

Enten-Alarm

Dein eigenes Spiel an einem Nachmittag

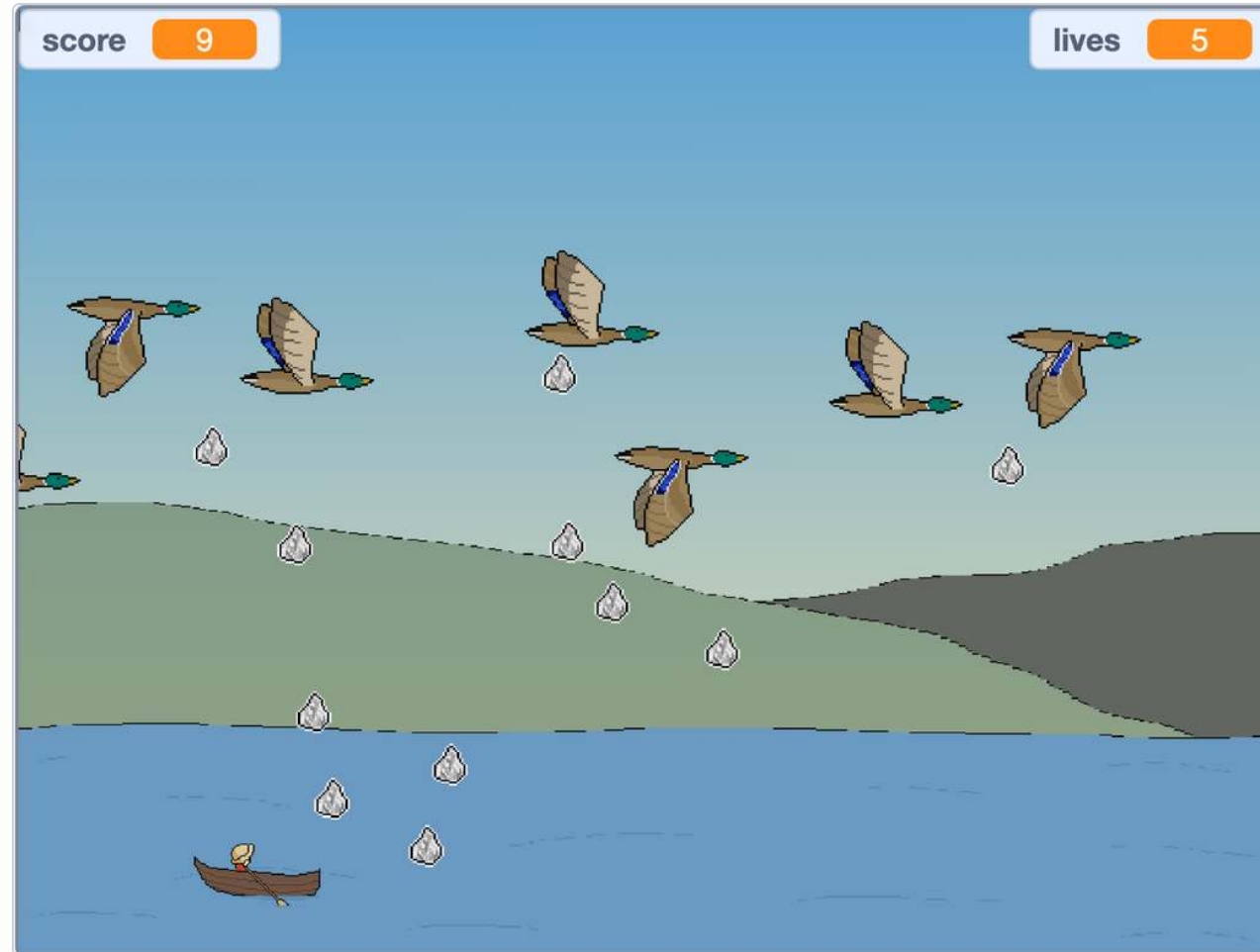
Boot ausweichen. Enten kacken. Fünf Leben.

Wer bin ich?

Und wer bist du?

- Name
- Alter
- Vorerfahrung
- ...

Das bauen wir heute



Das bauen wir heute

- Ein **Boot**, das deinem Finger folgt
- Eine **Ente**, die oben hin und her fliegt
- **Kacke**, die herunterfällt
- **Fünf Leben**, ein Treffer kostet eins
- **Punkte**, die hochzählen
- Mit der Zeit wird es **immer schwerer**

Und am Ende: deine eigene Idee dazu.

Der Tag

	Was	Dauer
1	Anmelden & Scratch kennenlernen	20 min
2-3	Boot steuern & Animation	35 min
	Pause	10 min
4-6	Ente, Kacke, Treffer & Leben	80 min
	Pause	10 min
7-8	Schwieriger machen, Feinschliff	35 min
9	Deine eigene Idee	35 min
10	Vorstellen	15 min

4 Dinge Vorab

1. **Speichern.** Immer wieder auf „Jetzt speichern“ tippen.
(sollte autom. speichern)
2. **Hand heben**, wenn etwas klemmt. Wir/Ich kommen zu dir.
3. **Ausprobieren ist erlaubt.** Kaputt geht nichts.
4. **Herausforderungen** = Bonus: Nur machen wenn ihr früh fertig seid

So laufen die Folien

Jedes Kapitel auf deinem Arbeitsblatt hat eine Nummer, zum Beispiel **2**. Die Nummer steht oben links auf jeder Folie. Und jedes Kapitel läuft gleich:

Warum?	Was fehlt dem Spiel gerade
Wie?	Welcher Scratch-Trick das löst
Was?	Screenshots vom Bauen, am Ende die Checkliste

Die Klick-Schritte stehen im Arbeitsblatt. Nummer oben links lesen, dort nachschlagen.

Bitte nicht den nächsten Block früher anfangen. Wenn ihr schneller seid: Herausforderung machen oder experimentieren.

Block 1

Ankommen & Scratch einrichten

Was ist Programmieren?

Stell dir einen Helfer vor, der **alles** macht — und **nichts** versteht.

- „Mach mir ein Brot.“ → Er steht nur da.
- „Nimm das Brot. Nimm das Messer. Streich Butter auf die obere Seite.“ → Klappt.

Programmieren = dem Helfer Schritt für Schritt aufschreiben, was er tun soll.
Die Liste heißt **Programm**.

Er rät nie, was du gemeint hast. Er macht genau, was dasteht.

Programmiersprachen

Der Helfer versteht kein Deutsch. Nur eine **Programmiersprache**.

Davon gibt es viele — wie Deutsch, Englisch, Türkisch:

Python · JavaScript · Java · C · Scratch

Alle sagen im Kern dasselbe:

mach das · wiederhole das · falls das, dann das

Scratch: Die Wörter sind **Blöcke**. Nichts tippen, nur zusammenstecken.
Sie passen nur dort, wo es Sinn ergibt.

WARUM?

Dein Spiel braucht ein Zuhause

Ohne Konto ist nach der Pause **alles weg**. Mit Konto liegt dein Spiel bei Scratch und wartet auf dich.

So geht das in Scratch

iPad **quer** halten. Du trittst unserer **Klasse** bei und denkst dir Name und Passwort selbst aus:

- **Benutzername:** 3–20 Zeichen, Buchstaben, Zahlen, `-` und `_`.
Fang mit `ente` an, z.B. `ente_blitz7`. **Nicht** dein echter Name.
- **Password:** mindestens 6 Zeichen. Haken bei „**Password anzeigen**“, dann siehst du, was du tippst.

Beides sofort auf die Karte schreiben! Ohne Passwort kommst du nicht zurück.

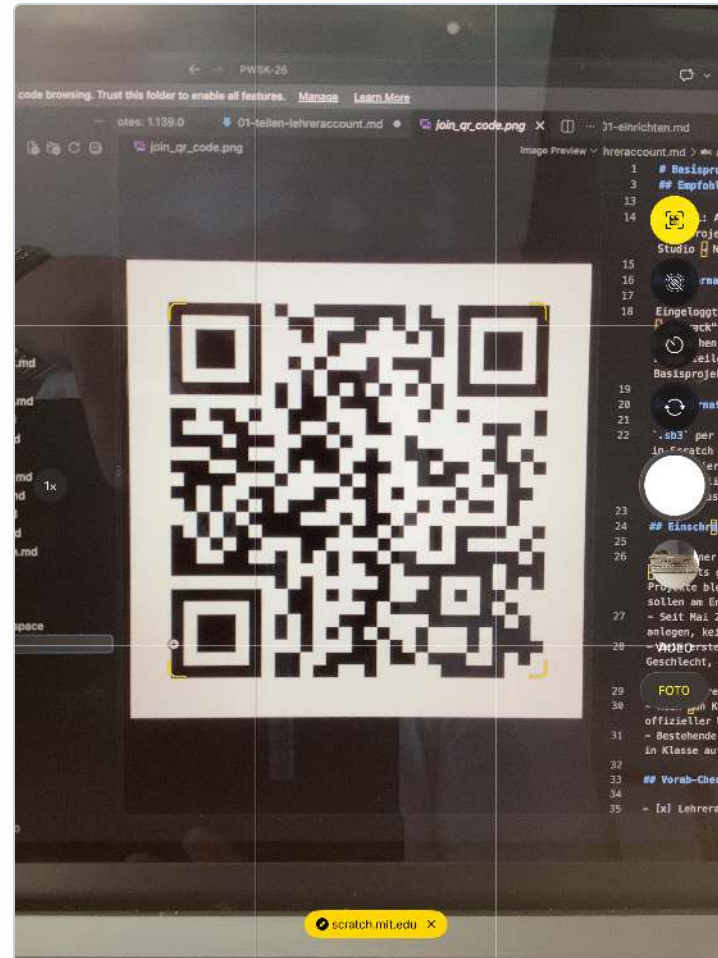
Scanne mich mit der Kamera



Kamera-App → draufhalten → unten auf das gelbe Feld tippen

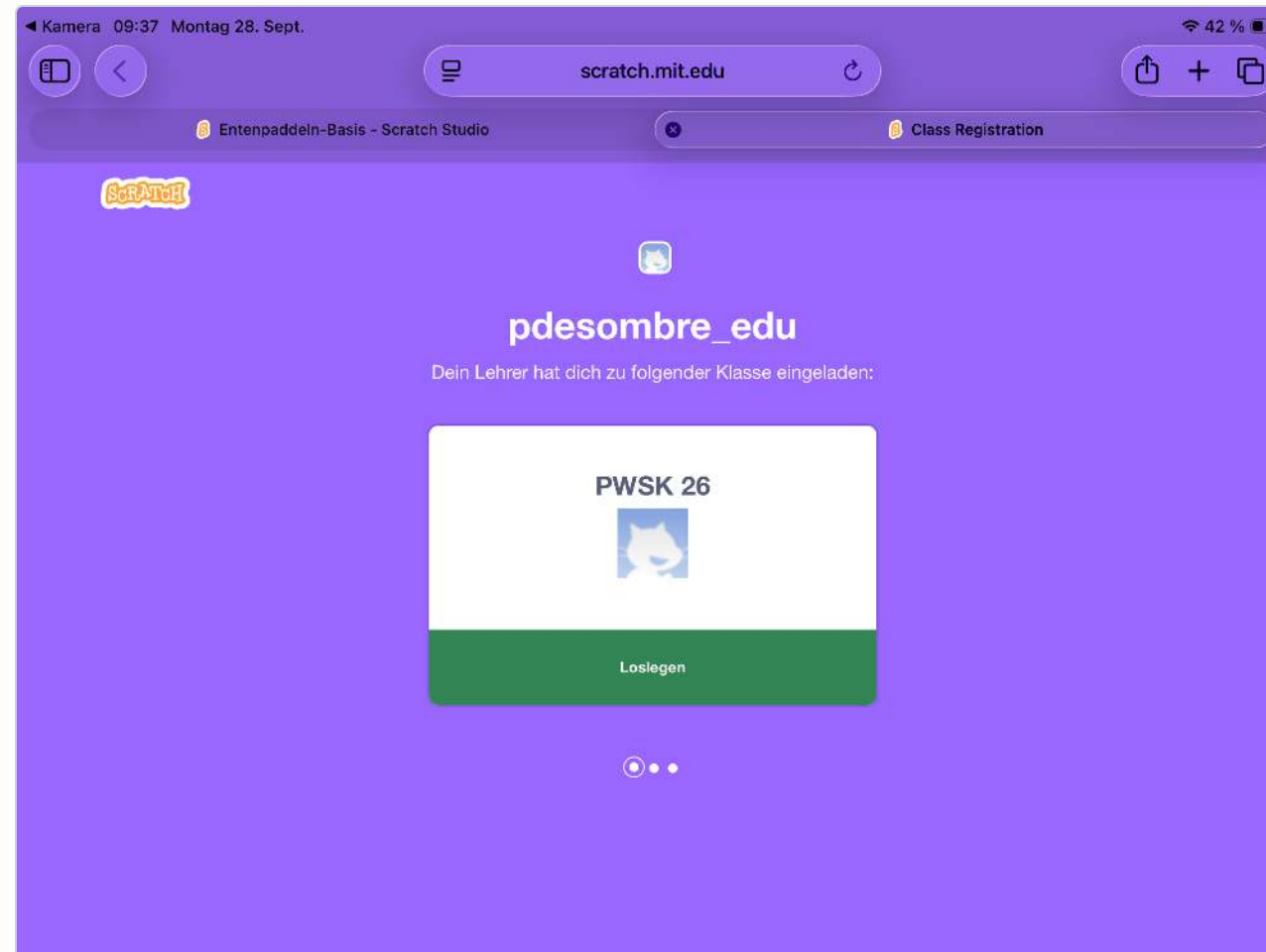
WAS?

Die Kamera erkennt den Code



WAS?

Grüner Knopf „Loslegen“



WAS?

Das Formular

The screenshot shows the Scratch account creation interface on a mobile device. The status bar at the top indicates 'Kamera', '09:37', 'Montag 28. Sept.', and a battery level of 42%. The URL bar shows 'scratch.mit.edu'. The main heading is 'Benutzerkonto'. Below it, a paragraph explains that Scratch is for creating games, animations, and stories, and that creating an account is simple and free. It asks if the user already has an account, with an information icon. The form itself is a white card with a green 'Nächster Schritt' button at the bottom. It has two sections: 'Wähle einen Benutzernamen' with a text input field and a blue callout bubble saying 'Benutze nicht deinen echten Namen', and 'Passwort' with a text input field and a checkbox for 'Passwort anzeigen'. A progress indicator at the bottom shows the first of three steps is active.

Benutzerkonto

Mit Scratch kannst du Spiele, Animationen und Geschichten erschaffen. Ein Benutzerkonto zu erstellen ist einfach und kostenlos. Fülle dazu das folgende Formular aus. Hast du bereits ein Scratch-Benutzerkonto? [i](#)

Wähle einen Benutzernamen
Vielleicht dein Lieblingsessen, dein liebstes Hobby oder Tier und ein paar Zahlen?

Passwort

☐ Passwort anzeigen

Nächster Schritt

Benutze nicht deinen echten Namen

WARUM?

Das Grundgerüst liegt schon bereit

Boot, Ente, Kacke, Hintergrund und alle Geräusche sind fertig.

Du brauchst nur noch **deine eigene Kopie** davon.

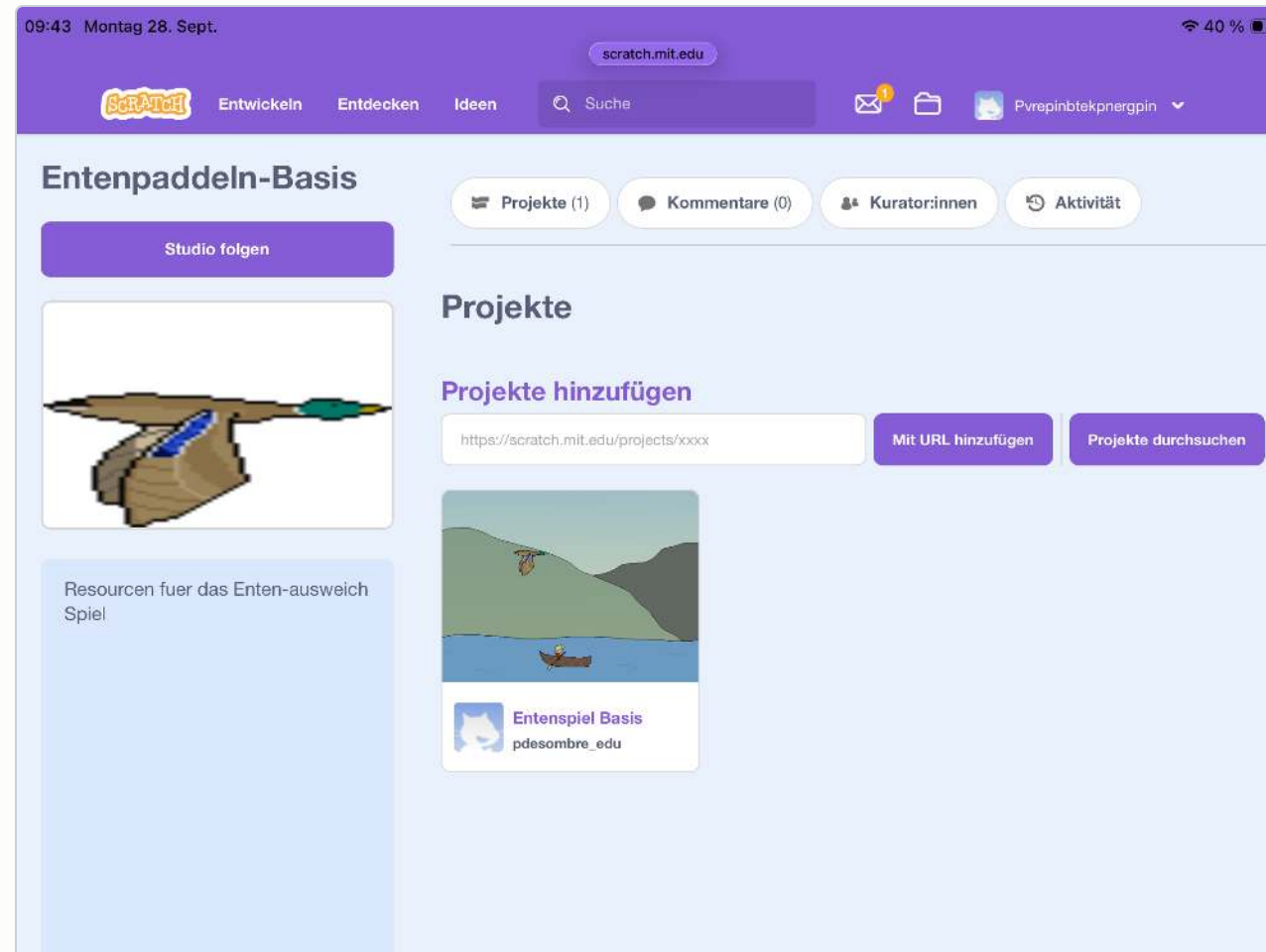
So geht das in Scratch

Eine eigene Kopie von einem fremden Projekt heißt **Remix**.

Das Startprojekt liegt in unserem **Studio** „Entenpaddeln-Basis“.

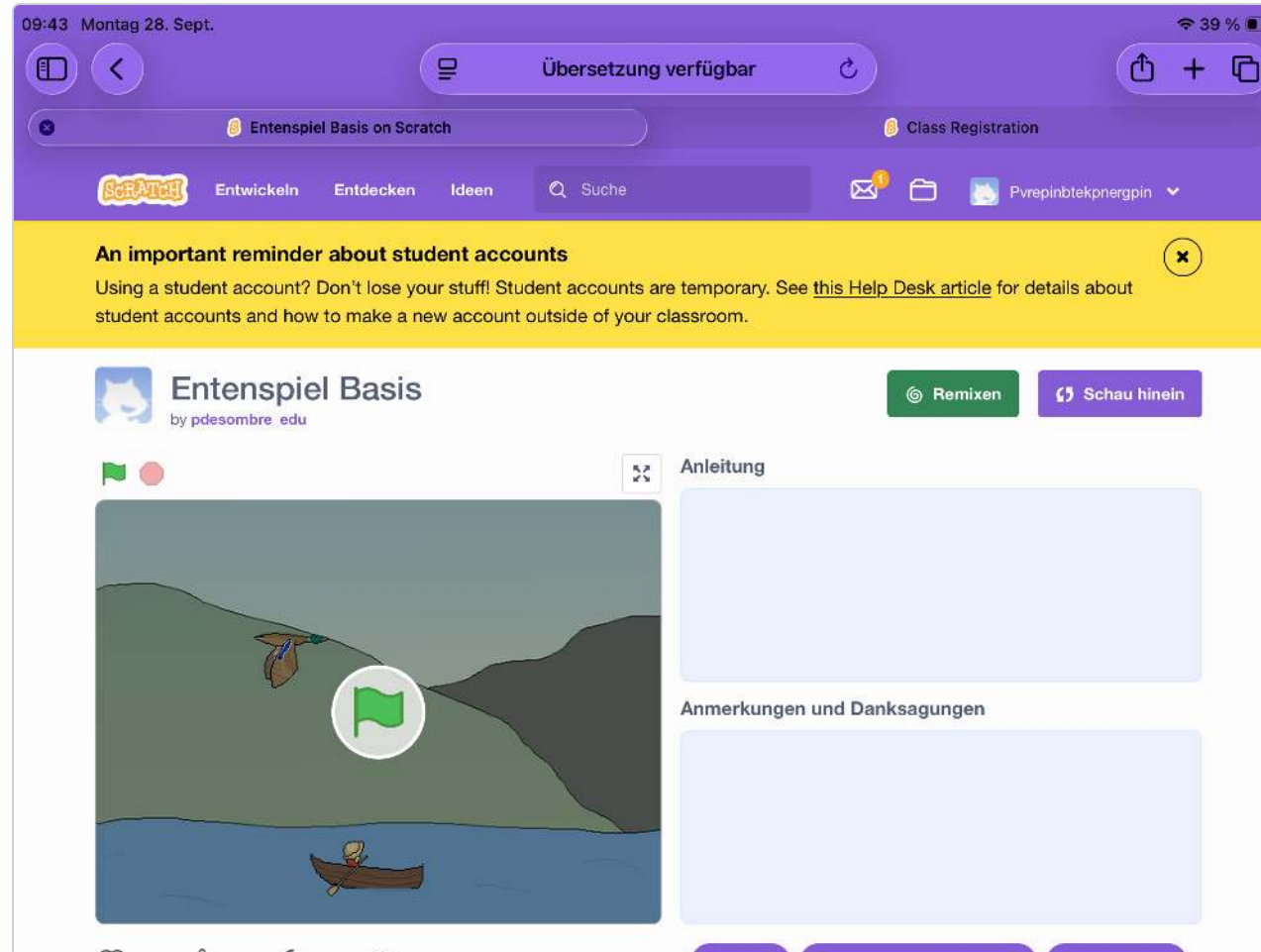
WAS?

Das Studio



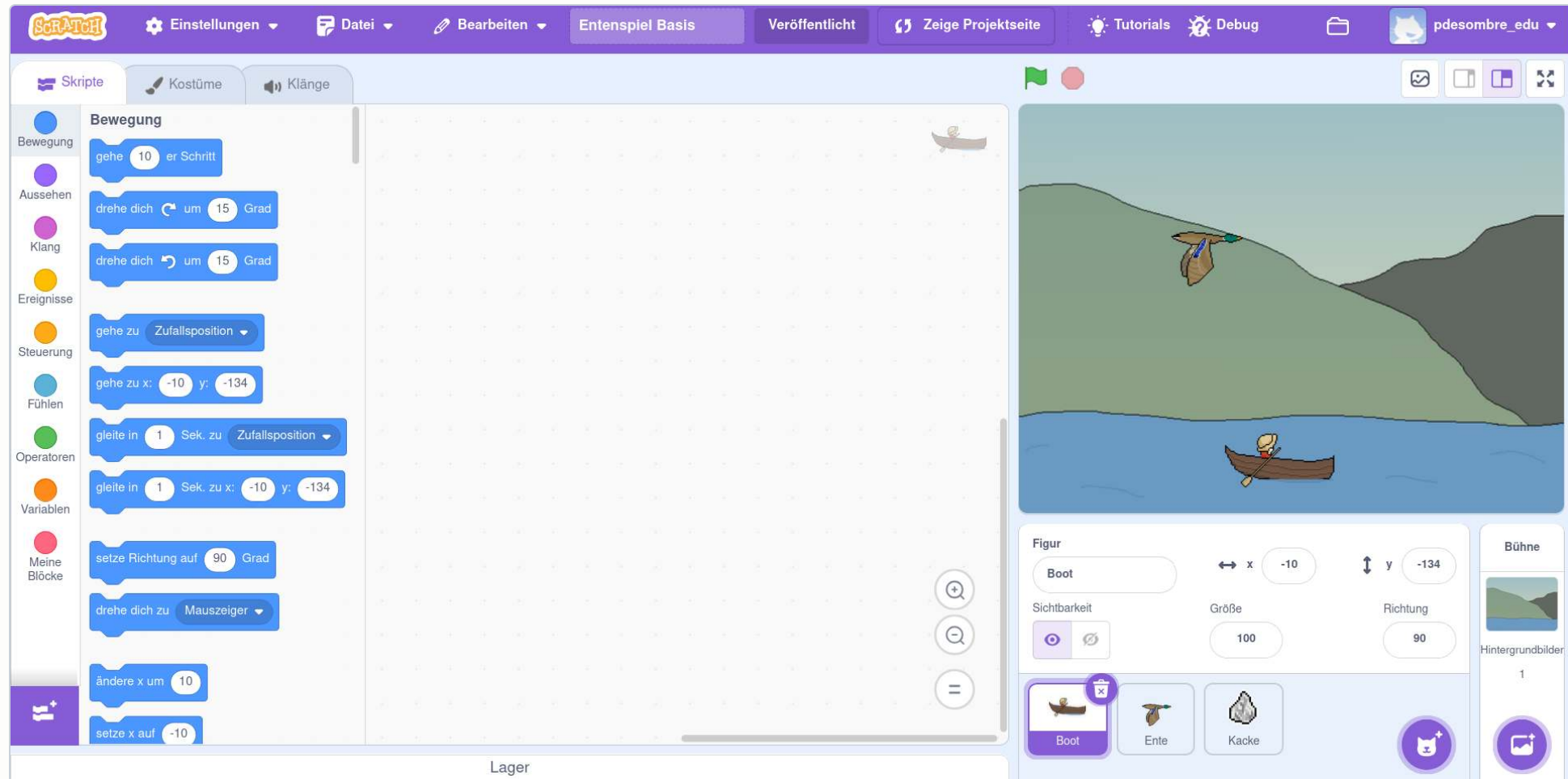
WAS?

Der grüne Knopf oben rechts



WARUM?

Wo baue ich, und wo läuft mein Spiel?



WIE?

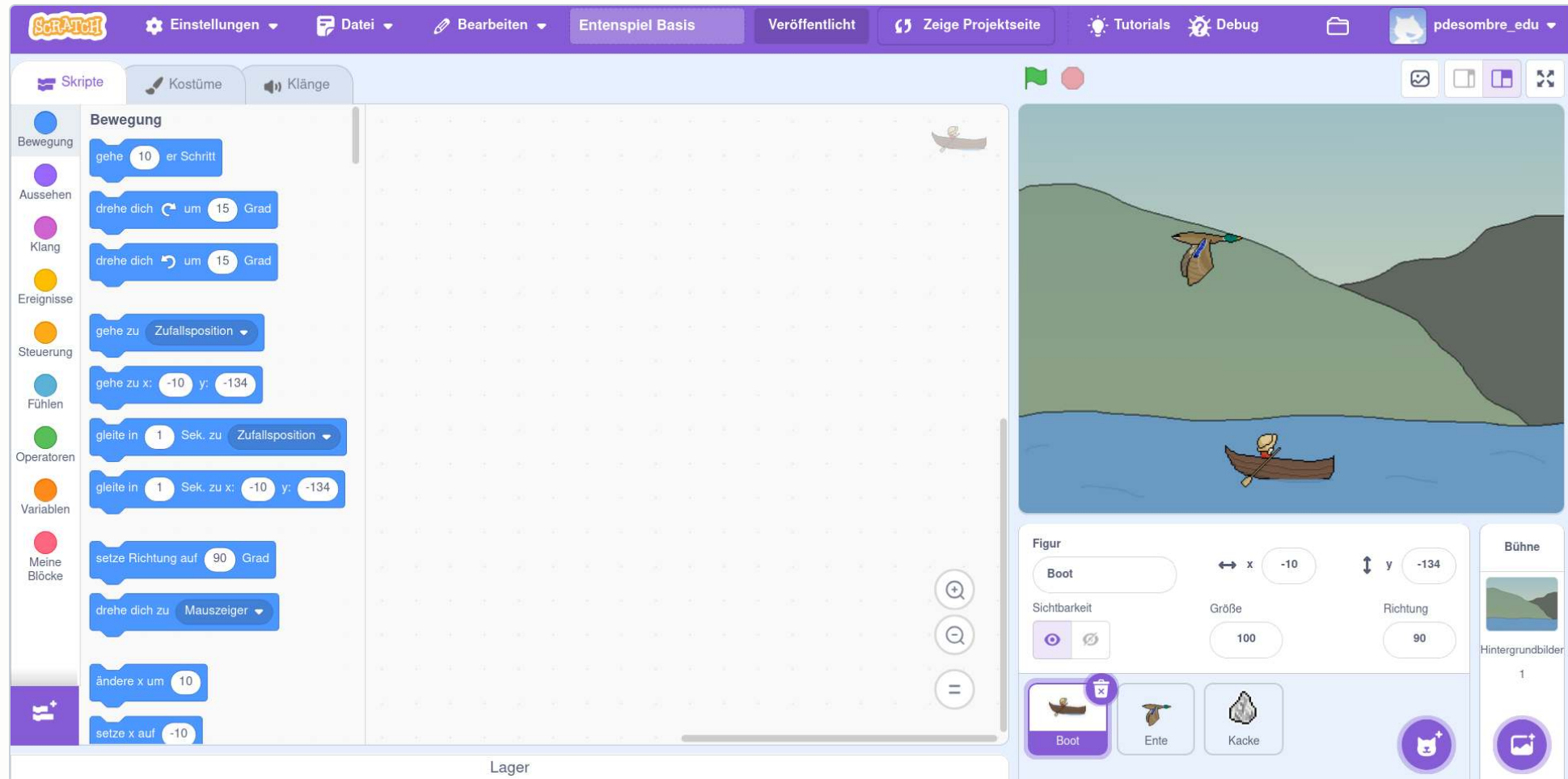
Die vier Bereiche

	Bereich	Wofür
1	Bühne rechts oben	Hier läuft dein Spiel
2	Figurenliste rechts unten	Boot, Ente, Kacke — und die Bühne
3	Blockpalette links	Alle Befehle, nach Farben sortiert
4	Skriptbereich Mitte	Hier baust du

Die Figur, die du antippst, ist die, die du gerade programmierst.

WAS?

Die vier Bereiche auf deinem Bildschirm



WAS?

Mit dem Finger

Scratch ist für die Maus gebaut. Auf dem iPad gilt:

- **Antippen, halten, ziehen** — so bewegst du Blöcke
- Es rastet ein, wenn der **graue Schatten** erscheint
- **Löschen:** zurück in die Palette ziehen
- **Lange drücken:** Menü mit Duplizieren und Löschen
- **Zoomen:** die Knöpfe **+ – =** unten rechts

Nicht mit zwei Fingern zoomen — dann zoomt Safari, nicht Scratch.

WARUM?

Erstes Skript: Funktioniert alles?

Bevor das Spiel losgeht, soll das Boot **einmal etwas sagen**.
Dann wissen wir: Blöcke, Flagge, Bühne — alles geht.

Ein *Script* bezeichnet einen zusammengesetzten Block.

WIE?

Ein Hut und ein Befehl

- Der **Hut-Block** (gelb, rund oben) wartet auf ein Ereignis — hier die grüne Flagge
- Alles, was **darunter hängt**, läuft von oben nach unten
- Ein Block **ohne Hut** passiert bei der Flagge nichts
- Tippst du einen Block direkt an, läuft er **einmal**

WAS?

Speichern

Scratch speichert **nicht immer** von allein bei jedem Schritt.

Nach jeder Übung: „Datei“ -> „Jetzt speichern“ tippen.

Oben in der Menüleiste steht danach kurz **„Projekt gespeichert“**.

WAS?

Checkliste Block 1

- ☐ Ich bin bei Scratch angemeldet.
- ☐ **Nutzername und Passwort stehen auf meiner Karte.**
- ☐ Ich habe mein eigenes Projekt `Entenspiel <Benutzername>` (Remix).
- ☐ Ich weiß, wo Bühne, Figurenliste, Blockpalette und Skriptbereich sind.
- ☐ Das Boot sagt etwas, wenn ich die grüne Flagge drücke.
- ☐ Ich habe auf „Jetzt speichern“ getippt.

Block 2

Das Boot steuern

WARUM?

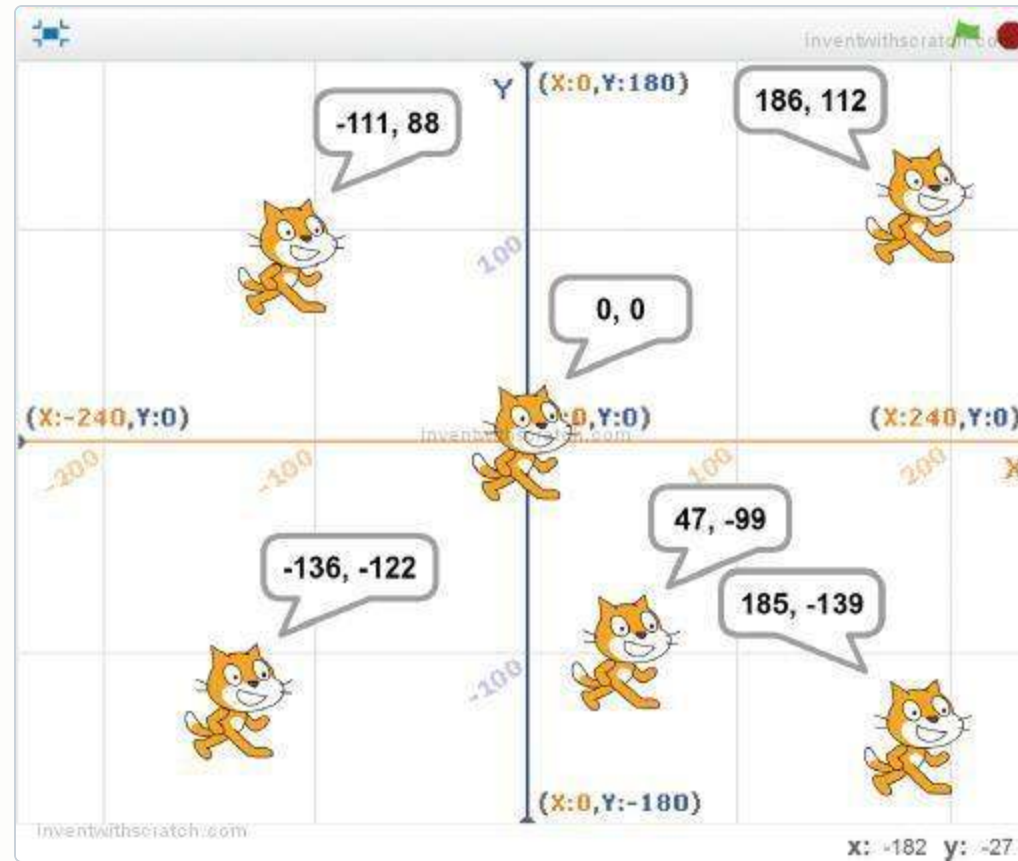
Das Boot soll da sein, wo dein Finger ist

Dafür muss Scratch wissen, **wo** dein Finger ist — und **wo** das Boot steht.

Scratch kennt keine Orte wie „links unten“. Es kennt nur **Zahlen**.

WIE?

Die Bühne ist ein Gitter



WIE?

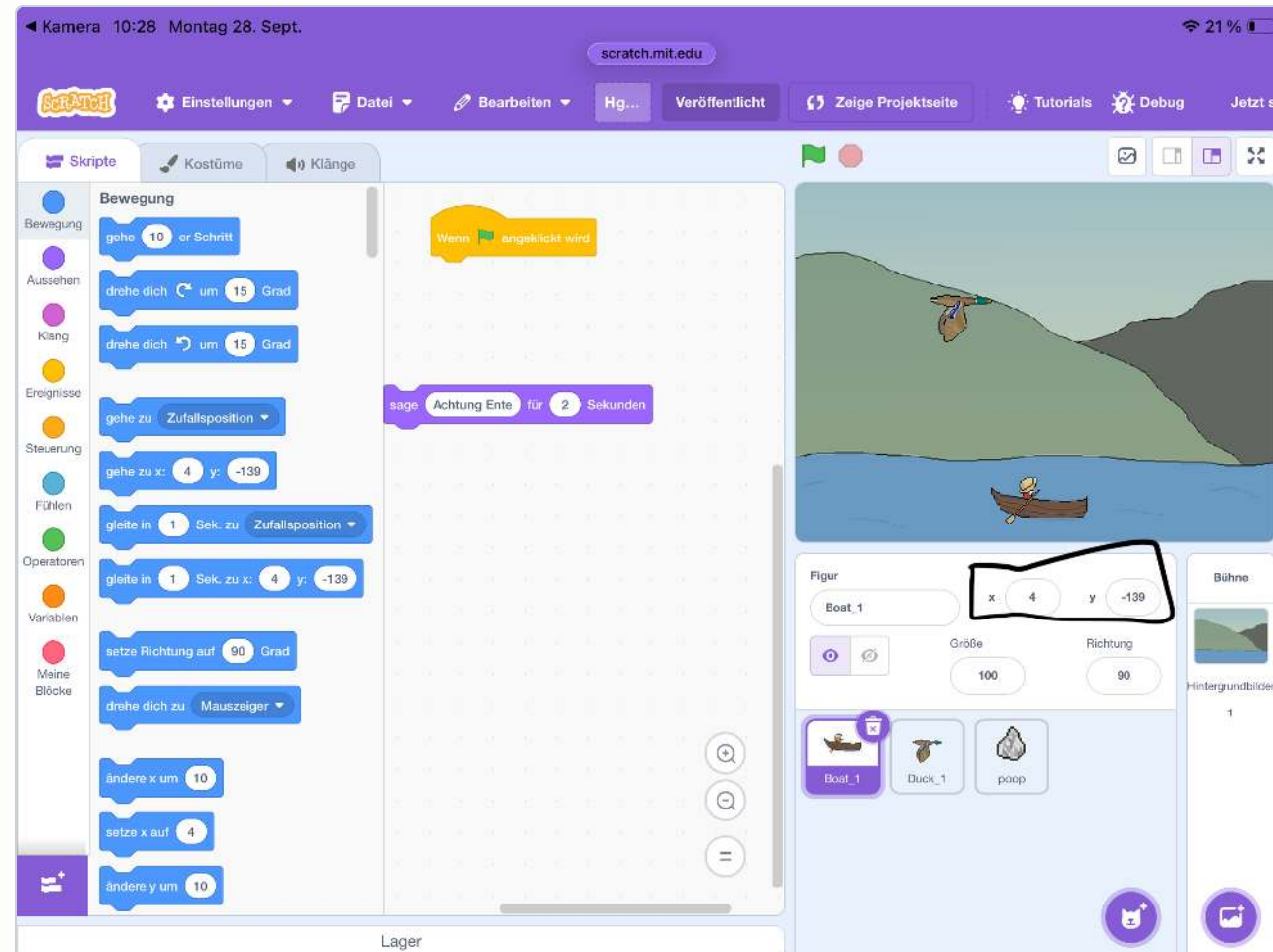
Zwei Zahlen für jeden Punkt

- **x** geht nach rechts: von **-240** bis **240**
- **y** geht nach oben: von **-180** bis **180**
- Mitte ist **x 0, y 0**
- Unten am Wasser: **y -140**

Das Boot soll seinem Finger folgen — also ändern wir **nur x**.

WAS?

Mini-Aufgabe: Wo steht das Boot gerade?



WARUM?

Nach dem Ziehen liegt das Boot irgendwo

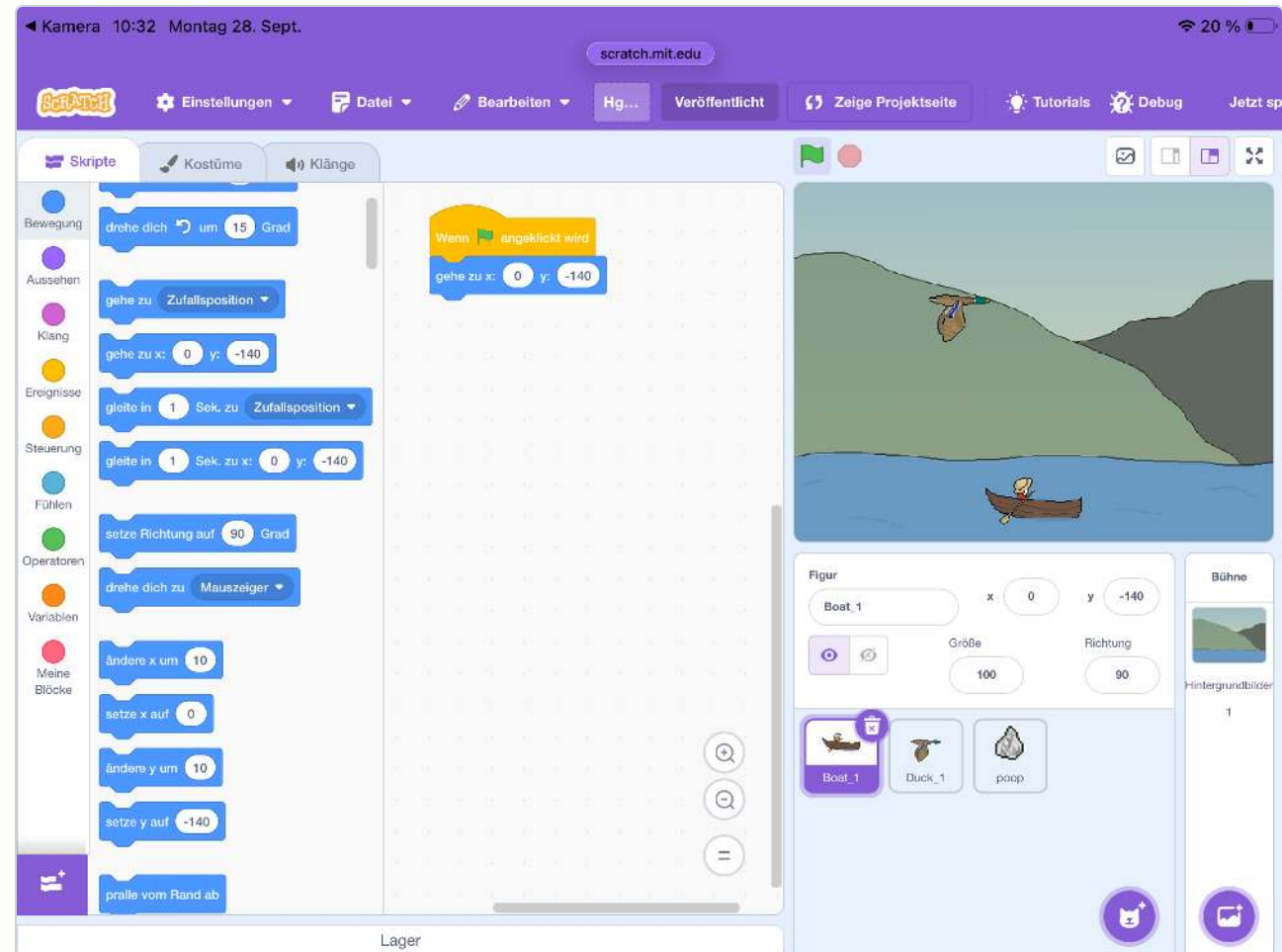
Beim Start soll es **immer** unten in der Mitte stehen.

So geht das in Scratch

„**gehe zu x: () y: ()**“ (Bewegung, blau) setzt die Figur sofort an einen Punkt. Gleich nach dem Hut-Block — dann passiert es bei jeder grünen Flagge zuerst.

WAS?

So sieht das aus



WARUM?

Einmal prüfen reicht nicht

Das Spiel soll **die ganze Zeit** nachschauen, wo dein Finger ist.
Nicht nur einmal beim Start.

WIE?

Ein Block, der nie aufhört

„**wiederhole fortlaufend**“ (Steuerung, orange) ist wie ein **Maul**.

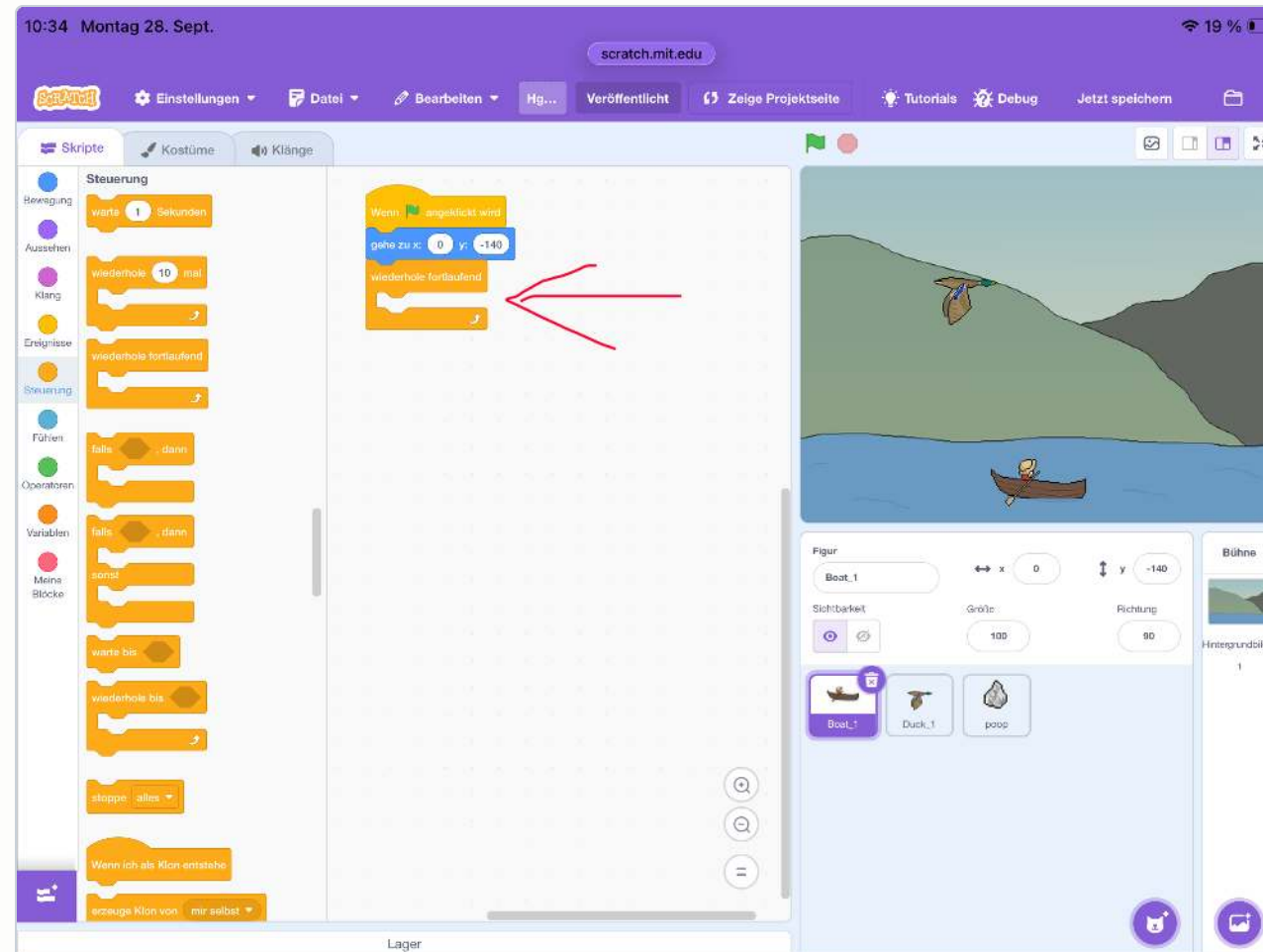
Alles, was drin liegt, läuft immer und immer wieder — bis du stoppst.

Etwa **30 Mal pro Sekunde**.

Probier es: Leg „**nächstes Kostüm**“ (Aussehen) ins Maul. Das Boot flackert wild.
Danach wieder raus — das machen wir in Block 3 ordentlich.

WAS?

Die Schleife unter „gehe zu“



WARUM?

Jetzt die Steuerung

Finger auf die Bühne → Boot fährt hin. Finger weg → Boot bleibt stehen.
Und es bleibt dabei **immer unten** auf dem Wasser.

WIE?

Scratch sagt Maus, du meinst Finger

Scratch sagt ...	Auf dem iPad heißt das ...
Maustaste gedrückt?	Dein Finger berührt die Bühne
Maus x-Position	Wo dein Finger links/rechts ist

Und eine **Bedingung** (Steuerung, orange):

„**falls** < ... > **dann**“ — nur wenn die Frage mit **Ja** beantwortet wird, passiert, was drin liegt.

Sechseckig oder rund?

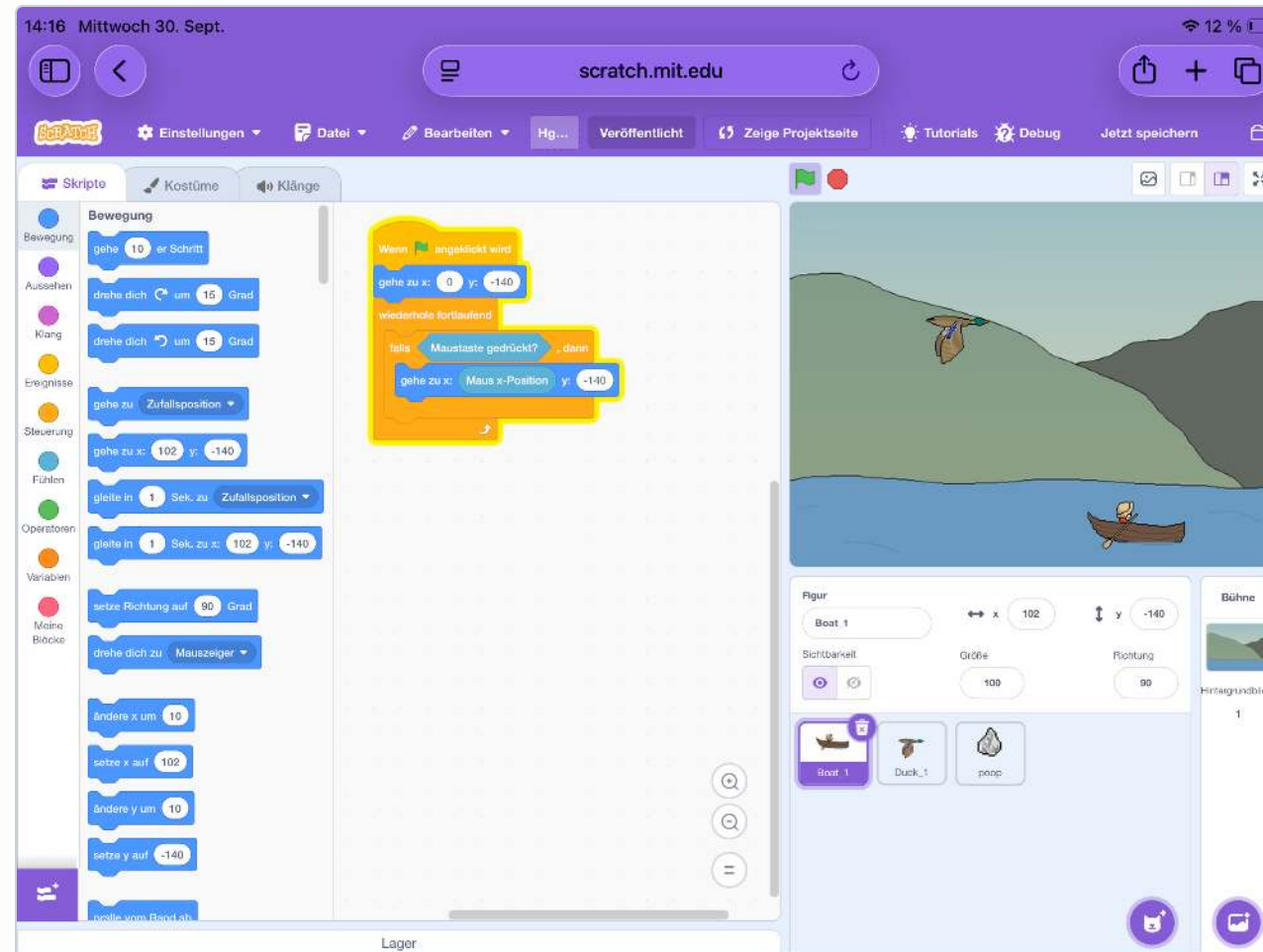
- **Frage-Blöcke** sind **sechseckig**: „Maustaste gedrückt?“
Antwort: Ja oder Nein. Passen nur in sechseckige Löcher.
- **Wert-Blöcke** sind **rund**: „Maus x-Position“
Liefern eine Zahl. Passen nur in runde Löcher — auch anstelle einer festen Zahl.



Wo eine Zahl stehen kann, darf auch ein runder Block stecken.

WAS?

Fertig: Das Steuer-Skript



MERKE

Lies das Skript wie einen Satz

„Wenn die Flagge gedrückt wird: Geh nach unten in die Mitte. Dann prüfe für immer:
Falls ein Finger auf der Bühne ist, **dann** geh zum x des Fingers — und bleib auf y -140.“

- **Endlosschleife:** fast jedes Spiel hat eine
- **Bedingung:** nur bei Ja wird ausgeführt, was drin liegt
- Weil **y** jedes Mal wieder -140 ist, bleibt das Boot unten — egal, wohin du tippst

WAS?

Checkliste Block 2

- ☐ Ich weiß, was x und y auf der Bühne bedeuten.
- ☐ Mein Boot steht beim Start unten in der Mitte (x: 0, y: -140).
- ☐ Ich habe eine „wiederhole fortlaufend“-Schleife im Skript.
- ☐ Das Boot folgt meinem Finger nach links und rechts.
- ☐ Ich habe gespeichert.

Block 3

Animation

WARUM?

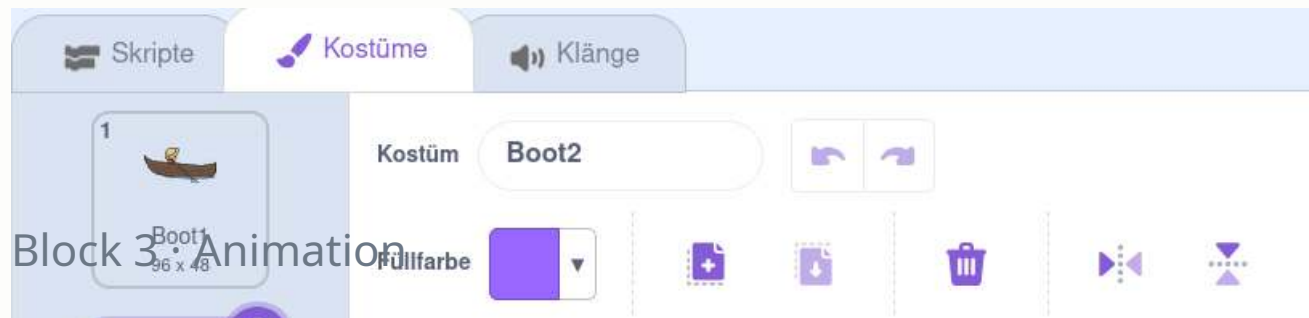
Das Boot steht starr im Wasser

Ein echtes Boot **paddelt**. Eine echte Ente **schlägt mit den Flügeln**.

So geht das in Scratch

Eine Figur kann **mehrere Bilder** haben. Die heißen **Kostüme**. Nur eins ist gerade zu sehen.

- Boot: **Boot1** und **Boot2** → das Paddel in zwei Stellungen
- Ente: **Ente1** und **Ente2** → die Flügel oben und unten



WARUM?

Ein Trickfilm ist schnelles Bilderwechseln

Boot1, Boot2, Boot1, Boot2 ... schnell genug hintereinander — und das Boot paddelt.

WIE?

Nächstes Kostüm, kurz warten, wieder von vorn

Ein **zweites** Skript beim Boot, **neben** dem Steuer-Skript — nicht dranhängen:

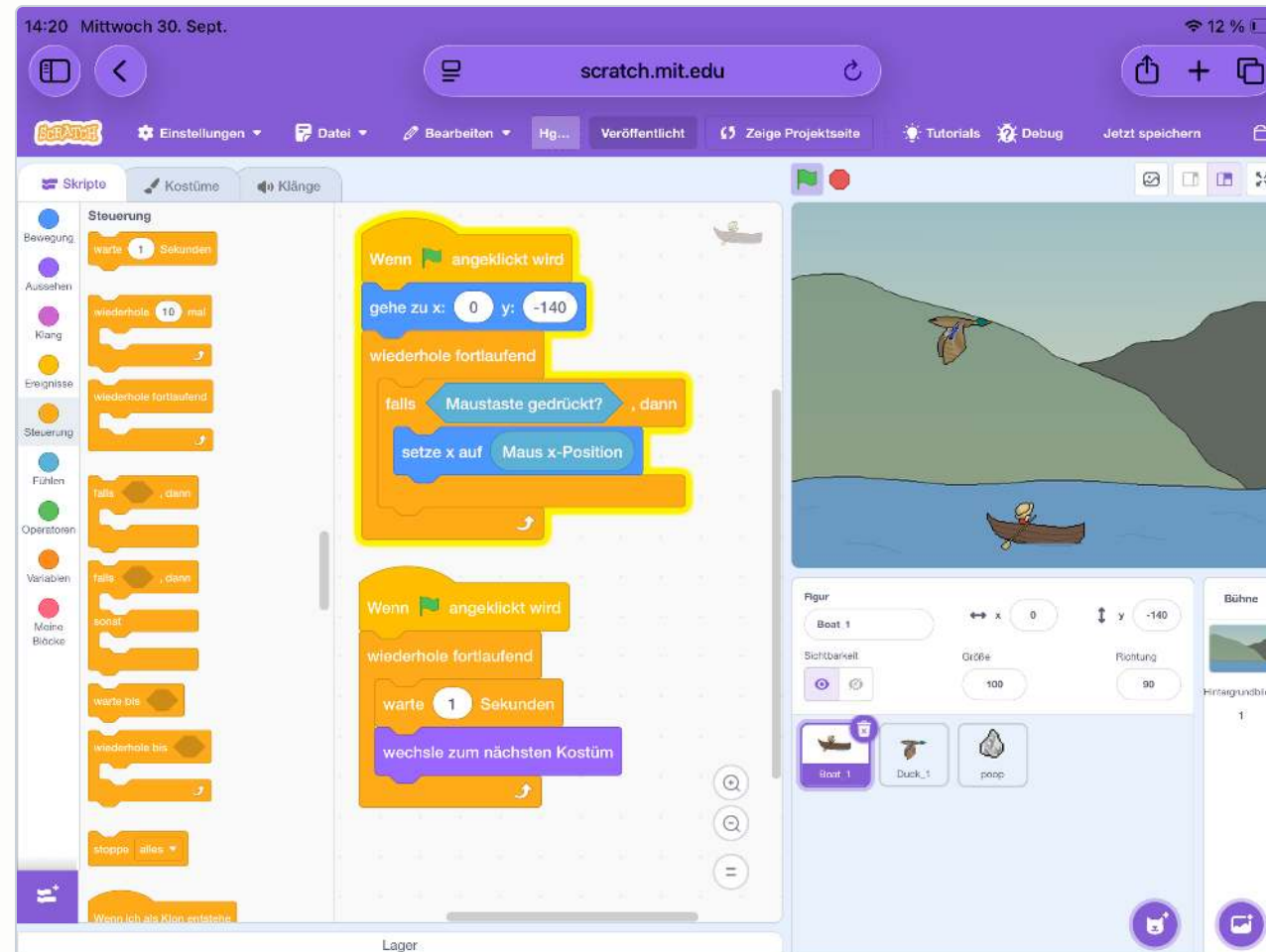
```
Wenn  angeklickt wird  
wiederhole fortlaufend  
  nächstes Kostüm  
  warte (0.3) Sekunden
```

- „**nächstes Kostüm**“ (Aussehen) springt zum nächsten Bild, nach dem letzten wieder zum ersten
- „**warte**“ (Steuerung) bremst die Schleife. Ohne warte: nur Flackern.

Dezimalzahlen mit **Punkt**: 0.3, nicht 0,3.

WAS?

Das Boot hat jetzt zwei Skripte



WARUM?

Die Ente genauso, nur schneller

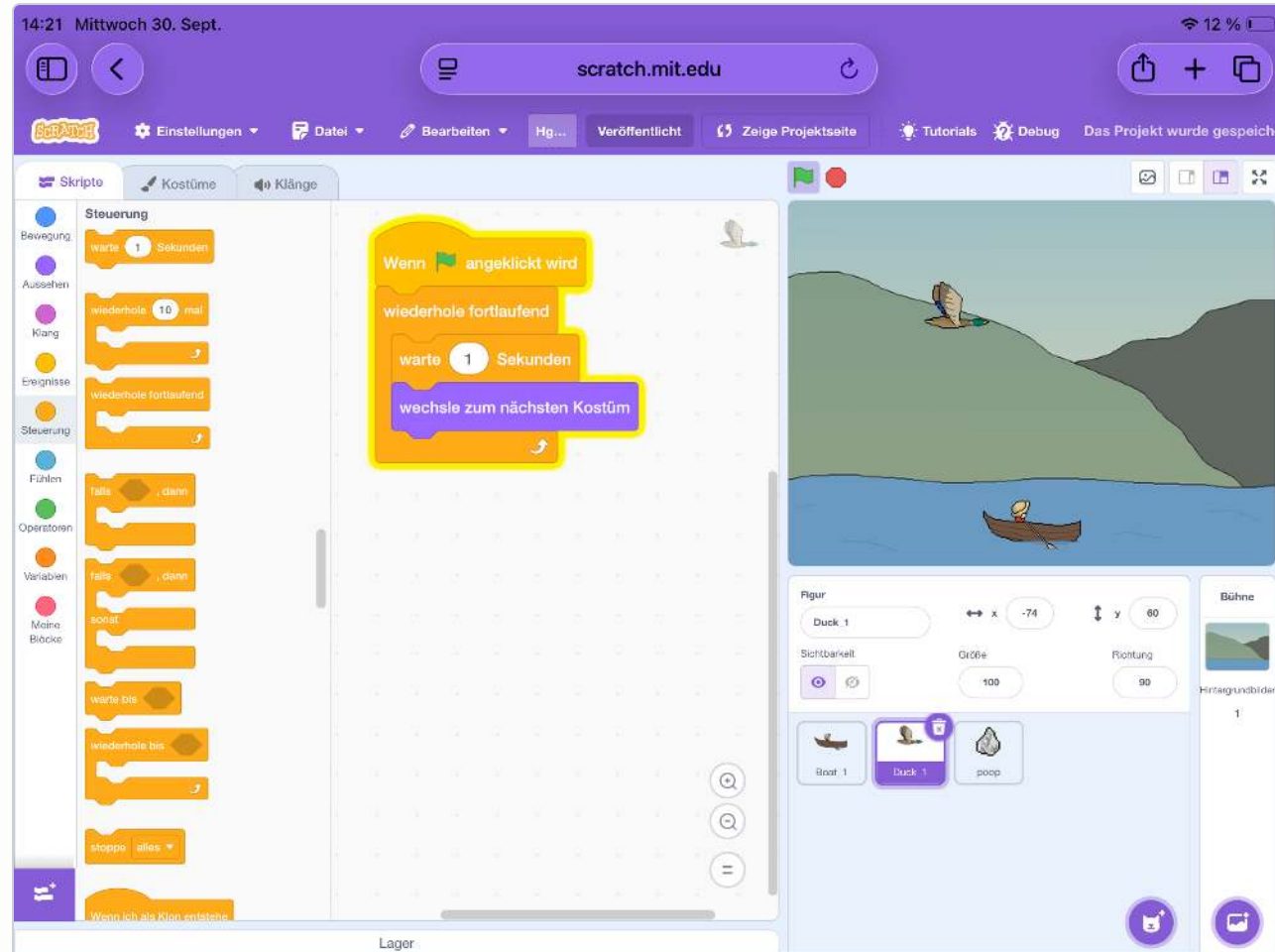
Figur **Ente**, dasselbe Skript, Wartezeit **0.2**:

```
Wenn  angeklickt wird  
wiederhole fortlaufend  
    nächstes Kostüm  
    warte (0.2) Sekunden
```

Grüne Flagge: Die Ente flattert, das Boot paddelt. **Jetzt speichern.**

WAS?

Und die Ente flattert



MERKE

Zwei Skripte gleichzeitig

Beide Skripte starten bei der grünen Flagge. Beide laufen **parallel**.

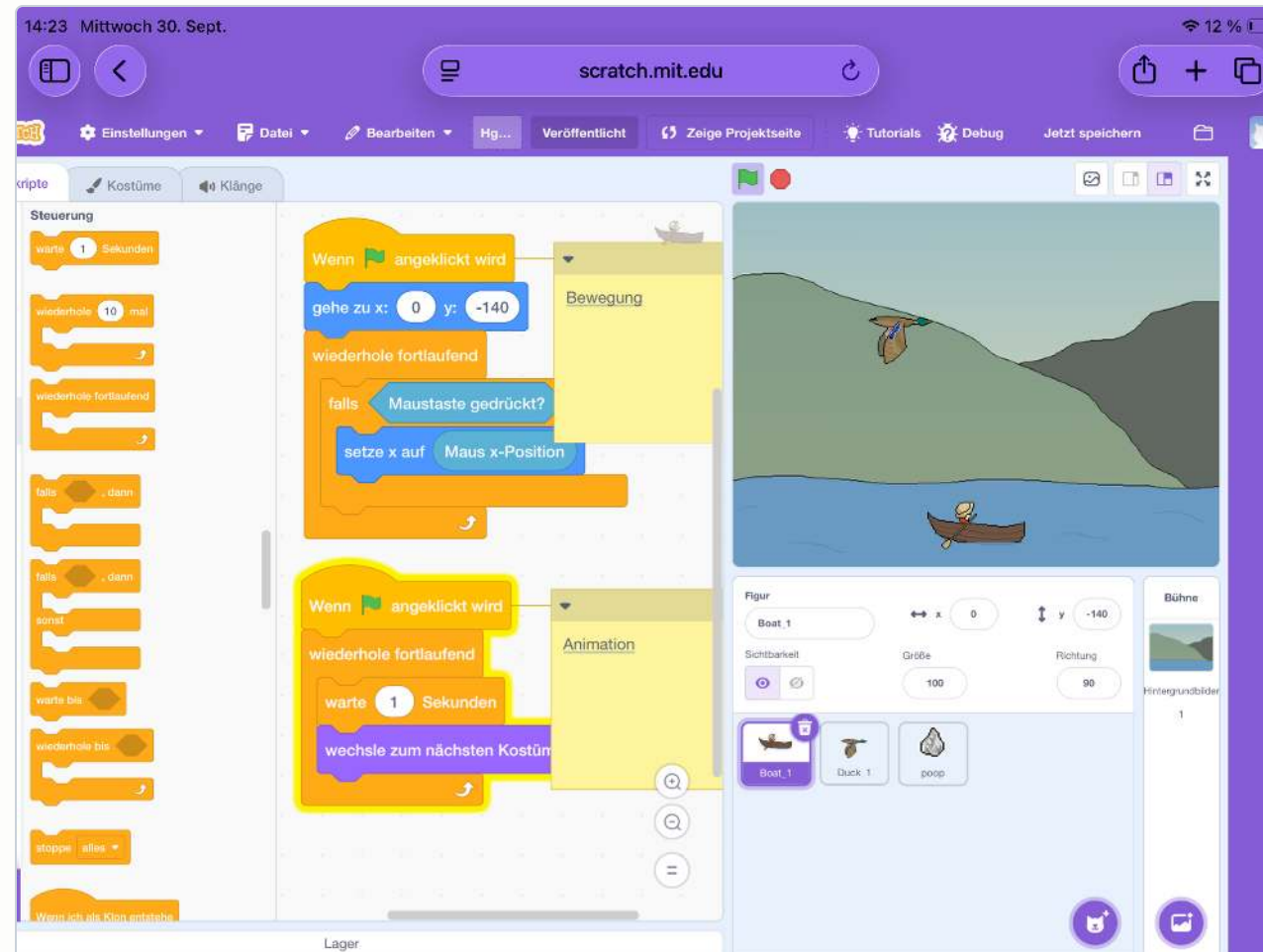
- Skript 1: Boot folgt dem Finger
- Skript 2: Boot wechselt das Kostüm

So bauen wir das ganze Spiel: **für jede Aufgabe ein eigenes kleines Skript.**

Ohne „warte“ flackert es wie verrückt. Probier ruhig 0.05 aus — und dann zurück auf 0.3.

MERKE

Bewegung und Animation nebeneinander



WAS?

Checkliste Block 3

- ☐ Ich weiß, wo die Kostüme einer Figur sind.
- ☐ Das Boot wechselt beim Spielen zwischen Boot1 und Boot2.
- ☐ Die Ente flattert.
- ☐ Ich verstehe: Eine Figur kann mehrere Skripte gleichzeitig laufen lassen.
- ☐ Ich habe gespeichert.

Pause

10 Minuten

Vorher: „Jetzt speichern“ tippen!

Block 4

Die Ente fliegt

WARUM?

Die Ente soll von selbst fliegen

Bis jetzt haben wir Figuren mit **x** und **y** an einen Ort **gesetzt**.
Die Ente soll aber **losfliegen** — ohne dass jemand sie steuert.

WIE?

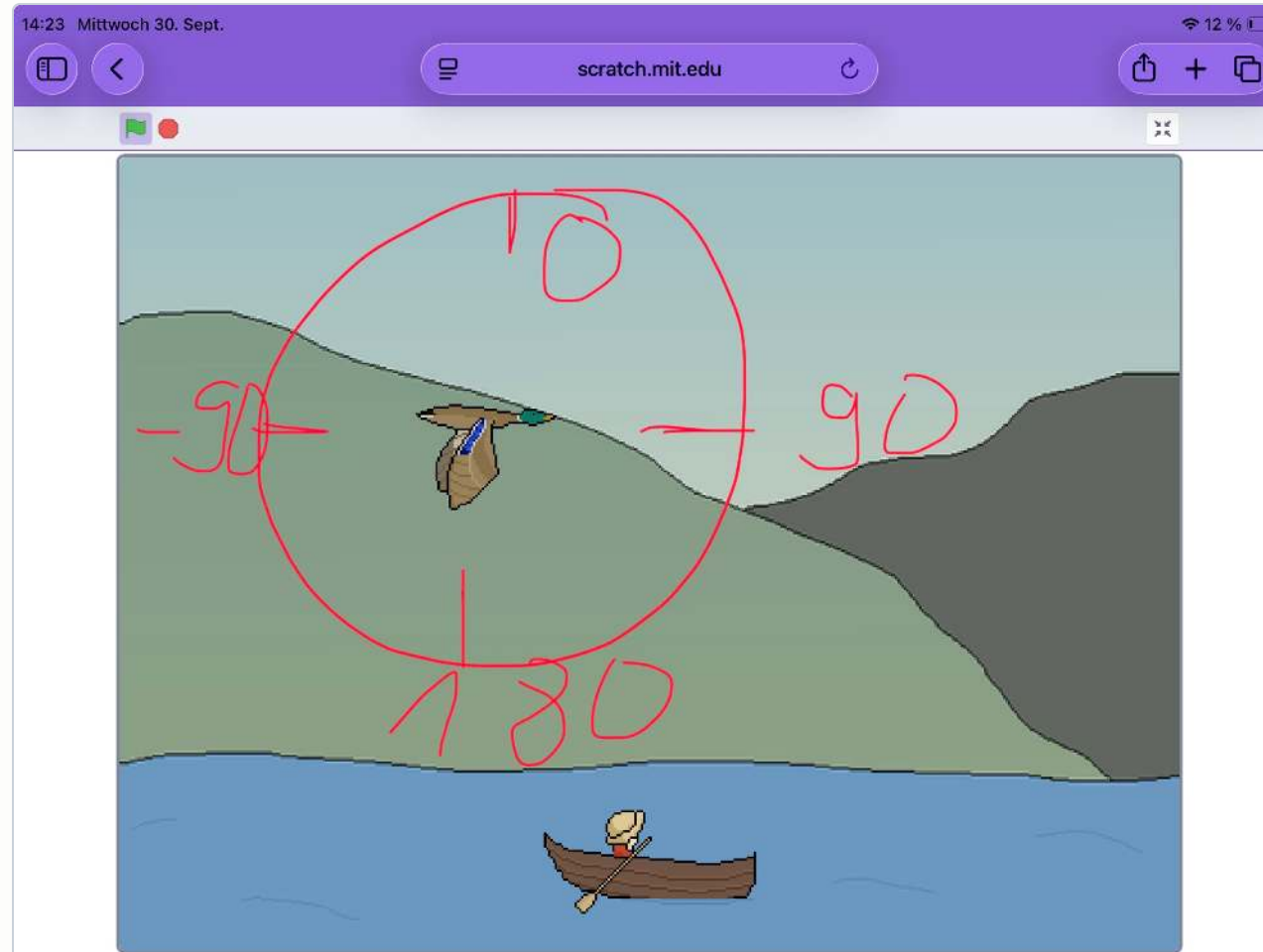
Richtung wie auf dem Kompass

- Jede Figur hat eine **Richtung** in Grad: **90 = rechts**, **-90 = links**, **0 = oben**, **180 = unten**
- **„gehe (10)er Schritt“** (Bewegung) bewegt sie 10 Pixel in diese Richtung

Zusammen wie ein Auto: **Lenken** und **Gas geben**.

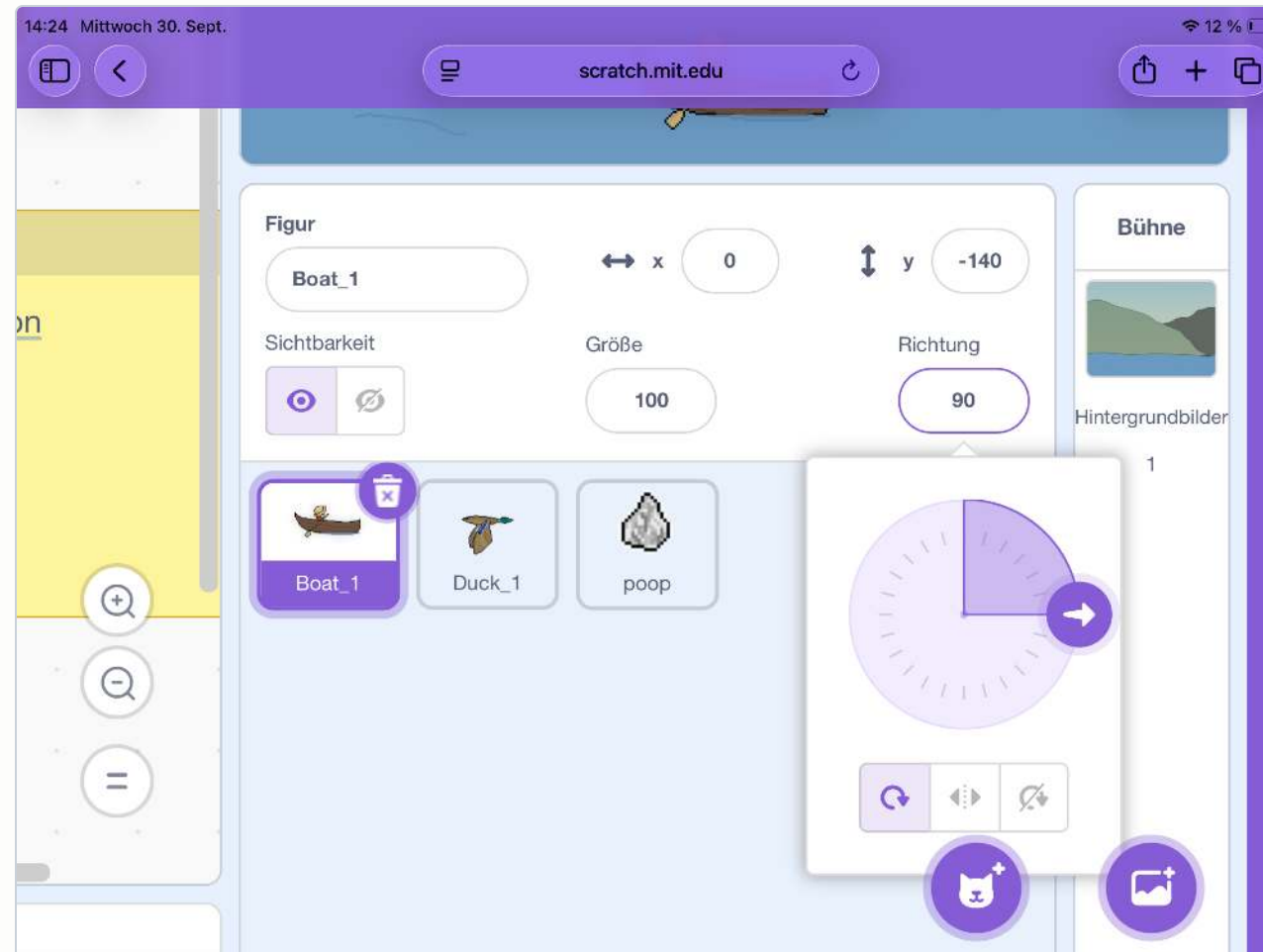
WIE?

Die Richtungen



WAS?

Mini-Aufgabe: Das Drehrad



WARUM?

Lenken und Gas geben

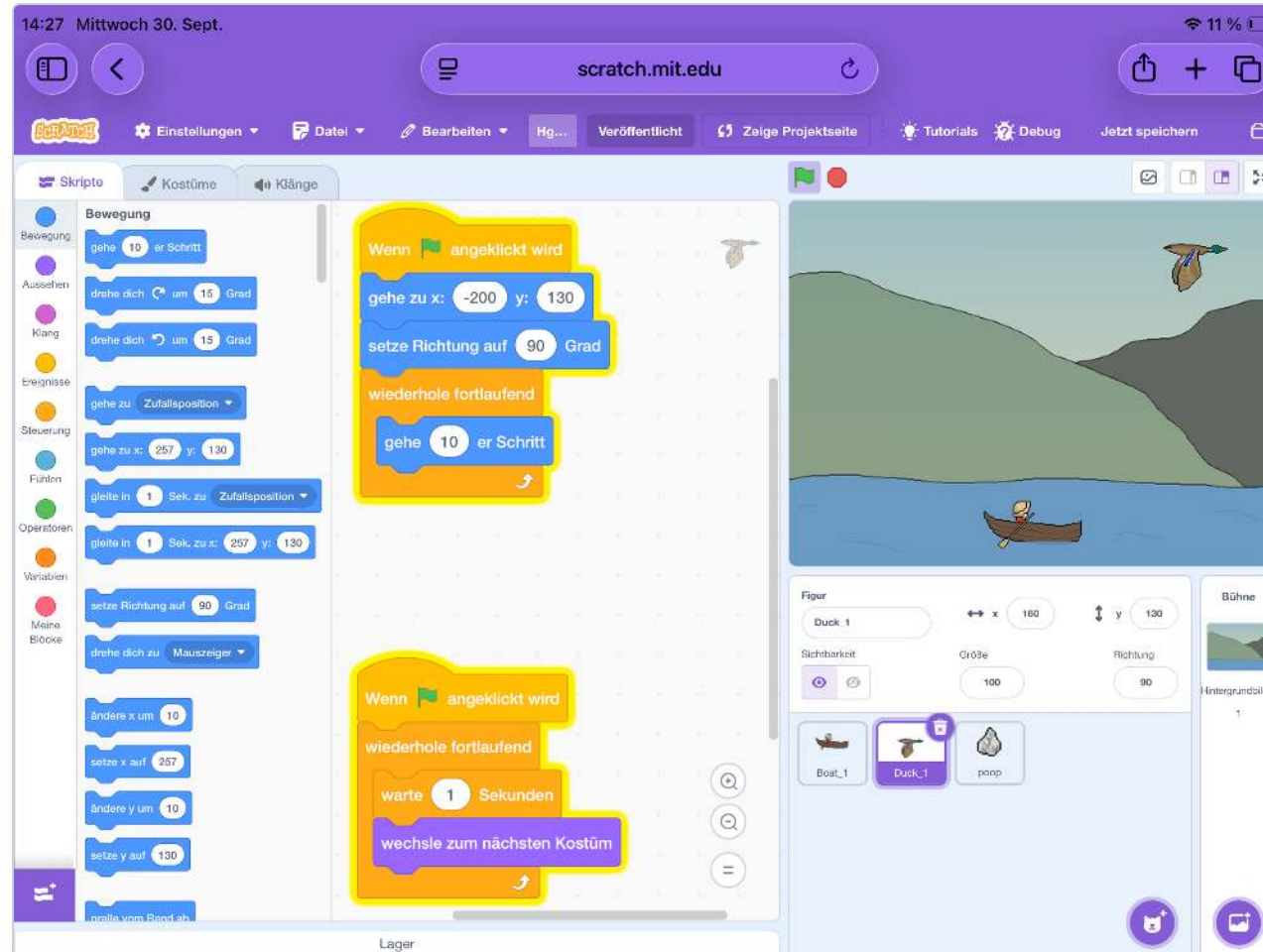
Die Ente startet oben links, schaut nach rechts und fliegt los.

So geht das in Scratch

Ein **zweites** Skript bei der Ente, neben dem Flatter-Skript:
Richtung setzen, dann in der Schleife immer wieder **„gehe (4)er Schritt“**.

WAS?

Was fehlt?



WARUM?

Sie klebt am Rand

Die Ente soll am Rand **umdrehen** — und dabei **richtig herum** bleiben.

WIE?

Abprallen — und der wichtigste Block

„**pralle vom Rand ab**“ (Bewegung) prüft bei jedem Durchlauf: Rand berührt? Dann Richtung umdrehen.

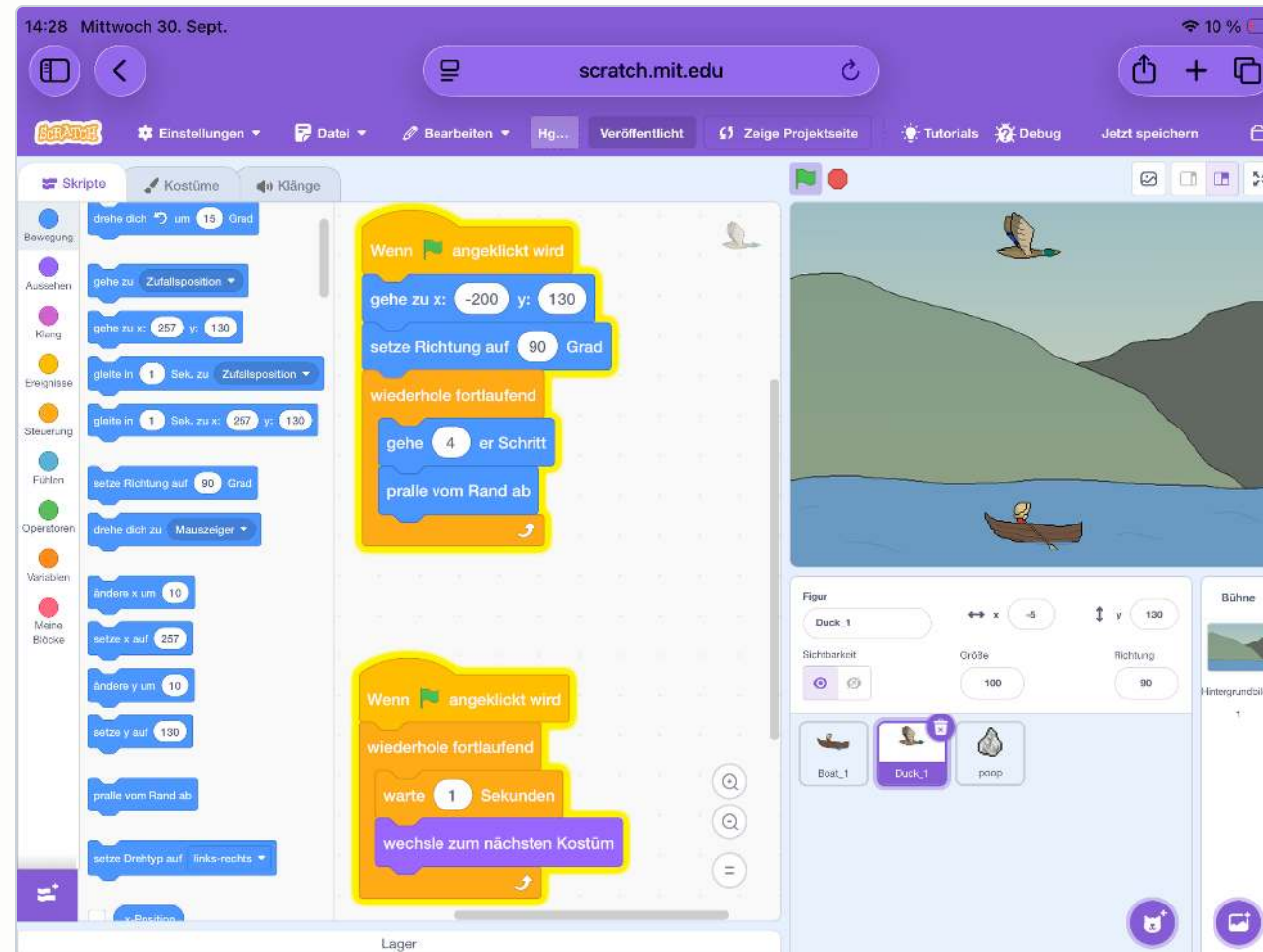
Aber: Scratch **dreht** die Ente dabei um 180 Grad wie einen Ball. Deshalb:

```
setze Drehtyp auf [links-rechts]
```

Drehtyp	Bild
rundum	dreht sich mit, kopfüber bei 180
links-rechts	wird nur gespiegelt — richtig für Tiere, Autos, Boote
nicht drehen	bleibt immer gleich

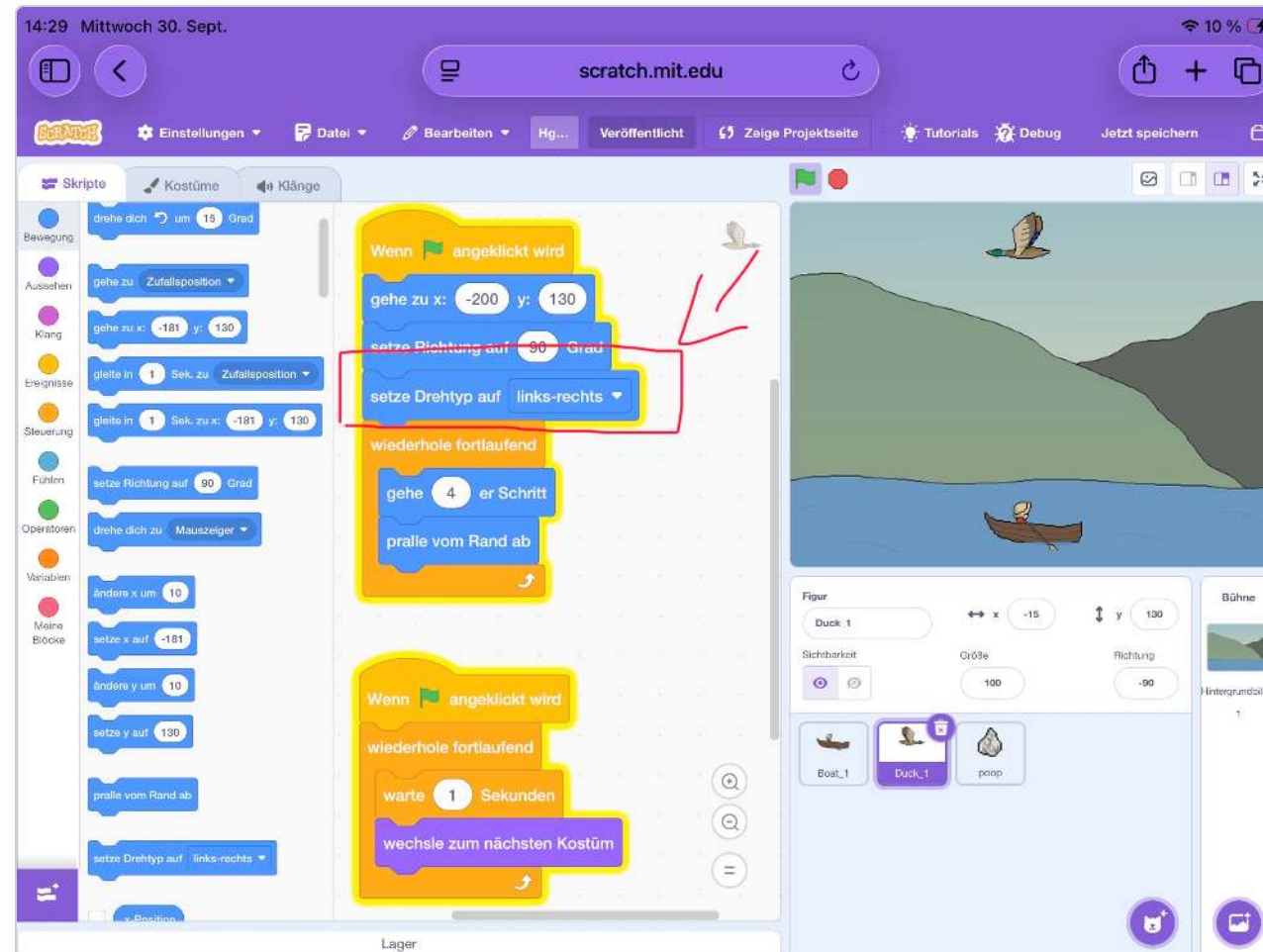
WAS?

Kopfüber!



WAS?

Das fertige Flug-Skript



WARUM?

Immer gleich ist langweilig

Ein Spiel, das jedes Mal genau gleich abläuft, kann man auswendig lernen.
Spiele brauchen **Zufall**.

WIE?

Der Würfel-Block

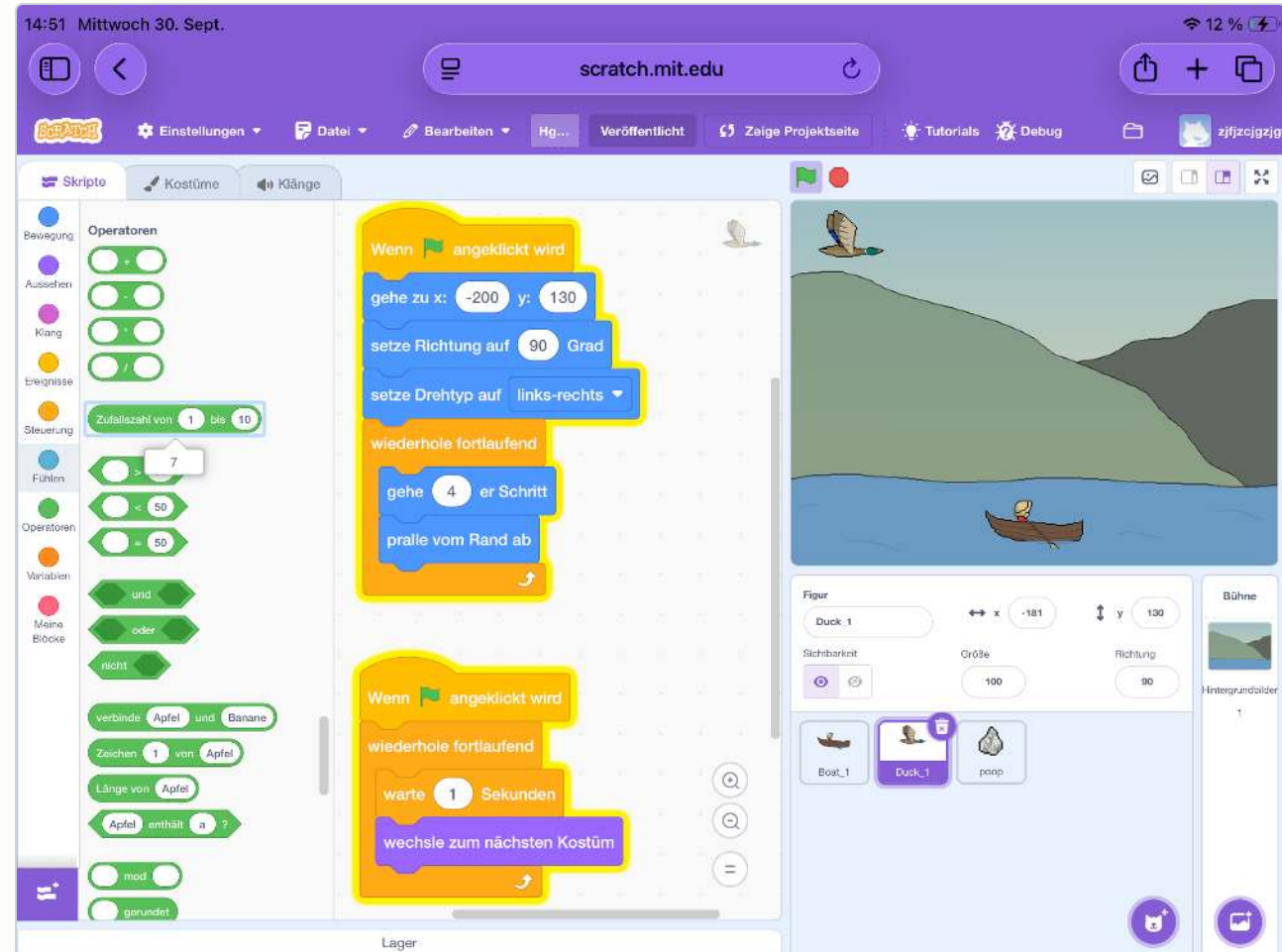
„**Zufallszahl von (1) bis (10)**“ (Operatoren, grün) würfelt **jedes Mal neu**.

Er ist **rund** — ein Wert-Block. Er passt in jedes runde Loch, auch anstelle einer festen Zahl.

Tipp ihn ein paar Mal direkt in der Palette an: Jedes Mal eine andere Zahl in der Blase.

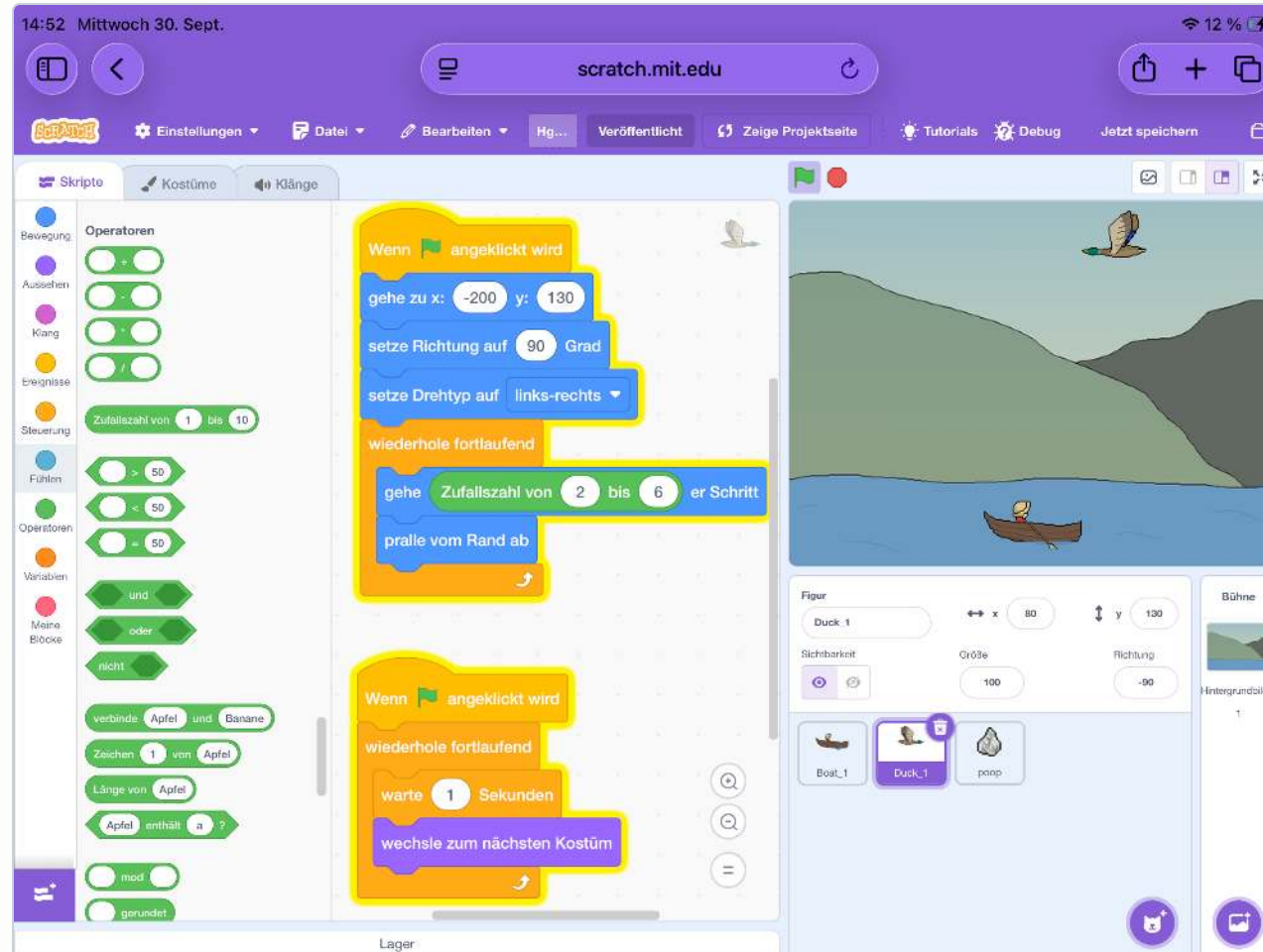
WIE?

Der Zufallszahl-Block in der Palette



WAS?

Zufall im gehe-Block



WAS?

Checkliste Block 4

- ☐ Die Ente startet oben links und fliegt nach rechts.
- ☐ Sie dreht am Rand um.
- ☐ Sie fliegt auf dem Rückweg **nicht** kopfüber (Drehtyp links-rechts).
- ☐ Ich weiß, was der Zufallszahl-Block macht und wo er hinpasst.
- ☐ Ich habe gespeichert.

Block 5

Es kackt

WARUM?

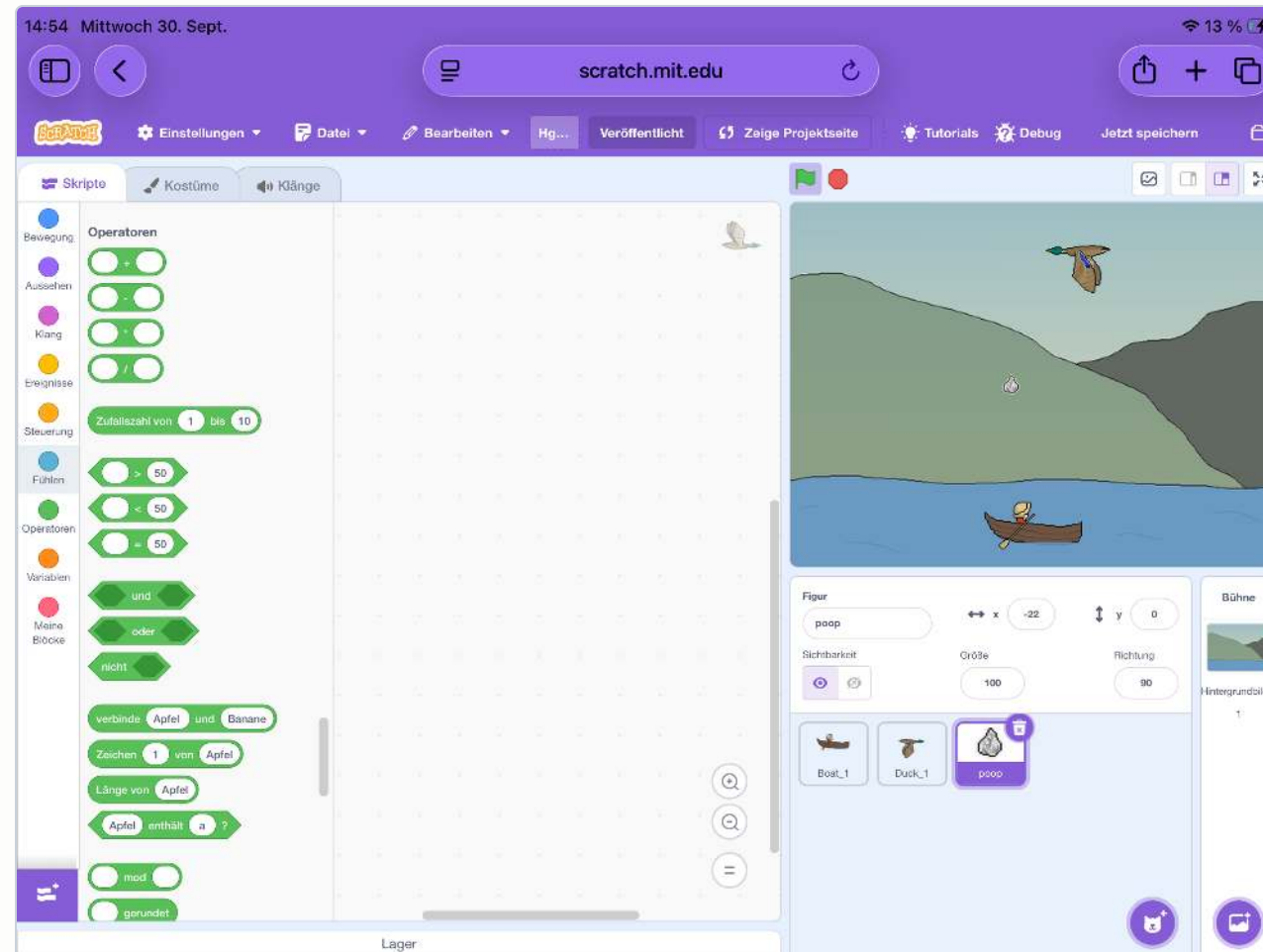
Eine Figur, viele Kacken

In der Figurenliste gibt es genau **eine** Kacke. Im Spiel sollen aber **viele** gleichzeitig fallen.

Zehnmal duplizieren? Viel Arbeit — und nach 10 Kacken wäre Schluss.

WARUM?

Nur eine Kacke, kein Skript



WIE?

Der Stempel: Klon

Ein **Klon** ist eine Kopie, die das Spiel **selbst** erzeugt, während es läuft.

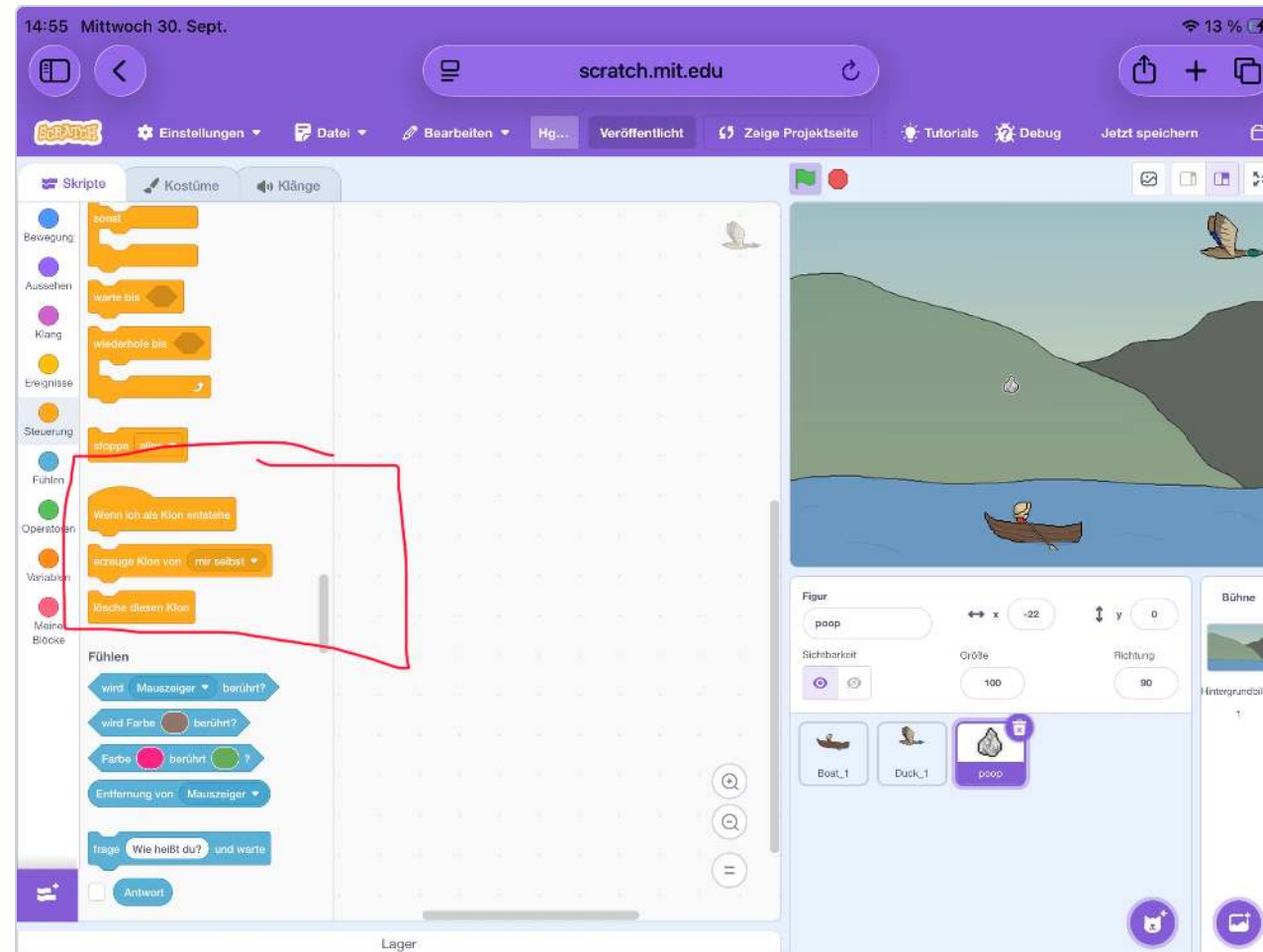
Die Figur Kacke ist ein **Stempel**. Er bleibt versteckt in der Schublade.

Jedes Mal, wenn die Ente kacken soll, drückt sie ihn auf die Bühne: ein **Abdruck** = ein Klon.
Der Abdruck fällt, und unten löst er sich auf.

Block (Steuerung, ganz unten)	Was er tut
„erzeuge Klon von (...)“	stempelt — darf auch eine andere Figur
„Wenn ich als Klon entstehe“	Hut-Block, startet nur für den Klon
„lösche diesen Klon“	der Klon verschwindet

WIE?

Die drei Klon-Blöcke



WARUM?

Alle 2 Sekunden einmal stempeln

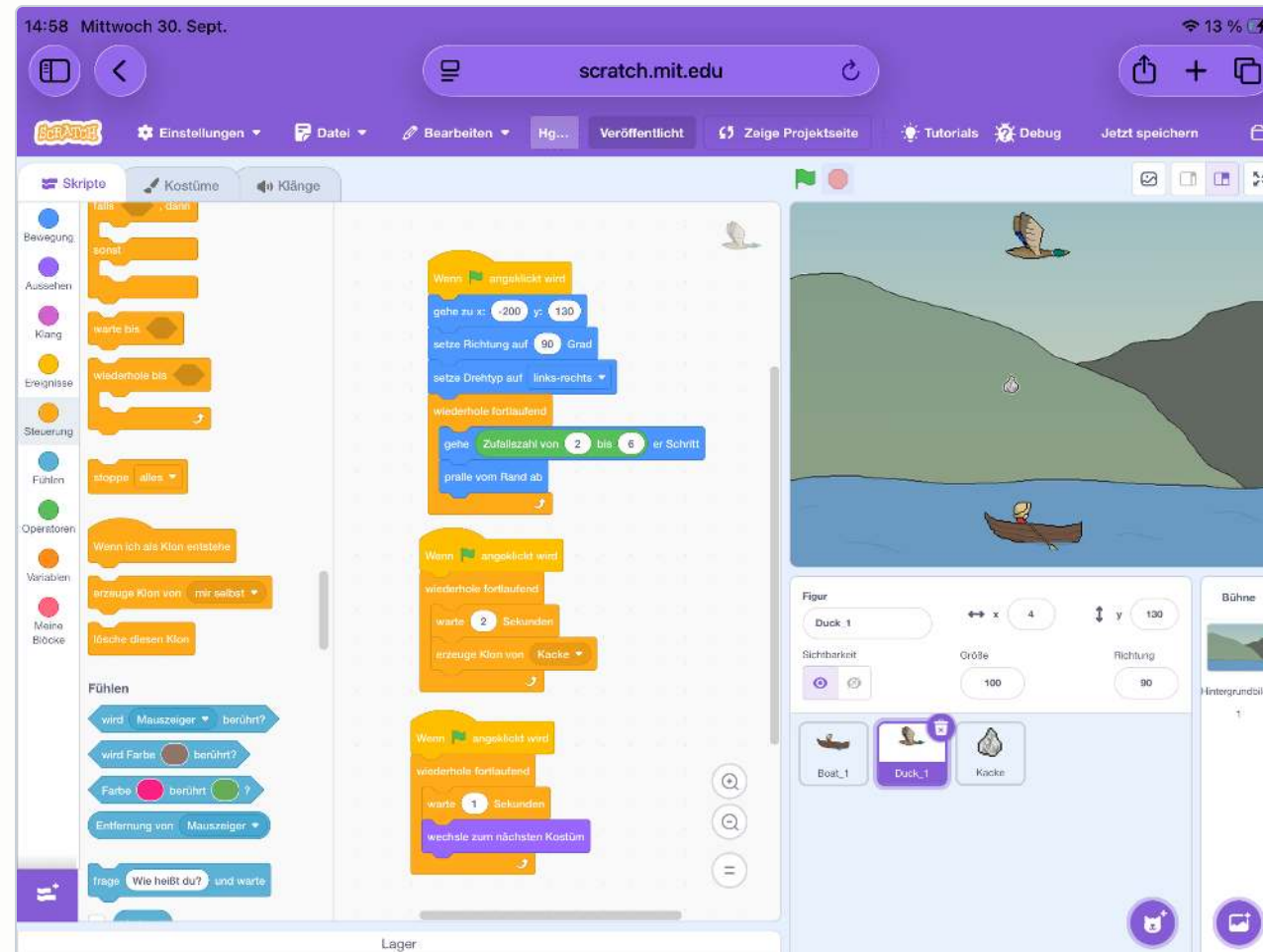
Die **Ente** soll regelmäßig eine Kacke fallen lassen.

So geht das in Scratch

Ein **drittes** Skript bei der Ente: Schleife, warten, „**erzeuge Klon von [Kacke]**“.
Die Ente darf das — obwohl sie nicht die Kacke ist.

WAS?

Die Ente hat jetzt drei Skripte



WARUM?

Der Stempel bleibt in der Schublade

Das Original soll **niemand sehen**. Nur die Abdrücke.

So geht das in Scratch

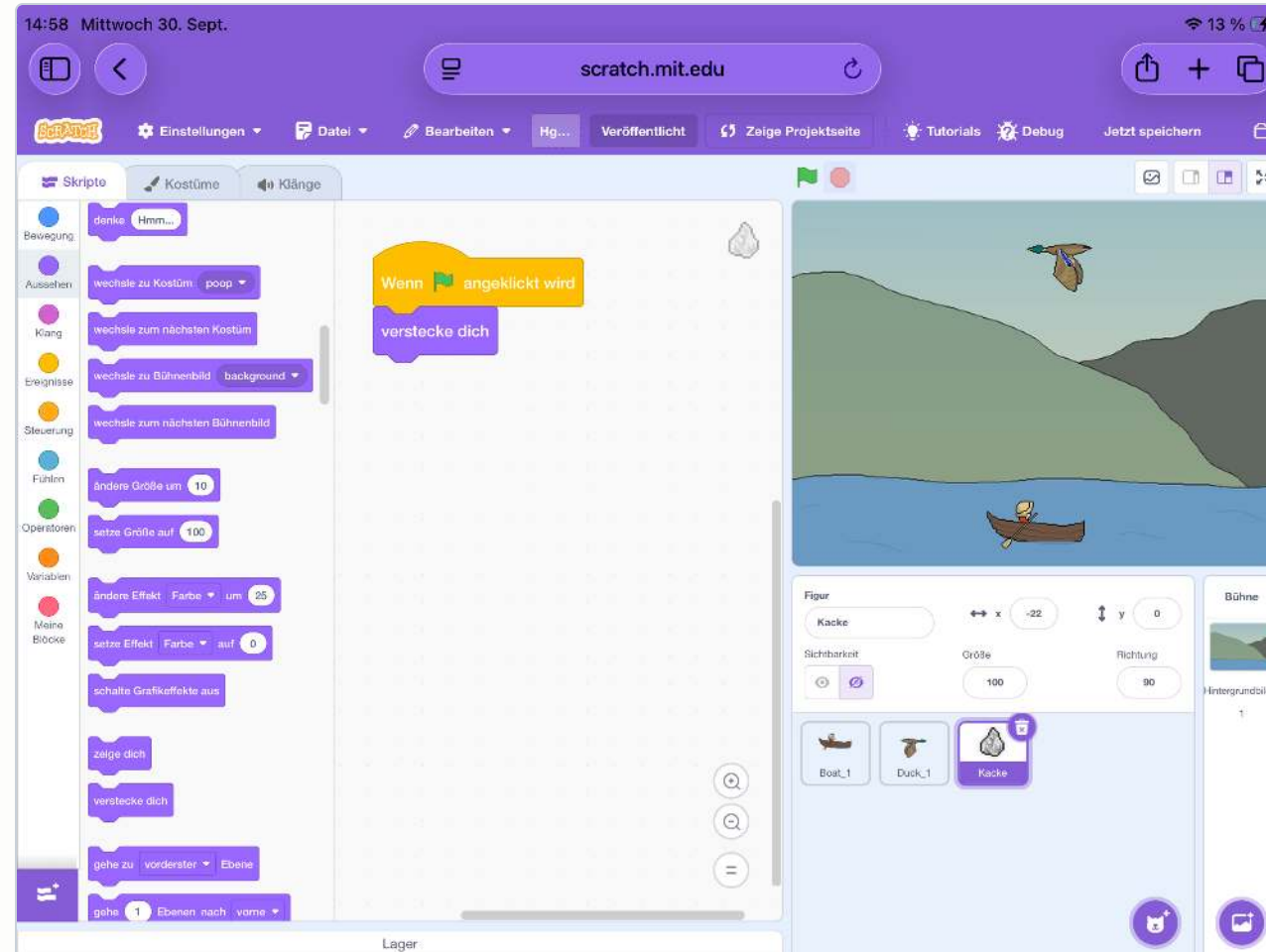
Figur **Kacke**, erstes Skript:

```
Wenn  angeklickt wird  
verstecke dich
```

Ereignisse → Hut-Block, **Aussehen** (lila) → „verstecke dich“. Grüne Flagge: Kacke weg. Gut so.

WAS?

Versteckt



WARUM?

Der Klon weiß noch nichts

Er soll bei der Ente **auftauchen**, nach unten **fallen** und unten **verschwinden**.

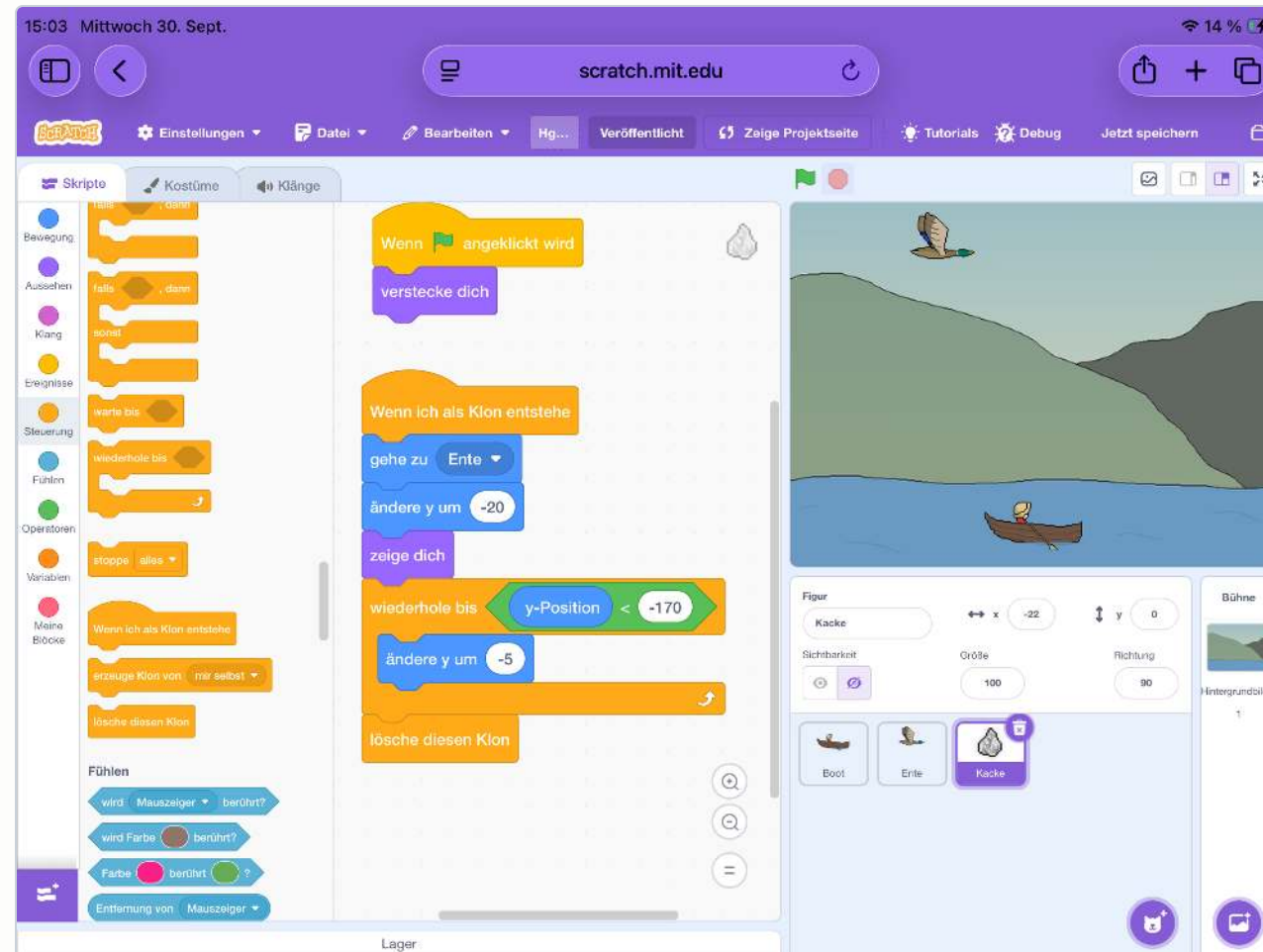
WIE?

Das Leben eines Klons

1. „**Wenn ich als Klon entstehe**“ — der Hut nur für Klone
2. „**gehe zu [Ente]**“ + „**zeige dich**“ — der Klon erbt das Versteckt-Sein, deshalb zeigen
3. „**wiederhole bis < ... >**“ — fallen, **bis** eine Frage mit Ja beantwortet wird
4. Die Frage: „**(y-Position) < (-170)**“ — „Bin ich fast ganz unten?“
Der grüne „<“-Block vergleicht zwei Zahlen
5. „**lösche diesen Klon**“ — **nach** der Schleife, nicht drin

WAS?

Das fertige Klon-Skript



MERKE

Klone

- Die **Ente** stempelt. Jeder Klon startet **sein eigenes** Skript „Wenn ich als Klon entstehe“.
Fünf Klone = fünf Skripte gleichzeitig.
- Der Klon **erbt alles** vom Original: Kostüm, Position — auch das Versteckt-Sein.
- „lösche diesen Klon“ trifft nur den einen. Die anderen fallen weiter.
- „<“ ist eine Frage: „Ist links kleiner als rechts?“ Antwort wahr oder falsch.
- „**wiederhole bis**“ wiederholt, **bis** die Antwort wahr ist.

MERKE

Warum nicht der Rand-Block?

Naheliegender wäre `wiederhole bis <wird [Rand] berührt?>`.

Funktioniert hier nicht: Die Ente berührt beim Abprallen den Rand — und der frisch

erzeugte Klon sitzt genau dort. Er würde sofort gelöscht.

Deshalb fragen wir die **y-Position** ab, mit dem grünen „<“-Block.

WAS?

Checkliste Block 5

- ☐ Ich kann in einem Satz erklären, was ein Klon ist.
- ☐ Die Ente erzeugt alle 2 Sekunden einen Klon der Kacke.
- ☐ Das Original der Kacke ist versteckt.
- ☐ Die Kacke-Klone fallen von der Ente nach unten und verschwinden.
- ☐ Ich habe auf „Jetzt speichern“ getippt.

Block 6

Treffer & Leben

WARUM?

Das Spiel muss sich etwas merken

Wie viele Leben hat das Boot noch? **Fünf. Nach einem Treffer vier.**
Irgendwo muss diese Zahl stehen — und sich ändern lassen.

WIE?

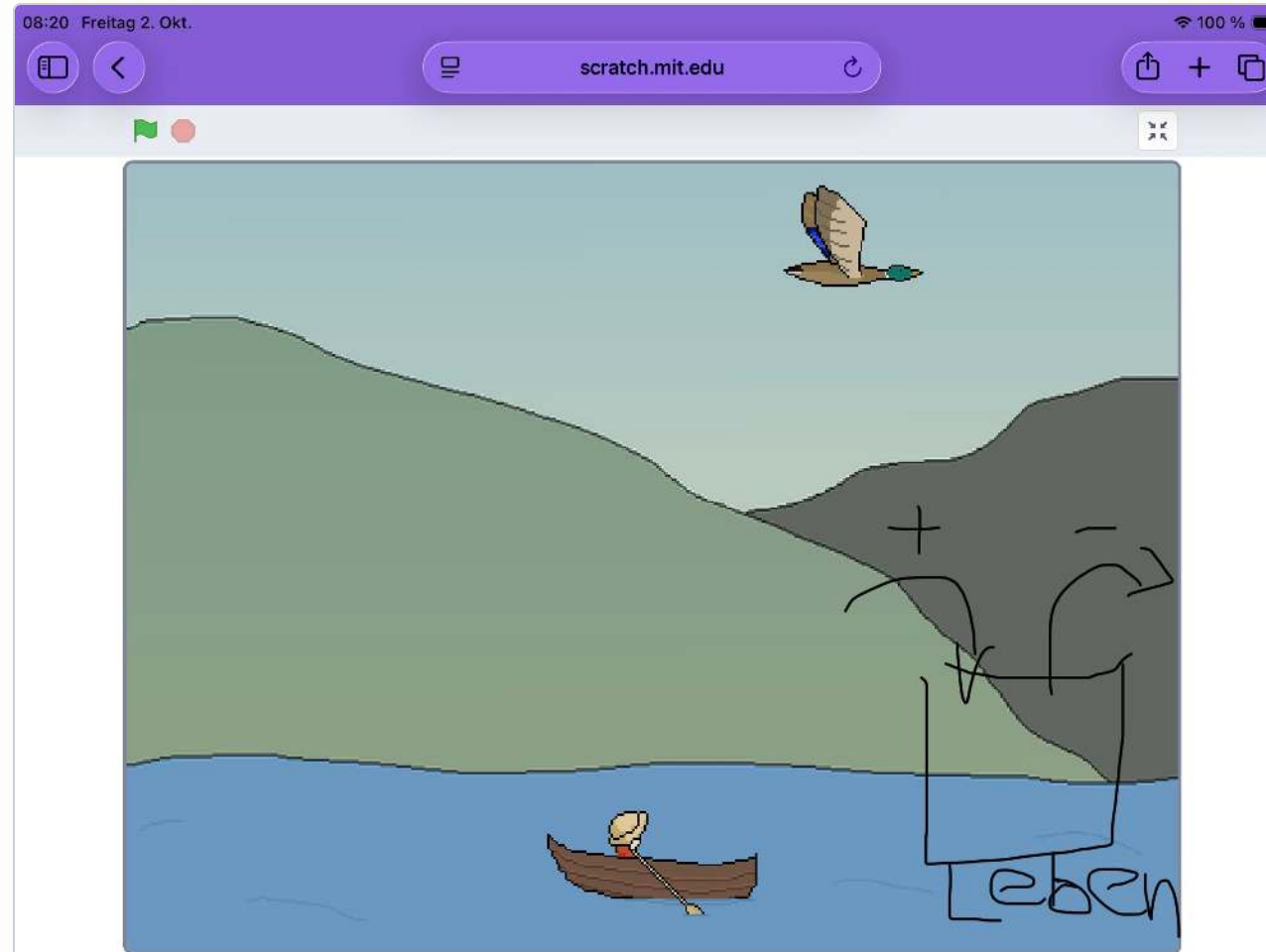
Eine Variable ist ein beschrifteter Becher

Auf dem Etikett steht der **Name** („Leben“), im Becher liegt eine **Zahl** (5).

Block (Variablen, dunkelorange)	Was er tut
„setze (Leben) auf (5)“	Becher leeren, 5 hineinlegen — am Spielstart
„ändere (Leben) um (-1)“	eins herausnehmen — bei jedem Treffer
„(Leben)“ (rund)	nachsehen, was drin ist — zum Vergleichen

WIE?

Der Becher „Leben“



WARUM?

Die Bühne ist der Spielleiter

Sie zählt die Leben und beendet später das Spiel. Deshalb legen wir die Variable **bei der Bühne** an.

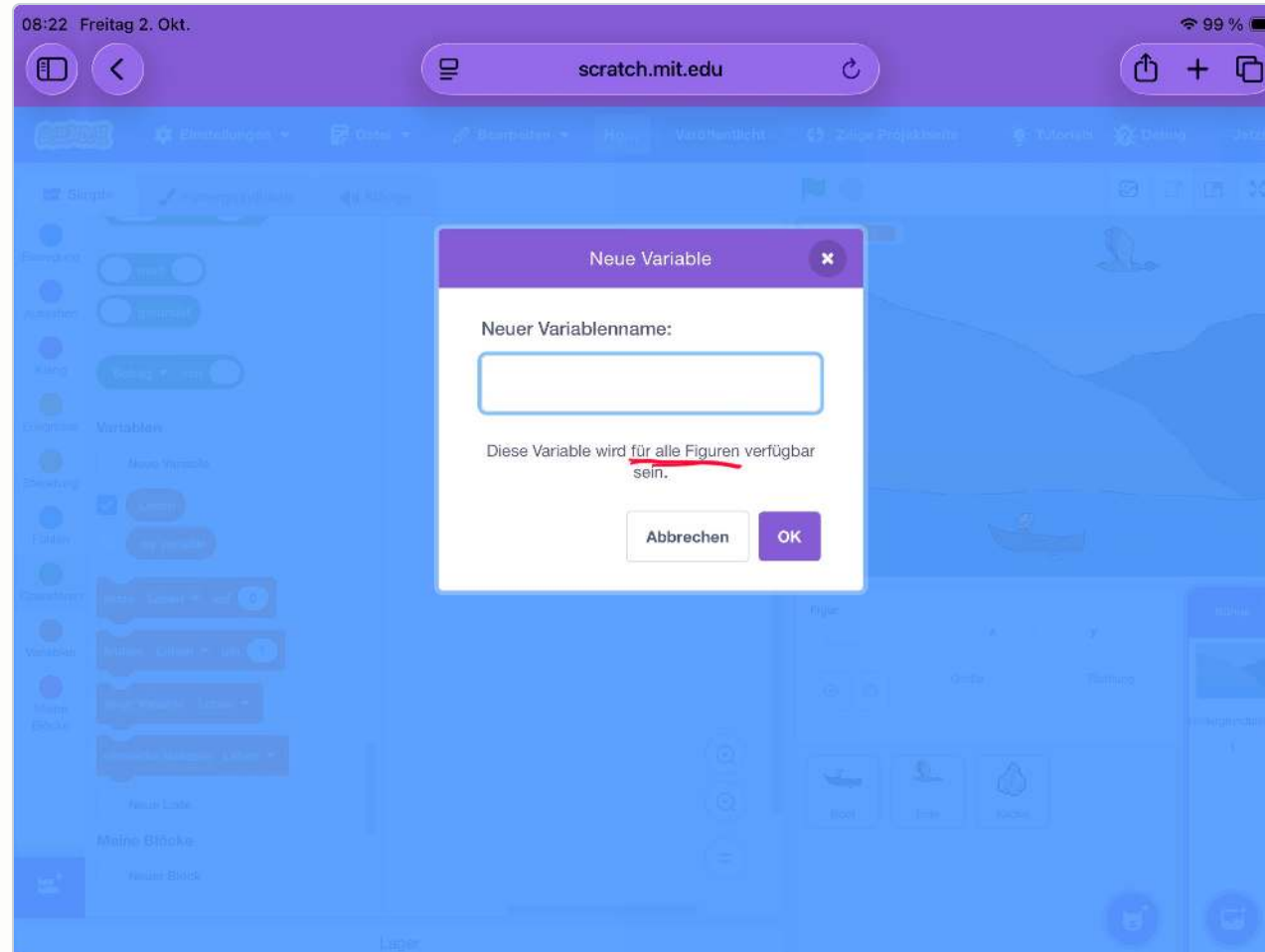
So geht das in Scratch

Rechts neben den Figuren die **Bühne** wählen → **Variablen** → „**Neue Variable**“ → **Leben** .

„**Für alle Figuren**“ muss ausgewählt sein. Sonst sehen Boot und Kacke den Becher nicht.

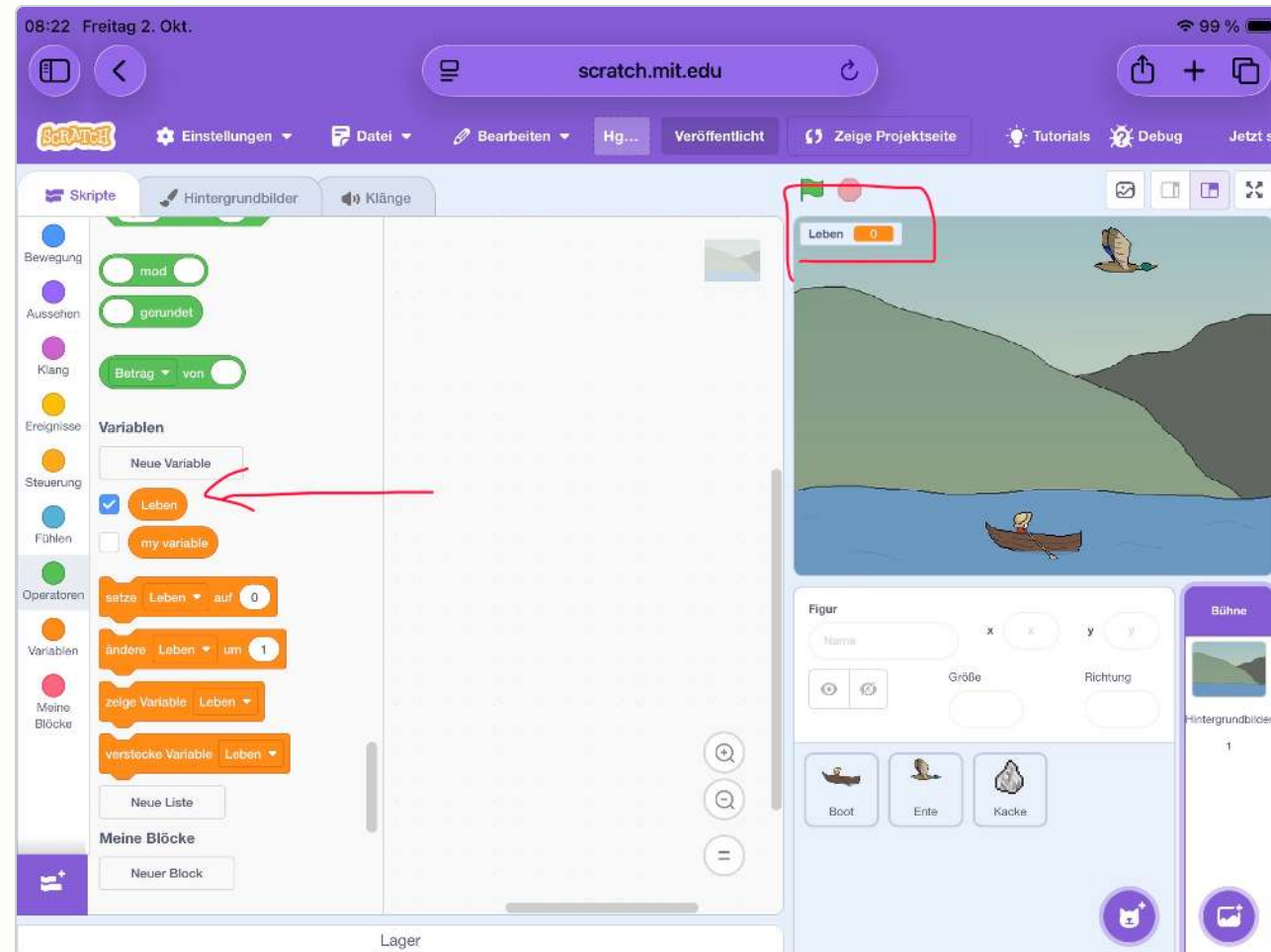
WAS?

Neue Variable



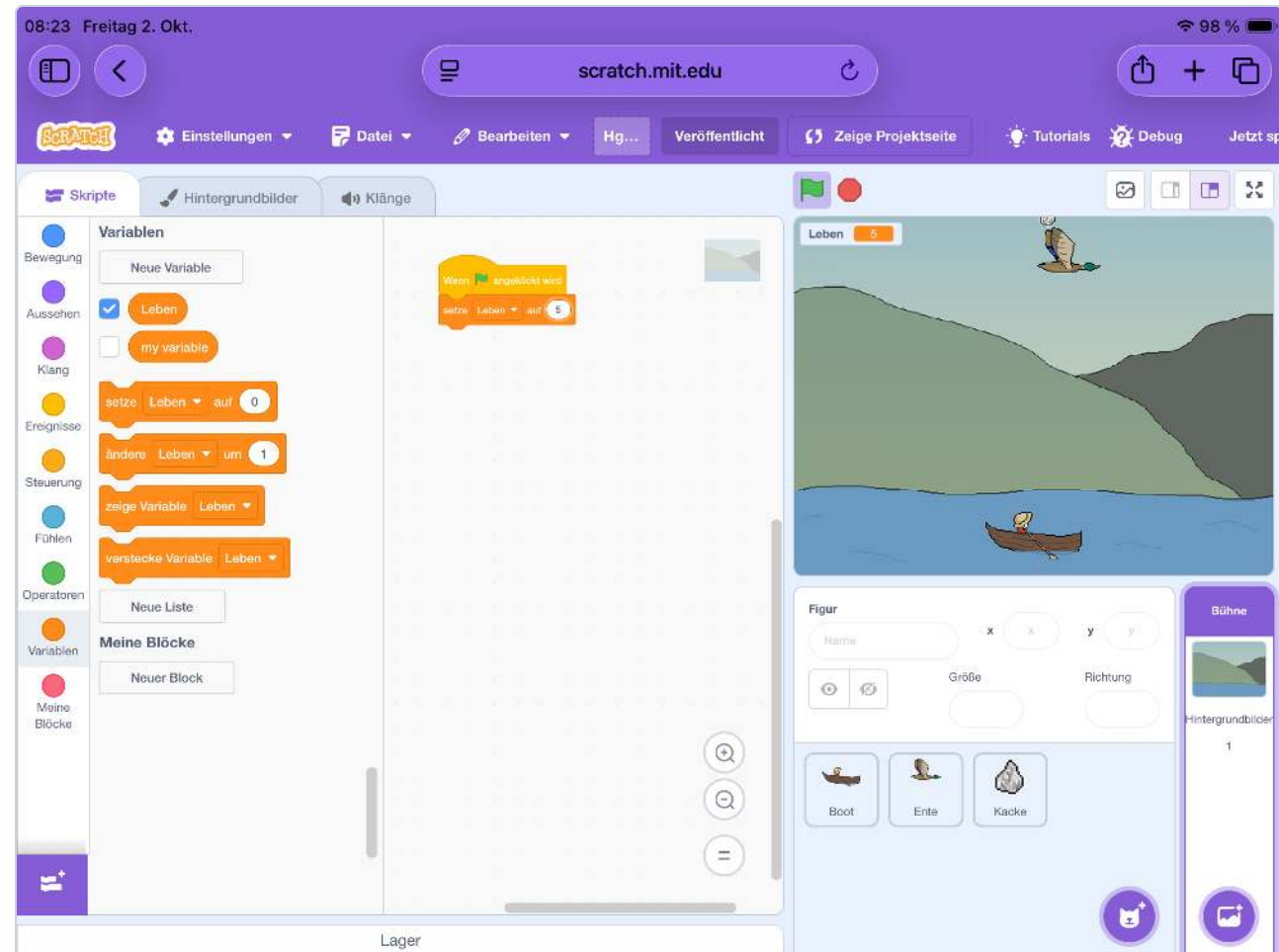
WAS?

Das Häkchen zeigt die Variable auf der Bühne



WAS?

Leben auf 5



WARUM?

Bei Berührung: aufhören und melden

Bis jetzt fällt die Kacke **durch das Boot hindurch**.

Sie soll stoppen, wenn sie das Boot berührt — und **Bescheid sagen**.

WIE?

Zwei Gründe zum Aufhören

Die Schleife soll stoppen, wenn die Kacke unten ist **oder** das Boot berührt:

„< > **oder** < >“ (Operatoren) — eine der beiden Antworten muss Ja sein.

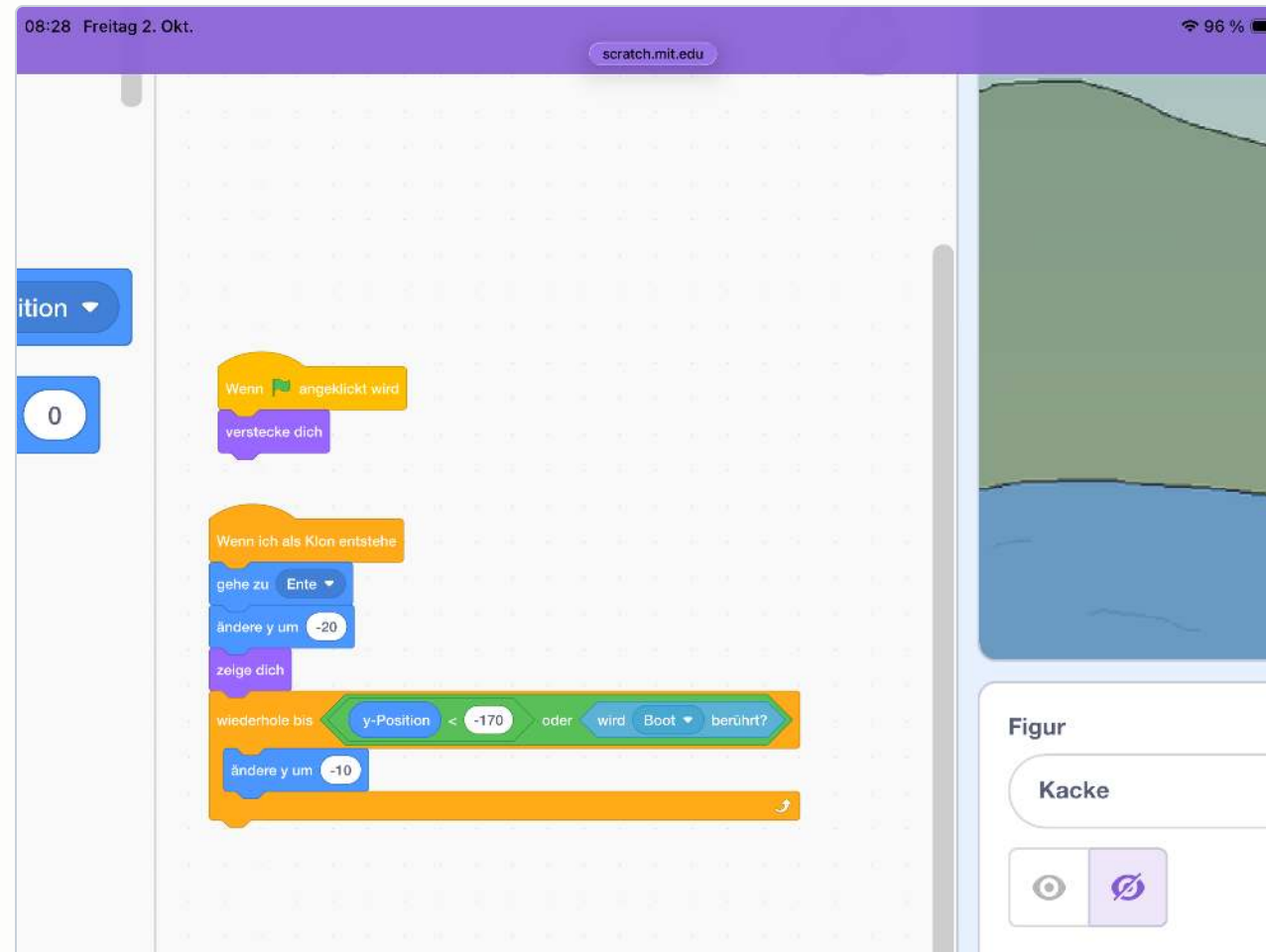
„**wird (Boot) berührt?**“ (Fühlen) — die zweite Frage.

Und nach der Schleife: **falls** das Boot berührt wurde → „**sende (Treffer) an alle**“ (Ereignisse).

Das ist ein Ruf, den jede Figur hören kann.

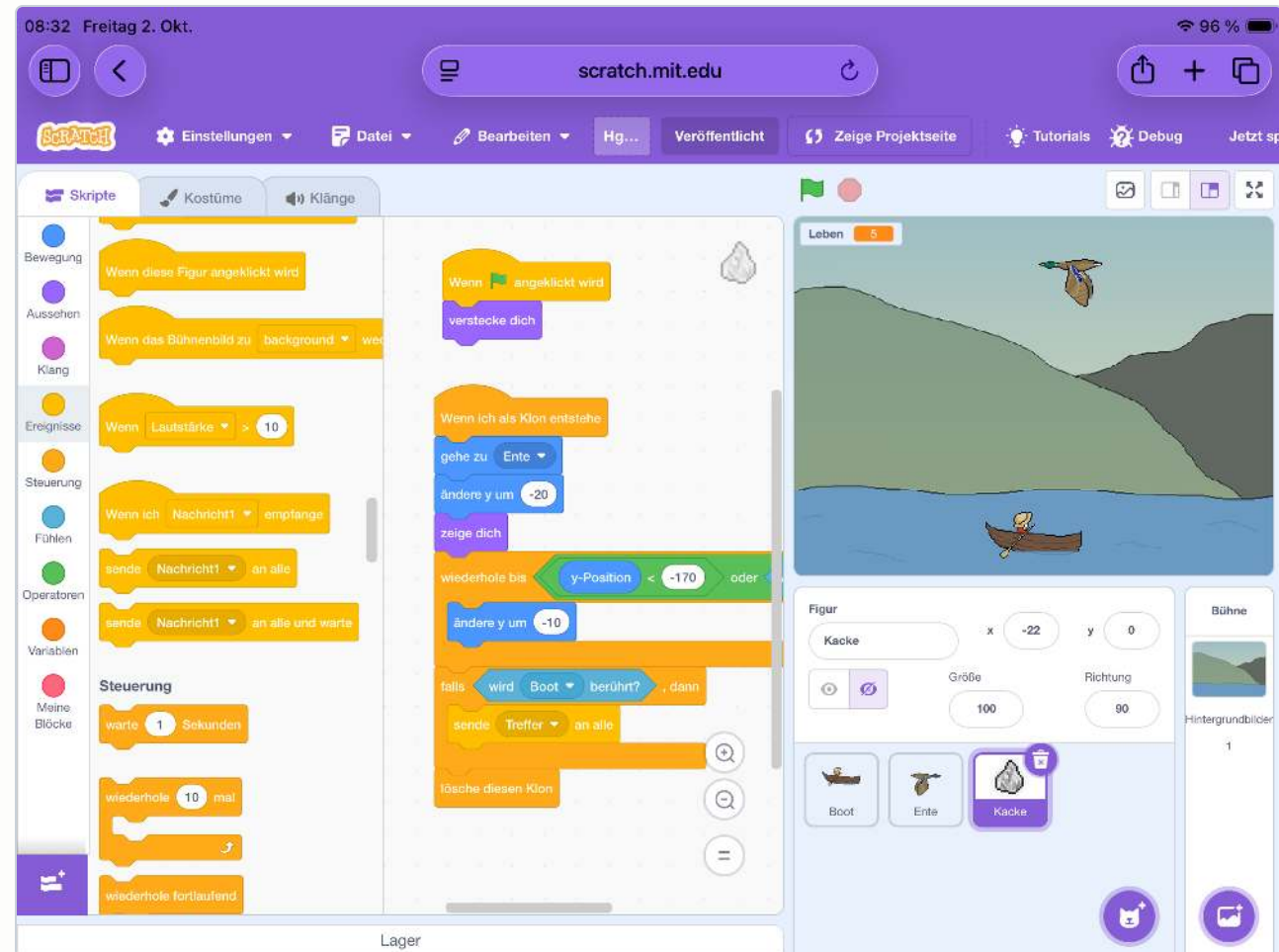
WAS?

Die oder-Bedingung



WAS?

Das Klon-Skript mit Treffer



WARUM?

Warum der Umweg über die Nachricht?

Die **Kacke** merkt den Treffer — aber das Blinken betrifft das **Boot**, und später vielleicht ein Geräusch oder ein Schild.

Mit einer Nachricht sagt die Kacke nur „**Treffer!**“.
Was dann passiert, entscheidet das Boot.

Nachrichten trennen Figuren voneinander. Das macht große Projekte erst beherrschbar.

WIE?

Die Lautsprecher-Durchsage

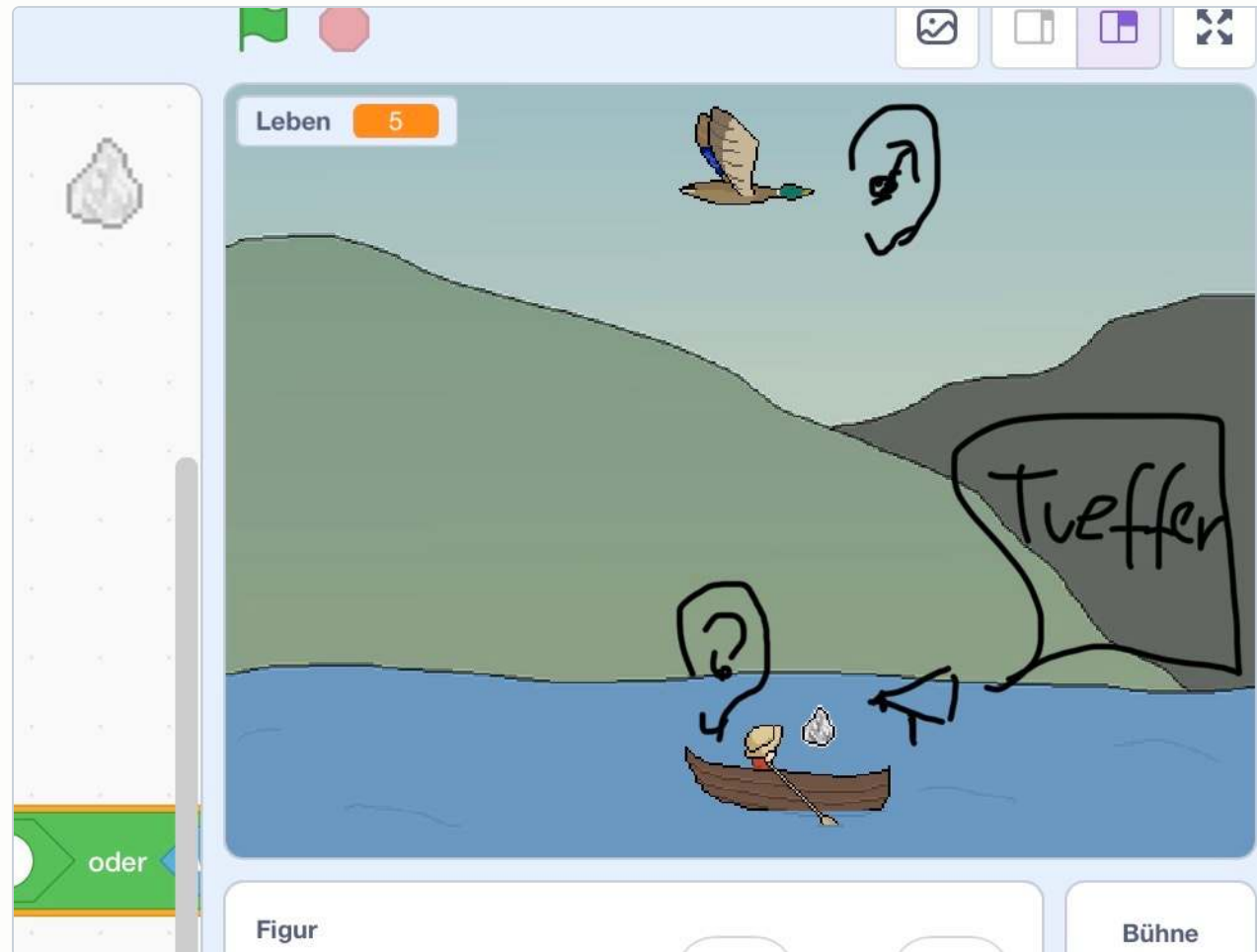
Wie im Schwimmbad: „Die Person mit dem roten Handtuch bitte zum Ausgang.“
Alle hören es. Nur die mit dem roten Handtuch steht auf.

- **„sende (Treffer) an alle“** = Durchsage machen
- **„Wenn ich (Treffer) empfangen“** = Hut-Block: „Ich höre zu und tue dann etwas“

Beide bei **Ereignisse** (gelb), ganz unten.

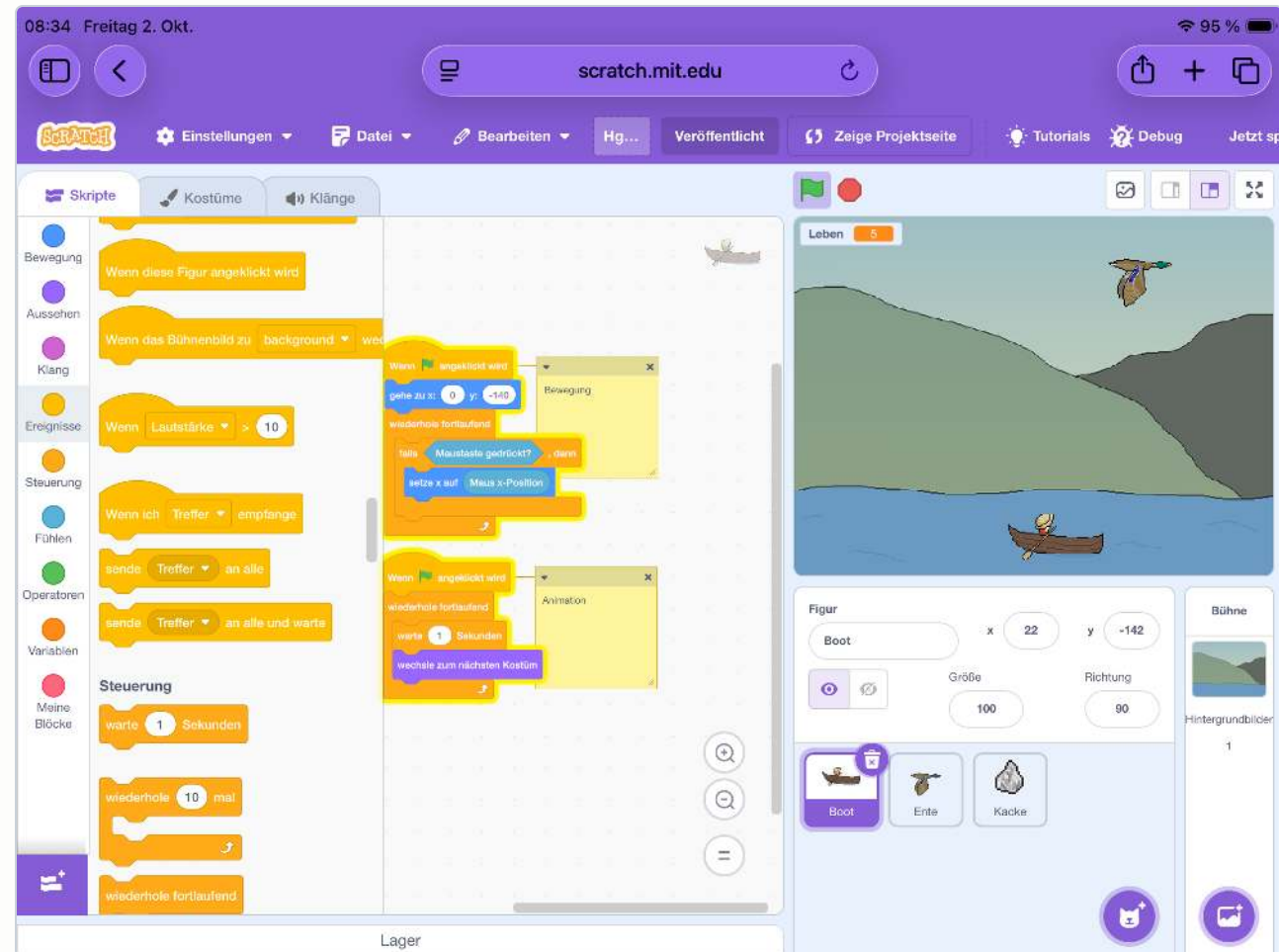
WIE?

Die Durchsage



WIE?

Senden und empfangen in der Palette



WARUM?

Das Boot hört „Treffer“

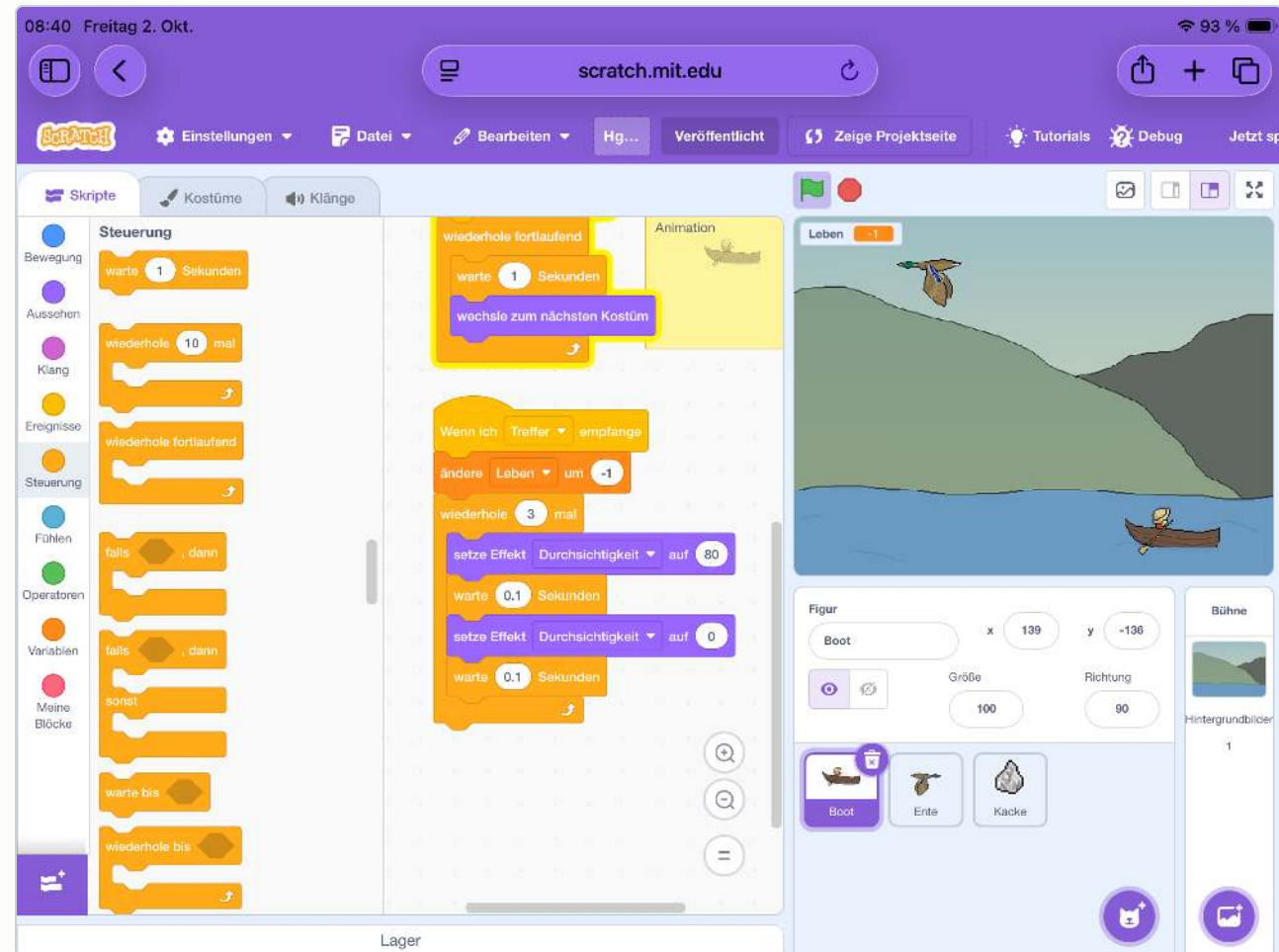
Dann: ein Leben weniger — und das Boot **blinkt** dreimal.

So geht das in Scratch

Hut-Block **„Wenn ich (Treffer) empfange“**, darunter **„ändere (Leben) um (-1)“**.
Blinken = dreimal **durchsichtig** und wieder sichtbar: **„setze Effekt (Durchsichtigkeit)“** (Aussehen).

WAS?

Das Treffer-Skript des Boots



WARUM?

Bei 0 Leben ist Schluss

Das Boot sagt „**Game Over!**“, alle Kacken verschwinden, alles stoppt.

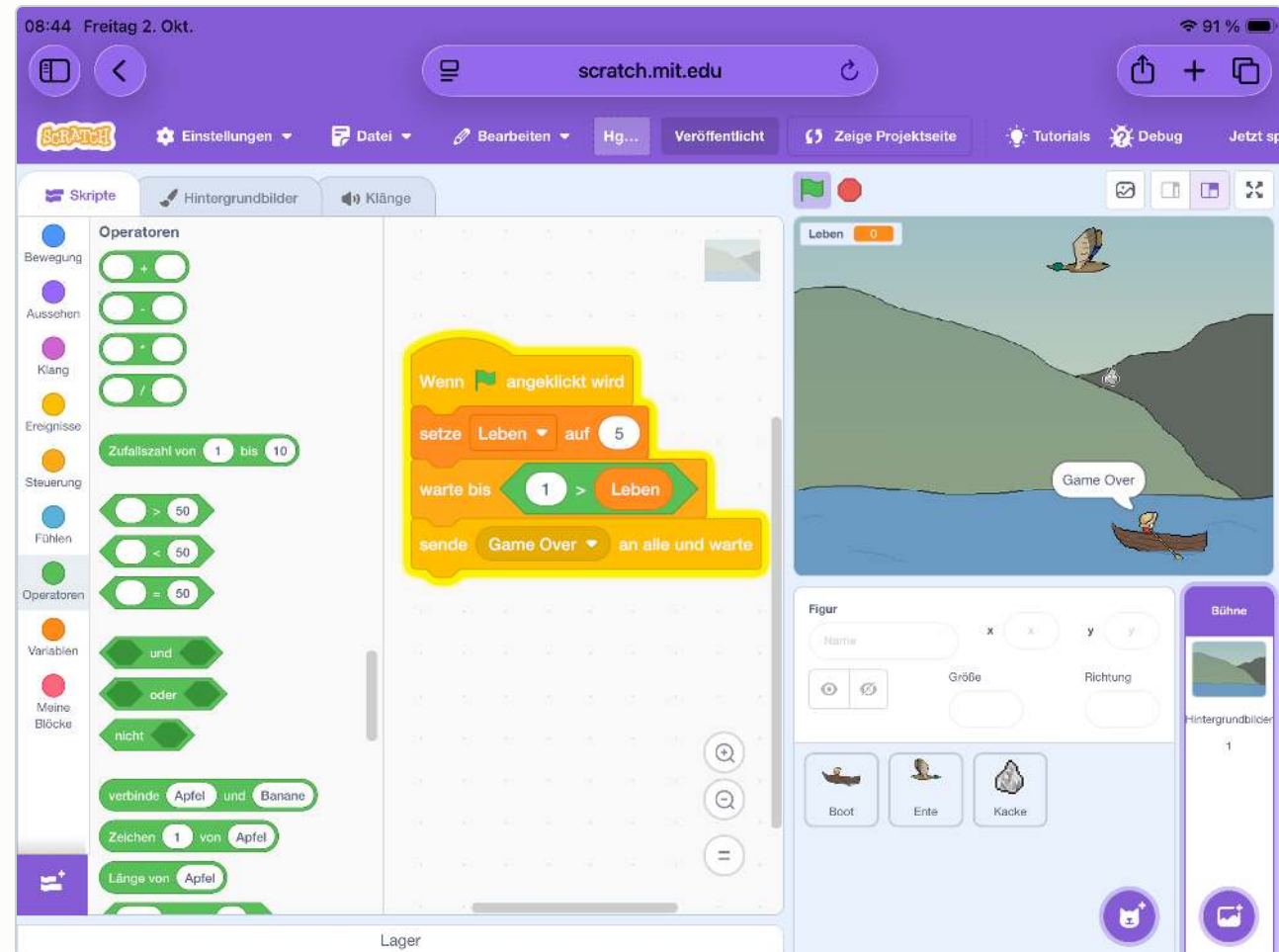
WIE?

Die Bühne wartet und ruft

- **„warte bis < ... >“** (Steuerung) hält das Skript an, bis die Frage Ja ist:
„(Leben) < (1)“
- Dann eine zweite Durchsage: **„sende (Game Over) an alle“**
- **Boot** und **Kacke** hören zu — jede Figur mit eigenem Hut-Block „Wenn ich (Game Over) empfangen“
- **„stoppe (alle)“** beendet alle Skripte wie das rote Stoppschild

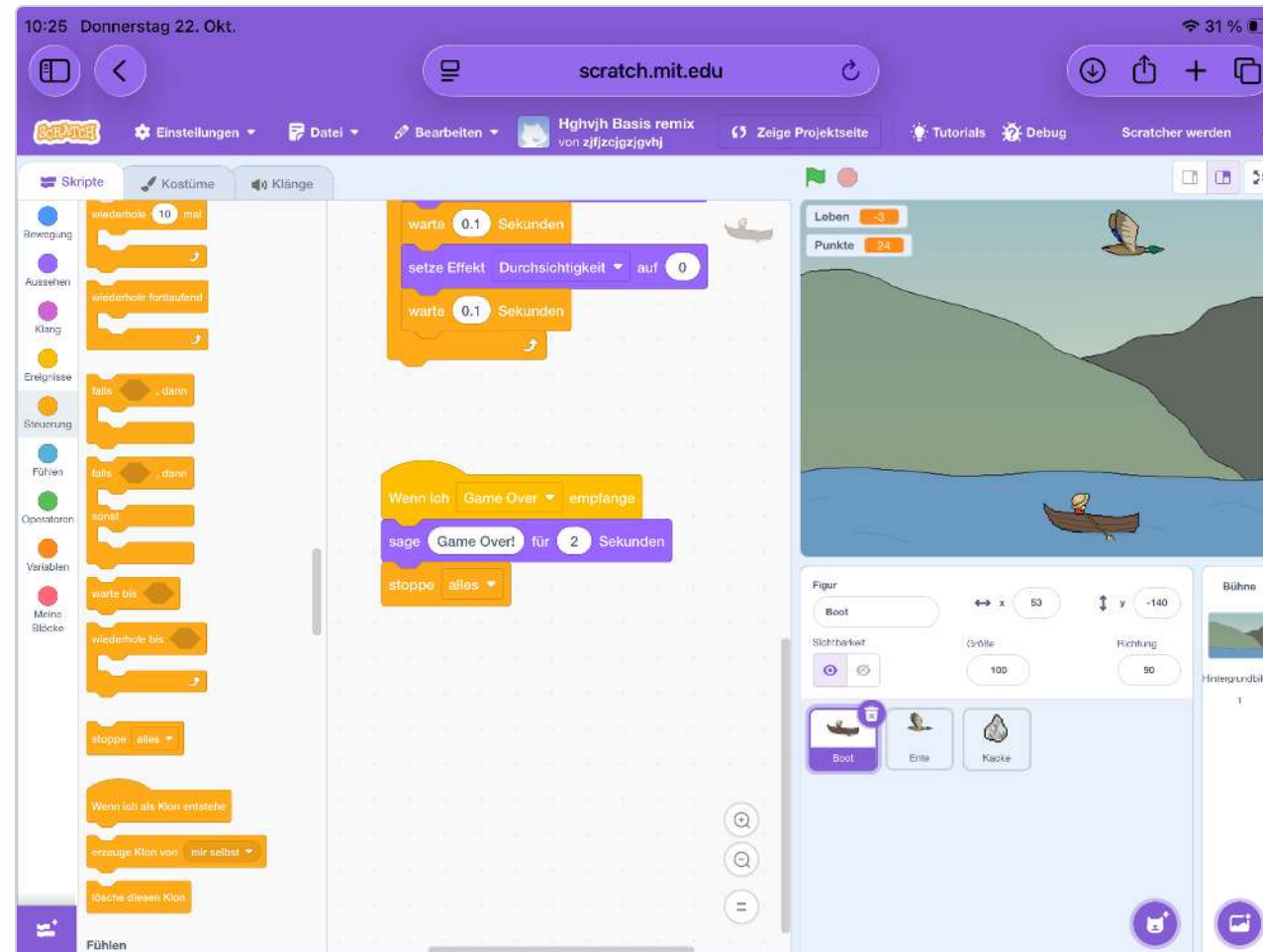
WAS?

Die Bühne wartet auf 0 Leben



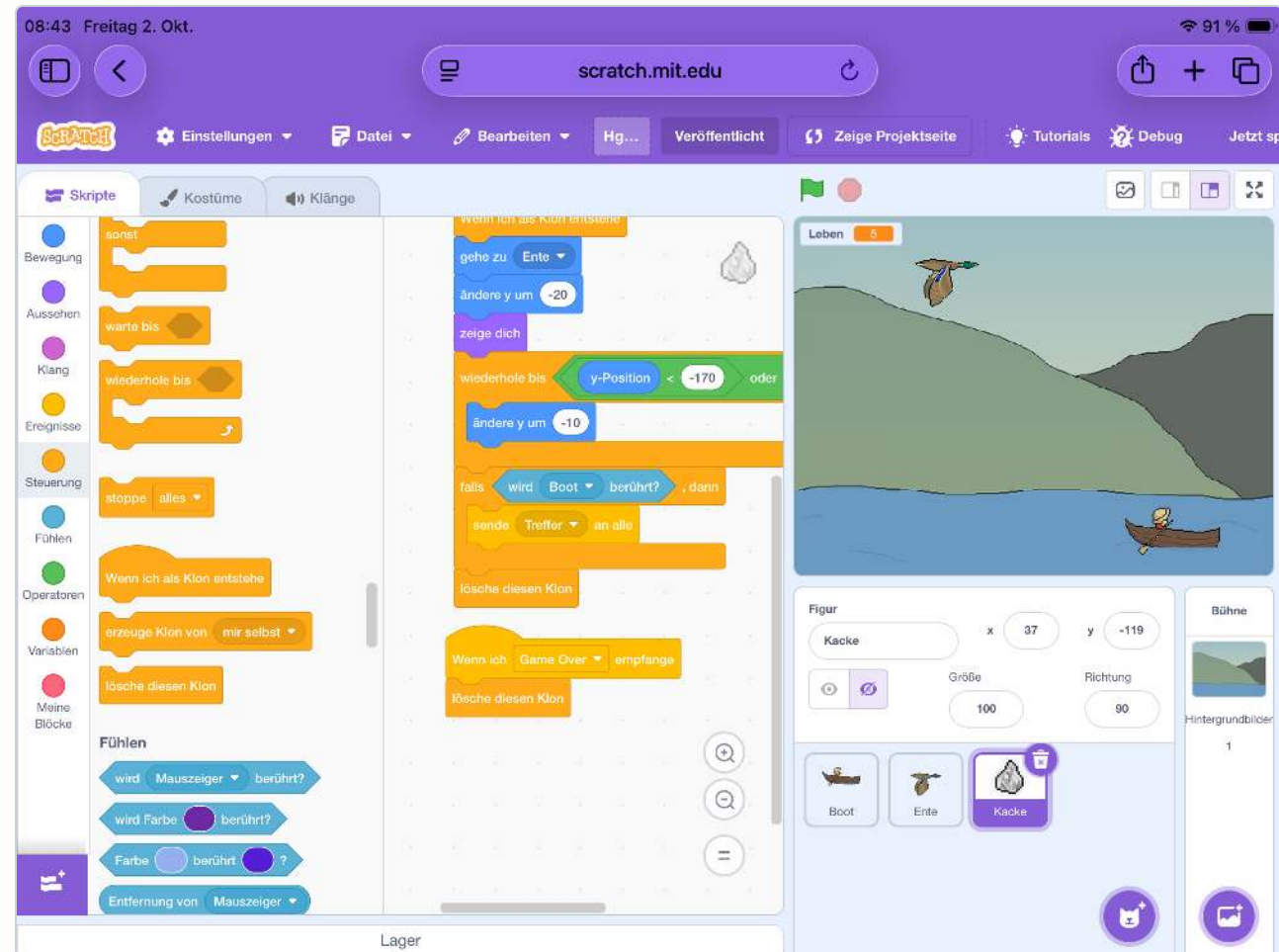
WAS?

Das Boot hört zu



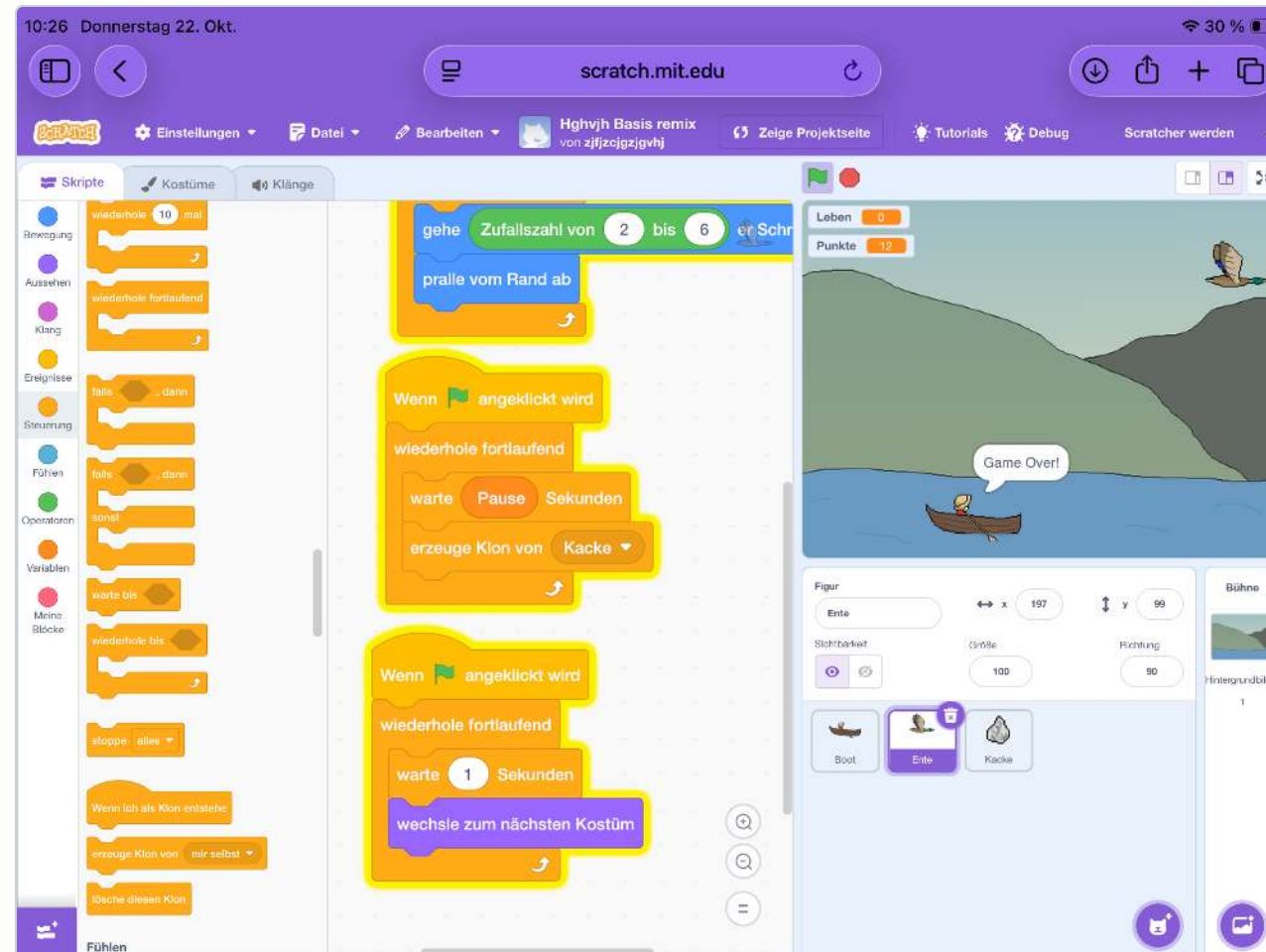
WAS?

Die Kacke hört zu



WAS?

Game Over!



MERKE

Becher, oder, Durchsage

- **Variable „Leben“:** ein Becher, in den alle Figuren schauen dürfen. Die Bühne füllt ihn, das Boot nimmt bei jedem Treffer 1 raus, die Bühne prüft ständig, ob er leer ist.
- **„oder“:** aufhören, wenn unten **oder** Boot berührt.
- **Nachrichten** koppeln Figuren locker: Die Kacke ruft nur. Das Boot regelt den Rest.
- **„warte bis“** blockiert, bis die Bedingung wahr ist.
- **„stoppe alle“** = rotes Stoppschild.

WAS?

Checkliste Block 6

- ☐ Die Variable „Leben“ ist bei der Bühne angelegt, „Für alle Figuren“, und wird auf der Bühne angezeigt.
- ☐ Am Start steht Leben auf 5.
- ☐ Trifft eine Kacke das Boot: Blinken, Leben –1.
- ☐ Bei 0 Leben sagt das Boot „Game Over!“ und alles stoppt.
- ☐ Ich habe auf „Jetzt speichern“ getippt.

Pause

10 Minuten

Vorher: „Jetzt speichern“ tippen!

Block 7

Schwieriger mit der Zeit

WARUM?

Die 2 Sekunden stehen fest im Block

Die Ente kackt **alle 2 Sekunden**. Immer. Das Spiel wird nie schwerer.

Am Anfang 2 Sekunden, nach einer Minute nur noch 1, später 0.5 — **während** das Spiel läuft.

WIE?

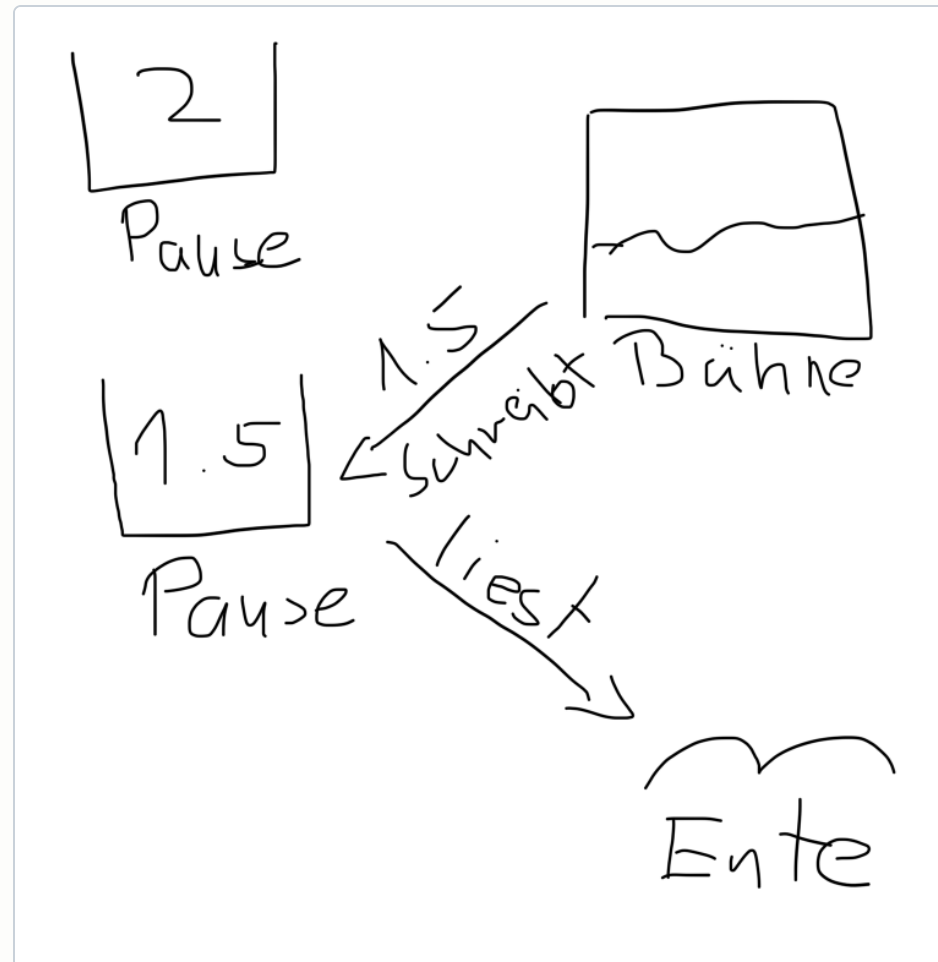
Die Zahl kommt in einen Becher

Variable „**Pause**“. Die Ente liest bei jedem Durchgang nach, was gerade drin liegt. Die **Bühne** dreht nebenbei an der Stellschraube und macht die Pause immer kleiner.

Der runde Block „**(Pause)**“ passt in jedes Zahlenfeld — auch in „**warte () Sekunden**“.

WIE?

Der Becher „Pause“



WARUM?

Jede Sekunde, die du überlebst, ein Punkt

Und im selben Skript dreht die Bühne an der Pause.

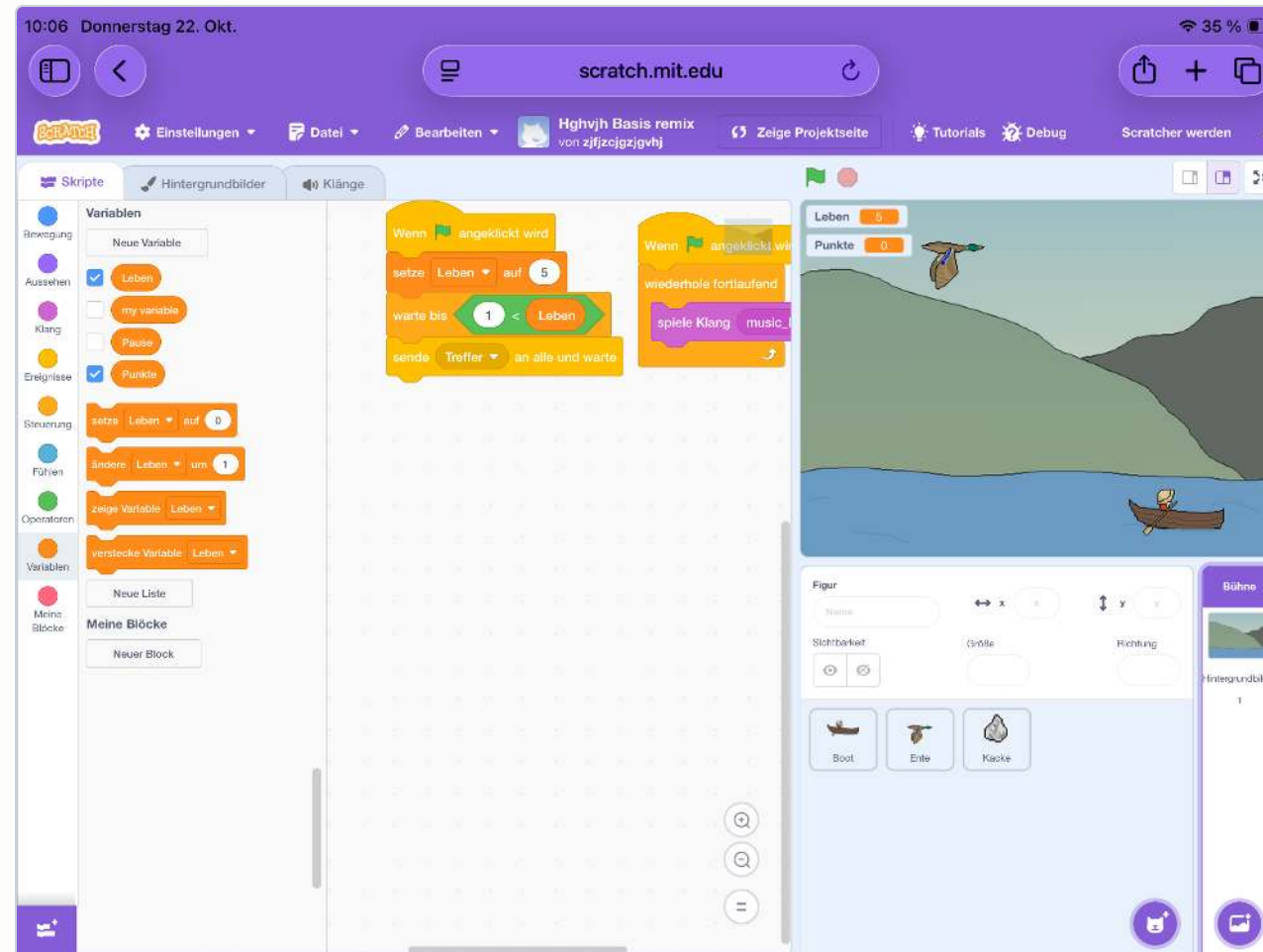
So geht das in Scratch

Zwei neue Variablen bei der **Bühne**, beide „Für alle Figuren“:

- **Punkte** — zählt hoch, Häkchen **an** (soll auf der Bühne stehen)
- **Pause** — Sekunden zwischen zwei Kacken, Häkchen **aus**

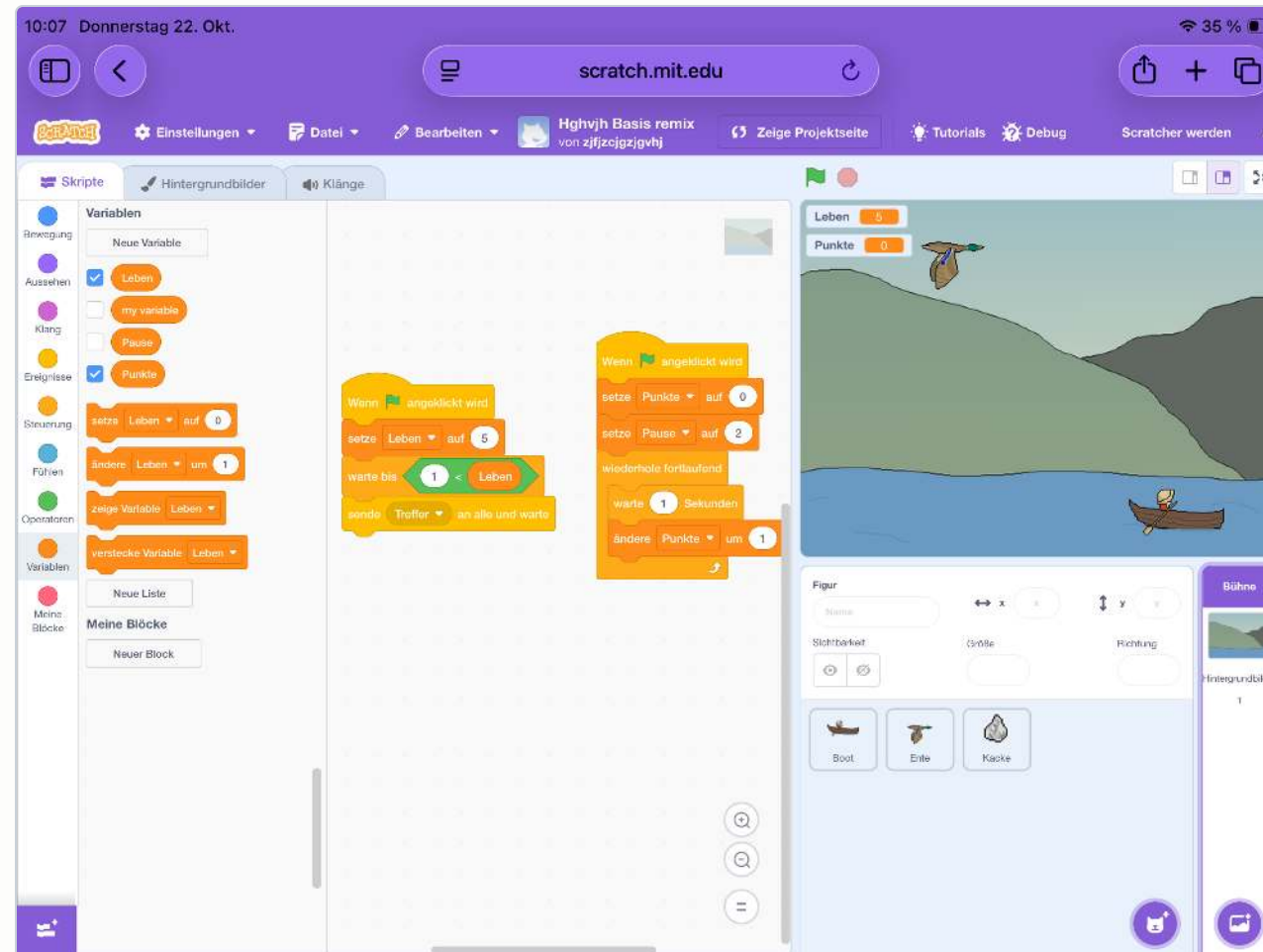
WAS?

Die Variablen-Palette



WAS?

Zwei Skripte auf der Bühne



WARUM?

Zahl raus, Variable rein

Im Kack-Skript der Ente ändern wir **eine Zahl** in eine **Variable**:

warte (2) Sekunden → warte (Pause) Sekunden

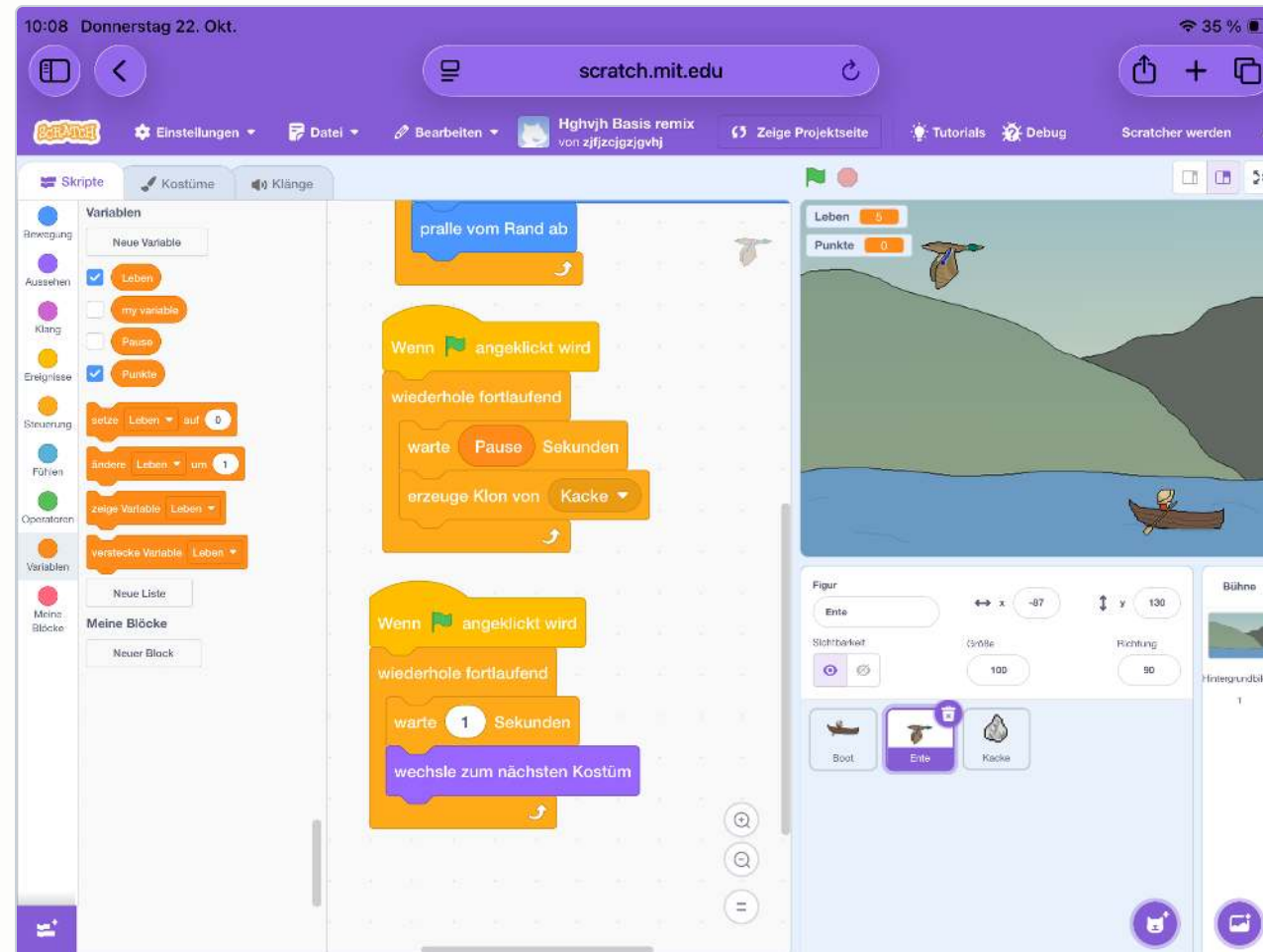
So geht das in Scratch

Variablen → runder Block „**(Pause)**“ ins Zahlenfeld des warte-Blocks ziehen. Die 2 verschwindet.

Grüne Flagge: noch genau wie vorher — die Bühne setzt Pause ja auf 2.

WAS?

„Pause“ im warte-Block



MERKE

Warum die Untergrenze 0.4?

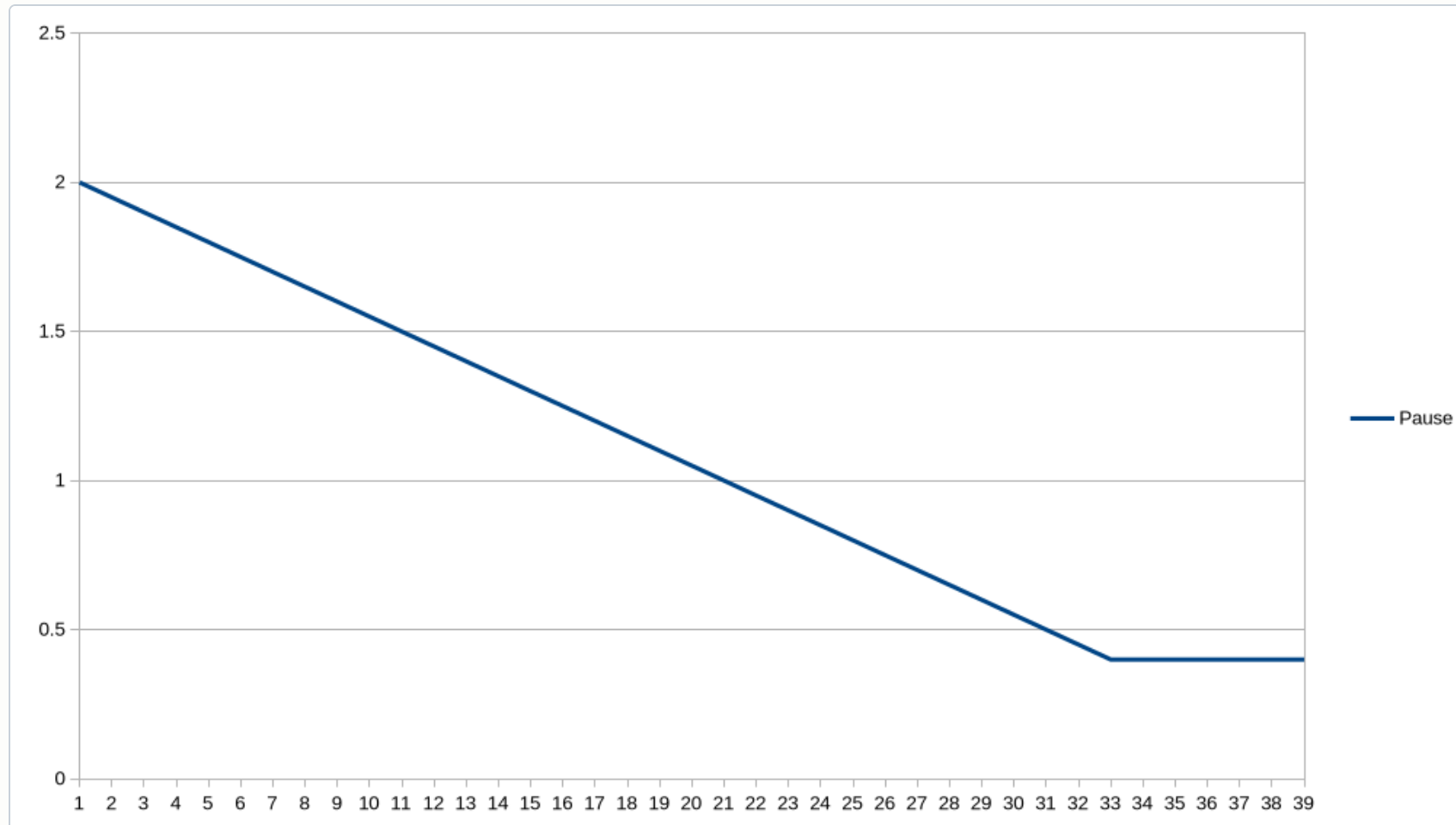
Ohne das „falls“ sinkt die Pause immer weiter: 0.1, 0, dann **negativ**.

Bei „warte –1 Sekunden“ wartet Scratch gar nicht mehr — hunderte Kacke pro Sekunde.

Also: nur kleiner machen, **solange** die Pause über 0.4 liegt. 0.4 ist die maximale Schwierigkeit.

MERKE

Pause über Zeit



MERKE

Die Bühne als Spielleiter

- **Variable statt fester Zahl:** Scratch schaut **jedes Mal neu** nach, was im Becher liegt. Die Ente merkt gar nicht, dass sich etwas ändert.
- **Rechnen:** „ändere Pause um -0.05 “ heißt $\text{Pause} = \text{Pause} - 0.05$. Nach 20 Sekunden: 1.
- Die Bühne hat zwei Skripte: eins zählt Leben, eins zählt Punkte und dreht an der Schwierigkeit.
Die Figuren kümmern sich nur um sich selbst. **Gute Aufteilung.**

WAS?

Checkliste Block 7

- ☐ Die Variable „Punkte“ zählt jede Sekunde hoch und wird angezeigt.
- ☐ Die Variable „Pause“ steuert die Wartezeit der Ente.
- ☐ Die Pause wird jede Sekunde kleiner, aber nie kleiner als 0.4.
- ☐ Ich kann erklären, warum die Untergrenze wichtig ist.
- ☐ Ich habe auf „Jetzt speichern“ getippt.

Block 8

Feinschliff

WARUM?

Ein Spiel ohne Musik ist still

Musik gehört zur **ganzen Bühne**, nicht zu einer Figur.

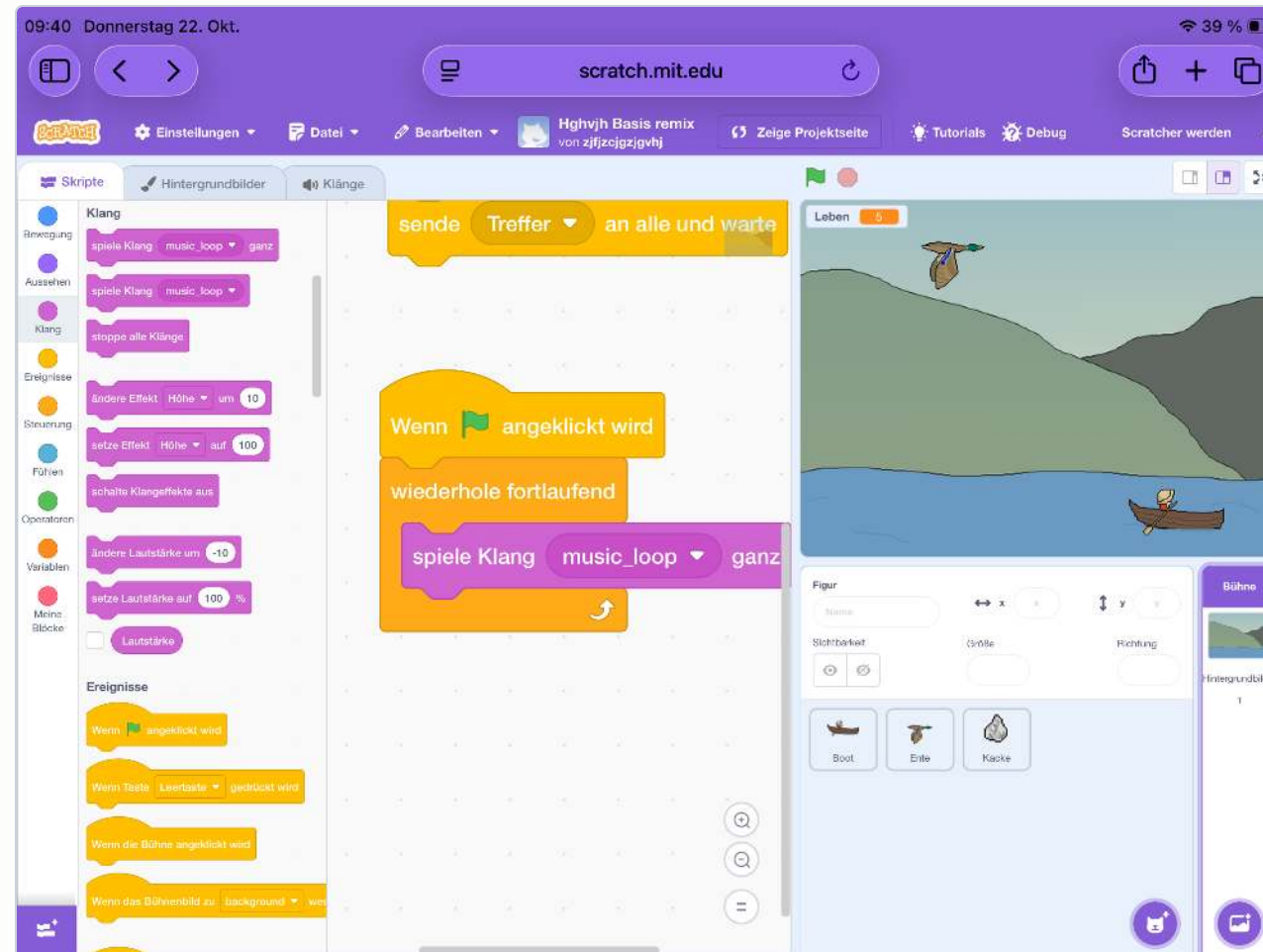
So geht das in Scratch

Die Bühne hat den Klang **music_loop** (Tab „Klänge“). In einer Schleife:
„spiele Klang (music_loop) ganz“ (Klang, pink).

„ganz“ heißt: warte, bis der Klang zu Ende ist. Dann fängt die Schleife ihn neu an.

WAS?

Das Musik-Skript der Bühne



WARUM?

Alle 10 Punkte ein Pling

Bei 10, 20, 30 ... Punkten soll es einen Belohnungs-Ton geben.

WIE?

mod: der Rest beim Teilen

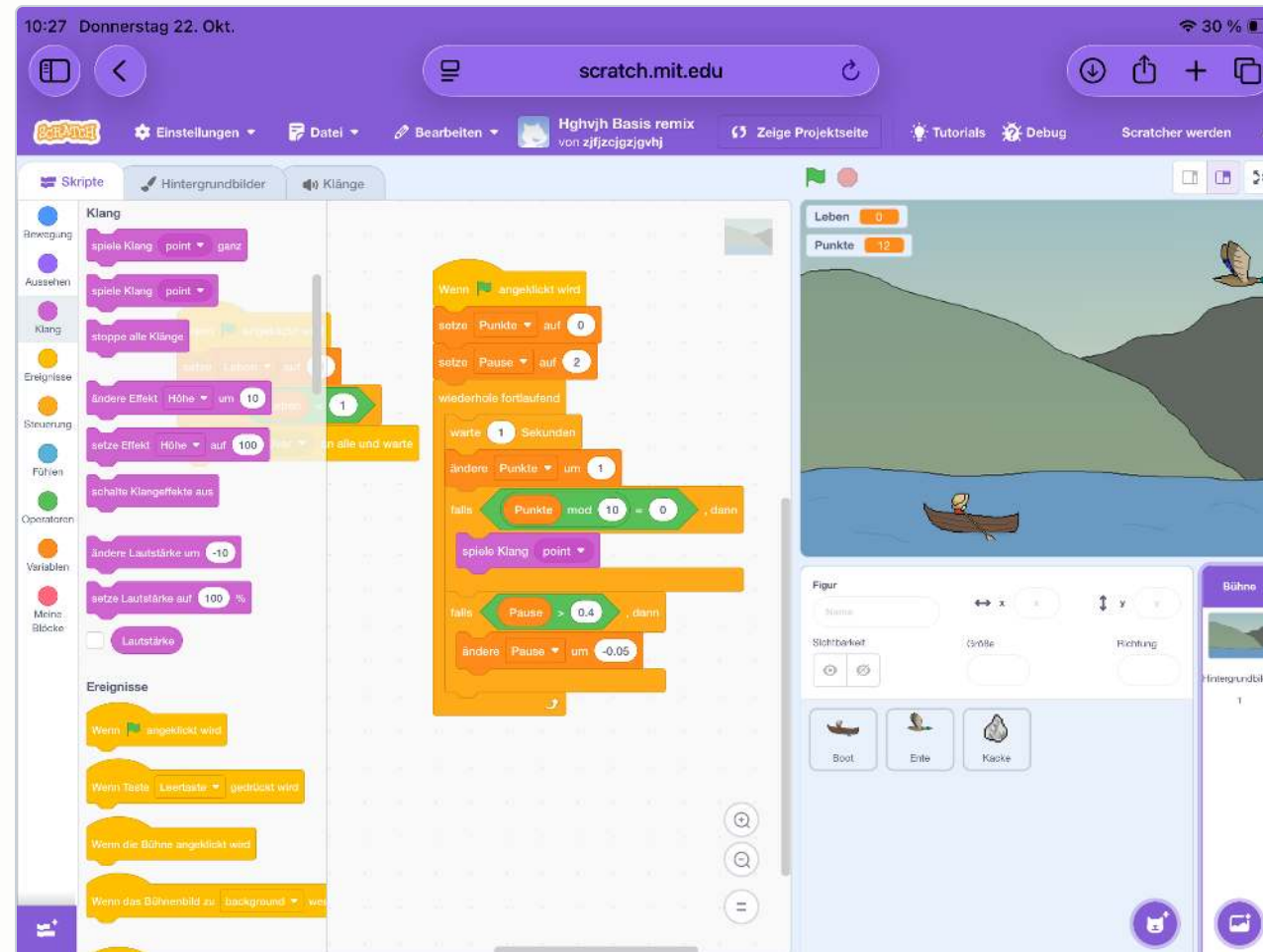
„**() mod ()**“ (Operatoren) rechnet den Rest beim Teilen aus.

Rechnung	Ergebnis
7 mod 10	7
10 mod 10	0
23 mod 10	3
30 mod 10	0

Rest **0** = durch 10 teilbar. Genau dann: „**spiele Klang (point)**“ — hier **ohne** „ganz“.

WAS?

Punkte mod 10



WARUM?

Wie viele Punkte hatte ich?

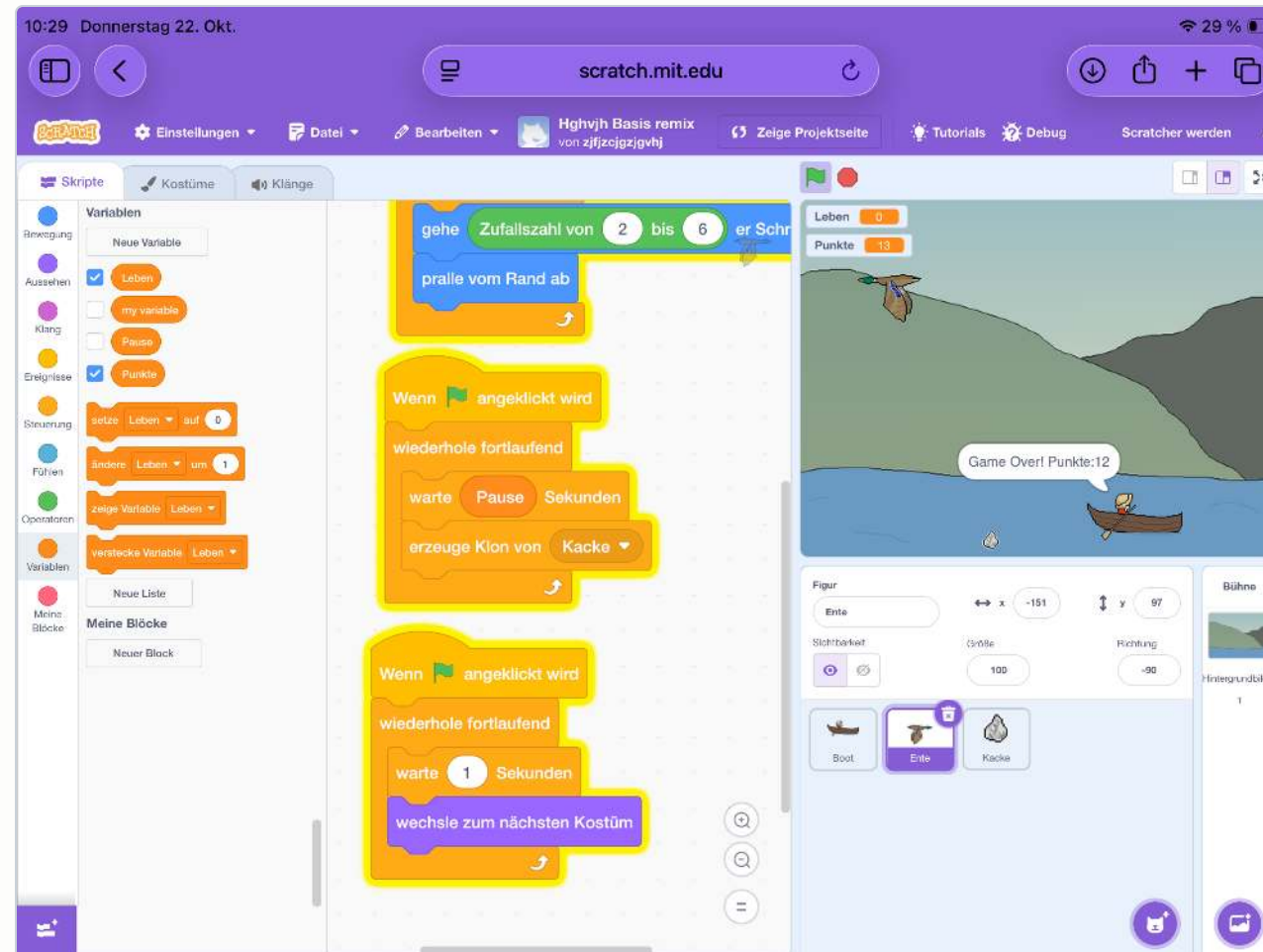
Beim Game Over soll das Boot die Punktzahl sagen.

So geht das in Scratch

„**verknüpfe () und ()**“ (Operatoren) klebt zwei Texte zusammen:
Text „Game Over! Punkte: “ + die Zahl aus dem Becher **Punkte**.

WAS?

Game Over! Punkte: 12



WARUM?

Halb durchsichtig beim Neustart

Startest du nach einem Treffer sofort neu, ist das Boot manchmal noch **halb durchsichtig** oder hat das falsche Kostüm.

WIE?

Der Startzustand

Alles, was bei „Wenn  angeklickt wird" **vor** der Schleife steht, ist der **Startzustand**:

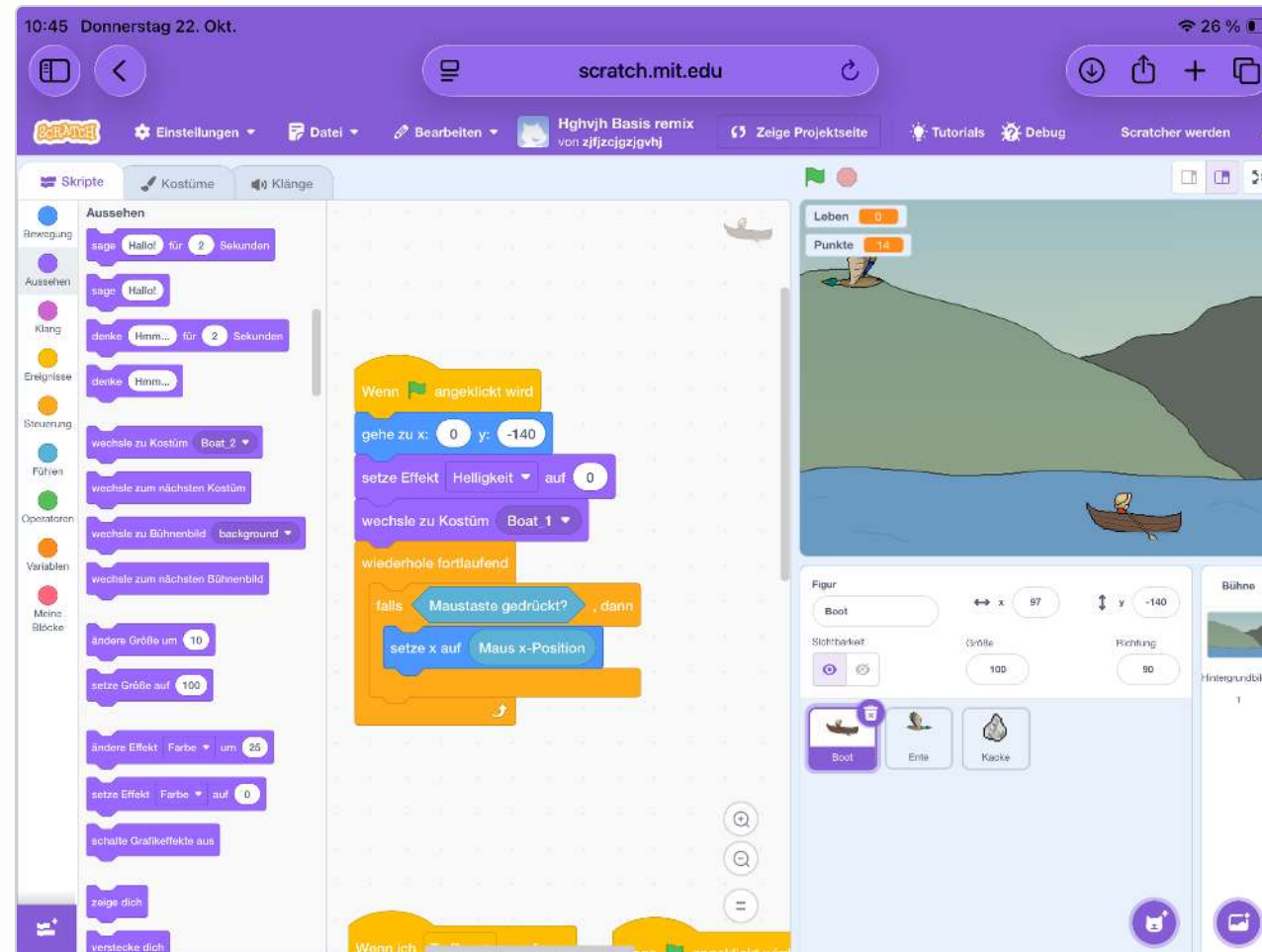
Position, Aussehen, Variablen.

Merke: Alles, was sich im Spiel verändert, muss am Start einmal zurückgesetzt werden.

Sonst schleppst du Reste vom letzten Durchgang mit.

WAS?

Die zwei Reset-Blöcke



WAS?

Checkliste Block 8

- ☐ Die Musik läuft in Dauerschleife.
- ☐ Alle 10 Punkte kommt ein Sound.
- ☐ Beim Game Over zeigt das Boot die Punktzahl.
- ☐ Nach Neustart ist das Boot wieder voll sichtbar und hat Kostüm Boot1.
- ☐ Ich habe gespeichert.

Block 9

Kreativzeit

Jetzt gehört das Spiel dir

Deine Idee

3 Minuten Ruhe. Kein Reden, kein iPad.

Was fehlt deinem Spiel? Schreib es auf deine Ideenkarte:

- die Idee in einem Satz
- der kleinste erste Schritt

Keine eigene Idee? Nimm eine von den Folien.

Ideen ★ — schnell gemacht

- **Eigene Kostüme malen:** Kostüme-Tab → Pinsel, oder Foto mit der Kamera
- **Eigenen Sound aufnehmen:** Klänge-Tab → Mikrofon
- **Treffer hörbar machen:** „spiele Klang (hit)" ins Treffer-Skript des Boots
- **Enten quaken lassen:** Klang „Duck" aus der Scratch-Bibliothek
- **Wolken am Himmel:** zweite Figur, die langsam vorbeizieht

Ideen ★★ — ein bisschen Arbeit

- **Herz-Power-up:** neue Figur, fällt wie die Kacke, berührt das Boot → **Leben +1**
- **Highscore:** Variable, die sich den besten Punktestand merkt
- **Level-Anzeige:** alle 20 Punkte ein Level höher, Ente wechselt die Farbe
- **Kacke fällt schneller:** `ändere y um ((0) - (Tempo))`
- **Schwierigkeit wählen:** „frage ... und warte" am Start → **Pause** setzen

Ideen ★★ — für Mutige

- **Das Boot schießt:** Klone nach oben, treffen die Ente → **Punkte +5**
- **Zweite Ente**, die nach 30 Sekunden dazukommt
- **Pause-Taste:** eigene Figur, die alle anderen Skripte anhält

Nimm **eine** Idee. Bau sie fertig. Erst dann die nächste.

So gehst du vor

1. **Idee aussuchen** — eine, nicht drei
2. **Kleinsten Schritt zuerst:** Figur anlegen, sichtbar machen, bewegen
3. **Nach jedem Schritt testen** — grüne Flagge
4. **Speichern**
5. Klemmt es? Schau in ein Skript, das schon funktioniert — vieles ist gleich

WAS?

Checkliste Block 9

- ☐ Ich habe mir eine Idee ausgesucht.
- ☐ Mindestens ein Teil davon funktioniert im Spiel.
- ☐ Das Spiel läuft noch ohne Fehler durch (grüne Flagge bis Game Over).
- ☐ Ich habe gespeichert.

Block 10

Vorstellen & Abschluss

Eine Minute für dein Spiel

Zeig uns:

1. **Dein Spiel** — kurz anspielen
2. **Eine Sache, die du dazugebaut hast**
3. **Eine Sache, die schwer war**

Wer will, zeigt sein Spiel

Vom **eigenen iPad** aus, auf dem Beamer. Niemand muss.

1. Bühne auf **Vollbild**
2. Vom **rechten oberen Rand** nach unten wischen
3. **Bildschirmsynchronisation** → Apple TV
4. Code von der Leinwand eintippen
5. Danach: **Synchronisation stoppen**

Zu Hause weitermachen

Einloggen -> Datei runterladen -> Neues Konto erstellen -> Datei hochladen

Dein Workshop-Account ist **nur vorübergehend**. Was du behalten willst, ist dann diese Datei.

Drei Fragen, ehrlich

Auf deiner Feedbackkarte — ankreuzen, kein Name:

			
Ich habe verstanden, was ich gebaut habe			
Das Tempo war richtig			
Ich will weitermachen			

Und zwei Sätze: **Was hat am meisten Spaß gemacht? Was war unnötig oder zu schwer?**

WAS?

Checkliste Block 10

- ☐ Ich habe „Jetzt speichern“ getippt.
- ☐ Ich habe mein Spiel vorgestellt, oder ich habe zugeschaut und mir eine Idee gemerkt.
- ☐ Ich weiß, wie ich zu Hause weitermache.

Geschafft

Du hast heute ein Spiel von null auf fertig programmiert

Steuerung · Animation · Klone · Variablen · Nachrichten · Schwierigkeitskurve

Aus genau diesen Bausteinen bestehen auch große Spiele.