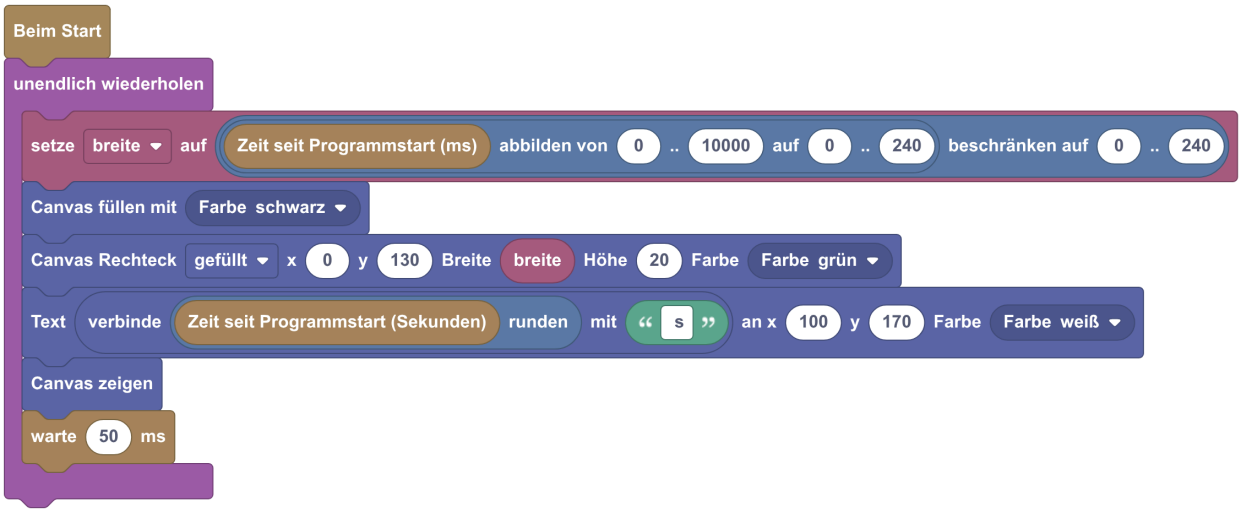
Aufgabe 2 – Der kleinste Wurf (`k17_loesung2.pbcp`)

Ein siebter Aufruf von `zeile` mit „Minimum von rot und gelb“ bei y = 220. Die Funktion `zeile` bleibt unverändert.



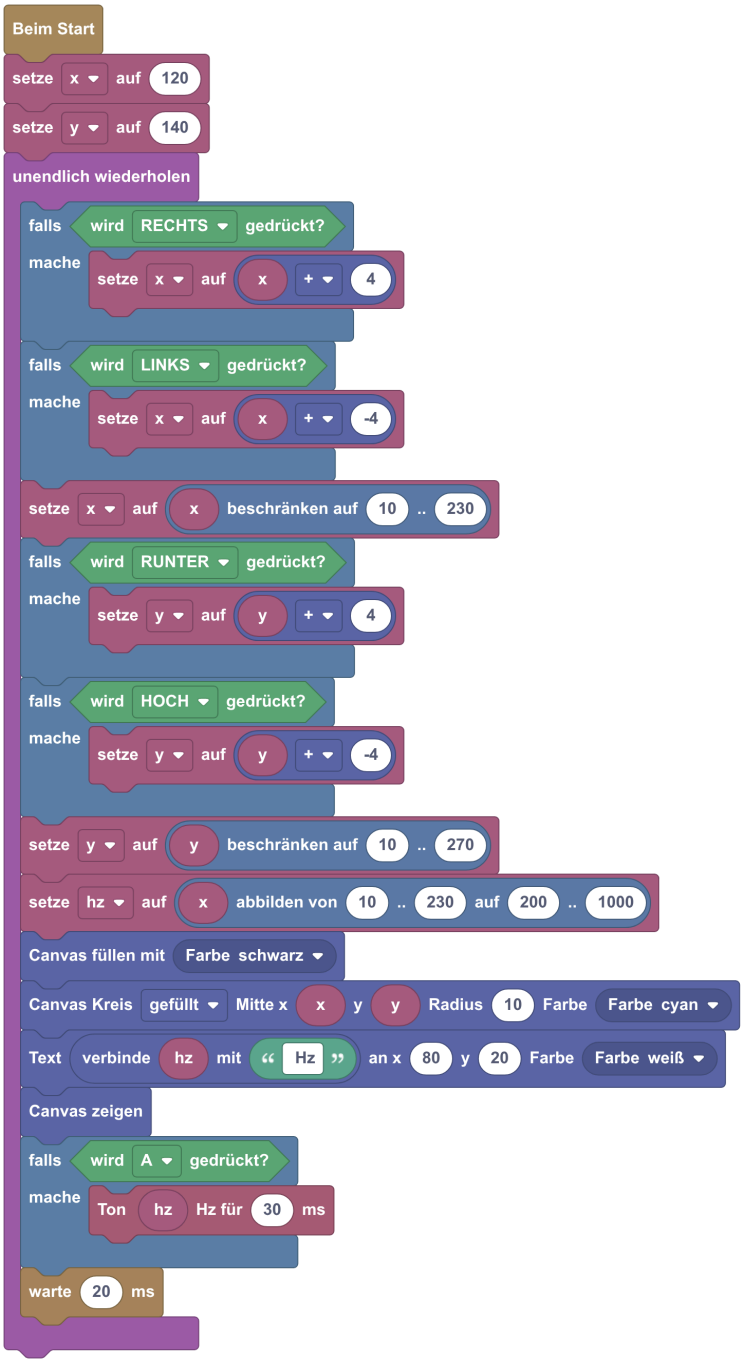
Aufgabe 3 – Zeitbalken (`k17_loesung3.pbcp`)

Nach 10 000 ms wäre der abgebildete Wert größer als 240. „beschränken auf 0 … 240“ hält ihn bei 240 fest – der Balken bleibt stehen.



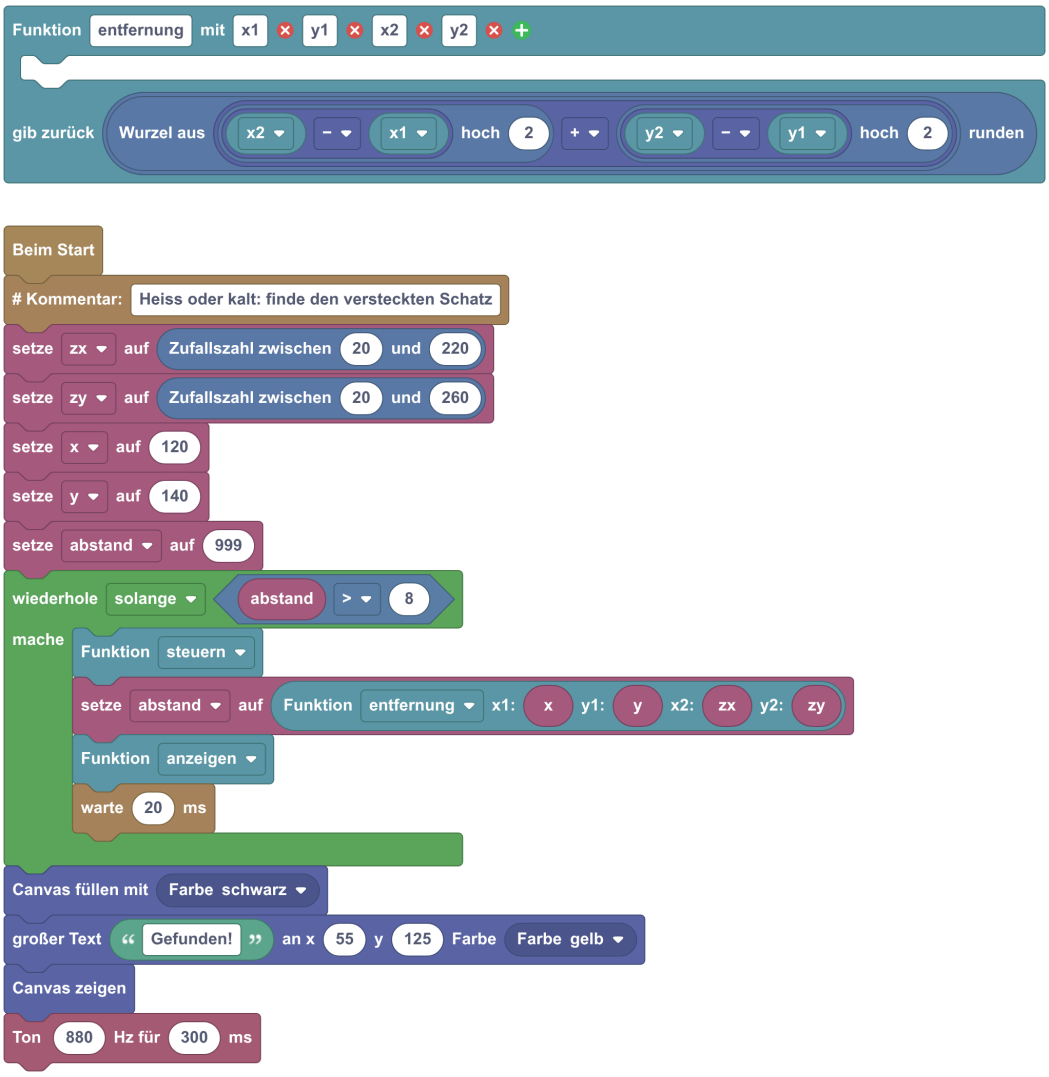
Aufgabe 4 – Hoch und runter (`k17_loesung4.pbcp`)

Dazu kommen eine Variable `y` (Start 140), zwei „falls“-Blöcke für HOCH und RUNTER und ein „setze y auf y beschränken auf 10 … 270“. Der Kreis bekommt `y` statt 140.



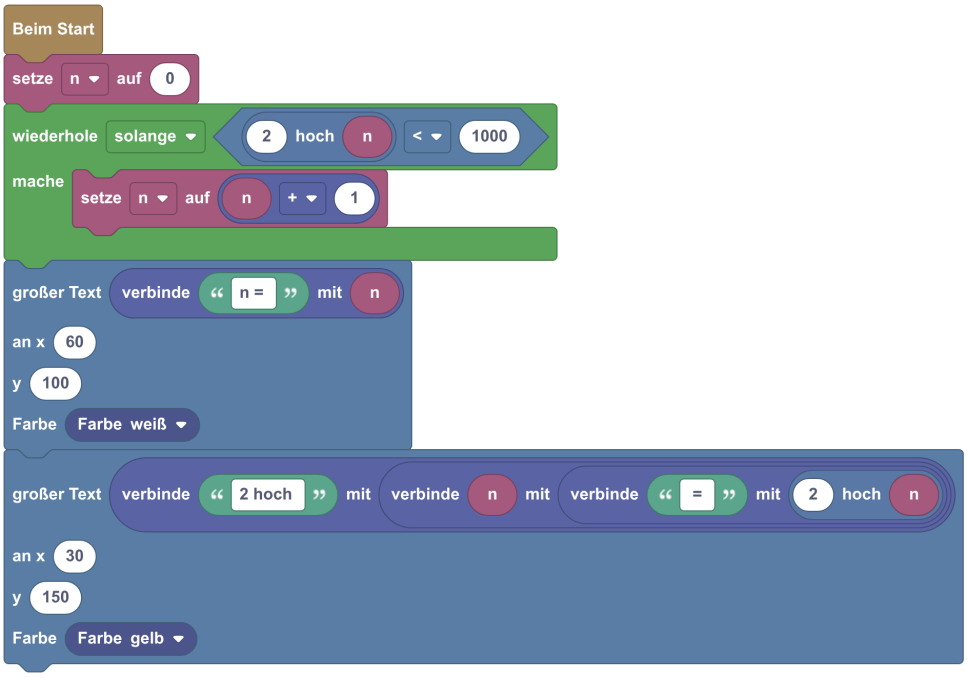
Aufgabe 5 – Abstand als Funktion (`k17_loesung5.pbcp`)

Die lange Rechnung wandert in die Funktion `entfernung`. Im Hauptprogramm steht nur noch „setze abstand auf entfernung(x, y, zx, zy)“. `steuern` und `anzeigen` bleiben wie im Kapitel.



Aufgabe 6 – Verdoppeln (`k17_loesung6.pbcp`)

Die Schleife hört bei `n` = 10 auf, denn 2 hoch 10 = 1024 ist die erste Zweierpotenz über 1000.



Lösungen zu Kapitel 18: Der Beschleunigungssensor

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern). Im Emulator laufen sie ohne Fehler, aber kippen kannst du nur das echte Gerät.

Aufgabe 1 – Forschungsauftrag

Die Werte sind ungefähr so (in m/s², gerundet). Kleine Abweichungen um 1 sind normal.

Haltung	X	Y	Z
flach, Bildschirm oben	0	0	10
auf der rechten Kante	−10	0	0
auf der linken Kante	10	0	0
auf der unteren Kante, Bildschirm zu dir	0	−10	0
flach, Bildschirm unten	0	0	−10

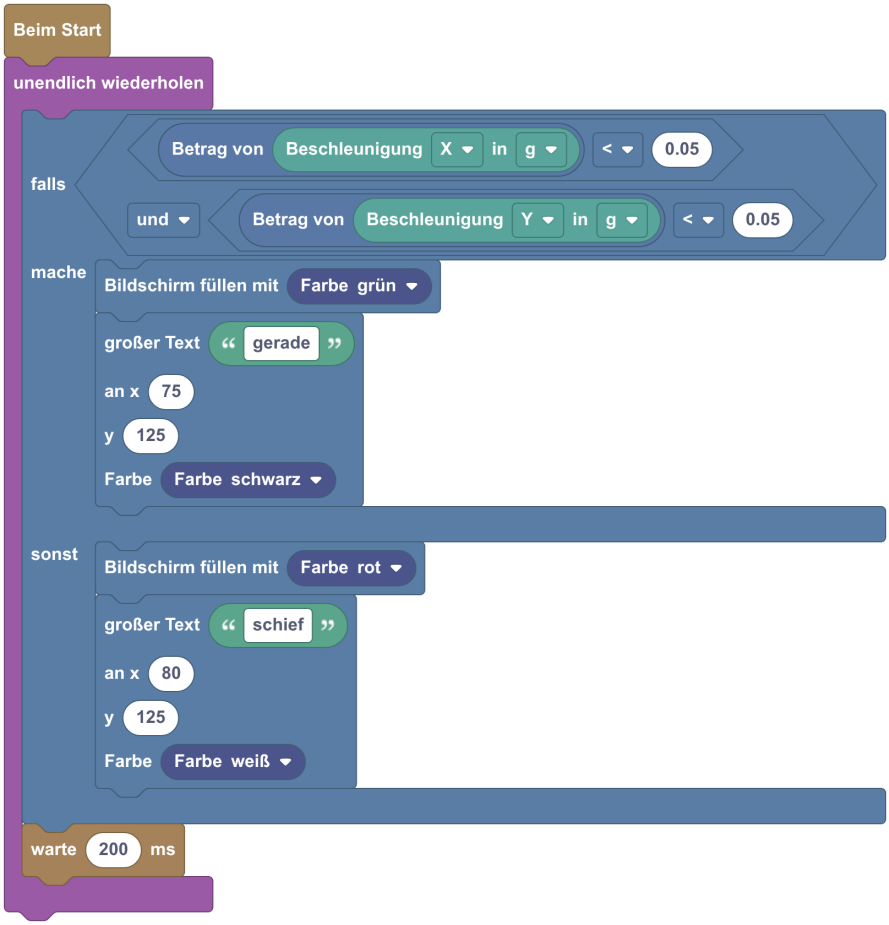
Es fällt auf: Immer ist genau **eine** Zahl etwa 10 oder −10, die anderen sind etwa 0. Die Erdanziehung (etwa 9,81 m/s²) steckt ganz in der Achse, die gerade senkrecht steht. Zeigt die Achse nach oben, ist die Zahl positiv, zeigt sie nach unten, ist sie negativ.

Aufgabe 2 – Wasserwaage (`k18_loesung2.pbcp`)

Der Block „falls … sonst“ entscheidet zwischen grün und rot. Mit „Betrag von“ ist es egal, ob der Wert ein bisschen über oder unter 0 liegt.



Im Emulator liegt der PBC+ immer gerade.

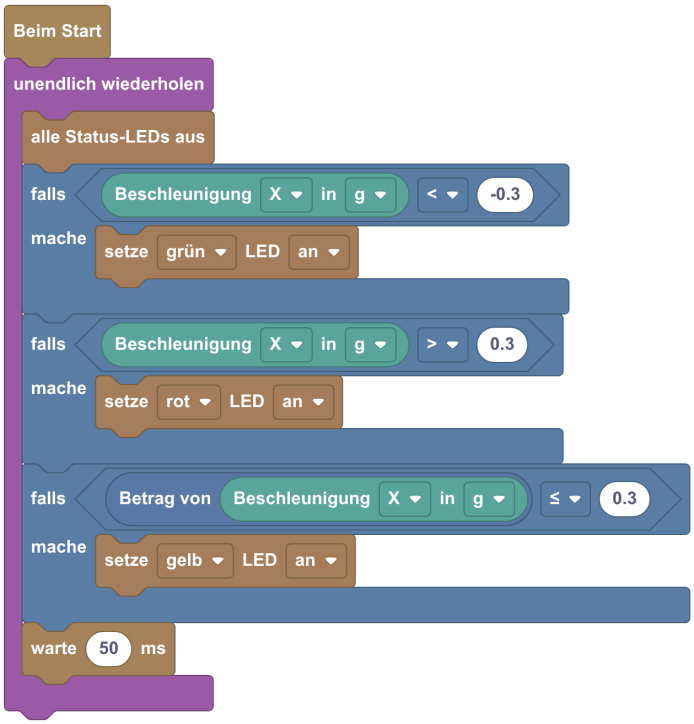


Aufgabe 3 – Schneller, langsamer, falsch herum

- Mit **−2** rollt die Kugel langsamer. Außerdem muss man stärker kippen, bevor sie überhaupt losrollt (erst ab 0,5 g ergibt sich 1 Pixel).
- Mit **−12** rollt sie doppelt so schnell wie mit −6 und reagiert schon auf kleine Neigungen.
- Mit **+6** rollt sie **bergauf**: Kippst du die rechte Seite nach unten, rollt sie nach links. Ohne das Minus bewegt sie sich in dieselbe Richtung wie das Vorzeichen des Messwerts – und das ist die Richtung nach oben.

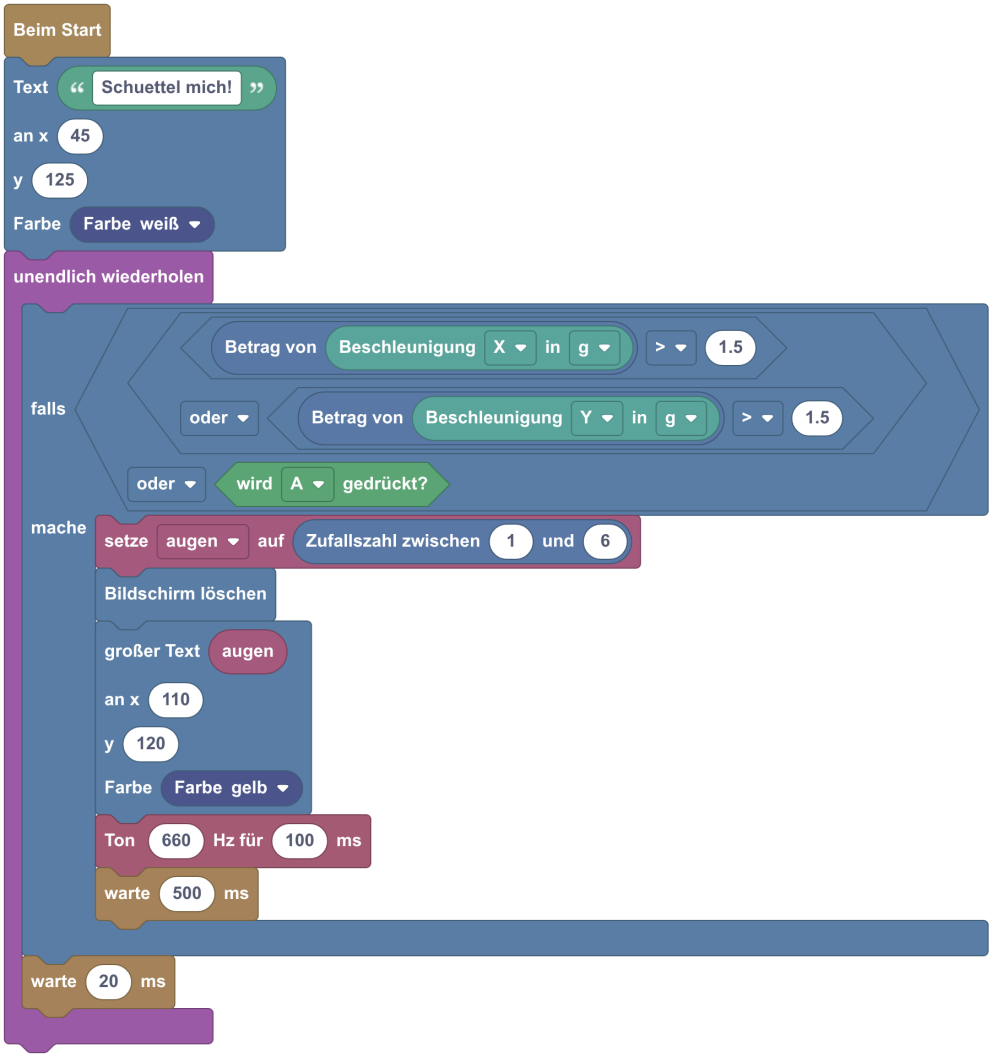
Aufgabe 4 – Kipp-Anzeige (`k18_loesung4.pbcp`)

Zuerst gehen alle LEDs aus, dann wird die passende eingeschaltet. Für „ungefähr flach“ fragt das Programm: Ist der Betrag von X höchstens 0,3?



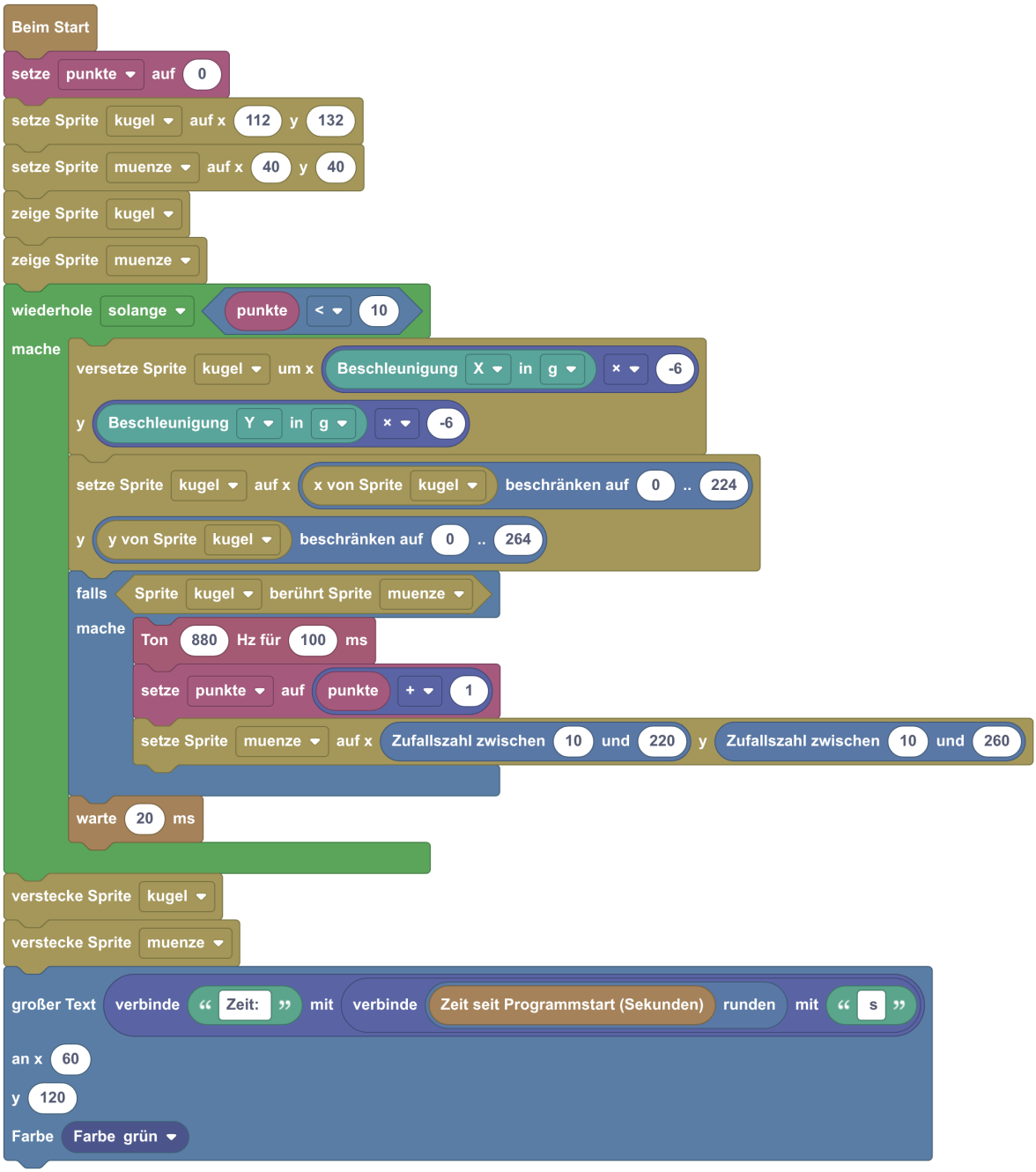
Aufgabe 5 – Schüttel-Würfel (`k18_loesung5.pbcp`)

Die Bedingung besteht aus drei Teilen, verbunden mit „oder“: Betrag von X größer als 1,5, Betrag von Y größer als 1,5 oder Taste A gedrückt. Das „warte 500 ms“ sorgt dafür, dass ein einziges Schütteln nicht gleich mehrmals würfelt.


Beim Start
Text “ Schuettel mich! ”
an x 45
y 125
Farbe Farbe weiß
unendlich wiederholen
falls
Betrag von Beschleunigung X in g > 1.5
oder
Betrag von Beschleunigung Y in g > 1.5
oder
wird A gedrückt?
mache
setze augen auf Zufallszahl zwischen 1 und 6
Bildschirm löschen
großer Text augen
an x 110
y 120
Farbe Farbe gelb
Ton 660 Hz für 100 ms
warte 500 ms
warte 20 ms

Aufgabe 6 – Münzen kippen (`k18_loesung6.pbcp`)

Das Spiel läuft wie mit dem Joystick – nur steuern statt der vier Joystick-Blöcke die beiden Kipp-Blöcke aus 18.2. Du brauchst die Sprites `kugel` und `muenze`.

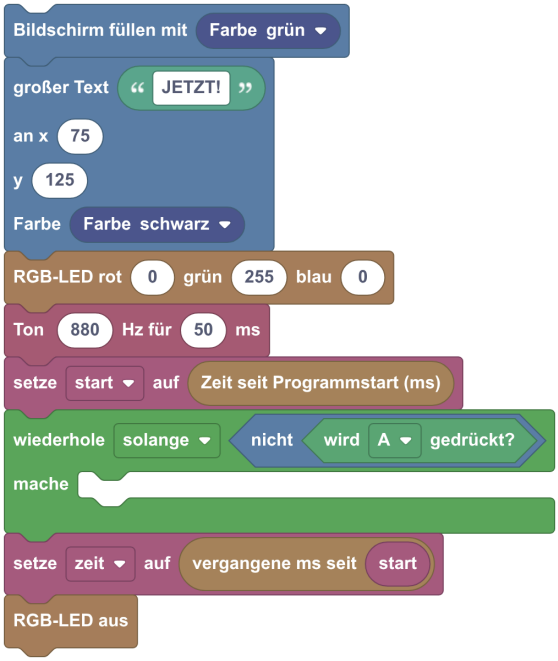


Lösungen zu Kapitel 19: Projekt: Reaktionstest

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern). Abgebildet ist jeweils nur der Teil, der sich gegenüber dem Kapitel ändert.

Aufgabe 1 – Mehr Signal (`k19_loesung1.pbcp`)

Der „sonst"-Teil aus Schritt 3 bekommt zwei neue Blöcke nach dem grünen Bildschirm: „RGB-LED rot 0 grün 255 blau 0" und den Ton. Nach der Messung schaltet „RGB-LED aus" die LED wieder aus.



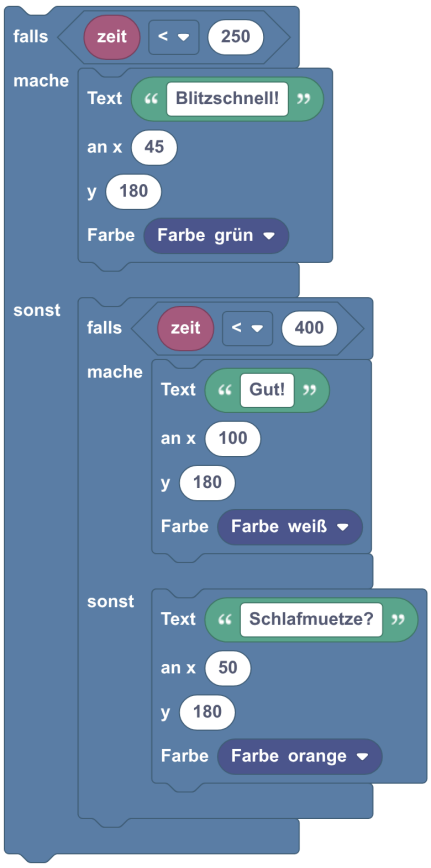
„Ton … für 50 ms" spielt den Ton im Hintergrund: Das Programm wartet nicht, bis er fertig ist, sondern läuft sofort weiter. Deshalb verfälscht der Ton die Messung nicht.

Aufgabe 2 – Schummel-Test

- **Schritt 2:** Es erscheint eine Zeit von fast 0 ms. Die Schleife „solange nicht A gedrückt" fragt erst, wenn das Signal schon da ist. Weil A schon unten ist, ist sie sofort fertig.
- **Schritt 3:** Der Bildschirm wird rot: „Zu frueh!". Hier fragt schon die Warte-Schleife nach Taste A. Wer vor dem Signal drückt (oder die Taste festhält), wird erwischt.

Aufgabe 3 – Wie gut war das? (`k19_loesung3.pbcp`)

Der Block steht in der Funktion `runde` direkt unter „setze bestzeit auf Minimum …". Er schreibt die Bewertung unter die Zeit (y = 180). Das zweite „falls … sonst" steckt im „sonst" des ersten: Es wird nur gefragt, wenn die Zeit **nicht** unter 250 ms war.

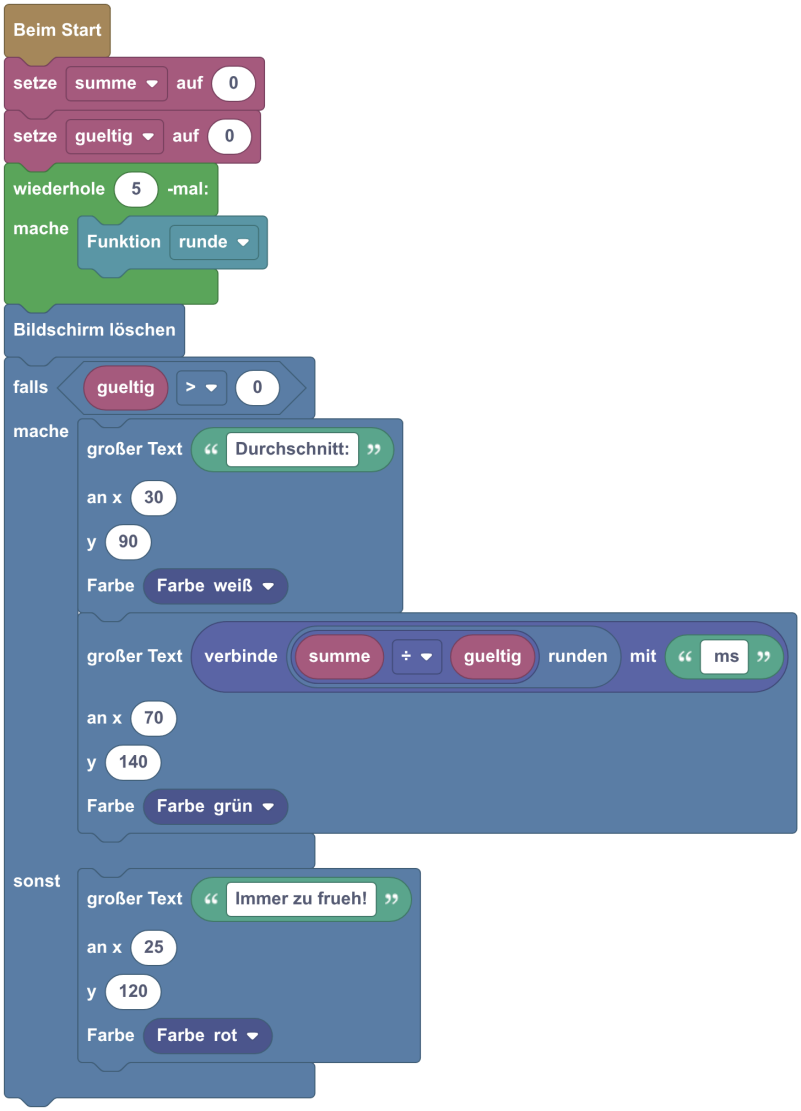


Aufgabe 4 – Durchschnitt (`k19_loesung4.pbcp`)

In der Funktion `runde` ersetzen diese zwei Blöcke das Merken der Bestzeit:



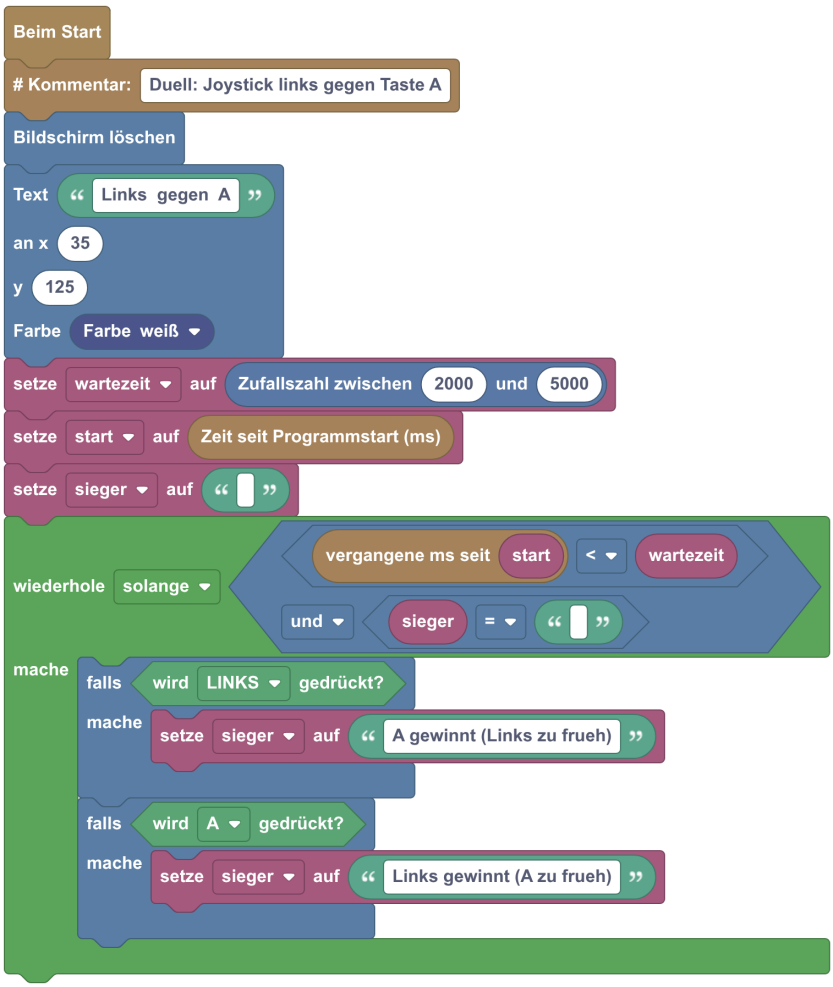
„Beim Start“ setzt `summe` und `gueltig` auf 0 und rechnet am Ende den Durchschnitt aus:



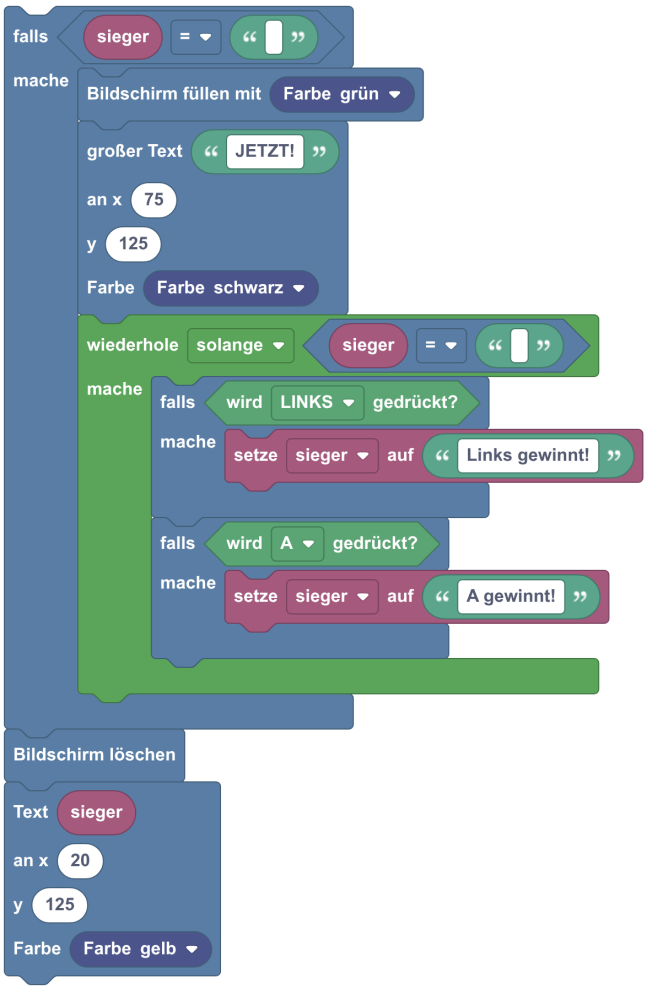
Waren alle Runden zu früh, ist `gueltig` gleich 0. Durch 0 kann man nicht teilen – das Programm würde mit einem Fehler abbrechen. Deshalb prüft das „falls“ vorher, ob `gueltig` größer als 0 ist, und zeigt sonst „Immer zu frueh!“.

Aufgabe 5 – Das Duell (`k19_loesung5.pbcp`)

Die Variable `sieger` ist ein Text (Kategorie **Variablen**, „+ neuer Text…“). Sie ist am Anfang leer. Die erste Schleife wartet zufällig lange. Drückt dabei jemand, steht der Sieger schon fest: der andere.



Ist `sieger` danach noch leer, kommt das Signal. Die zweite Schleife läuft, bis einer drückt – wer zuerst gefragt wird und unten ist, gewinnt. Am Ende steht der Sieger auf dem Bildschirm.

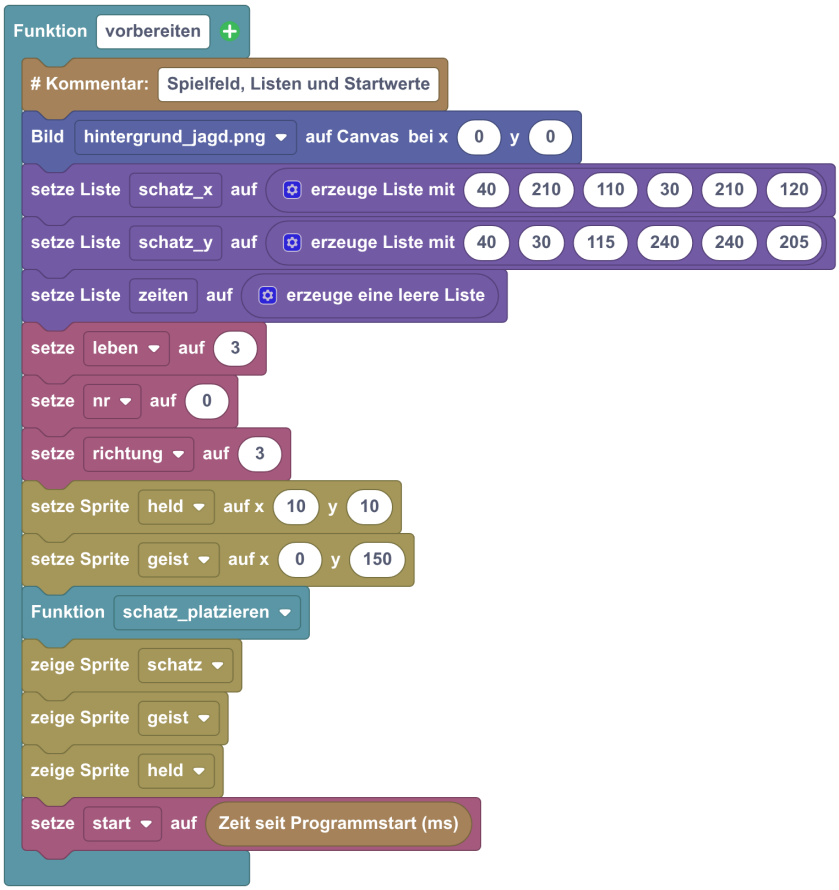

falls
sieger
=
“ ”
mache
Bildschirm füllen mit
Farbe grün
großer Text
“ JETZT! ”
an x
75
y
125
Farbe
Farbe schwarz
wiederhole
solange
sieger
=
“ ”
mache
falls
wird
LINKS
gedrückt?
mache
setze
sieger
auf
“ Links gewinnt! ”
falls
wird
A
gedrückt?
mache
setze
sieger
auf
“ A gewinnt! ”
Bildschirm löschen
Text
sieger
an x
20
y
125
Farbe
Farbe gelb

Lösungen zu Kapitel 20: Projekt: Schatzjagd

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Spiel tut, was die Aufgabe verlangt. Alle Lösungen sind das fertige Spiel `k20_schatzjagd.pbcp` mit einer Änderung. Sie liegen als Datei im Ordner `buch/programme` (Name in Klammern). Abgebildet ist nur, was sich ändert.

Aufgabe 1 – Ein sechster Schatz (`k20_loesung1.pbcp`)

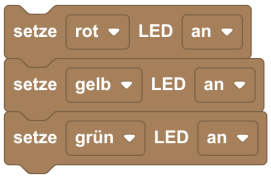
In `vorbereiten` bekommen **beide** Listen einen sechsten Platz (Zahnrad am Block „erzeuge Liste mit“): 120 in `schatz_x` und 205 in `schatz_y`.



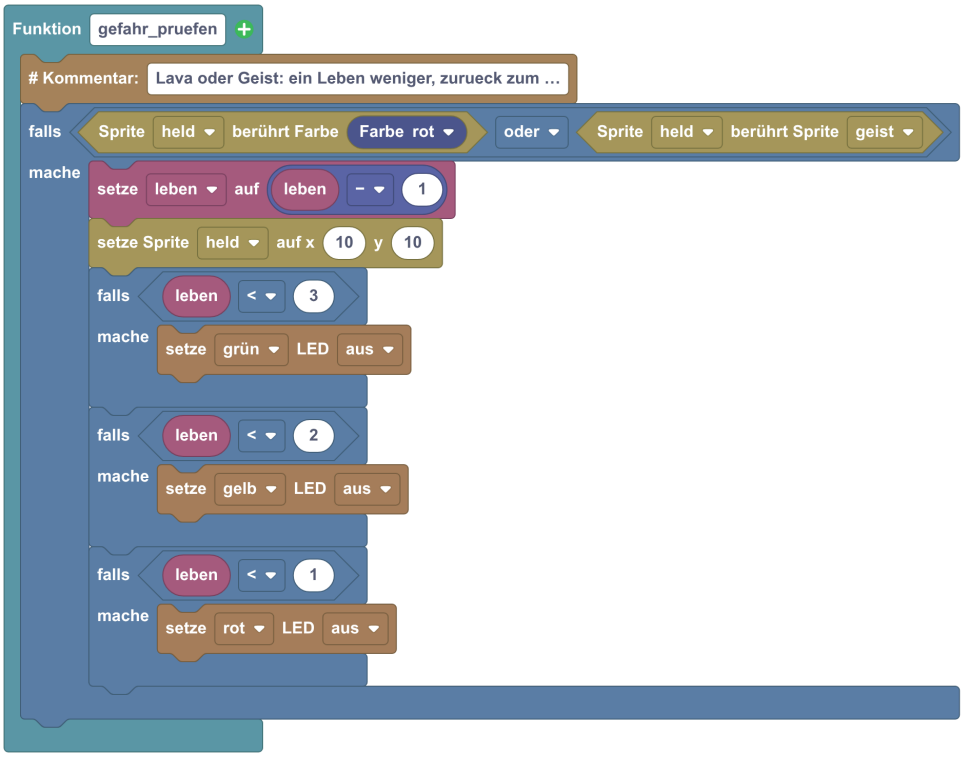
Sonst musst du nichts ändern. Die Schleife fragt nach der **Länge** der Liste und nicht nach der festen Zahl 5. Darum läuft sie automatisch weiter, bis auch der sechste Schatz gefunden ist. Vergisst du den Wert in einer der beiden Listen, sind sie verschieden lang – dann bricht das Spiel beim sechsten Schatz mit einem Fehler ab.

Aufgabe 2 – Leben-Anzeige (`k20_loesung2.pbcp`)

Am Ende von `vorbereiten` schalten diese drei Blöcke alle LEDs ein:



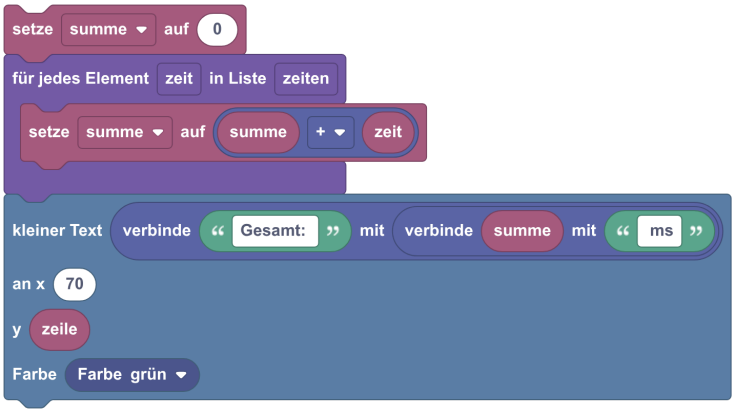
In `gefahr_pruefen` schaltet das Spiel nach dem Abziehen eines Lebens die LEDs aus, die zu viel sind:



Aufgabe 3 – Gesamtzeit (`k20_loesung3.pbcp`)

Diese Blöcke kommen in `ende_anzeigen` unter die Schleife, die die Zeiten zeigt – also noch in den „mache“-Teil von „falls leben > 0“. Die erste Schleife hat `zeile` schon weitergezählt, deshalb steht die Gesamtzeit genau unter der letzten Zeit.





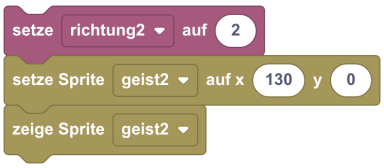
Aufgabe 4 – Der Geist wird schneller (`k20_loesung4.pbcp`)

Schwebt der Geist gerade nach rechts, ist `richtung` positiv: Dann wird 1 dazugezählt. Schwebt er nach links, ist `richtung` negativ: Dann wird 1 abgezogen. In beiden Fällen wird er schneller. Würdest du immer 1 dazuzählen, würde ein Geist, der nach links fliegt, langsamer.

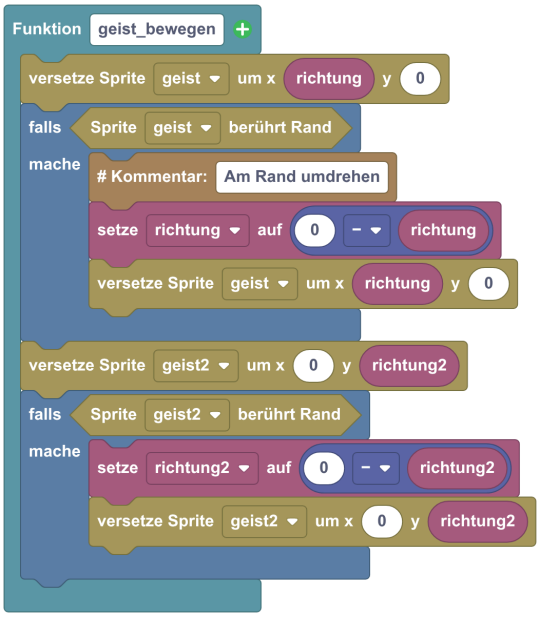


Aufgabe 5 – Ein zweiter Geist (`k20_loesung5.pbcp`)

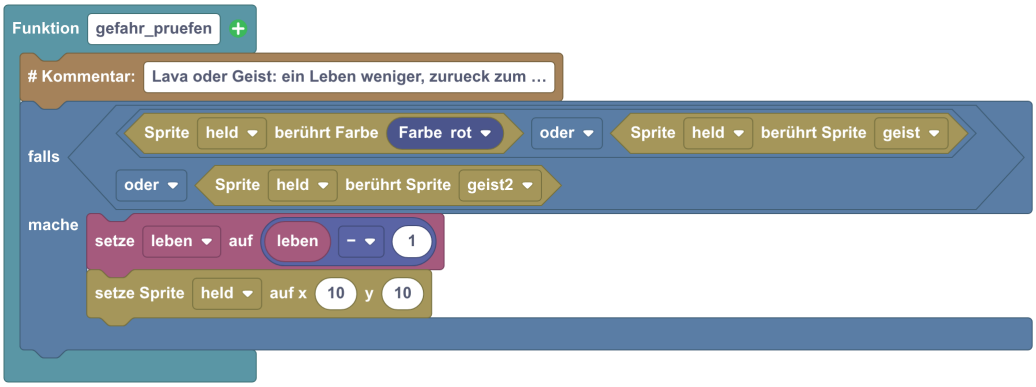
Lege das Sprite `geist2` mit dem Bild `geist.png` an. Der zweite Geist braucht eine eigene Richtung `richtung2`. In `vorbereiten` kommt dazu:



In `geist_bewegen` bewegt sich `geist2` genauso wie der erste Geist – nur in y-Richtung:



`gefahr_pruefen` bekommt ein zweites „oder“:

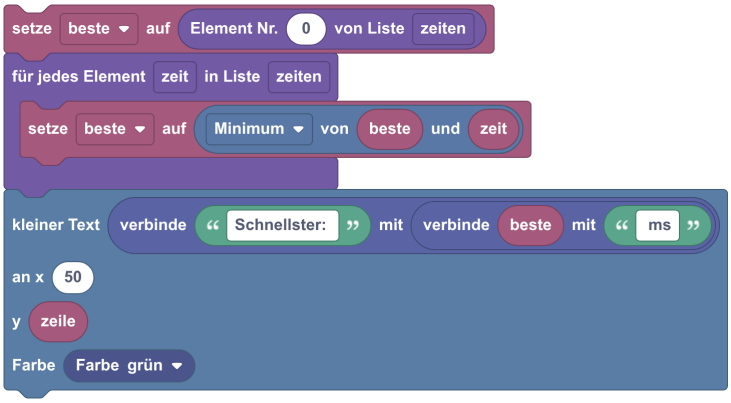


Und in `ende_anzeigen` muss auch `geist2` versteckt werden.

Aufgabe 6 – Der schnellste Schatz (`k20_loesung6.pbcp`)

`beste` startet mit der ersten Zeit (Element Nr. 0). Die Schleife vergleicht dann jede Zeit mit `beste` und behält die kleinere. Die Blöcke stehen an derselben Stelle wie in Aufgabe 3. Weil „Schnellster Schatz“ zu lang für eine Zeile wäre, steht nur „Schnellster:“ da.





ANHANG C – LÖSUNGEN

Lösungen zu Kapitel 21: Ein eigenes Spiel entwickeln

Oft gibt es mehrere richtige Lösungen. Wichtig ist, dass das Programm tut, was die Aufgabe verlangt. Alle Lösungen liegen als Datei im Ordner `buch/programme` (Name in Klammern). Lade vorher die Bilder `raumschiff.png`, `meteor.png` und – für Aufgabe 5 – `stern.png` hoch.

Aufgabe 1 – Finde den Fehler

Im „falls“-Block steht „y von Sprite meteor < 280“ statt „> 280“. Der Meteor startet oben bei y = 0. 0 ist kleiner als 280 – die Bedingung ist also sofort wahr: Es gibt einen Punkt, und der Meteor springt an eine neue Stelle, wieder ganz oben. Das passiert in jeder Runde der Schleife. Der Meteor zappelt oben hin und her, die Punkte steigen rasend schnell, und das Schiff wird nie getroffen. Reparatur: das Zeichen im Vergleich auf „>“ stellen.

Aufgabe 2 – Plane „Fang den Apfel“

Ein möglicher Planungsbogen:

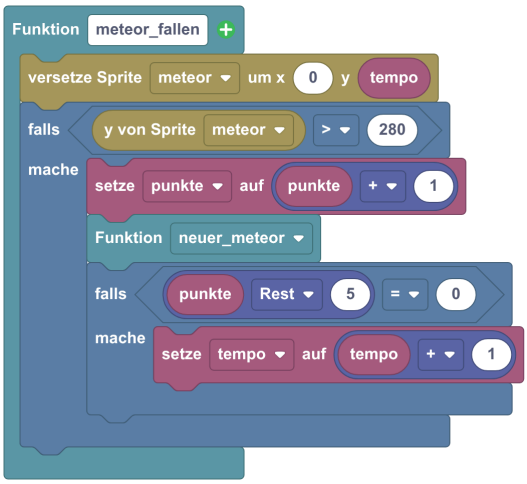
Frage	Fang den Apfel
Worum geht es?	Ein Korb unten fängt Äpfel, die von oben fallen.
Figuren (Sprites)	`korb`, `apfel`
Steuerung	Joystick links und rechts bewegen den Korb.
Wie endet das Spiel?	Gewonnen bei 10 gefangenen Äpfeln, verloren bei 3 verpassten.
Variablen	`gefangen` (Start: 0), `verpasst` (Start: 0), `tempo`

Wenn …	dann …
der Joystick links gedrückt ist (und der Korb nicht am Rand steht),	fährt der Korb nach links.
der Joystick rechts gedrückt ist (und der Korb nicht am Rand steht),	fährt der Korb nach rechts.
der Korb den Apfel berührt,	ist `gefangen` um 1 größer, ein neuer Apfel startet oben.
der Apfel unten herausfällt,	ist `verpasst` um 1 größer, ein neuer Apfel startet oben.
`gefangen` 10 ist,	erscheint „Gewonnen!“.
`verpasst` 3 ist,	erscheint „Verloren“.

Funktionen: `vorbereiten`, `steuern`, `neuer_apfel`, `apfel_fallen`, `fangen_pruefen`, `spiel_ende`. Die Schleife läuft „solange gefangen < 10 und verpasst < 3“. In `spiel_ende` entscheidet ein „falls / sonst“, ob „Gewonnen!“ oder „Verloren“ erscheint.

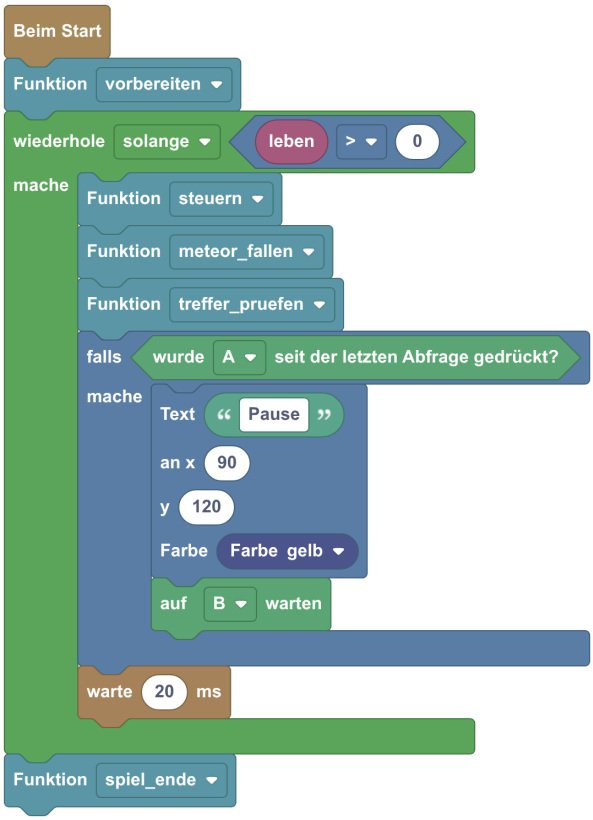
Aufgabe 3 – Immer schneller (`k21_loesung3.pbcp`)

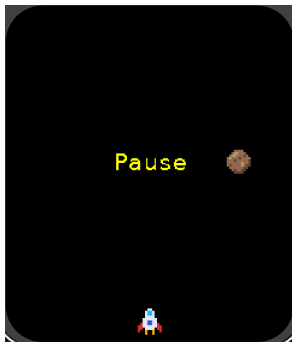
Nur `meteor_fallen` ändert sich. Nach dem Punkt prüft ein zusätzlicher „falls“-Block, ob `punkte` Rest 5 gleich 0 ist – also ob `punkte` durch 5 teilbar ist. Dann wird `tempo` um 1 größer.



Aufgabe 4 – Pause! (`k21_loesung4.pbcp`)

In die Schleife des Hauptprogramms kommt ein „falls“-Block. „wurde A seit der letzten Abfrage gedrückt?“ merkt sich auch einen kurzen Druck. „auf B warten“ hält das Programm an, bis B gedrückt und losgelassen wird. Der Text „Pause“ verschwindet von selbst: Sobald sich der Meteor wieder bewegt, wird der Bildschirm neu gemalt.

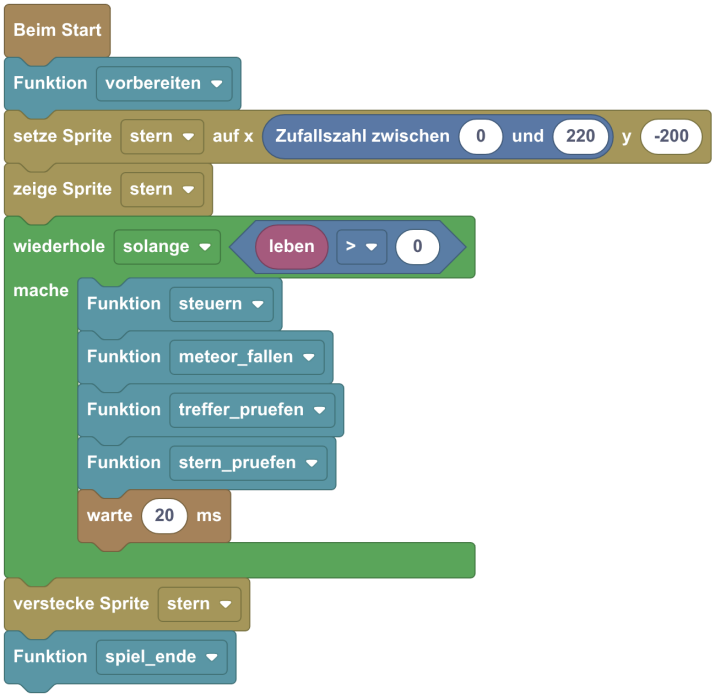




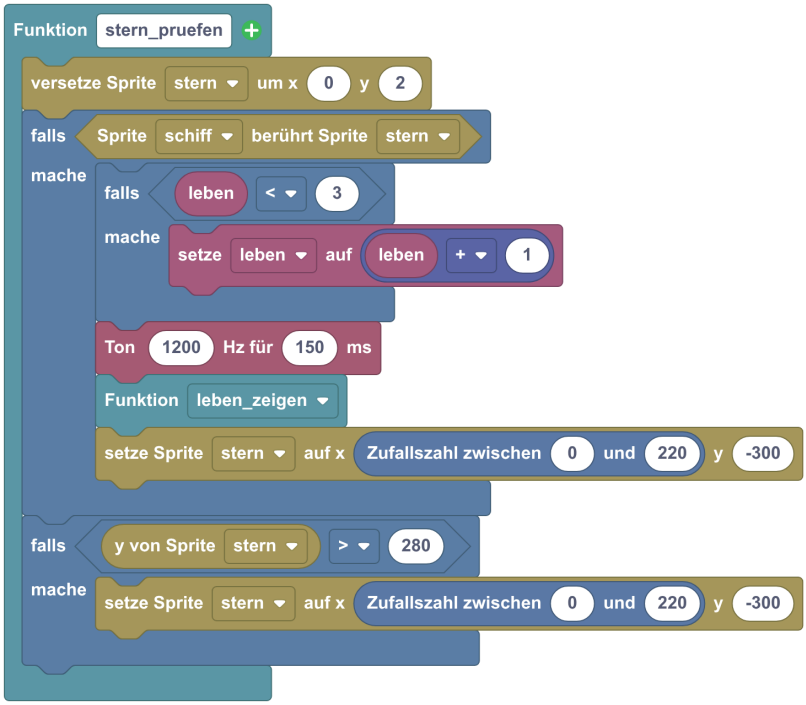
Warum nicht zweimal Taste A? Wenn das Programm den Druck bemerkt, ist A oft noch unten. „auf A warten“ wäre dann sofort erfüllt, und die Pause wäre vorbei, sobald du loslässt.

Aufgabe 5 – Der Bonus-Stern (`k21_loesung5.pbcp`)

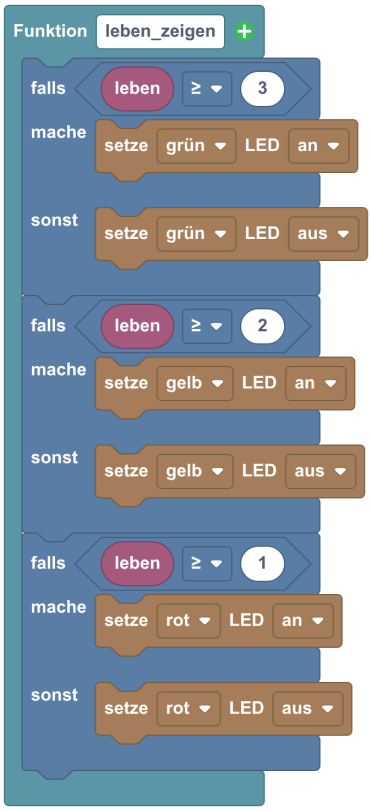
Der Stern wird ein drittes Sprite `stern`. Er startet weit über dem Bildschirm (y = −200) und fällt nur 2 Pixel pro Runde – so dauert es eine Weile, bis er auftaucht. Das Hauptprogramm ruft zusätzlich `stern_pruefen` auf und versteckt am Ende auch den Stern:



`stern_pruefen` bewegt den Stern. Fängt das Schiff ihn, gibt es ein Leben zurück – aber nur, wenn `leben` kleiner als 3 ist. Danach (und wenn er unten herausfällt) startet der Stern wieder weit oben.



`leben_zeigen` muss die LEDs jetzt auch wieder **einschalten** können. Deshalb hat jede LED ein „falls / sonst“:



Aufgabe 6 – Dein eigenes Spiel

Hier gibt es keine Musterlösung. Prüfe dein Spiel mit diesen Fragen:

Ist der Planungsbogen ganz ausgefüllt, mit mindestens vier „Wenn …, dann …“-Regeln? Gibt es für jede Regel einen „falls“-Block in der Schleife? Liest sich das Hauptprogramm wie ein Inhaltsverzeichnis? Hast du nach jeder Stufe getestet und gespeichert? Kann jemand anderes dein Spiel spielen, ohne dass du etwas erklären musst?

ANHANG D

Glossar

Hier stehen die wichtigsten Fachwörter aus diesem Buch – von A bis Z. Die Zahl in Klammern sagt dir, in welchem Kapitel das Wort genauer erklärt wird.

A

Anweisung
Ein Befehl, den das Programm ausführt, z. B. „Bildschirm löschen“. Anweisungs-Blöcke haben oben eine Nase und unten eine Kerbe und werden untereinander gehängt. (2)

Arbeitsfläche
Der große Bereich in PBC+ Studio, auf dem du dein Programm aus Blöcken baust. (1)

Aufruf
Der Block „Funktion …“, der eine Funktion startet. Danach geht das Programm nach dem Aufruf weiter. (14)

Ausgabe
Alles, was das Programm nach außen gibt: Bilder auf dem Bildschirm, Töne, Licht der LEDs oder Text im Terminal. (9, 16)

B

Bedingung
Eine Frage, deren Antwort wahr oder falsch ist, z. B. „punkte > 5“ oder „wird A gedrückt?“. Bedingungen stecken in „falls“ und „solange“. (7)

Beim Start
Der Block, mit dem jedes Programm beginnt. Alles, was darunter hängt, läuft von oben nach unten ab. (2)

Beschleunigungssensor
Ein Bauteil im PicoBoy, das spürt, wie das Gerät gekippt oder bewegt wird. (18)

Bildschirm
Der Farbbildschirm des PicoBoy: 240 Pixel breit und 280 Pixel hoch. (1, 2)

Block
Ein Baustein eines Programms. Jeder Block steht für einen Befehl, einen Wert oder eine Frage. (1)

Blockleiste
Die Leiste links in PBC+ Studio. Dort liegen alle Blöcke, sortiert in Kategorien wie Display, Sprites oder Sound. (1)

Bug
Englisch für „Käfer“: ein Fehler im Programm. Fehler suchen und beheben nennt man auch *debuggen*. (21)

C

Canvas
Englisch für „Leinwand“: ein Bild im Speicher, auf das du malst. Man sieht es erst nach „Canvas zeigen“. Sprites laufen vor dem Canvas, und „berührt Farbe“ prüft die Farben des Canvas. (11)

D

Dateien
Das Menü in PBC+ Studio zum Speichern, Öffnen und Hochladen von Bildern. Programme werden als `.pbcp`-Datei gespeichert. (1)

Debuggen
Fehler in einem Programm suchen und beheben. (21, Anhang B)

E

Eingabe
Alles, was von außen ins Programm kommt: Tasten, Joystick, Beschleunigungssensor oder Text, den du im Terminal eintippst. (8, 16)

Element
Ein einzelner Wert in einer Liste. (15)

Emulator
Ein nachgebauter PicoBoy im Browser. Er tut so, als wäre er das echte Gerät. Die Tasten bedienst du mit Pfeiltasten, Leertaste, A und B. (1)

Endlosschleife
Eine Schleife, die nie von selbst aufhört, z. B. „unendlich wiederholen“. Sie braucht fast immer ein „warte“. (5)

F

falls
Der Block für Entscheidungen: Die Blöcke darin laufen nur, wenn die Bedingung wahr ist. Mit „sonst“ gibt es auch einen Weg für „falsch“. (7)

Farbe
Farben wählst du nach Namen, mit dem Farbwähler oder gemischt aus Rot, Grün und Blau (je 0 bis 255). (3)

Fehlermeldung
Eine Nachricht von PBC+ Studio, wenn etwas nicht stimmt – als Fenster oder als roter Text im Terminal. (Anhang B)

Frage-Block
Ein Block mit spitzen Enden. Er antwortet mit wahr oder falsch, z. B. „Sprite … berührt Rand“. (7, 13)

Funktion
Ein Stück Programm mit eigenem Namen. Du legst es einmal an und kannst es überall aufrufen. So wird ein großes Programm übersichtlich. (14, 21)

G

Ganze Zahl
Eine Zahl ohne Komma, z. B. 3 oder −12. Zufallszahlen und Pixel-Positionen sind ganze Zahlen. (6)

Gerät
Der echte PicoBoy Color Plus – im Gegensatz zum Emulator. In PBC+ Studio heißt dieses Ziel „PBC+“. (1)

H

Hertz (Hz)
Die Einheit für die Tonhöhe. 440 Hz ist der Kammerton a; je größer die Zahl, desto höher der Ton. (9)

Hintergrund
Das, was hinter den Sprites zu sehen ist. Am besten malst du ihn auf den Canvas oder legst ein Bild darauf. (11, 12)

I

Index
Die Nummer eines Platzes in einer Liste oder eines Zeichens in einem Text. Gezählt wird ab 0. (15, 16)

J

Joystick
Der kleine Steuerknüppel am PicoBoy: hoch, runter, links, rechts. Drückst du ihn hinein, ist das die Taste MITTE. (1, 8)

K

Kategorie
Eine Gruppe von Blöcken in der Blockleiste, z. B. Sound oder Listen. Ein Klick klappt sie auf. (1)

Kollision
Wenn sich zwei Dinge im Spiel berühren, z. B. Held und Münze. Dafür gibt es die Blöcke „berührt Sprite“, „berührt Rand“ und „berührt Farbe“. (13)

Kommentar
Eine Notiz im Programm für Menschen. Der PicoBoy überspringt sie. (2)

Konsole
Die Blöcke, die Text ins Terminal schreiben („ausgeben“) oder dort lesen („einlesen“). (16)

Koordinaten

Zwei Zahlen x und y, die eine Stelle auf dem Bildschirm angeben. (0, 0) ist oben links, x zählt nach rechts, y nach unten. Bei der Turtle ist (0, 0) die Mitte. (2, 10)

L

LED

Ein kleines Lämpchen. Der PicoBoy hat drei Status-LEDs (rot, gelb, grün) und eine RGB-LED, die jede Farbe kann. (9)

Liste

Eine Variable, in der viele Werte nacheinander stehen, z. B. alle Punktestände. Die Plätze werden ab 0 gezählt. (15)

Lokale Variable

Eine Variable, die nur innerhalb einer Funktion gilt. (14)

M

Millisekunde (ms)

Der tausendste Teil einer Sekunde. 1000 ms sind 1 Sekunde, „warte 20 ms“ ist also sehr kurz. (4)

N

Name

Variablen, Listen, Sprites und Funktionen brauchen einen Namen. Nimm nur Buchstaben, Ziffern und `_` – keine Umlaute, kein ß, keine Leerzeichen: `muenze`, `spiel_ende`. (6)

P

Parameter

Ein Wert, den du einer Funktion beim Aufruf mitgibst, z. B. wie hoch eine Figur springen soll. (14)

PBC+ Studio

Das Programm im Browser, mit dem du den PicoBoy Color Plus programmierst. (1)

Pixel

Ein einzelner Bildpunkt auf dem Bildschirm. Der PicoBoy hat 240 × 280 Pixel. (2, 3)

Planungsbogen

Ein Blatt, auf dem du ein Spiel planst, bevor du es baust: Figuren, Steuerung, Regeln, Ende, Variablen. (21)

Programm

Eine Folge von Anweisungen, die ein Computer der Reihe nach ausführt. (1, 2)

R

Rand

Die Grenze des Bildschirms. „Sprite … berührt Rand“ ist wahr, wenn ein Stück des Sprites darüber hinausragt. (13)

Rest

Was beim Teilen übrig bleibt: 7 geteilt durch 2 ist 3, Rest 1. Mit dem Rest prüfst du z. B., ob eine Zahl gerade ist. (6)

RGB

Rot, Grün, Blau. Aus diesen drei Lichtfarben lässt sich jede Farbe mischen. (3, 9)

Rückgabewert

Das Ergebnis, das eine Funktion mit „gib zurück“ liefert. Ihr Aufruf ist dann ein Wert-Block. (14)

S

Schleife

Ein Block, der andere Blöcke wiederholt: N-mal, „für i von … bis…“, „solange“ oder unendlich. (5, 7)

Sensor

Ein Fühler, der etwas misst. Der PicoBoy hat einen Beschleunigungssensor. (18)

solange

Eine Schleife, die wiederholt, solange eine Bedingung wahr ist. (7)

Spielschleife
Die Schleife eines Spiels: Tasten abfragen, Figuren bewegen, Regeln prüfen, kurz warten – und wieder von vorn. (13, 21)

Sprite
Ein Bild, das sich über den Bildschirm bewegen kann, z. B. eine Spielfigur. Seine Position x, y ist die Ecke oben links. (12)

Stop
Der Knopf ■ Stop hält ein laufendes Programm an. (1)

Stufe
Ein Zwischenstand beim Bauen eines Spiels. Nach jeder Stufe wird getestet und gespeichert. (21)

T

Terminal
Das Textfeld unter dem Emulator. Hier erscheinen Meldungen, Fehler und alles, was „ausgeben“ schreibt. (1, 16)

Testen
Ausprobieren, ob ein Programm tut, was es soll. Mit ▶ Testen startest du es. Gutes Testen heißt: vorher überlegen, was passieren soll. (1, 21)

Testprotokoll
Eine Tabelle: Was teste ich? Was soll passieren? Was passiert wirklich? (21)

Text
Eine Folge von Zeichen, z. B. „Hallo“. Mit „verbinde“ hängst du Texte und Zahlen aneinander. Auf dem Bildschirm erscheinen ä, ö, ü und ß als „?“. (2, 16)

Turtle
Englisch für „Schildkröte“: eine Figur mit Stift, die beim Laufen Linien zeichnet. (10)

U

und / oder / nicht
Blöcke, die Bedingungen verknüpfen: „und“ – beide müssen wahr sein, „oder“ – eine reicht, „nicht“ dreht die Antwort um. (7)

V

Variable
Ein Speicherplatz mit Namen, z. B. `punkte`. Sie merkt sich einen Wert, den du lesen und ändern kannst. (6)

Vergleich
Ein Block, der zwei Werte vergleicht: =, ≠, <, ≤, >, ≥. Das Ergebnis ist wahr oder falsch. (7)

W

wahr / falsch
Die zwei möglichen Antworten auf eine Frage oder Bedingung. (7)

Wert
Eine Zahl, ein Text oder eine Farbe. Wert-Blöcke haben links einen Stecker und passen in die Lücken anderer Blöcke. (2, 6)

Z

Zählschleife
Die Schleife „zähle i von … bis …“. Die Variable `i` zählt dabei mit. (5)

Ziel
Die Auswahl oben rechts in PBC+ Studio: Läuft das Programm im Emulator oder auf dem Gerät (PBC+)? (1)

Zufallszahl
Eine Zahl, die das Programm zufällig wählt – wie beim Würfeln. Beide Grenzen können vorkommen. (6)