Übungsbuch

für Anfängerinnen und Anfänger

CoderDojo LINZ

Übungsbuch

für Anfängerinnen und Anfänger

4. Auflage

Herausgeber: Coding Club Linz e.V., https://linz.coderdojo.net
Umschlagfoto: Rainer Stropek

August 2023

Inhalt

Scratch

Scratch (https://scratch.mit.edu) ist eine Blockprogrammiersprache, die ideal für Einsteigerinnen und Einsteiger ins Coding geeignet ist. Du musst noch nicht gut tippen können, um deine ersten Computerspiele zu bauen. Dieses Buch enthält vier Übungen, bei denen du von Spiel zu Spiel neue Dinge über Scratch lernen wirst. Viel Spaß beim Programmieren!

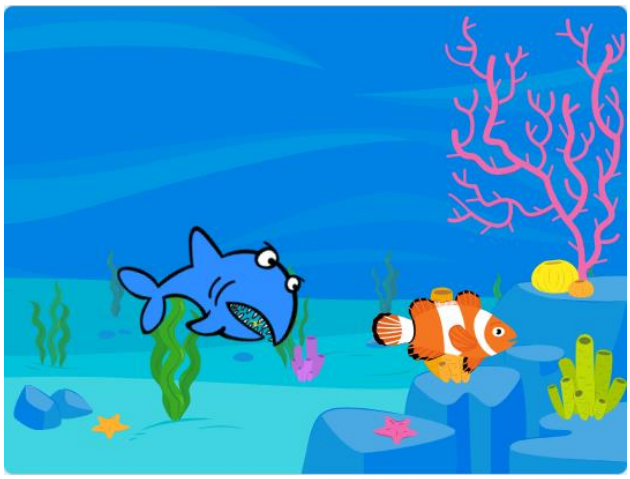
TypeScript

Im zweiten Teil dieses Buches lernst du, wie man textuell programmiert, indem du kleine Spiele mit einer Programmiersprache namens *TypeScript* baust. Im Gegensatz zu Scratch musst du den Code auf der Tastatur eintippen, genauso wie es die Programmier-Profis auch tun. Am Anfang ist es vielleicht eine Herausforderung, die richtigen Tasten zu finden. Du wirst aber sehen, dass du schon nach kurzer Zeit Code flott tippen kannst.

Python

Im dritten Teil geht es weiter mit textueller Programmierung. Diesmal lernst du aber eine neue Programmiersprache kennen: *Python*. Probiere diese Beispiele aus, nachdem du mit *TypeScript* experimentiert hast. Welche gefällt dir besser?

Fang mich - mein erstes Spiel mit Scratch



In diesem Spiel bist du ein kleiner Fisch, der dem großen Haifisch entkommen muss.

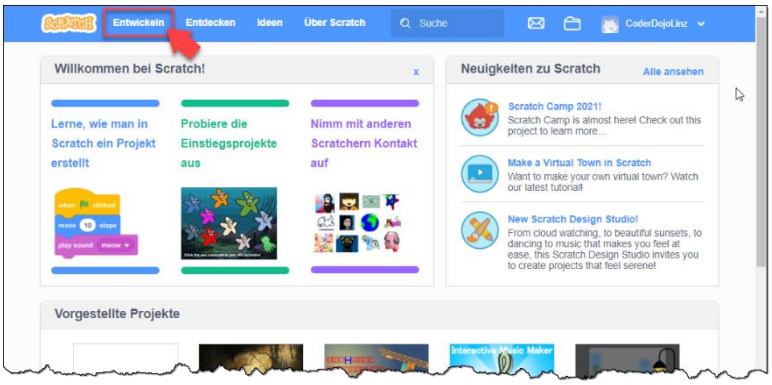
Schaffst du es?

Scratch starten

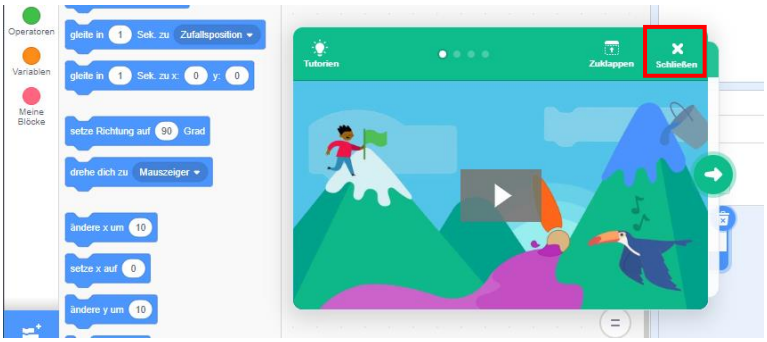
Öffnen einen Browser wie Chrome oder Firefox und öffne die Seite https://scratch.mit.edu.

Falls die Seite englisch angezeigt wird, scrolle ganz nach unten und ändere die Sprache auf Deutsch.

Klicke dann auf den ersten Menüpunkt *Entwickeln*, um mit dem Programmieren zu beginnen.



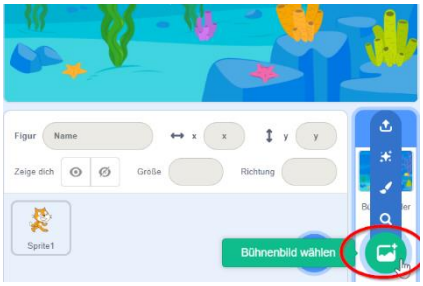
Das grüne Kärtchen mit den Tutorien kannst du schließen.



Bühne und Figuren anlegen

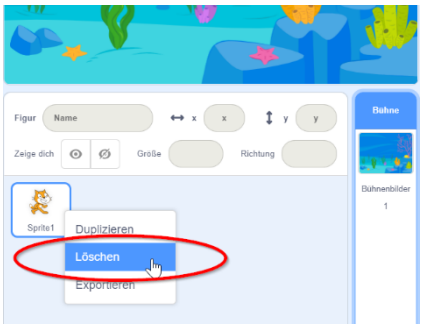
Spielfeld

Als erstes legst du fest, wie dein Spielfeld aussehen soll. Wir brauchen zuerst das Aquarium, in dem die Fische schwimmen. Wähle als links unten unter *Bühnenbild aus der Bibliothek wählen* ein Bühnenbild aus - zum Beispiel ein Aquarium.



Scratchy löschen

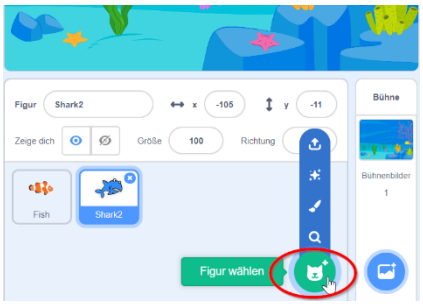
Als nächstes lösche die Figur Scratchy mit dem Namen *Sprite1* oder *Figur1*, indem du mit der rechten Maustaste daraufklickst. Im angezeigten Menü kannst du Scratchy löschen.



Figuren anlegen

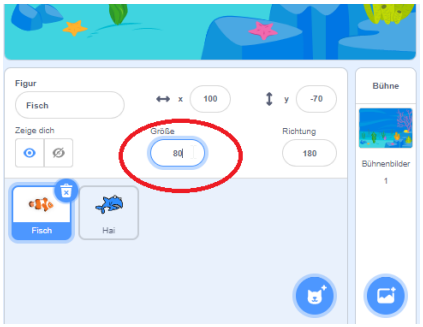
Jetzt brauchen wir einen Haifisch und einen Fisch, mit dem wir dem Haifisch entkommen wollen. Klicke dazu auf *Figur aus der Bibliothek wählen* und füge einen Fisch und einen Haifisch hinzu.

Natürlich können es auch andere Figuren sein, zum Beispiel ein Käfer, der einem Vogel davonläuft.



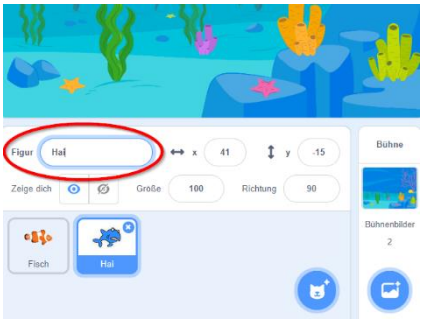
Fisch verkleinern

Damit der kleine Fisch auch kleiner ist als der große Haifisch, müssen wir den Fisch verkleinern. Wähle dazu die Figur aus und ändere die Größe, so dass die Figur gut in dein Bühnenbild passt.

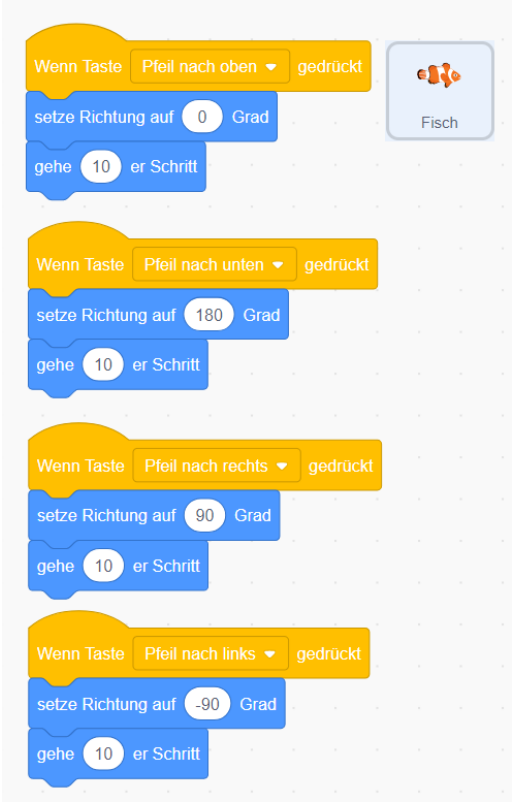


Namen vergeben

Damit du später die Figuren leichter verwenden kannst, gib ihnen Namen wie *Haifisch* und *Fisch*. Du kannst die Eigenschaften von Figuren ändern, indem du auf das blaue *i* links über der Figur klickst.



Fisch bewegen

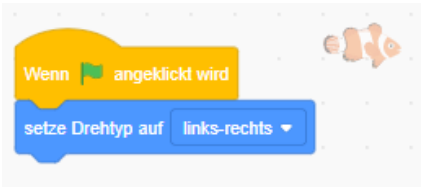


Damit du den Fisch bewegen kannst, musst er nach links und rechts sowie oben und unten bewegt werden können.

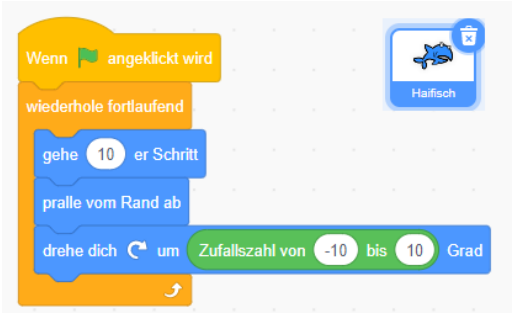
- Wähle zuerst den Fisch aus, damit er blau umrandet ist.
- Im Tab *Skripte* kannst du deinen Fisch nun bewegen. Verwende das Ereignis *Wenn Taste … gedrückt* unter *Ereignisse*.
- Verknüpfe es jeweils mit einer Drehung *setze Richtung auf …* unter *Bewegung*, damit der Fisch in die richtige Richtung schaut.
- Außerdem brauchen wir *gehe …er Schritt*, um den Fisch zu bewegen.

- Achte auf die Einstellungen der Kommandos.
  Für *Pfeil nach oben* gedrückt: Richtung 0 Grad, gehe 10er-Schritte.
  Für *Pfeil nach unten* gedrückt: Richtung 180 Grad, gehe 10er-Schritte.
  Für *Pfeil nach rechts* gedrückt: Richtung 90 Grad, gehe 10er-Schritte.
  Für *Pfeil nach links* gedrückt: Richtung -90 Grad, gehe 10er-Schritte.

- Experimentiere mit der Schrittanzahl. Je größer die Schrittanzahl, desto schneller ist dein Fisch.
- Damit der Fisch nicht auf dem Kopf steht, wenn es sich nach links bewegt, kannst du den Drehmodus ändern



Haifisch bewegen



Jetzt soll der Haifisch im Aquarium herumschwimmen.

- Wähle dazu den Haifisch aus, damit er blau umrandet ist.
- Im Tab *Skripte* kannst du den Haifisch nun bewegen.
- Unter *Ereignisse* wähle *Wenn … angeklickt*.
- Anschließend wähle *wiederhole fortlaufend* bei *Steuerung* aus.
- Bewege den Haifisch mit *gehe 10er-Schritt, pralle vom Rand ab* und *drehe dich um … Grad*
- Um etwas mehr Zufall reinzubringen, nimm im Menü *Operatoren* den Block *Zufallszahl von -10 bis 10* und ziehe ihn an die Stelle der *15 Grad*.

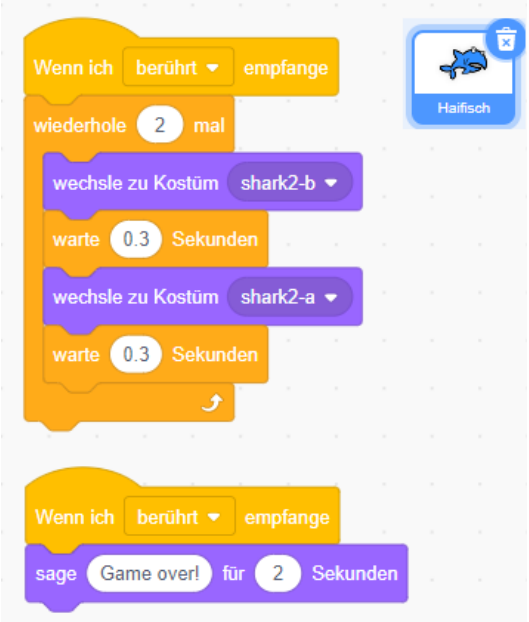
Fisch fangen

Haifisch berührt Fisch

Wenn der Haifisch den Fisch berührt, soll der Fisch ausgeblendet und wieder ins linke obere Eck gesetzt werden.

- Wähle den Fisch aus, damit er blau umrandet ist.
- Im Tab *Skripte* kannst du den Fisch verschwinden lassen, sobald er den Haifisch berührt.
- Unter *Ereignisse* wähle *Wenn … angeklickt*.
- Setze den Fisch an Position -230 und 170 mittels *gehe zu x: -230, y: 170*, um den Fisch ins linke obere Eck zu setzen, und *zeige dich*.
- Falls jetzt der Hai berührt wird (*Steuerung falls … dann*), dann *sende "berührt" an alle*, *verstecke dich*, *warte 5 Sekunden*, *zeige dich*, und gehe wieder ins linke obere Eck mit *gehe zu x: -230, y: 170*. Anschließend sage *Willkommen zurück* für 2 Sekunden.

Hai schnappt nach Fisch



Wenn der Haifisch den Fisch berührt, soll er zweimal schnappen und das Spiel "Game Over" sein.

- Wähle dazu den Haifisch aus, damit er blau umrandet ist.
- Im Tab *Skripte* kannst du den Haifisch "Game Over" sagen lassen.
- Unter *Ereignisse* wähle, *Wenn ich … empfange*, der Hai reagiert somit auf die vom Fisch ausgelöste Nachricht. Anschließend wähle *wiederhole 2-mal* bei *Steuerung* aus.
- Um den Haifisch schnappen zu lassen, gibt es unter *Aussehen* verschiedene Varianten des Hais. Füge folgende Blöcke in den Wiederhol-Block: *wechsle zu Kostüm b*, *warte 0.3 Sek.*, *wechsle zu Kostüm a*, *warte 0.3 Sek.*
- Und um den Haifisch "Game over" sagen zu lassen, füge einen neuen *Wenn ich … empfange* Block hinzu und *sage "Game Over!" für 4.5 Sekunden*.

Du kannst das fertige Projekt unter https://meet.coderdojo.net/scratch-fang-mich ausprobieren.

Weitere Ideen

- Mach das Spiel schwieriger, indem du einen zweiten, langsameren Haifisch dazu gibst.
- Baue eine Uhr ein, um zu sehen, wie lange du dem Haifisch entkommen kannst.
- Steuere den Fisch mit der Maus anstatt der mit der Tastatur.

Tastatur-Rennen – Wer tippt am schnellsten?

In diesem Spiel treten mehrere Personen in einem Rennen gegeneinander an

Wer tippt am schnellsten?

Scratch starten

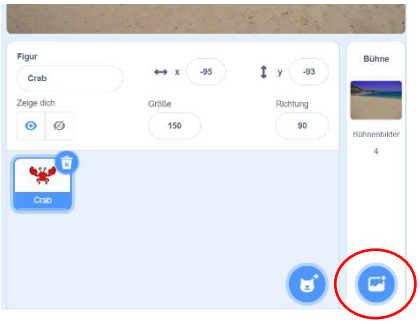
Für dieses Spiel sollten mehrere Spieler gleichzeitig programmieren, damit anschließend jeder und jede mit seinem oder ihrem Programm am Rennen teilnehmen kann. Schnapp dir also ein paar Freundinnen und Freunde und lade Sie ein, herauszufinden, wer am schnellsten tippen kann.

Du startest als erstes Scratch genauso, wie du es im vorigen Spiel gelernt hast.

Bühne und Figuren anlegen

Spielfeld

Genau wie im vorigen Spiel wählst du ein Bühnenbild. Alle die mitmachen können sich individuelle Bilder aussuchen. Wir verwenden hier einen Sandstrand. Du kannst aber gerne ein ganz anderes Bild verwenden.

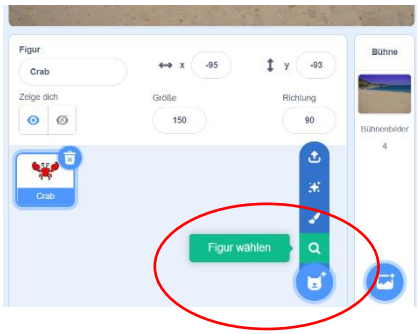


Scratchy löschen

Als nächstes lösche die Figur Scratchy mit dem Namen *Sprite1* oder *Figur1*. Du hast bereits im vorigen Programm gelernt wie das geht.

Figur anlegen

Jetzt brauchen wir die Figur, mit der wir am Rennen teilnehmen wollen. Du kannst dir eine beliebige Figur aussuchen. Wir verwenden hier eine Krabbe, du kannst aber eine andere Figur auswählen wenn du magst. Klicke dazu auf *Figur aus der Bibliothek wählen*.

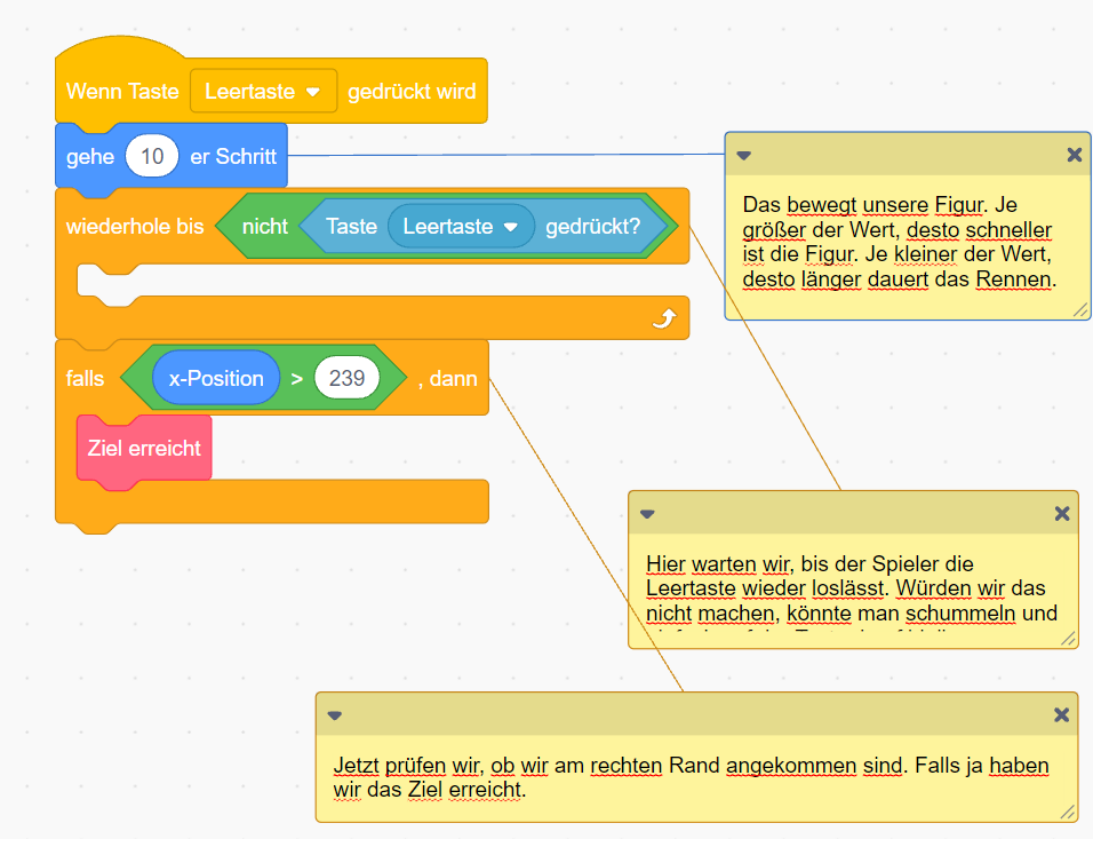


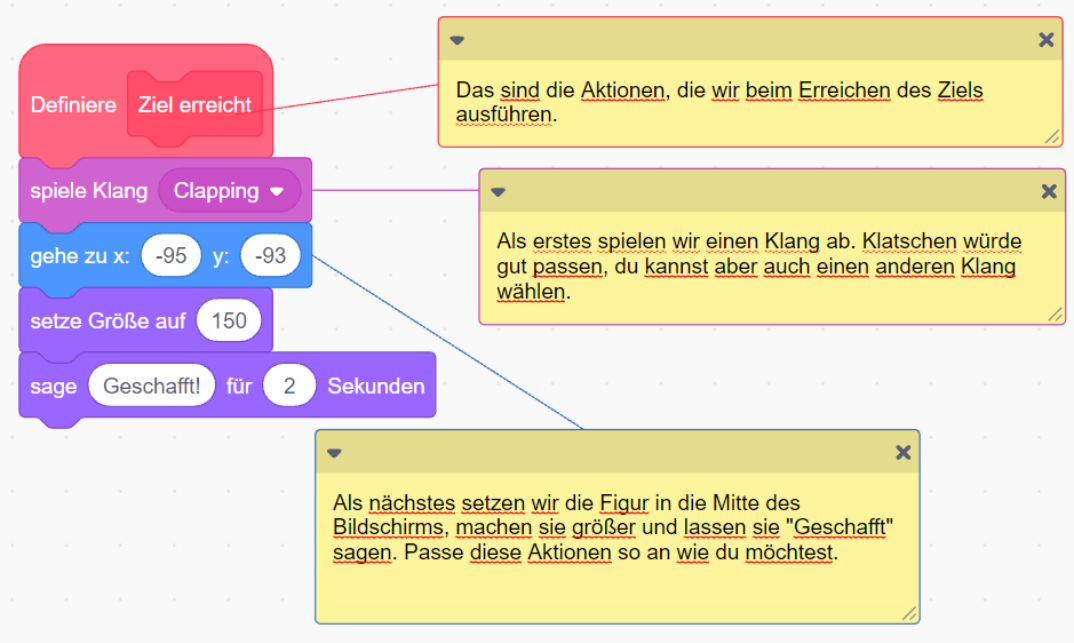
An den Start gehen

Setze deine Figur an den Start. Wichtig ist, dass du bei der X-Koordinate den Wert *-240* eingibst. Das ist der äußerst linke Rand. Diese Einstellung müssen alle machen, die beim Rennen mitmachen, damit niemand einen Vorteil hat.

Welchen Wert du bei der Y-Koordinate auswählst, ist dir überlassen. Wähle einen Wert, der gut zu einer Figur und dem Hintergrundbild passt.

Das Rennen





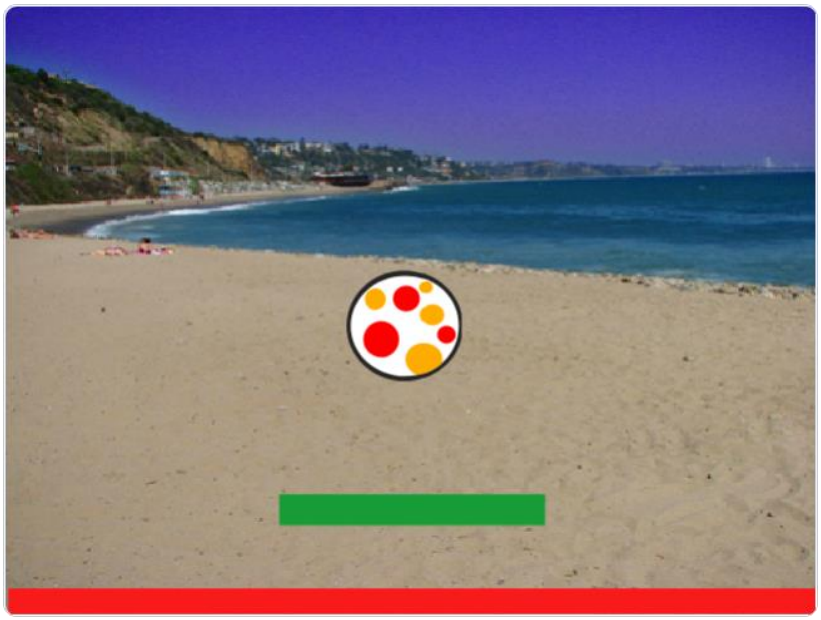
Programmiere die oben gezeigten Blöcke nach. Die gelben Felder sind Hinweise, die erklären, was die Blöcke machen. Sie sind nur zur Erklärung da, du musst sie nicht in dein Programm einfügen.

Los geht's!

Alles fertig programmiert? Dann drücken alle, die mitmachen wollen, auf die grüne Fahne. Dann sagt jemand „Auf die Plätze, fertig, LOS!“. Alle drücken so schnell wie möglich immer wieder auf die Leertaste.

Wer erreicht als erstes das Ziel?

Paddle Game

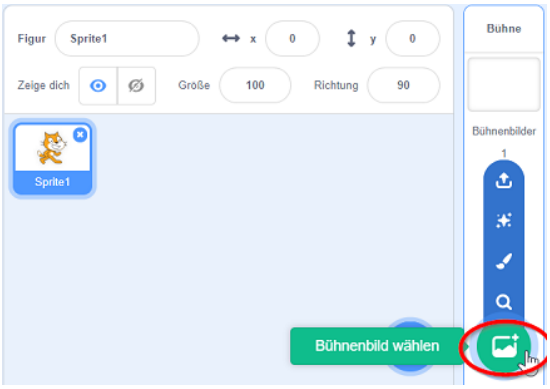


In dieser Übung baust du ein kleines Spiel, indem du versuchst, einen Ball mit einem Schläger in der Luft zu halten.

Bühne und Figuren anlegen

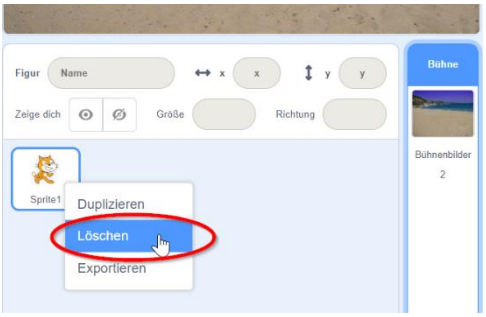
Spielfeld

Als erstes legst du fest, wie dein Spielfeld aussehen soll. Wir brauchen die Bühne, hier der Strand, einen Ball, einen Schläger und einen Bereich, der markiert, wo der Ball im Out ist. Wenn du ein neues Projekt startest, siehst du eine weiße Bühne mit Scratchy, der Katze. Wähle als erstes rechts unten ein Bühnenbild aus.



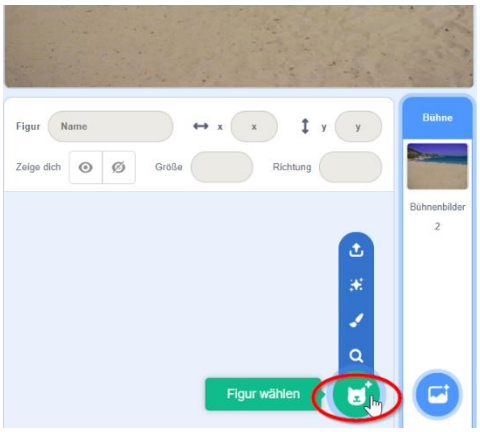
Scratchy löschen

Als nächstes lösche die Figur Scratchy mit dem Namen Sprite 1 indem du mit der rechten Maustaste daraufklickst. Im angezeigten Menü kannst du Scratchy löschen.



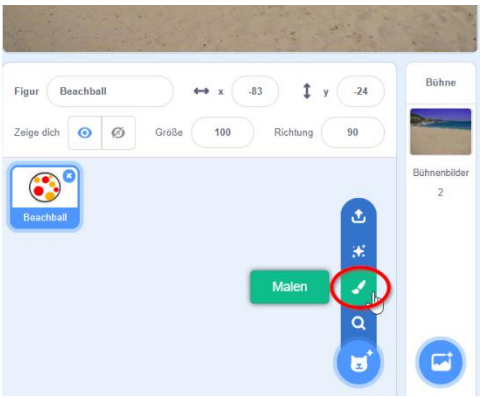
Ball

Jetzt brauchen wir einen Ball und einen Schläger. Links neben dem Bühnenbild findest du einen Menüpunkt, mit dem du neue Figuren hinzufügen kannst. Bei Thema Sport findest du verschiedene Bälle. Wähle einen davon aus.

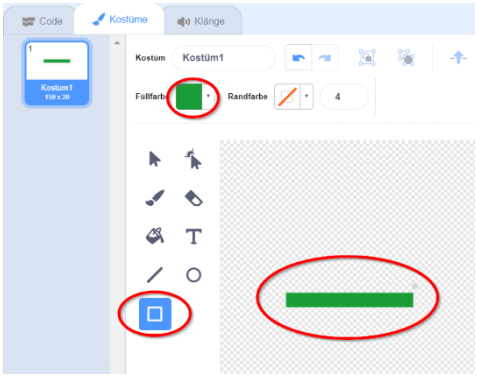


Schläger

Als nächstes ist der Schläger dran. Diesen malen wir selbst. Dafür führst du deine Maus zuerst auf das Feld Figur wählen, darüber siehst du nun neue Felder. Klicke auf das Feld Malen und du bekommst links eine Zeichenfläche.



Male ein einfaches Rechteck, dass den Ball daran hindern wird, am Boden aufzuschlagen. Du brauchst dazu das Werkzeug, um Rechtecke zu malen. Wenn du das Rechteck gemalt hast, wähle noch eine schöne Füllfarbe aus. Danach kannst du es im rechten Bereich auf der Bühne verschieben. Verschiebe es in den unteren Bereich der Bühne, aber nicht ganz nach unten. Hier brauchen wir noch ein wenig Platz für den Bereich, indem der Ball im Out ist.



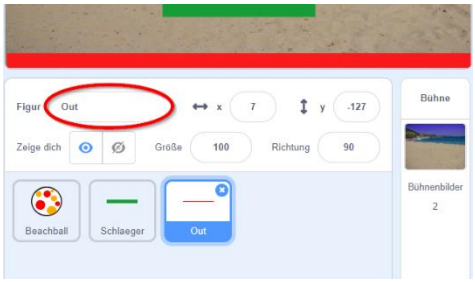
Achte darauf, dass der „Schläger“ genau in der Mitte der Figur ist. Du kannst dich beim Ausrichten am angezeigten Fadenkreuz orientieren.

Out-Bereich

Jetzt brauchst du noch den Out Bereich. Dafür malst du eine weitere Figur - ein Rechteck in einer anderen Farbe. Das Rechteck muss so breit wie der ganze Zeichenbereich sein. Das Rechteck schiebst du dann auf der Bühne ganz nach unten.

Figurennamen

Damit du später die Figuren leichter verwenden kannst, gib ihnen Namen wie Schläger und Out anstelle von Sprite 1 und Sprite 2. Du kannst den Namen unter der Bühne neben dem Wort Figur im Namensfeld ändern.



Schläger nach links und rechts bewegen

Schläger bewegen

Damit du den Ball in der Luft halten kannst, musst du den Schläger nach links und rechts bewegen können. Verwende dazu das Ereignis *Wenn Taste … gedrückt*. Du kannst einmal *Pfeil nach links* und einmal *Pfeil nach rechts* auswählen.

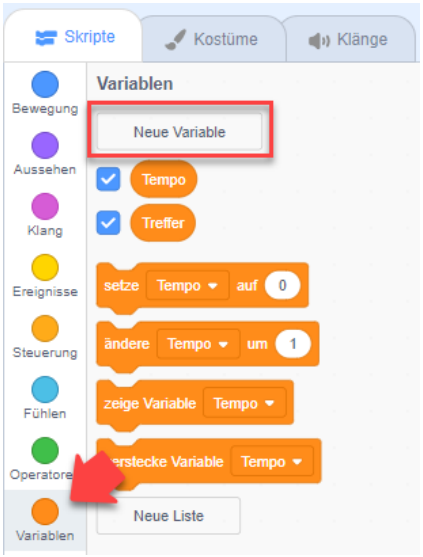
Wird der Pfeil nach links gedrückt, setze die Richtung auf - 90 Grad. Das bedeutet der Schläger bewegt sich nach links. Dann bewege ihn 15 Schritte. Für den Pfeil nach rechts setze die Richtung stattdessen auf 90 Grad.

Damit der Schläger nicht am linken oder rechten Rand verschwindet, kannst du prüfen, ob die x-Position des Schlägers noch nicht zu klein oder groß ist, bevor du den Schläger bewegst.

Wenn du jetzt die Pfeiltasten nach links oder rechts drückt, bewegt sich der Schläger. Beim Start des Spiels soll der Schläger in der Mitte sein. Verwende dafür den Block *Wenn Fahne angeklickt* und setze die x-Position des Schlägers auf 0.

Ball herumhüpfen lassen

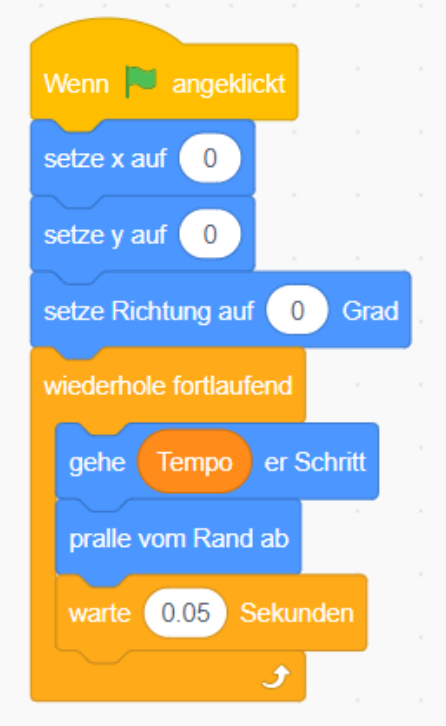
Variablen



Für unser Spiel brauchen wir zwei *Variablen*: *Tempo* und *Treffer*. *Tempo* legt fest, wie schnell der Ball ist. *Treffer* zählt, wie oft wir den Ball schon nach oben geschossen haben.

Wähle auf der linken Bildschirmseite die Kategorie *Variablen* und klicke auf *Neue Variable*. Tippe den Variablennamen ein (erst *Tempo*, dann *Treffer*) und lege fest, dass die Variable *für alle Figuren* gelten soll.

Ball steuern



Als nächstes kannst du den Ball bewegen. Wenn das Spiel gestartet wird, soll der Ball in die Mitte des Spielfelds gesetzt werden. Dazu setzt du die x- und y-Position auf 0. Setze außerdem die Richtung auf 0 - das heißt der Ball bewegt sich nach oben.

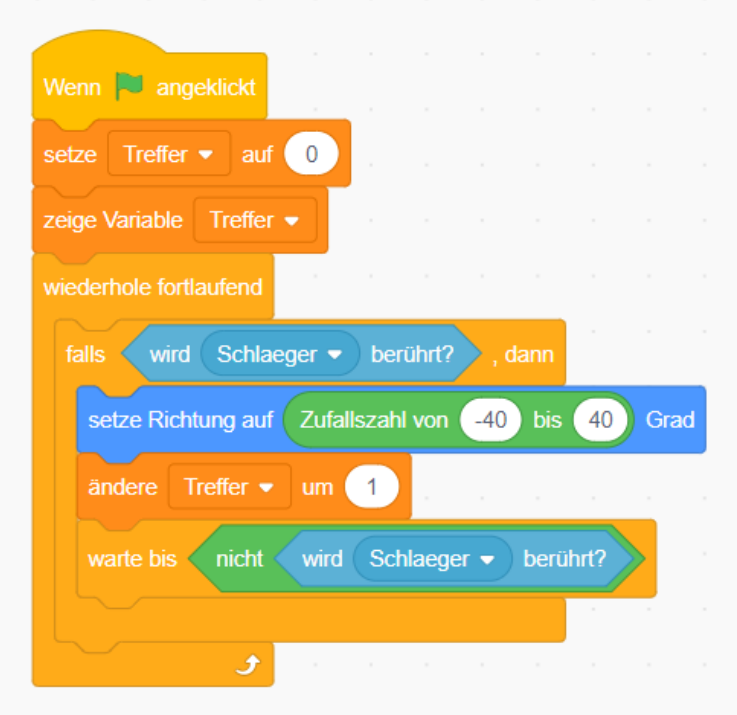
Dann kannst du die Bewegung starten. Wiederhole dazu drei Blöcke: gehe 10er-Schritt, pralle vom Rand ab - damit wechselt der Ball die Richtung, wenn er den Rand erreicht und warte 0.05 Sek. - damit bestimmst du die Geschwindigkeit des Balls.

Wenn du diesen Block ausführst, pendelt der Ball zwischen dem oberen und unteren Rand der Bühne. Den Schläger ignoriert er aber noch.

Um die Richtung des Balls zu ändern, wenn er den Schläger berührt, brauchst du einen weiteren Block, der beim Start des Spiels ausgeführt wird.

Hier wird fortlaufend geprüft, ob der Ball gerade den Schläger berührt. Wenn ja, wird die Richtung auf einen zufälligen Wert zwischen -40 und 40 geändert. Die Richtung 0 Grad bedeutet gerade nach oben. Ein Wert zwischen -40 und 40 sagt aus, dass der Ball gerade nach oben oder aber auch etwas links oder rechts davonfliegt.

Dann warte, bis der Ball den Schläger nicht mehr berührt, bevor der Block weiter ausgeführt wird.



Als letztes musst du noch erkennen, wann der Ball den Out-Bereich berührt. Dann ist das Spiel vorbei.

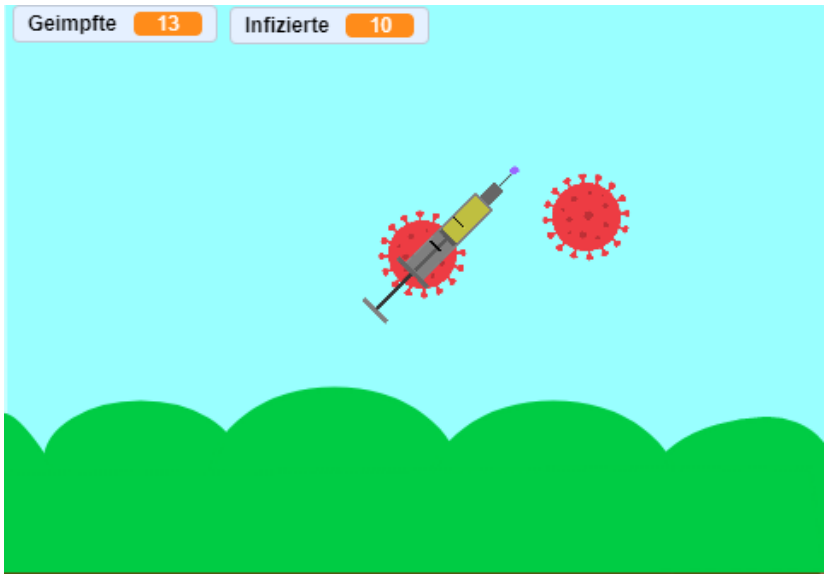
Fertiges Spiel

Gratuliere! Dein Spiel ist fertig. Probiere es gleich aus! Du kannst das fertige Projekt unter http://meet.coderdojo.net/scratch-paddle-game ausprobieren.

Weitere Ideen

- Steuere den Schläger mit der Maus statt mit der Tastatur.
- Bringe nach einiger Zeit einen weiteren Ball ins Spiel.

Virus-Buster

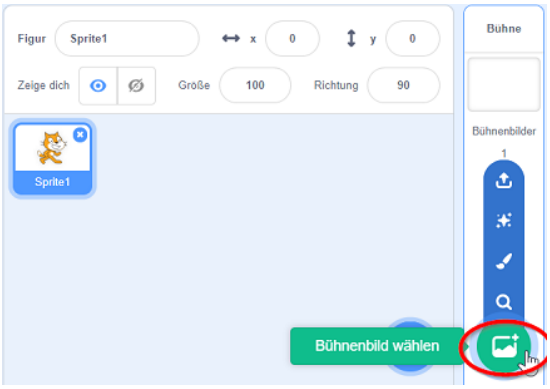


Durch dein schnelles Handeln kannst du in diesem Spiel bösartige Viren unschädlich machen.

Bühne und Figuren anlegen

Bühne

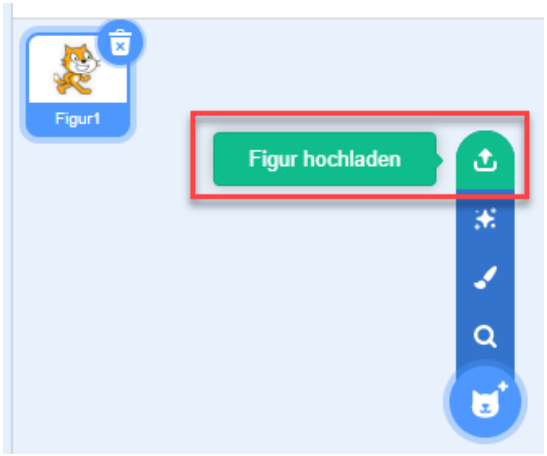
Als erstes legst du fest, wie dein Spielfeld aussehen soll. Wir brauchen einen Hintergrund, die Viren und die Impfspritze. Wenn du ein neues Projekt startest, siehst du eine weiße Bühne mit Scratchy, der Katze. Wähle als erstes rechts unten ein beliebiges Bühnenbild aus.



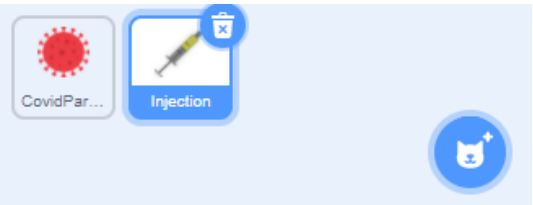
Figuren

Wir haben die Figuren für das Virus und die Impf-spritze vor dich vorbereitet.

Lade die Virus-Figur von unserer Webseite herunter (https://meet.coderdojo.net/virus-buster-virus). Merke dir, wo du die Figur speicherst. Mit *Figur hochladen* kannst du anschließend die Figur in dein Scratch-Projekt übernehmen. Wiederhole den Vorgang mit der Figur für die Spritze (http://meet.coderdojo.net/virus-buster-injection). Abschließend kannst du die *Scratchy*-Figur (Katze) löschen.



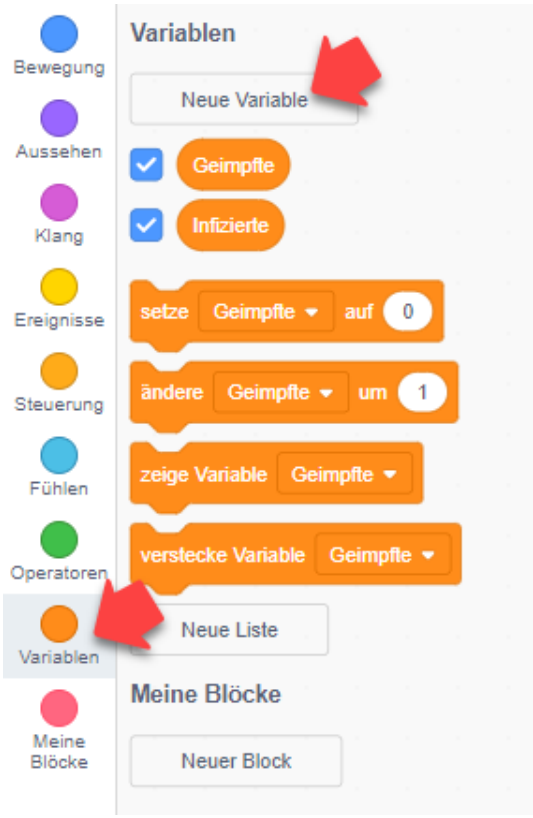
Jetzt solltest du zwei Figuren in deinem Projekt haben: Den Virus und die Spritze.



Variablen

In unserem Spiel möchten wir 2 Variable verwenden. Achte darauf, dass die Variablen mit einem Häkchen bei *Für alle Figuren* angelegt werden.

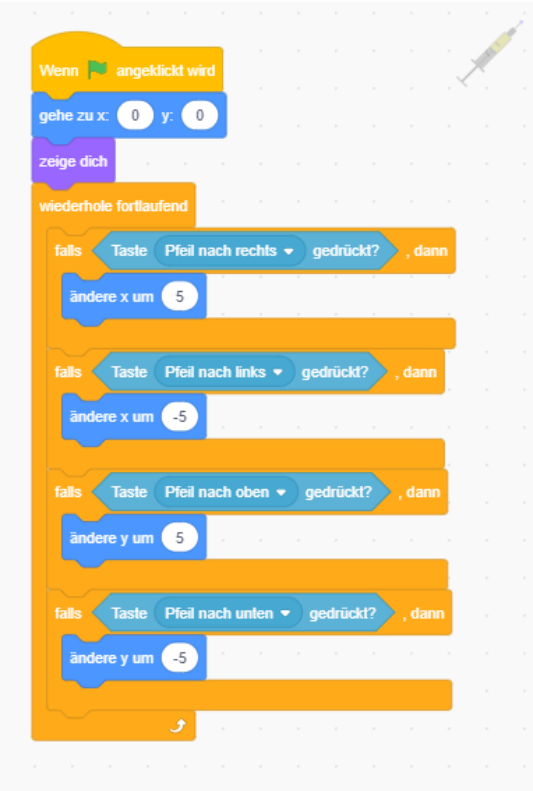
- *geimpfte*
- *infizierte*



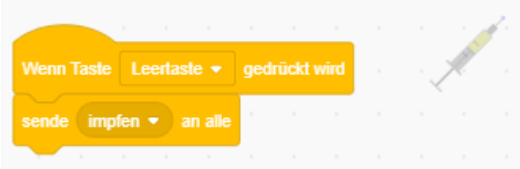
Spritze

Spritze steuern

Um die Spritze zu steuern, werden wir die Pfeil-Tasten verwenden. Da es bei diesem Spiel um schnelles Reagieren geht, können wir leider die Ereignisblöcke (z.B. *Wenn Pfeil nach unten gedrückt wird*) nicht verwenden. Wir werden stattdessen die Abfrage, ob eine Taste gerade nach unten gehalten wird, verwenden.

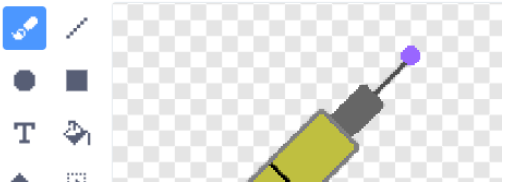


Impfung verabreichen



Für das Verabreichen der Spritze können wir das Ereignis *Wenn Taste Leertaste gedrückt wird* nehmen. Wir senden damit eine neue Nachricht (z.B. *impfen*) an alle.

Wir müssen noch einen kleinen Farbklecks an die Spitze der Spritze machen. Bitte verwende dazu eine Farbe, die sonst im Spiel nicht vorkommt (nicht im Hintergrund und nicht auf anderen Figuren, die das Virus berühren könnten). Dieser wird als unser *Trefferpunkt* dienen.



Virus

Spielstand

Unsere Viren sollen nach dem Freisetzen von rechts nach links über den Bildschirm fliegen. Wir müssen sie unschädlich machen, bevor sie den rechten Rand erreichen und weitere Menschen infizieren.

Beim Spielstart sollten wir die Punktestände auf 0 stellen.

Töne

Als nächstes holen wir uns noch Töne für das Freisetzen des Virus und deren Bekämpfung. Du kannst diese Klänge ganz einfach aus der Scratch-Bibliothek importieren.

Du kannst natürlich auch eigene Klänge verwenden, wenn dir diese besser gefallen.

Viren erzeugen

Da wir mehrere Viren auf dem Bildschirm haben möchten, werden wir zuerst die Figur unsichtbar machen. Nach einer zufälligen Zeit erzeugen wir dann einen Klon.

Virus bewegen

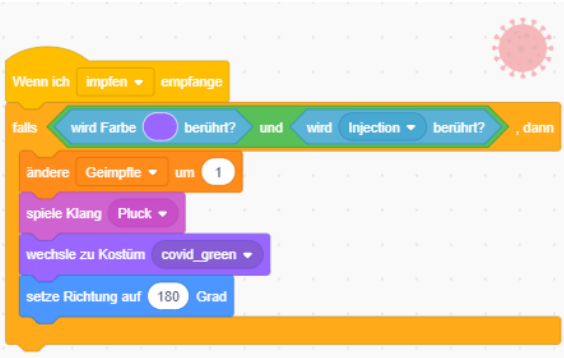
Den Klon bringen wir anschließend an eine zufällige Position an der rechten Seite und machen ihn sichtbar. Von dort aus tritt das Virus seine Reise über den Bildschirm an.

Das verwendete Kostüm soll uns Auskunft darüber geben, ob das Virus noch gefährlich oder schon unschädlich ist. Daher stellen wir es am Anfang auf das erste Kostüm (*covid_red*).

Gelangt das Virus zum Rand können wir so feststellen, ob noch jemand infiziert wurde oder nicht.

Als Abschluss zerstören wir den Klon.

Impfen



Nun fehlt uns nur noch das eigentliche Impfen. Wir haben bei der Spritze die Nachricht *impfen* abgeschickt und wollen nun wissen, ob wir einen Virus erwischt haben oder nicht.

Dabei wird uns unser Trefferpunkt (siehe Spritze) helfen. Bitte achte darauf, dass du genau die Farbe des Trefferpunktes beim *Wird Farbe .. berührt* Block einstellst. Du kannst dazu das *Pipetten*-Werkzeug verwenden.

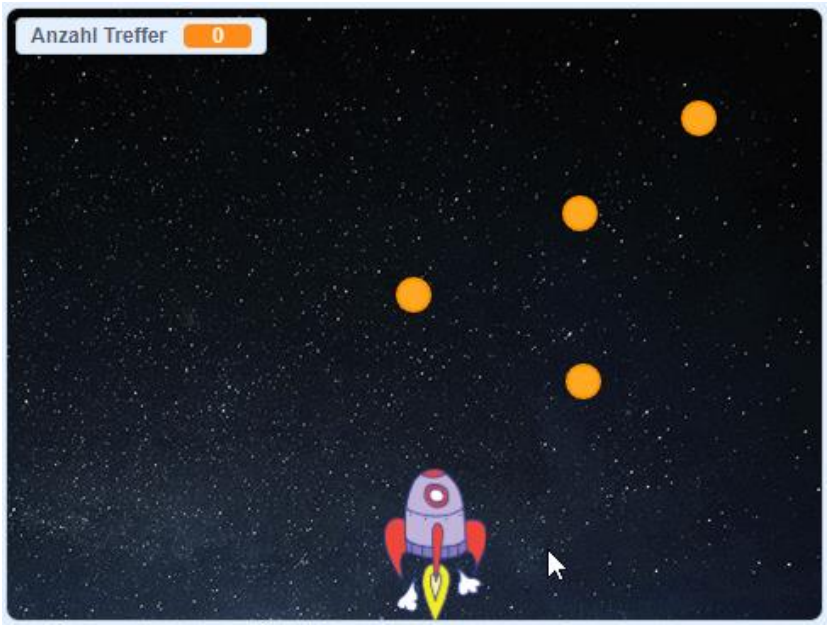
Fertig!

Gratuliere! Dein Spiel ist fertig. Probiere es gleich aus! Du kannst das fertige Projekt unter https://meet.coderdojo.net/virus-buster ausprobieren.

Weitere Ideen

- Bei den Hustentönen muss nicht jedes Mal der gleiche Ton verwendet werden. Du kannst zufällig entweder *Cough1* oder *Cough2* abspielen.
- Kannst du dein Skript so erweitern, dass nur 3 Spritzen verabreicht werden können bevor du z. B. 2 Sekunden warten musst? Wenn du magst, kannst du dafür die Figur *Impfung* verwenden (herunterzuladen unter http://meet.coderdojo.net/virus-buster-vaccine).

Space Shooter

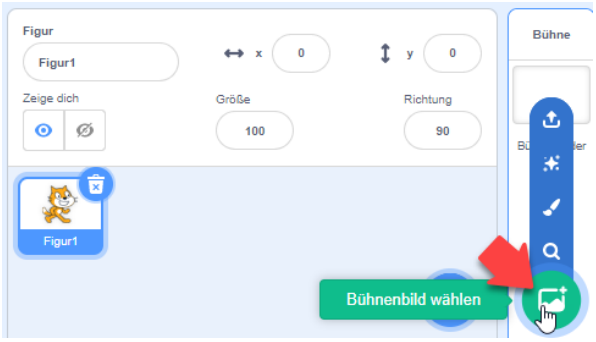


In dieser Übung schießt du mit deinem Raumschiff herabfallende Meteoriten ab bevor sie dein Raumschiff zerstören.

Bühne und Figuren anlegen

Spielfeld

Als erstes legst du fest, wie deine Bühne aussehen soll. Für dieses Spiel brauchst du das Weltall als Hintergrund. Wähle ein passendes Bild aus oder male selbst eines.

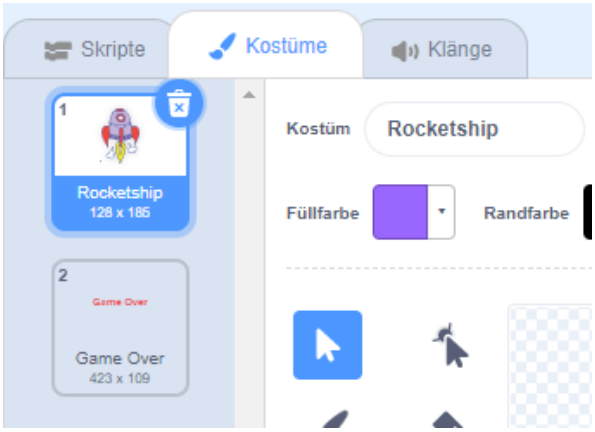


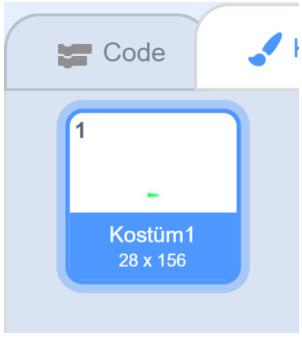
Figuren

Als erste Figur brauchst du das Raumschiff. Du fügst es mit *Figur wählen* ein.

Die Figur muss aus zwei Kostümen bestehen: dem Raumschiff selbst und einem weiteren Kostüm, das angezeigt wird, wenn das Spiel vorbei ist.

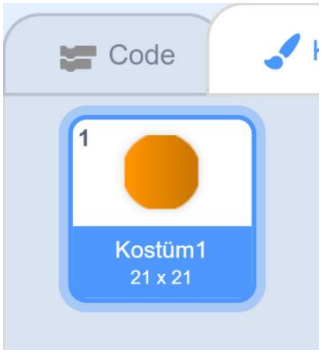
Erstelle dazu ein eigenes Kostüm mit dem Text *Game Over* oder male ein passendes Bild.

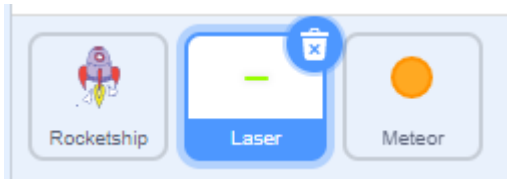




Die nächste Figur ist der Laserstrahl, der von der Rakete abgefeuert werden kann. Diesen kannst du dir selbst malen.

Und als letzte Figur brauchst du noch einen Meteoriten. Diesen kannst du dir selbst malen.

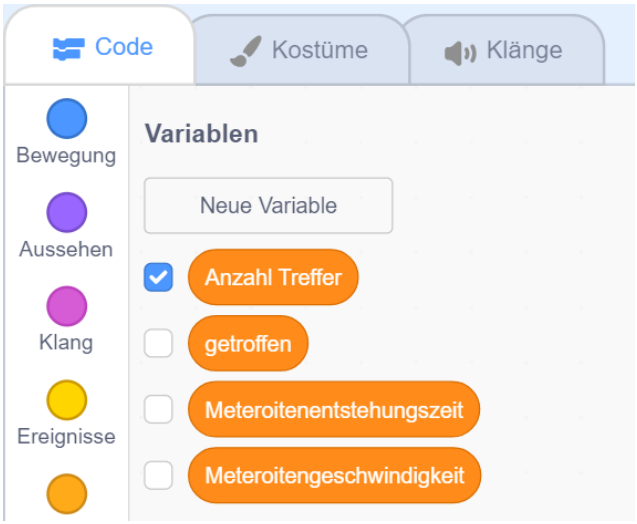




Am Ende solltest du drei Figuren haben: Raumschiff, Laser und Meteor.

Variablen

Wir brauchen für den Space Shooter einige *Variablen*: die Anzahl der Treffer, die Entstehungszeit von Meteoriten, deren Geschwindigkeit und einen Indikator, ob ein Meteorit getroffen wurde.

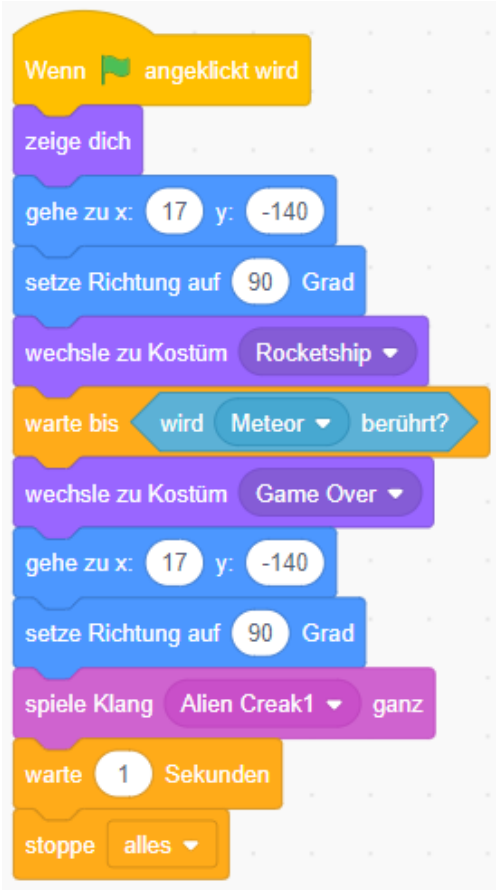


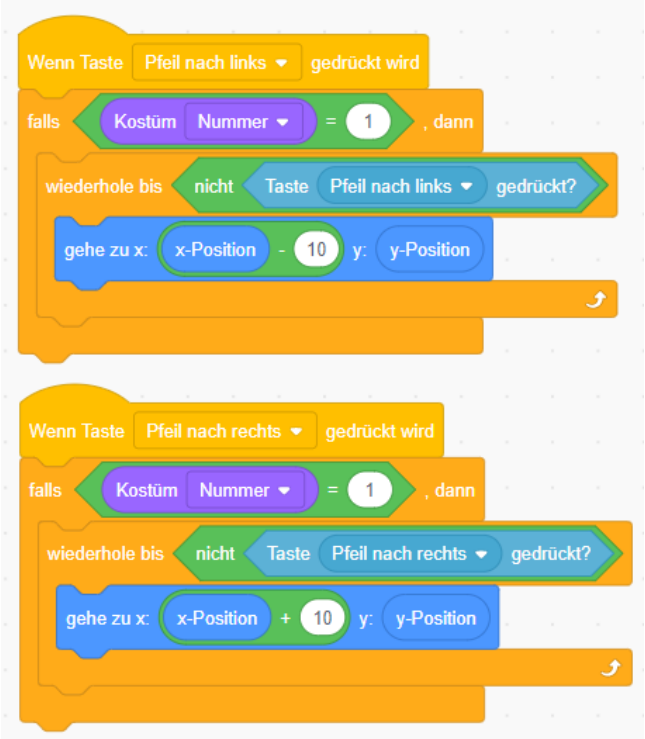
Achtung: **Die Variable getroffen gilt nur für die Figur "Meteorit", alle anderen Variablen gelten *für alle Figuren*.**

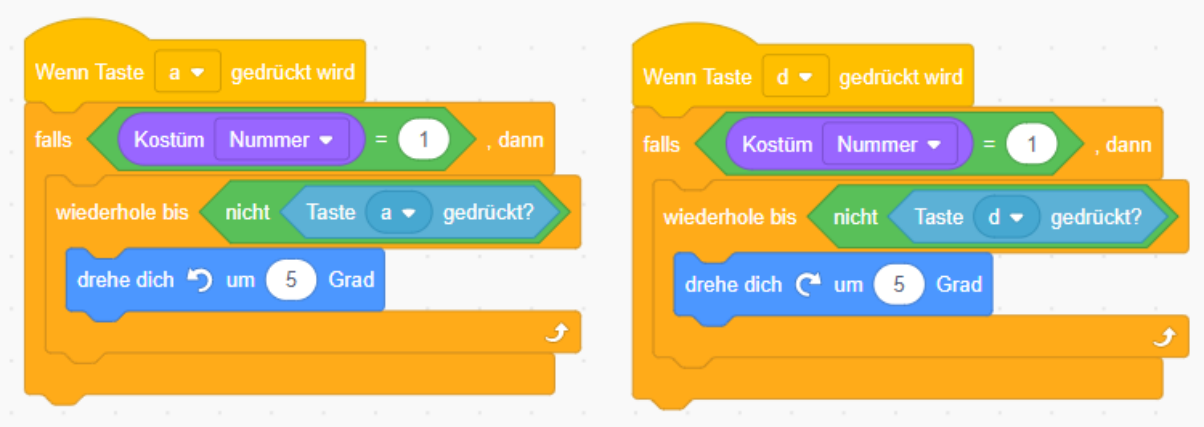
Skripte für das Raumschiff

Das Raumschiff hat drei Aufgaben:

- Es muss erkennen, wann es von einem Meteoriten getroffen wurde und dann das Spiel beenden.
- Mit den Pfeiltasten kann es nach links und rechts bewegt werden.
- Mit den Tasten *a* und *d* kann es nach links und rechts gedreht werden.

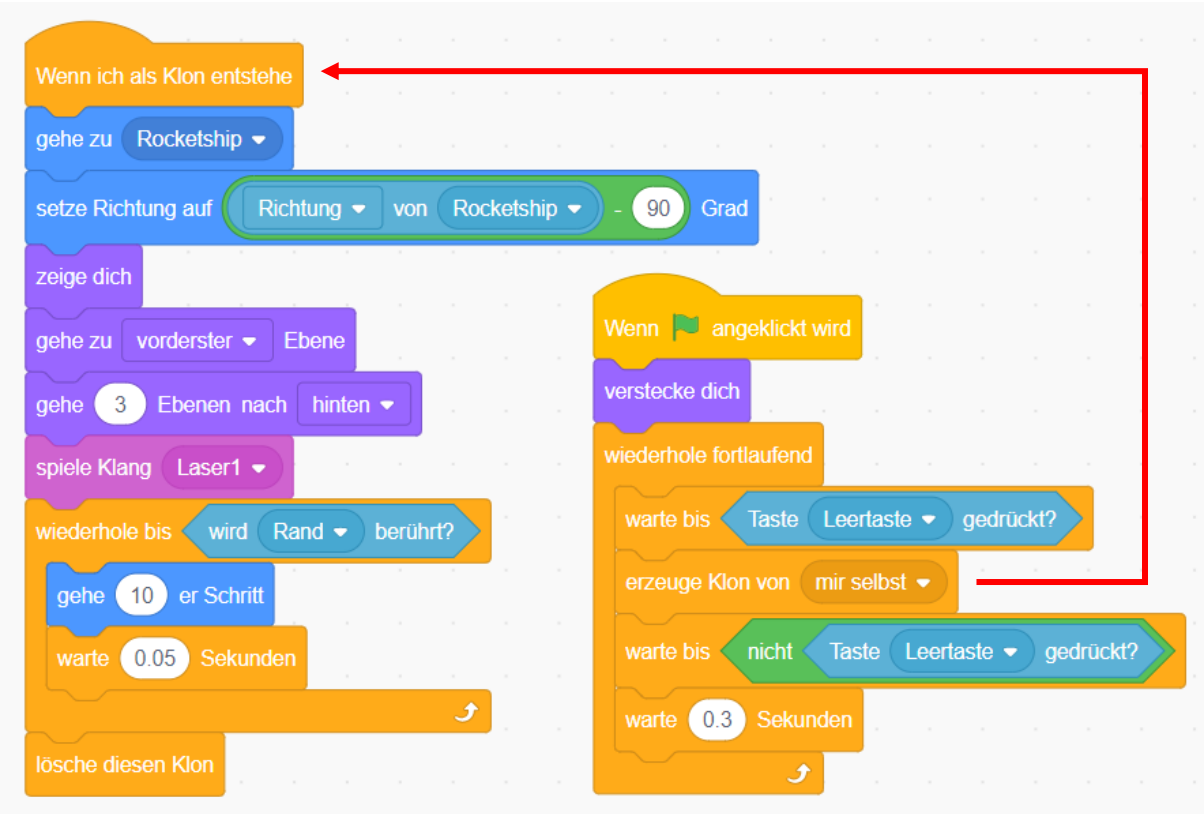






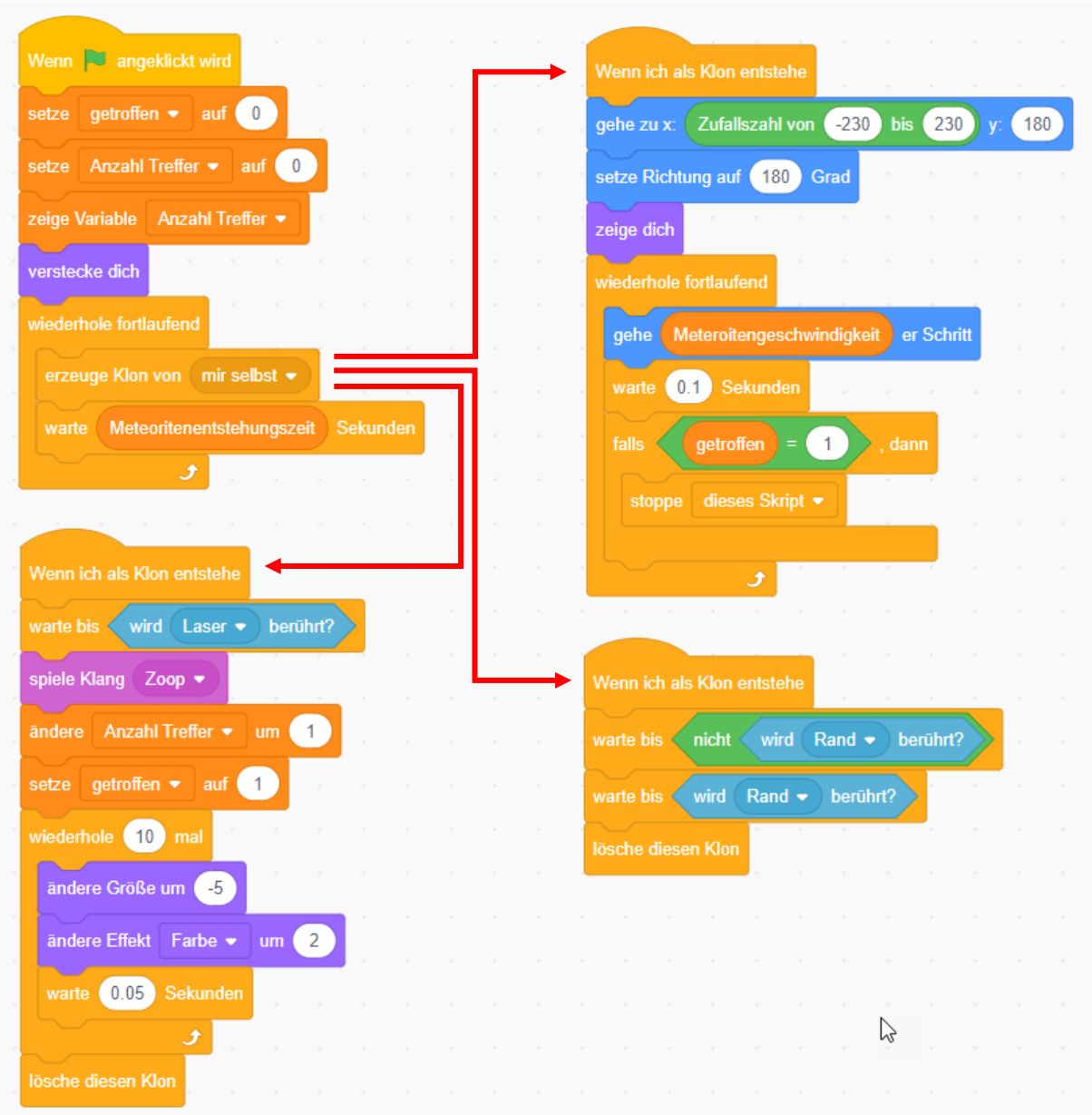
Skripte für den Laser

Jedes Mal, wenn die Leertaste gedrückt wird, muss ein neuer Laserstrahl erzeugt und abgefeuert werden.



Skripte für den Meteoriten

Für die Meteoriten brauchen wir eine Schleife, in der alle paar Sekunden ein neuer Meteorit erzeugt wird. Die Meteoriten müssen langsam nach unten fallen. Sie müssen verschwinden, wenn sie am unteren Bildschirmrand angekommen sind oder von einem Laser getroffen wurden.



Fertig!

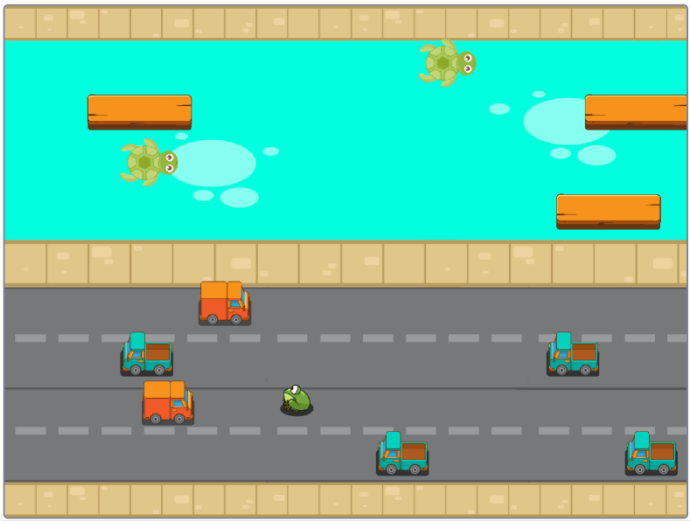
Gratuliere! Dein Spiel ist fertig. Probiere es gleich aus! Du kannst das fertige Projekt unter https://meet.coderdojo.net/scratch-spaceshooter ausprobieren.

Weitere Ideen

- Füge verschiedene Arten von Meteoriten hinzu.
- Ändere den Hintergrund, je länger das Spiel läuft.
- Zeige den Feuerstrahl des Raumschiffs nur an, wenn es sich bewegt.

Froggy – Die Straße überqueren

In diesem Spiel werden wir uns ansehen, wie wir eine größere Menge an Figuren in der Welt mit wenig Aufwand erzeugen. Ganz nebenbei helfen wir *Froggy* über die Straße.



Figuren

Bitte lade dir folgende Figuren herunter:

- Unsere Spielfigur
  https://cddataexchange.blob.core.windows.net/images/frog-jump/Spieler.sprite3
- Die Autos
  https://cddataexchange.blob.core.windows.net/images/frog-jump/Autos.sprite3
- Das Treibgut für den Fluss
  https://cddataexchange.blob.core.windows.net/images/frog-jump/Treibgut.sprite3

Importiere alle diese Figuren in Scratch (*Figur -> Figur hochladen*).

Der Hintergrund

Bitte lade dir folgenden Hintergrund herunter:

- https://cddataexchange.blob.core.windows.net/images/frog-jump/frog.png

Verwende dieses Bild als Hintergrund.

Steuerung des Frosches

Wichtig: Alle Blöcke in diesem Kapitel beziehen sich auf den Spieler, also den Frosch!

Um den kleinen Frosch über die Straße zu bringen, müssen wir ihn irgendwie steuern können. Damit das ganze gut aussieht, animieren

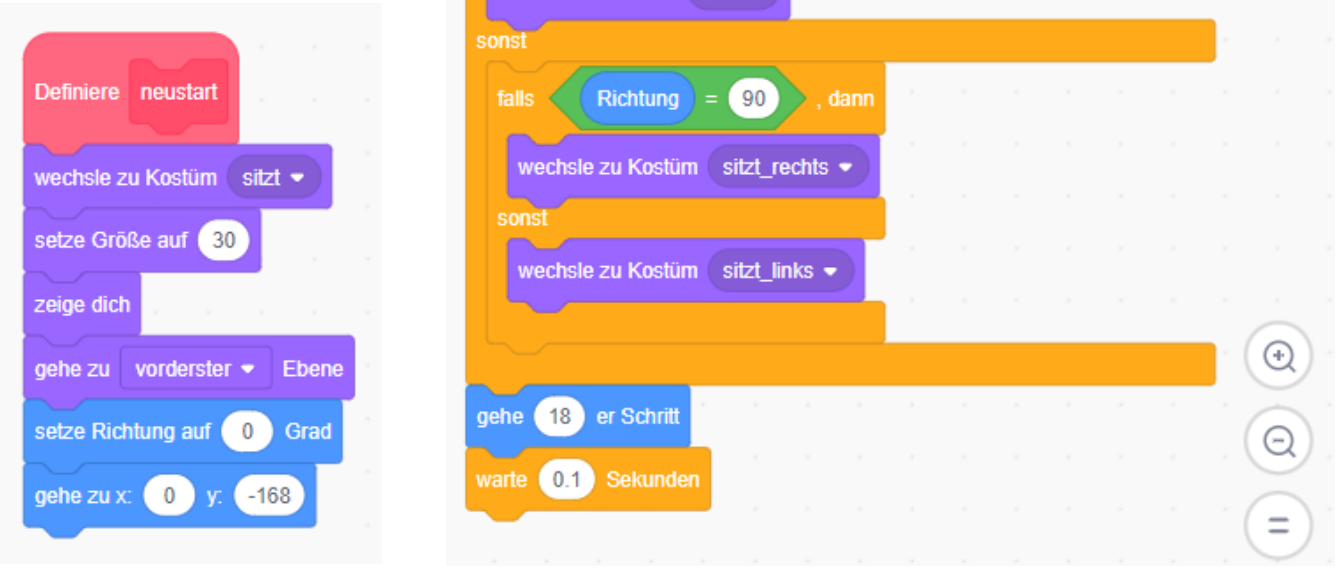
wir den Frosch bei jedem Schritt. Außerdem erlauben wir nur einen Schritt pro Tastendruck, damit das ganze etwas interessanter wird.

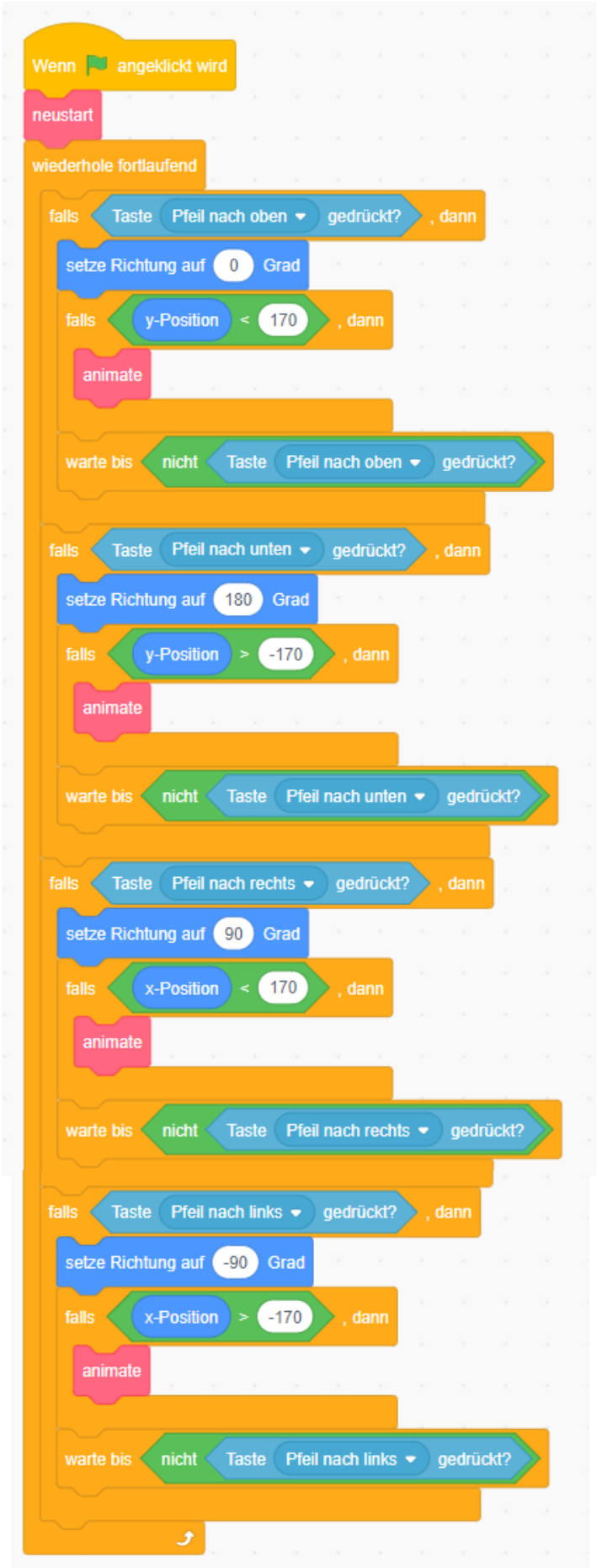
Am Anfang rufen wir einen kleinen Block auf, der unseren Frosch an die Anfangsposition bringt, und sein Aussehen sowie seine Größe einstellt.

Danach bauen wir für die für jede der Pfeiltasten einen Programmteil, der dafür sorgt, dass

- der Frosch nicht über den Rand hinauslaufen kann,
- der Frosch animiert und bewegt wird
- die Taste losgelassen werden muss, bevor die nächste Taste gedrückt werden kann.

Das Animieren erledigen wir auch in einem Block, damit wir es nicht jedes Mal aufs neue programmieren müssen.



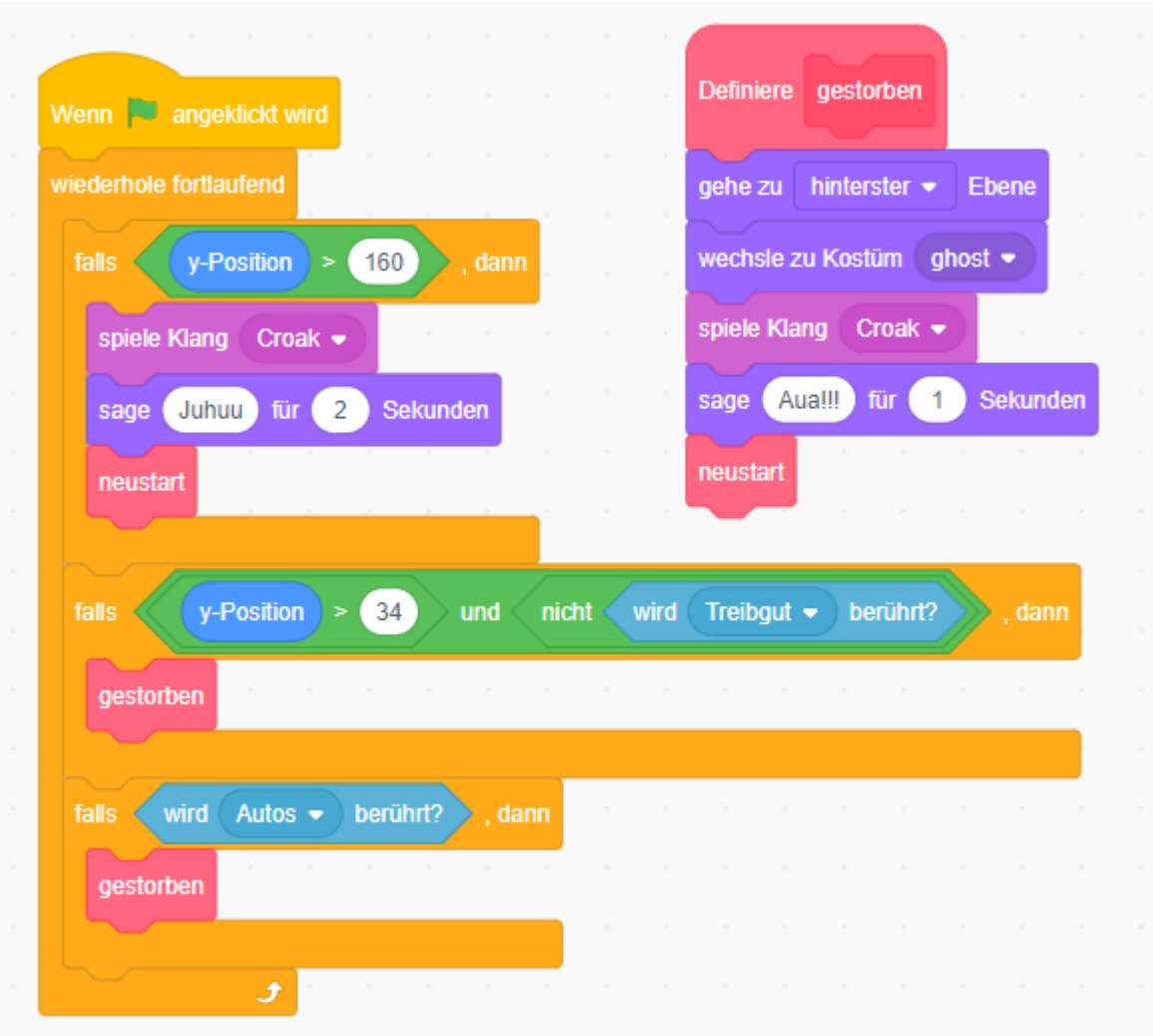

Wenn angeklickt wird
neustart
wiederhole fortlaufend
falls Taste Pfeil nach oben gedrückt? , dann
setze Richtung auf 0 Grad
falls y-Position < 170 , dann
animate
warte bis nicht Taste Pfeil nach oben gedrückt?
falls Taste Pfeil nach unten gedrückt? , dann
setze Richtung auf 180 Grad
falls y-Position > -170 , dann
animate
warte bis nicht Taste Pfeil nach unten gedrückt?
falls Taste Pfeil nach rechts gedrückt? , dann
setze Richtung auf 90 Grad
falls x-Position < 170 , dann
animate
warte bis nicht Taste Pfeil nach rechts gedrückt?
falls Taste Pfeil nach links gedrückt? , dann
setze Richtung auf -90 Grad
falls x-Position > -170 , dann
animate
warte bis nicht Taste Pfeil nach links gedrückt?

So … das war ein ordentliches Stück Arbeit, aber wir können den Frosch nun über den Bildschirm hüpfen lassen. Probier es einfach mal aus …

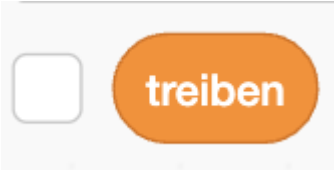
Um drei Sachen müssen wir uns allerdings noch kümmern:

- Wir sollten es auch bemerken, wenn wir die Hindernisse erfolgreich überquert haben,
- der Frosch darf keine Autos berühren und
- den Fluss kann er nur auf einem Treibgut-Stück überqueren.

Sollten wir einen der beiden letzten Punkte erfüllen, so sind wir “gestorben” und müssen nochmal von vorne anfangen. Auch dafür haben wir einen Block.

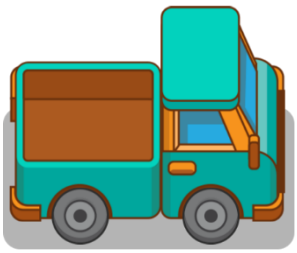


Jetzt müssen wir beim Frosch noch etwas Vorarbeit leisten, damit wir ihn später mit dem Treibgut mitschwimmen lassen können. Lege dir dafür eine Variable “für alle Figuren” namens *treiben* an.



Damit wir mitgekommen, dass wir auf einem Treibgut stehen, das sich bewegt, lassen wir uns eine Nachricht schicken.

Die Autos



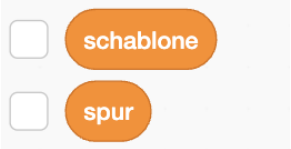
Wichtig: Alle Blöcke in diesem Kapitel beziehen sich auf die Autos!

Damit wir mit nur einer Figur für die Autos auskommen, verwenden wir einen kleinen Trick. Am Anfang des Spiels werden wir Klone erzeugen, die als “Schablone” für die einzelnen Autos auf den Fahrspuren dienen. Während des Spiels brauchen wir dann nur mehr die “Schablonen” klonen.

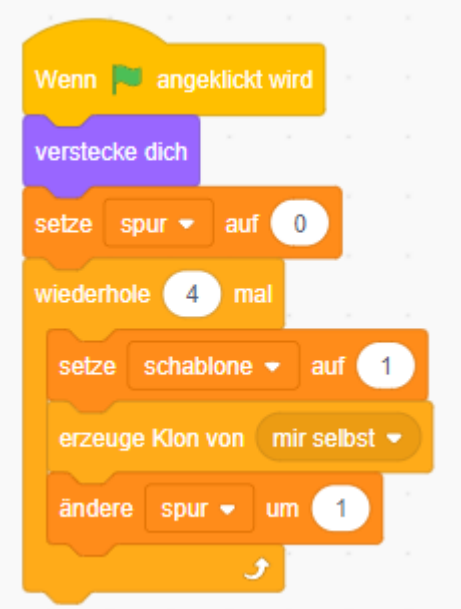
Damit wir die Schablonen auch als solche erkennen, merken wir uns in einer Variablen, ob es sich gerade um eine Schablone handelt, oder ein fahrendes Auto. Außerdem brauchen wir noch eine Variable, in der wir uns die Fahrspur merken.

Bitte lege also folgende Variablen **nur für diese Figur** an:

- schablone
- spur



In einer Schleife legen wir vier Schablonen-Autos an und setzen sie jeweils auf die richtige Spur.



Kümmern wir uns nun um die Behandlung der Schablonen. Als Erstes müssen wir überprüfen, ob es sich bei dem aktuellen Klon um eine Schablone handelt. Wenn nicht, wird hier gar nichts gemacht. Auch die Schablonen bleiben unsichtbar, werden aber bereits auf die richtige Fahrspur gestellt.

Die Position der Fahrspur berechnen wir einfach mit der Formel: (spur * 35) - 135.

Zusätzlich berechnen wir auch die Kostüm-Nummer aus der spur: (spur mod 2) + 1.

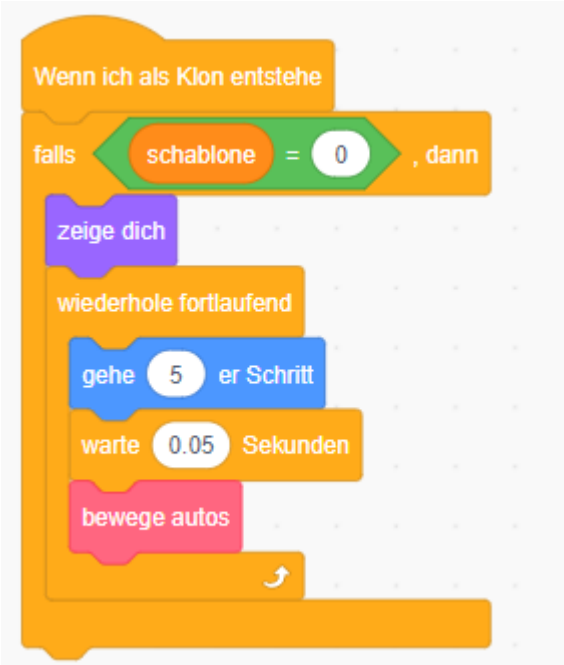
Anmerkung: mod ist der Divisionsrest. Das +1 am Schluss benötigen wir, da die Kostüme mit Nummer “1” anfangen.

Danach sorgen wir dafür, dass jede 2. Fahrspur von der rechten Bildschirmseite beginnt. Auch dafür können wir “mod 2”. Damit erhalten wir “1” für ungerade Fahrspuren und “0” für gerade.

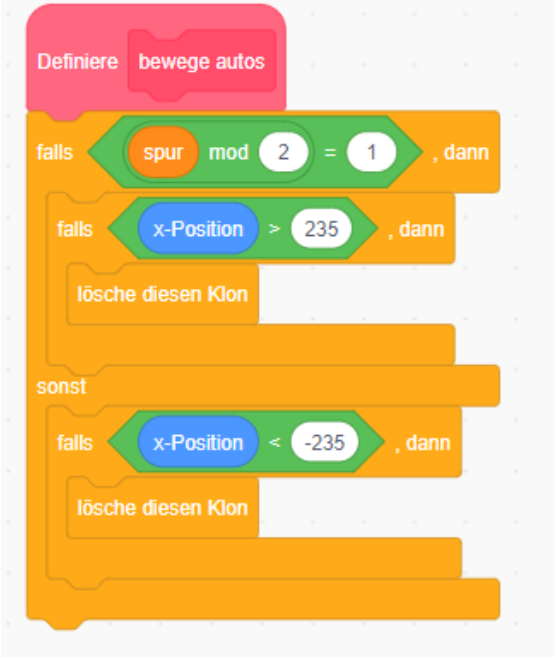
Ganz am Schluss werden wir in einer Schleife noch Klone der Schablone in zufälligen Abständen erzeugen. Das “Warte” vor der Schleife ist nur da, damit Scratch etwas Zeit hat zuerst alle Schablonen zu erzeugen, bevor das Spiel wirklich startet.

Das Erzeugen der Autos ist dann recht einfach. Nicht vergessen, dass wir natürlich nur “nicht-Schablonen” als Autos ansehen dürfen.

Wir machen das Auto sichtbar und fahren so lange, bis wir an der anderen Bildschirmseite angekommen sind. Auch da hilft uns wieder die Überprüfung, ob die Fahrspur gerade ist, oder nicht.



Das Bewegen des Autos ist hier in einem Code-Block ausgeführt, damit die einzelnen Programmteile nicht zu gross werden.



Wenn du das Spiel nun ausprobierst, solltest du bereits den Verkehr auf der Straße sehen. Aber nimm dich in Acht - Die Autos könnten unseren kleinen Frosch bereits überfahren…

Das Treibgut

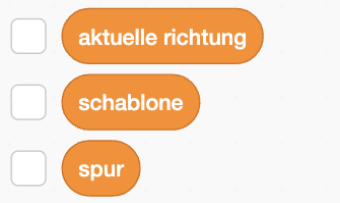
Wichtig: Alle Blöcke in diesem Kapitel beziehen sich auf das Treibgut!

Die Blöcke des Treibguts sind nahezu die gleichen wie die für die Autos. Du kannst dir hier das Leben einfach machen, und die Blöcke aus “Autos” kopieren. Du musst nur aufpassen, dass du die Position richtig einstellst und die Programmblöcke umbenennst.

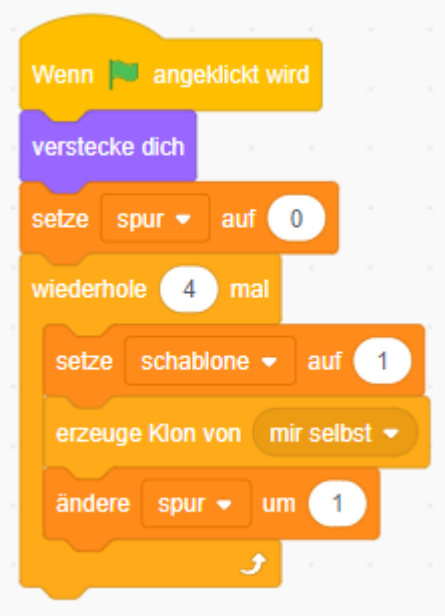
Wir erzeugen die Schablonen - achte darauf, dass du die Variablen *spur* und *schablone* **nur für diese Figur** anlegst. Außerdem müssen wir uns die aktuelle X-Richtung merken, damit wir dem Spieler später sagen können, wohin der treibt.

Leg also bitte die Variablen nur für diese Figur an:

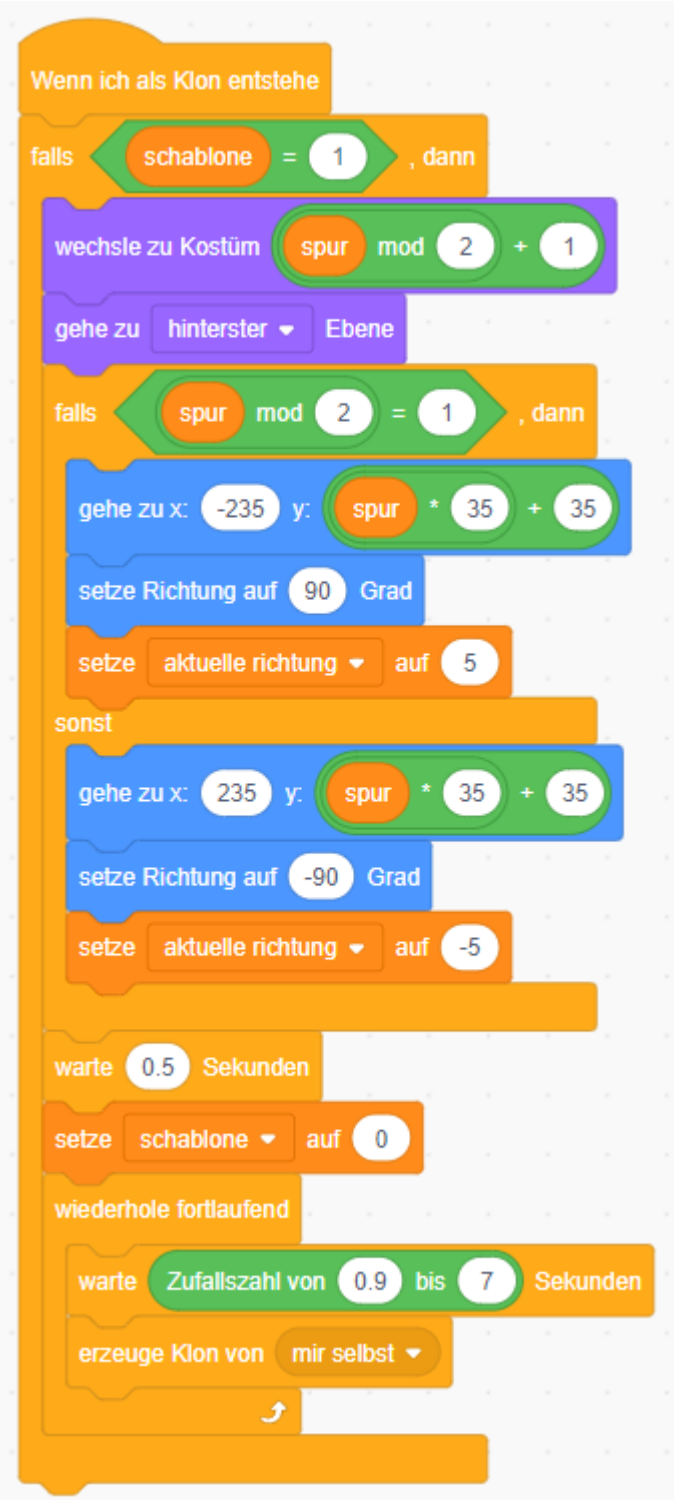
- akutelle richtung
- schablone
- spur



In einer Schleife legen wir vier Treibgut-Schablonen an und setzen sie jeweils auf die richtige Spur.

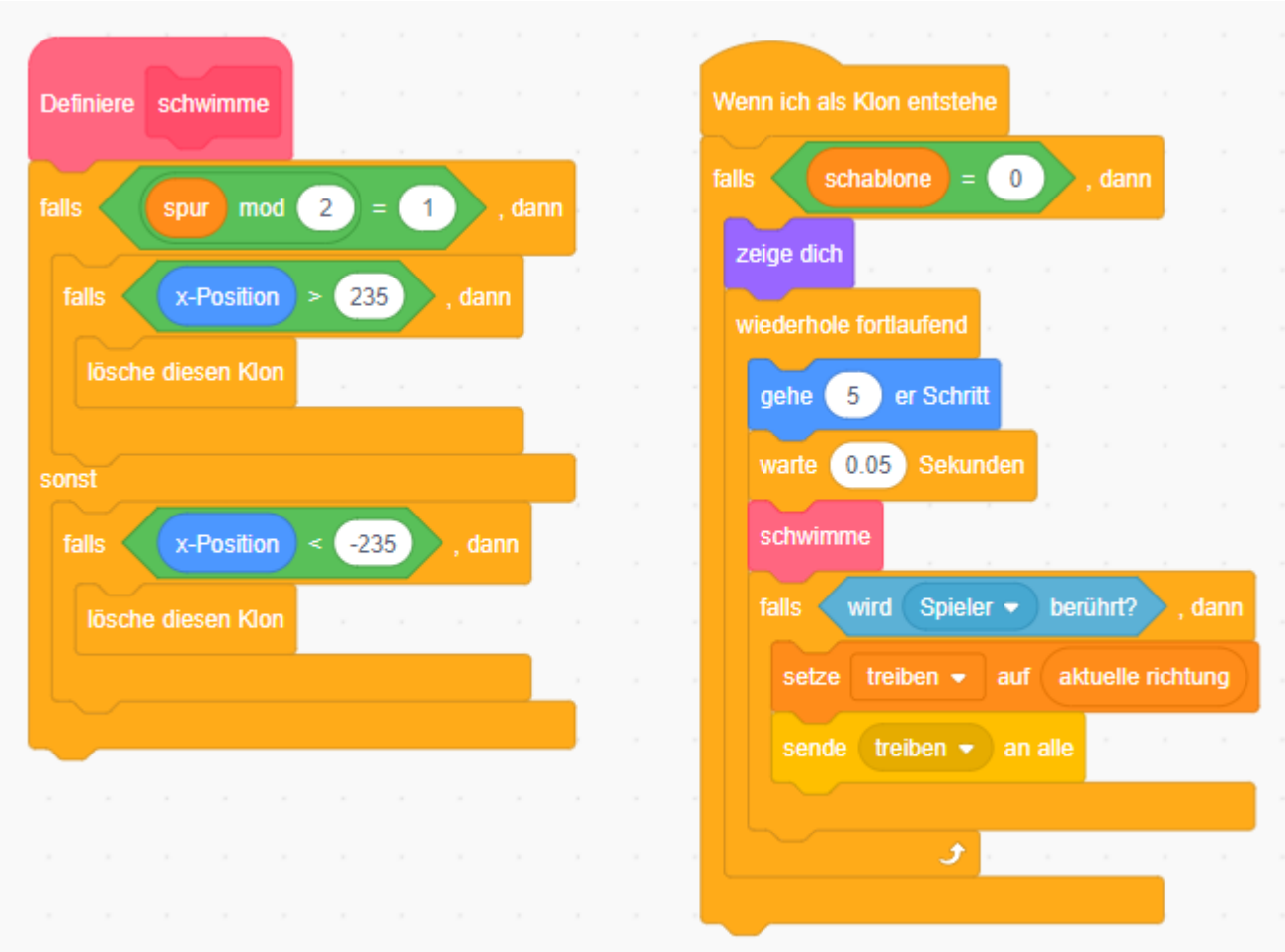


Die Behandlung der Schablonen läuft genau wie bei den Autos. Einzig die Positionen sind anzupassen, und die Variable “aktuelle richtung” muss gesetzt werden.



Auch das “Schwimmen” des Treibguts funktioniert wie das Fahren der Autos. Ein zusätzlicher Teil kommt allerdings dazu: Wenn der Spieler auf dem Treibgut sitzt, müssen wir ihn wissen lassen, dass er mitkommen muss. Also senden wir die Nachricht treiben.

Auch hier haben wir das eigentliche Treiben in einen eigenen Block ausgelagert.



Fertig! Viel Spaß beim Spielen! Das Spiel funktioniert am besten, wenn du den “Vollbild-Modus” aktivierst.

Erweiterungen

Vielleicht kannst du das Spiel noch ein bisschen erweitern. Ein paar Ideen:

- Du könntest Punkte sammeln, wenn du die Straße überquert hast.
- Eventuell gibt es nur eine begrenzte Anzahl von Leben
- Du könntest auch Level einbauen, indem sich z.B. die Autos schneller bewegen, wenn du bereits mehrere Punkte hast

Malen nach Zahlen

Wir lernen, wie man im Browser programmiert und bauen dabei ein kleines Mal-programm.

Ziel der Übung

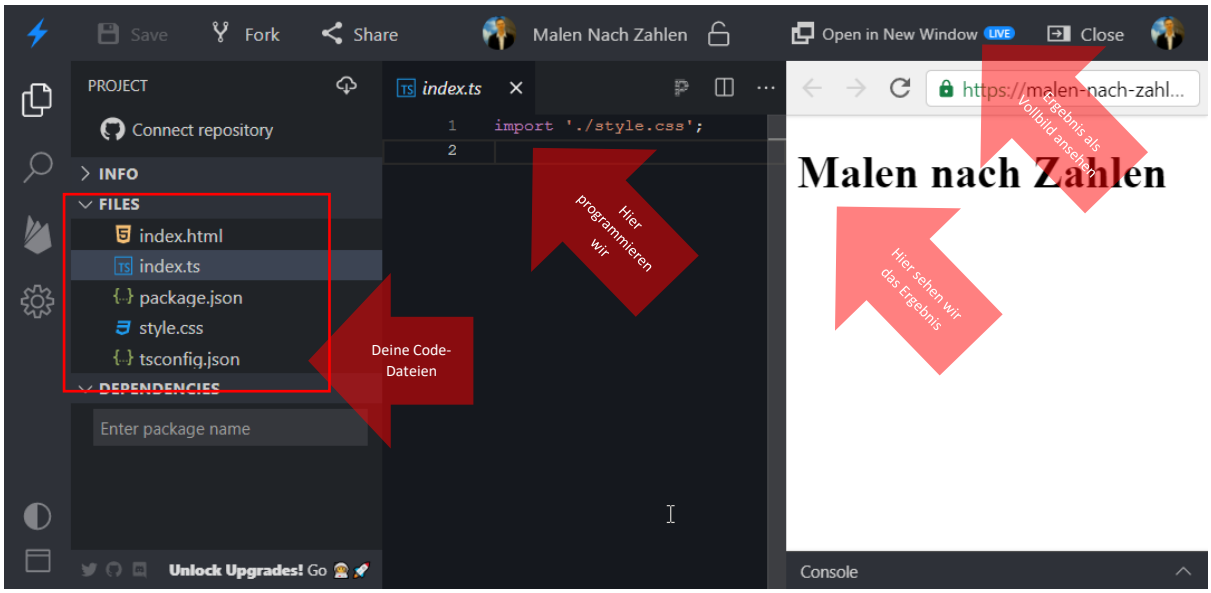
Wenn du noch nie textuell programmiert hast, ist das die richtige Übung zum Einsteigen für dich. Du kannst das Tippen von Code auf der Tastatur üben und lernst Grundlagen der Browser-Programmierung wie HTML, Variablen und Schleifen kennen.

Du solltest für diese Übung schon ein wenig Programmiererfahrung mit einer Blockprogrammiersprache wie *Scratch* oder *Snap!* haben. Falls du noch nie Programmcode eingetippt hast, solltest du auch etwas Geduld mitbringen. Es ist am Anfang nicht immer leicht, die vielen Sonderzeichen im Code auf der Tastatur zu finden. Halte durch, Übung macht den Meister!

Start

Du brauchst für diese Übung keine spezielle Software auf deinem Computer zu installieren. Es ist nur ein aktueller Web-Browser (vorzugsweise Chrome) notwendig.

Um loszulegen, öffne die URL https://stackblitz.com/edit/malen-nach-zahlen-starter. Du wirst dadurch im Web-Browser ein Fenster sehen, das in etwa so aussieht:



- Links siehst du eine Liste der Dateien. Wir programmieren heute in *index.html* und *index.ts*.
- Unseren Programmcode schreiben wir im Textfenster in der Mitte.
- Rechts sehen wir nach jeder Codeänderung gleich das Ergebnis.
- Mit *Open in New Window* kannst du das Ergebnis deines Programms im Vollbildmodus ansehen.

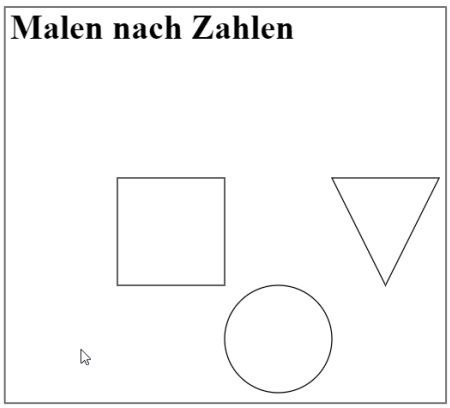
In dieser Übung programmierst du mit HTML und der Programmiersprache *TypeScript*.

Programmieren mit HTML

Worum geht es?

In unserem Malprogramm starten Benutzer mit einem Schwarzweißbild. Wir werden mit einfachen Formen wie Rechtecken und kreisen beginnen, wollen am Ende aber dann einen schönen Schmetterling sehen. Anschließend wählt man eine Farbe und klickt auf eine weiße Fläche im Bild. Dadurch färbt man die Fläche entsprechend ein. Es entsteht ein schönes, buntes Bild.

Erste Formen



Als erstes zeichnen wir einfache Formen auf den Bildschirm. Dafür verwenden wir sogenannte *Vektorgrafiken*. In Englisch heißen sie *Scalable Vector Graphics*, kurz *SVG*.

Öffne *index.html* und füge die in Rot dargestellten Codezeilen hinzu. Das Ergebnis sollte so aussehen wie rechts angezeigt.

```
<h1>Malen nach Zahlen</h1>

<svg id="image" width="1000" height="600">
   <rect x="100" y="100" width="100" height="100" stroke="black" />
   <circle cx="250" cy="250" r="50"  stroke="black" />
   <path d="m300,100 h100 l-50,100 Z" stroke="black" />
</svg>
```

Hier ein paar Tipps zum Code, den du gerade geschrieben hast:

- *Width* heißt auf Deutsch *Breite*
- *Height* heißt auf Deutsch *Höhe*
- *Rect* ist eine Abkürzung für *Rectangle* und heißt auf Deutsch *Rechteck*. Mit *rect* zeichnen wir also das Rechteck.
- *Circle* heißt auf Deutsch *Kreis*. Mit *circle* zeichnen wir also den Kreis.
- *Path* heißt auf Deutsch *Pfad*. Damit können wir beliebige Linienpfade zeichnen. Bei uns malen wir ein Dreieck.
- *Stroke* heißt auf Deutsch *Pinselstrich*. Damit legst du die Farbe fest. Im Moment steht überall *black*, auf Deutsch *Schwarz*. Probiere zum Beispiel einmal, *black* durch *red* (Rot) zu ersetzen. Siehst du den Unterschied?

Auf die Details der Koordinaten (*x*, *y*, *cx*, *cy* etc.) wollen wir hier im Moment noch nicht eingehen. Du kannst aber mit den Zahlen experimentieren. Was passiert, wenn du größere oder kleinere Werte verwendest?

Farbauswahl

Als nächstes möchten wir, dass man sich eine Malfarbe aussuchen kann. Dazu müssen wir wieder *index.html* erweitern.

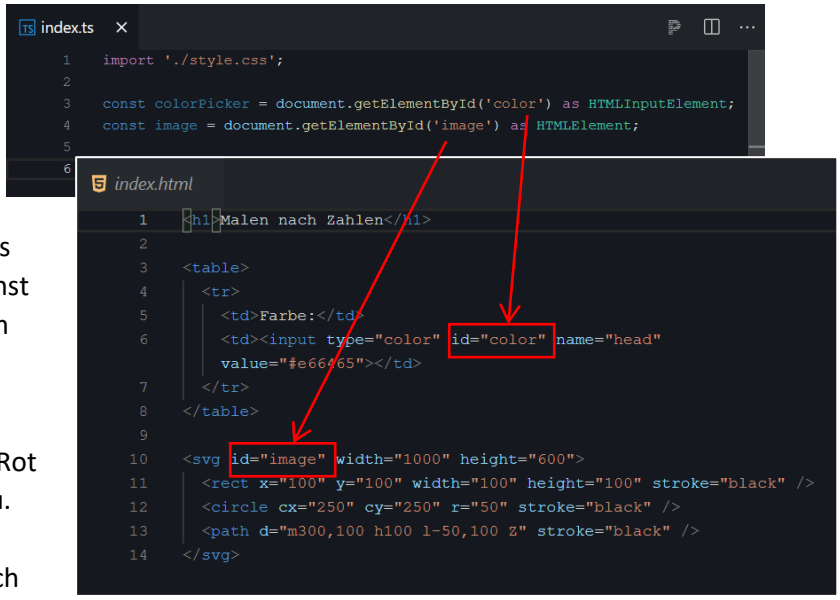
Öffne *index.html* und füge die in Rot dargestellten Codezeilen hinzu. Das Ergebnis sollte so aussehen wie rechts angezeigt.

```
<h1>Malen nach Zahlen</h1>

<table>
  <tr>
    <td>Farbe:</td>
    <td><input type="color" id="color" name="head" value="#e66465"></td>
  </tr>
</table>

<svg id="image" width="1000" height="600">
  <rect x="100" y="100" width="100" height="100" stroke="black" />
  <circle cx="250" cy="250" r="50" stroke="black" />
  <path d="m300,100 h100 l-50,100 Z" stroke="black" />
</svg>
```

Super gemacht! Wir haben Formen und eine Farbauswahl. Jetzt können wir mit TypeScript den Code schreiben, der es uns ermöglicht, die Formen einzufärben.

Programmieren mit TypeScript

Elemente suchen

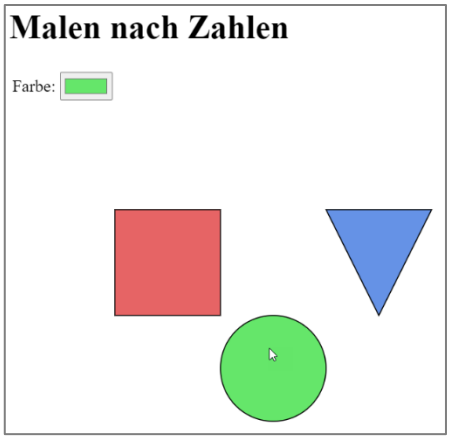


Als erstes müssen wir in TypeScript die HTML-Elemente, die du zuvor in HTML programmiert hast, heraussuchen. Wir machen das über ihre *id*. Im Bild rechts siehst du, wie die *id* in HTML mit dem Code zusammenhängt, den du gleich schreiben wirst.

Öffne *index.ts* und füge die in Rot dargestellten Codezeilen hinzu. An der Ausgabe deines Programms ändert sich dadurch noch nichts.

```
import './style.css';

const colorPicker = document.getElementById('color') as HTMLInputElement;
const image = document.getElementById('image') as HTMLElement;
```

Auf Mausklick reagieren



Jetzt können wir die Formen ausmalen, wenn man auf sie mit der Maus klickt.

Öffne *index.ts* und füge die in Rot dargestellten Codezeilen hinzu. Danach wähle eine Farbe aus und klicke auf eine der Formen. Ihre Füllfarbe sollte sich verändern, wenn du alles richtig gemacht hast.

```
import './style.css';

const colorPicker = document.getElementById('color') as HTMLInputElement;
const image = document.getElementById('image') as HTMLElement;

for(let i = 0; i < image.children.length; i++) {
  const child = image.children[i] as SVGElement;

  child.onclick = () => {
    child.style.fill = colorPicker.value;
  }
}
```

Schmetterling

Jetzt wollen wir die einfachen Formen durch den Schmetterling ersetzen. Der Code der Vektorgrafik mit dem Schmetterling ist natürlich viel länger als der Code für die drei Formen. Damit du nicht so viel tippen musst, haben wir den Schmetterling vorbereitet. Deine Aufgabe ist es, den Code zu kopieren und an der richtigen Stelle in deinem Programm einzufügen.

Code für Schmetterling kopieren

Öffne https://meet.coderdojo.net/malen-nach-zahlen-schmetterling in einem eigenen Browser-Fenster. Markiere mit der Maus den gesamten SVG-Code. Im Bild oben siehst du, was damit gemeint ist. **Achtung:** Der Code ist lang! Du musst weit nach unten scrollen, um alles zu erwischen.

Anschließend klicke mit der rechten Maustaste auf den markierten Code und wähle *Kopieren*.

Schmetterling einfügen

Öffne *index.html* und ersetze den bestehenden SVG-Codeblock durch den Code des Schmetterlings. Du musst dafür den unten durchgestrichen dargestellten Code löschen und stattdessen den kopierten Code für den Schmetterling einfügen.

```
<h1>Malen nach Zahlen</h1>

<table>
  <tr>
    <td>Farbe:</td>
    <td><input type="color" id="color" name="head" value="#e66465"></td>
  </tr>
</table>

<svg id="image" width="1000" height="600">
  <rect x="100" y="100" width="100" height="100" stroke="black" />
  <circle cx="250" cy="250" r="50" stroke="black" />
  <path d="m300,100 h100 l-50,100 Z" stroke="black" />
</svg>

<svg id="image" width="1000" height="600"  viewBox="0 10 204 172"
xmlns:svg="http://www.w3.org/2000/svg" xmlns="http://www.w3.org/2000/svg">
  <path style="stroke:#000000;stroke-width:0.26458332"
        d="m 57.689061,177.30516 ..." id="path3782" />
  ...
</svg>
```

Diesen Code löschen

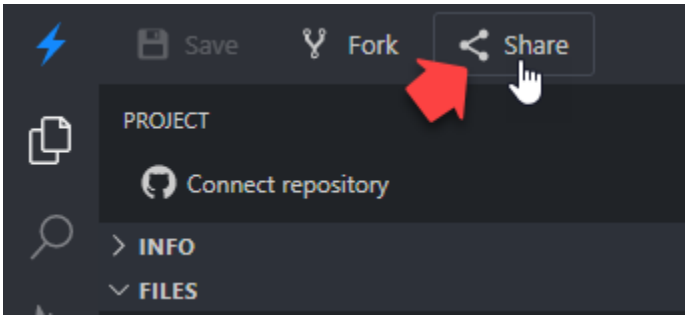
Kopierten Code einfügen

Fertig!

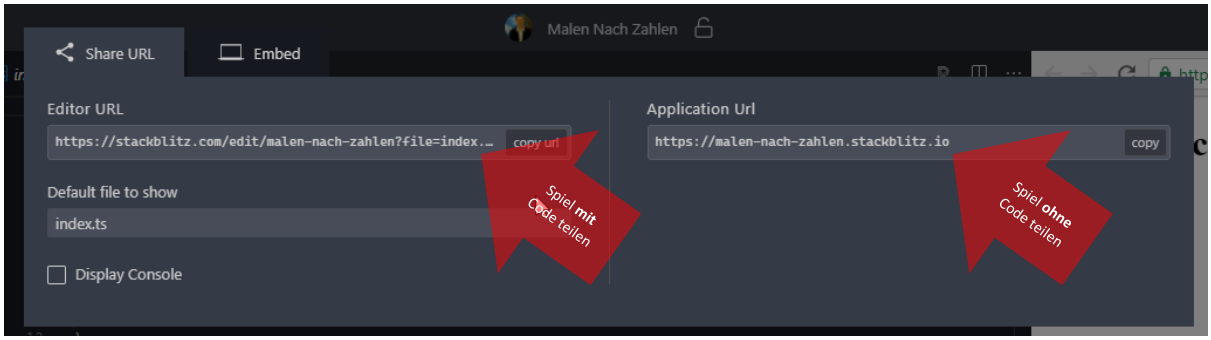
Gratuliere! Dein Spiel ist fertig. Probiere es gleich aus und male den Schmetterling mit schönen Farben aus.

Spiel teilen

Möchtest du das Spiel mit anderen teilen, damit sie es auch ausprobieren können? Wenn du etwas in Stackblitz programmierst, kannst du mit *Share* einen Link abrufen, den du weitergeben kannst.

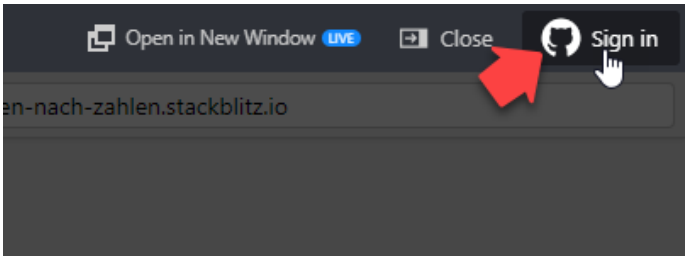


Du kannst wählen, ob du dein Spiel mit oder ohne Code teilen möchtest. Wähle mit Code, wenn die Empfängerin oder der Empfänger weiterprogrammieren möchte. Ohne Code passt besser wenn das Spiel nur gespielt werden soll und der Code für die Empfängerin oder der Empfänger weniger interessant ist.



Spiele speichern

Wenn du öfter in Stackblitz programmierst, ist es praktisch, wenn alle deine Spiele automatisch für dich gespeichert werden. Falls du das möchtest, lege dir ein kostenloses Konto bei GitHub an und melde dich damit bei Stackblitz an. Du kannst auch ein bestehendes Konto verwenden, falls du schon eines hast.



Feuerwerk mit TypeScript

Heute programmieren wir einen Feuerwerk-Simulator. Dabei kannst du das Tippen von Code auf der Tastatur üben und lernst Grundlagen der textuellen Programmierung kennen.

Ziel der Übung

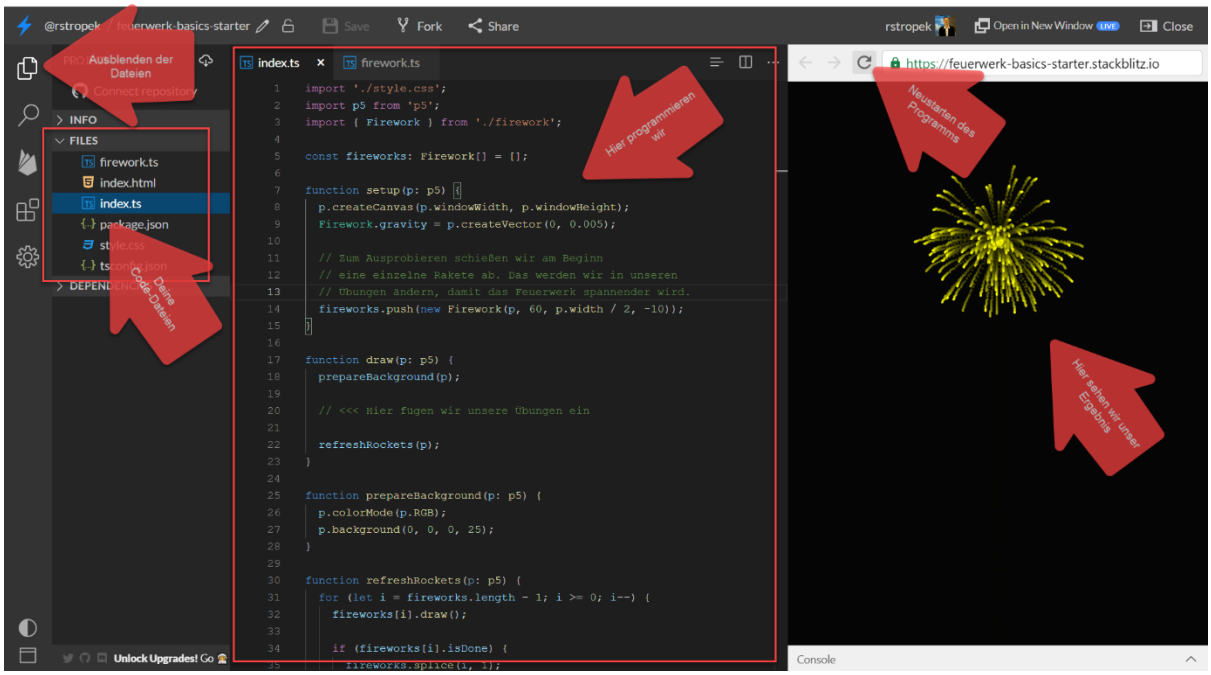
Heute wollen wir gemeinsam einen Feuerwerk-Simulator programmieren. Dabei kannst du das Tippen von Code auf der Tastatur üben und lernst Grundlagen der Programmierung wie Variablen, Schleifen und Funktionen kennen.

Du solltest für diese Übung schon ein wenig Programmiererfahrung mit einer Blockprogrammiersprache wie *Scratch* oder *Snap!* haben. Falls du noch nie Programmcode eingetippt hast, solltest du auch etwas Geduld mitbringen. Es ist am Anfang nicht immer leicht, die vielen Sonderzeichen im Code auf der Tastatur zu finden. Halte durch, Übung macht den Meister!

Start

Du brauchst für diese Übung keine spezielle Software auf deinem Computer zu installieren. Es ist nur ein aktueller Web-Browser (vorzugsweise Chrome) notwendig.

Um loszulegen, öffne die URL https://stackblitz.com/edit/feuerwerk-basics-starter?file=index.ts. Du wirst dadurch im Web-Browser ein Fenster sehen, das in etwa so aussieht:



- Links siehst du eine Liste der Dateien. Wir programmieren heute nur in *index.ts*, daher kannst du die Dateien mit dem Symbol links oben ausblenden (siehe Pfeil Ausblenden der Dateien im Bild oben).
- Unseren Programmcode schreiben wir im Textfenster in der Mitte.
- Rechts sehen wir nach jeder Codeänderung gleich das Ergebnis.

In dieser Übung programmierst du mit der Programmiersprache *TypeScript*. Außerdem verwenden wir die Game Engine *p5js*.

Experimente mit der ersten Rakete

Wir haben dir unter der oben genannten URL schon ein wenig Code zum Starten vorbereitet. Ein Programm wie den Feuerwerksimulator ganz von vorne zu programmieren, ist nichts für AnfängerInnen. Dafür braucht man etwas Übung.

Der von uns vorbereitete Code ist aber ziemlich dumm. Er schießt nach dem Laden der Ergebnisseite nur eine einzelne Rakete ab. Die dafür verantwortliche Zeile in unserem Code ist:

fireworks.push(new Firework(p, 60, p.width / 2, 75, 15));

Deine Aufgabe ist, diese Codezeile zu suchen. Tipp: Sie ist in der Funktion *setup*.

Kommentare

Die Zeilen, die mit // beginnen, sind nur Kommentare. Kommentare schreibt man in den Code, damit man später noch versteht, was man sich gedacht hat, als man den Code geschrieben hat. In diesem Fall verwenden wir Kommentare, um dir zu erklären, was die Codezeilen machen.

Abschießen der ersten Rakete

Hier ein paar Erklärungen zu der Codezeile zum Abschießen der Rakete:

- *fireworks.push* bedeutet, dass wir eine Rakete zu einer Liste von Raketen hinzufügen, die wir auf den Bildschirm zeichnen wollen. Push ist Englisch und bedeutet *schieben* oder *schubsen*. Wir werden später *fireworks.push* oft aufrufen, um mehr als eine Rakete zu zeichnen.
- *new Firework* legt eine Rakete (*Firework* heißt auf Deutsch *Feuerwerkskörper* und *new* heißt *neu*) an. In den Klammern gibt man die Parameter für die Rakete an. Sie sind durch Kommas getrennt. **Achtung:** Achte beim Experimentieren darauf, dass du die Kommas nicht löschst.

Deine Aufgabe ist, mit den Parametern zu experimentieren. Hier ein paar Hinweise dazu:

- Den ersten Parameter *p* ändere bitte nicht. Er ist aus technischen Gründen notwendig und wir gehen heute nicht näher darauf ein.
- Ändere den zweiten Parameter *60* auf einen anderen Wert zwischen *0* und *360*. Hier siehst du eine Darstellung, welcher Wert zu welcher Farbe wird:

- Beim dritten Parameter *p.width / 2* wird die Mitte des Bildschirms errechnet. Du kannst hier statt der Formel auch eine Zahl angeben. Je kleiner die Zahl, desto weiter links erscheint die Rakete, je größer desto weiter rechts. Probiere mal 20, 200 und 400 aus. Siehst du, wie sich die Position der Rakete ändert?
- Der vierte Parameter 75 steuert, wie hoch die Rakete fliegen wird. 0 bedeutet, dass sie kaum abhebt und 100 bedeutet, dass sie bis zum oberen Bildschirmrand fliegen wird. Probiere mal 0, 50, 75 und 100 aus. Siehst du, wie sich die Höhe der Rakete ändert?
- Der fünfte Parameter 15 bestimmt die Größe der Explosion. Probiere mal 10, 15 und 25 aus.

Mehrere Raketen

Code kopieren

Statt einer Rakete möchten wir jetzt mehrere Raketen gleichzeitig abschießen. Deshalb müssen wir die Codezeile zum Abschießen der Rakete, mit der wir gerade experimentiert haben, kopieren. Weißt du schon, wie das geht?

- Stelle deinen Cursor in die Codezeile, die du kopieren möchtest.
- Drücke Strg + C (erst Strg, Strg gedrückt halten, dann C, danach beide Tasten loslassen), um die Zeile zu kopieren.
- Drücke Strg + V, um die Zeile einzufügen.
- Jetzt hast du die Zeile zwei Mal im Code.
- Deine Aufgabe ist es, die Zeile so oft zu kopieren, dass sie fünf Mal im Code ist.

Es macht natürlich keinen Sinn, wenn alle fünf Raketen an der gleichen Stelle losfliegen. Deine nächste Aufgabe ist es, den Code so zu ändern, dass die Raketen über die ganze Bildschirmbreite verteilt sind. So würde das aussehen:

```
fireworks.push(new Firework(p, 60, p.width * 0 / 4, 75, 15));
fireworks.push(new Firework(p, 60, p.width * 1 / 4, 75, 15));
fireworks.push(new Firework(p, 60, p.width * 2 / 4, 75, 15));
fireworks.push(new Firework(p, 60, p.width * 3 / 4, 75, 15));
fireworks.push(new Firework(p, 60, p.width * 4 / 4, 75, 15));
```

Konstanten

Fällt dir auf, dass du die Farbe (60) in jeder Zeile drinnen hast? Wenn du die Farbe ändern willst, musst du fünf Mal die Zahl ändern. Das ist unpraktisch, oder? In einem solchen Fall verwendet man eine Konstante. Deine Aufgabe ist es, den Code so zu ändern, dass die Farbe nur noch einmal enthalten ist und dadurch das Ändern der Farbe einfacher wird. So muss der Code nach der Änderung aussehen:

```
const color = 180;
fireworks.push(new Firework(p, color, p.width * 0 / 4, 75, 15));
fireworks.push(new Firework(p, color, p.width * 1 / 4, 75, 15));
fireworks.push(new Firework(p, color, p.width * 2 / 4, 75, 15));
fireworks.push(new Firework(p, color, p.width * 3 / 4, 75, 15));
fireworks.push(new Firework(p, color, p.width * 4 / 4, 75, 15));
```

Deine nächste Aufgabe ist, nach dem gleichen Prinzip wie bei *color* auch für die Flughöhe der Raketen (vorletzter Parameter) und für die Größe der Explosion (letzter Parameter) Konstanten einzufügen und zu verwenden. So muss der Code nach der Änderung aussehen:

```
const color = 180;
const height = 75;
const size = 15;
fireworks.push(new Firework(p, color, p.width * 0 / 4, height, size));
fireworks.push(new Firework(p, color, p.width * 1 / 4, height, size));
fireworks.push(new Firework(p, color, p.width * 2 / 4, height, size));
fireworks.push(new Firework(p, color, p.width * 3 / 4, height, size));
fireworks.push(new Firework(p, color, p.width * 4 / 4, height, size));
```

Zufallszahlen

Immer die gleiche Farbe ist langweilig, oder? Beim Programmieren kannst du ganz einfach Zufallszahlen erzeugen. Das funktioniert ähnlich wie bei Spielen durch Würfeln. Beim Programmieren kann man aber nicht nur Zufallszahlen zwischen 1 und 6 “würfeln”, sondern beliebige Zahlen zufällig erzeugen lassen. Deine Aufgabe ist es, die Farbe zufällig vom Computer auswählen zu lassen. Du musst dafür nur den Wert der Konstanten *color* wie folgt anpassen:

```
const color = p.random(360);
```

Jedes Mal, wenn du das Programm neu startest, wird der Computer eine zufällige Farbe verwenden.

Raketen im Intervall abschießen

Endlosschleife

Wir möchten unser Programm jetzt so erweitern, dass jede Sekunde Raketen abgeschossen werden. Dafür brauchen wir eine Schleife. Bei unserem ersten Experiment möchten wir fortlaufend neue Raketen abschießen. Deshalb verwenden wir eine Endlosschleife, also eine Schleife, die für immer läuft. In Scratch wäre das die Wiederhole fortlaufend-Schleife.

Deine Aufgabe ist es, den Code so zu ändern, dass jede Sekunde Raketen starten. So muss der Code nach der Änderung aussehen:

```
while (true) {
  const color = p.random(360);
  const height = 75;
  const size = 15;
  fireworks.push(new Firework(p, color, p.width * 0 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 1 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 2 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 3 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 4 / 4, height, size));
  await delay(1000);
}
```

Die Zeile *await delay(1000);* wartet 1000 Millisekunden, also eine Sekunde. Deine Aufgabe ist es, mit dieser Zahl zu experimentieren. Wie sieht das Ergebnis aus, wenn du 2000 statt 1000 verwendest? Wie beim Wert von 500?

for-Schleife

Bereit für eine Herausforderung? Lass uns genau 10 Mal Raketen abschießen, dann soll das Feuerwerk zu Ende sein. In solchen Fällen verwendet man eine for-Schleife. Deine Aufgabe ist es, die *while*-Schleife in eine for-Schleife umzubauen. So muss der Code nach der Änderung aussehen:

```
for (let i = 0; i < 10; i++) {
  const color = p.random(360);
  const height = 75;
  const size = 15;
  fireworks.push(new Firework(p, color, p.width * 0 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 1 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 2 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 3 / 4, height, size));
  fireworks.push(new Firework(p, color, p.width * 4 / 4, height, size));
  await delay(1000);
}
```

Hier ein paar Tipps zur *for*-Schleife:

- Sie beginnt bei Null zu zählen (*i = 0*).
- Sie läuft solange *i* kleiner als 10 ist, also zehn Mal.
- Bei jedem Schleifendurchlauf wird i um eins erhöht (*i++*).

Eine weitere *for*-Schleife

Hmmm, könnten wir die for-Schleife nicht auch verwenden, um die vielen, kopierten Aufrufe von *fireworks.push* zu vereinfachen. Deine Aufgabe ist es, eine *for*-Schleife zu verwenden, um aus den fünf *fireworks.push*-Aufrufen einen zu machen. So muss der Code nach der Änderung aussehen:

```
for (let i = 0; i < 10; i++) {
  const color = p.random(360);
  const height = 75;
  const size = 15;
```

```
  for (let j = 0; j < 5; j++) {
    fireworks.push(new Firework(p, color, (p.width * j) / 4, height,
        size));
  }
  await delay(1000);
}
```

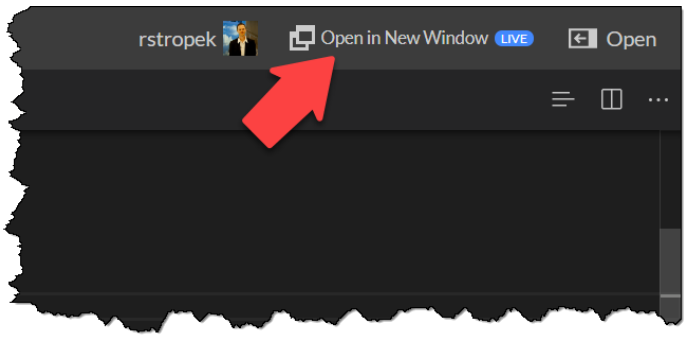
Noch mehr Raketen…

Was müssten wir ändern, damit wir nicht nur fünf Raketen parallel abschießen können, sondern beliebig viele? Wir müssen die Obergrenzen für j anpassen und die Berechnungsformel für die X-Koordinate (horizontale Abschussposition) ändern. Das ist deine nächste Aufgabe. So muss der Code nach der Änderung aussehen:

```
for (let i = 0; i < 10; i++) {
  const color = p.random(360);
  const height = 75;
  const size = 15;
  const numberOfRockets = 10;
  for (let j = 0; j < numberOfRockets; j++) {
    fireworks.push(new Firework(p, color,
        (p.width * j) / (numberOfRockets - 1), height, size));
  }
  await delay(1000);
}
```

Wow, da tut sich ordentlich was, oder?

Hier noch ein Tipp: Wenn du ein richtig großes Feuerwerk sehen willst, das über den ganzen Bildschirm geht, klicke auf Open in *New Window* (rechts oben im Bildschirmfenster), drücke danach *F11* (Vollbildschirm) und zum Schluss F5 zum neu Laden. Cool, oder? Um aus der Vollbildschirmanzeige rauszukommen, drücke einfach nochmals F11.



Raketen in Formation

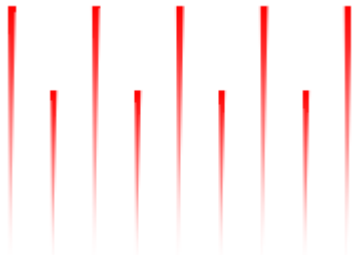
Abwechselnde Höhe

Jetzt wollen wir unsere Raketen in Formation fliegen lassen. Damit wir vom gleichen Startpunkt ausgehen, ist deine Aufgabe, den Code zu ändern, damit er so aussieht:

```
for (let i = 0; i < 100; i++) {
  const height = 75;
  const size = 15;
  const numberOfRockets = 5;
  for (let j = 0; j < numberOfRockets; j++) {
    fireworks.push(new Firework(p, p.random(360),
        (p.width * j) / (numberOfRockets - 1), height, size));
  }
```

```
  await delay(1000);
}
```

Als erstes möchten wir probieren, wie es aussieht, wenn man jede zweite Rakete nur 3⁄4 mal so hoch fliegen lässt. Hier eine Skizze, wie die Flughöhe aussehen soll:



Damit wir diese Herausforderung meistern können, brauchen wir ein wenig Mathematik. In der Schule hast du sicher schon Operationen wie Addition, Subtraktion oder Multiplikation gelernt. Es gibt beim Programmieren jedoch eine weitere Operation, die man häufig braucht: *Modulo*. Modulo gibt den Rest einer Division zurück.

Hier ein paar Beispiele:

Division	Ergebnis	Rest
9 / 5	1	4
10 / 5	2	0
5 / 2	2	1
6 / 2	3	0
0 / 3	0	0

Was hilft uns das bei unserer Herausforderung? Wenn man fortlaufende Zahlen (0, 1, 2, 3, 4, 5 etc.) modulo 2 rechnet, erhält man abwechselnd 0 und 1. Um genau zu sein erhalten wir bei geraden Zahlen die 0 als Ergebnis der Modulo-Operation und bei ungeraden 1. Hier als Tabelle dargestellt:

Zahl	Ergebnis der Zahl modulo 2
0	0
1	1
2	0
3	1
4	0
5	1

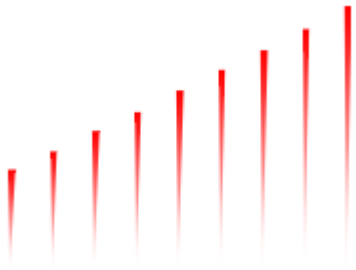
Modulo schreibt man im Code mit dem Zeichen *%*. Deine Aufgabe ist jetzt, mit diesem Wissen den geforderten Formationsflug unserer Raketen zu programmieren. So muss der Code nach der Änderung aussehen:

```
for (let i = 0; i < 100; i++) {
  const size = 15;
  const numberOfRockets = 8;
  for (let j = 0; j < numberOfRockets; j++) {
    let height: number;
    if (j % 2 == 0) height = 90;
    else height = 67.5;
    fireworks.push(new Firework(p, p.random(360),
        (p.width * j) / (numberOfRockets - 1), height, size));
  }
```

```
  await delay(1000);
}
```

Diagonale

Als zweite Formation möchten wir, dass die Raketen eine Diagonale bilden. Die erste Rakete soll auf Höhe 25 fliegen, die letzte auf Höhe 100. Hier eine Skizze, wie die Flughöhe aussehen soll:



Hier brauchen wir wieder etwas Mathematik. In diesem Fall hilft uns Prozentrechnung. Falls du das in der Schule noch nicht gelernt hast, ist das kein Problem. Wenn du magst, bitte eine ältere Person, eine CoderDojo-Mentorin oder einen CoderDojo-Mentor, dir das Prinzip der Prozentrechnung zu erklären. Du kannst aber auch den unten angeführten Code abtippen und Geduld haben, bis ihr Prozentrechnung in der Schule lernt.

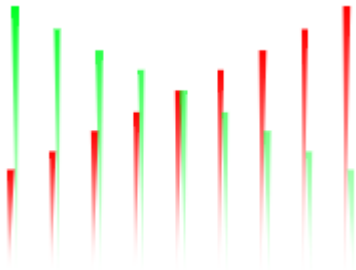
Die Höhe in unserem Fall berechnet sich wie folgt:

- Das Minimum ist 25.
- Zu diesem Minimum addieren wir den Abstand zur Maximalhöhe *(100 - 25 = 75)* multipliziert mit dem Index der Rakete *j* dividiert durch die Anzahl der Raketen *numberOfRockets minus 1*.
- Die Formel lautet also: *height = 25 + 75 * j / (numberOfRockets - 1)*

Deine Aufgabe ist es, diese Formel in unseren Code einzubauen. So muss der Code nach der Änderung aussehen:

```
for (let i = 0; i < 100; i++) {
  const size = 15;
  const numberOfRockets = 8;
  for (let j = 0; j < numberOfRockets; j++) {
    let height = 25 + 75 * j / (numberOfRockets - 1);
    fireworks.push(new Firework(p, p.random(360),
        (p.width * j) / (numberOfRockets - 1), height, size));
  }

  await delay(1000);
}
```

Einen besonders schönen Effekt bekommst du, wenn du zwei Diagonalen gegenläufig erzeugst. Eine steigt von links nach rechts und die andere von rechts nach links.



Das ist schwer theoretisch zu erklären, man muss es probieren. Deine Aufgabe ist es, den Code wie folgt zu ändern:

```
for (let i = 0; i < 100; i++) {
  const size = 15;
  const numberOfRockets = 8;
  for (let j = 0; j < numberOfRockets; j++) {
    let height = 25 + 75 * j / (numberOfRockets - 1);
    fireworks.push(new Firework(p, 60,
        (p.width * j) / (numberOfRockets - 1), height, size));
    fireworks.push(new Firework(p, 120,
        (p.width * j) / (numberOfRockets - 1), 25 + 100 - height, size));
  }

  await delay(3000);
}
```

Text hinzufügen

Zum Abschluss möchten wir noch einen Glückwunsch (z. B. Neujahrswünsche, Geburtstagswünsche) zu unserem Feuerwerk hinzufügen. Das kannst du mit wenigen Zeilen Code in der *draw*-Funktion erledigen. Deine letzte Aufgabe für heute ist es, die *draw*-Funktion zu suchen und die folgenden Zeilen für die Textausgabe hinzuzufügen:

```
function draw(p: p5) {
  p.colorMode(p.RGB);
  p.background(0, 0, 0, 25);

  p.fill('red');
  p.textSize(60);
  p.textStyle(p.BOLD);
  p.textAlign(p.CENTER, p.CENTER);
  p.text('Schönes,\nneues Jahr\n🍾🥂🎉', p.width / 2, p.height / 4);

  // Hier folgt die for-Schleife, die wir zuvor programmiert haben
  for (let i = fireworks.length - 1; i >= 0; i--) {
    ...
  }
}
```

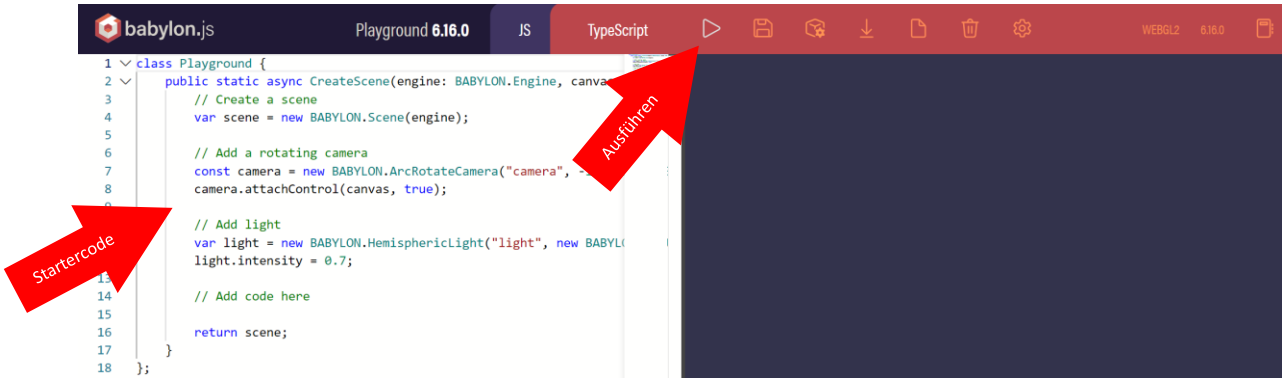
Erste Schritte bei 3D-Programmierung

In dieser Übung probieren wir die Programmierung von dreidimensionalen Welten mit TypeScript und Babylonjs aus. Im Vergleich zu Scratch oder den vorigen TypeScript-Übungen haben wir es – wie der Name schon sagt – nicht nur mit zwei (X-Achse und Y-Achse), sondern drei (Z-Achse) Dimensionen zu tun. Viele von euch kennen dreidimensionale Welten aus Spielen wir Minecraft oder von VR-Brillen.

Wer diese Übung programmieren möchte, sollte vorher Erfahrung mit Scratch und auch ein wenig TypeScript gesammelt haben. Am besten geht das mit den in diesem Buch zuvor enthaltenen Beispielen.

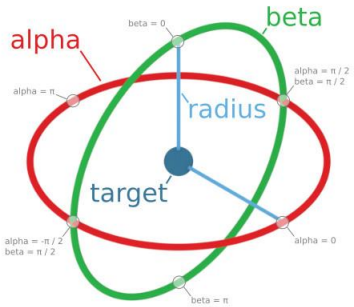
Start

Wir programmieren diese Übung in der Babylonjs-Übungsumgebung (wird in Englisch als *Playground*, also „Spielplatz“, bezeichnet). Damit du etwas leichter starten kannst, haben wir etwas Startercode vorbereitet. Klicke auf den Link https://playground.babylonjs.com/#5FUGGN, um mit dieser Übung zu beginnen.



Nimm dir etwas Zeit, den Code durchzusehen.

- Der erste für uns wichtige Schritt ist das Anlegen einer ***Szene*** (ähnlich einer Szene im Theater). Das geschieht in Zeile 4.
- Als nächstes fügen wir in Zeile 7 eine ***Kamera*** hinzu und sagen Babylonjs, dass man die Kamera mit der Maus steuern können soll (Zeile 8). Wir verwenden in diesem Beispiel eine Kamera, die – wie rechts gezeigt – rund um unsere Figuren auf einer Kreisbahn rotiert.
- Die dritte Komponente unseres Beispiels ist die ***Beleuchtung*** (Zeilen 11 bis 12). Ohne Licht würden wir Figuren, die wir zur Szene hinzufügen, nicht sehen.

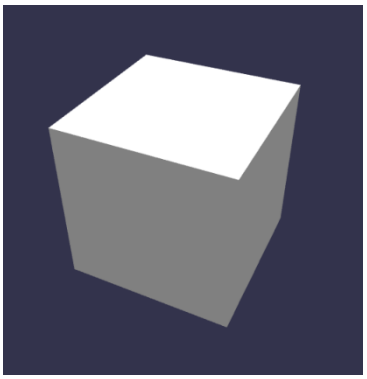


Noch ist unsere Szene leer. Der Startercode enthält einen Hinweis *Add code here*. Dort werden wir unsere Figuren hinzufügen.

Hinzufügen einer Box

Als erstes fügen wir eine Box zu unserer Szene hinzu. Ersetze dafür den Hinweis *Add code here* durch folgenden Code:

```
const box = BABYLON.MeshBuilder.CreateBox("box", { size: 1 });
```



Nach dem Tippen des Codes, drücke auf den *Ausführen*-Knopf (siehe Hinweispfeil in Abbildung oben). Du solltest eine Box am Bildschirm sehen. Versuche, die Ansicht mit Hilfe der Maus zu verändern (Klicken-und-Ziehen, Mausrad).

Hinweis: Es sieht auf den ersten Blick vielleicht so aus, als würdest du die Box drehen. Das ist aber nicht so. Du bewegst in Wirklichkeit die Kamera rund um die Box.

Experimentiere jetzt ein wenig mit dem Code:

- Ändere die Lichtintensität (*light.intensity*), um die Box stärker und weniger stark zu beleuchten.
- Ändere die Größe der Box.

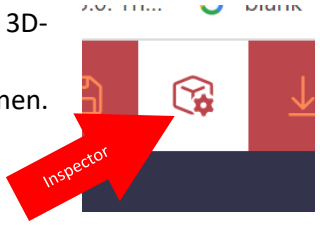
Vergiss nicht, nach jedem Experiment den *Ausführen*-Knopf zu drücken.

Hinzufügen einer Kugel

Fügen wir eine zweite Figur zu unserer Szene hinzu, diesmal eine Kugel (Englisch *Sphere*).

```
const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 0.1});
```

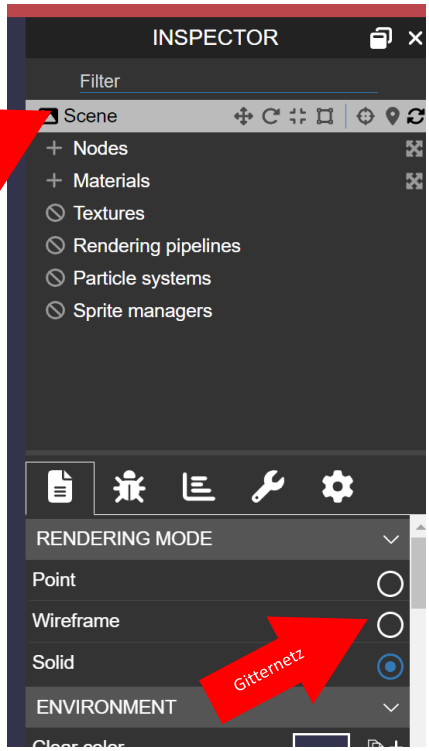
Drücke auf *Ausführen*, siehst du die Kugel? Nein? Stimmt, denn sie ist *innerhalb* der Box und daher im Moment unsichtbar. Vielleicht ist es dir in Scratch schon einmal passiert, dass du eine Figur an die gleiche Stelle wie eine andere gesetzt hast und dadurch die eine Figur die andere verdeckt hat. Genauso ist es hier mit Box und Kugel – nur eben in drei Dimensionen.



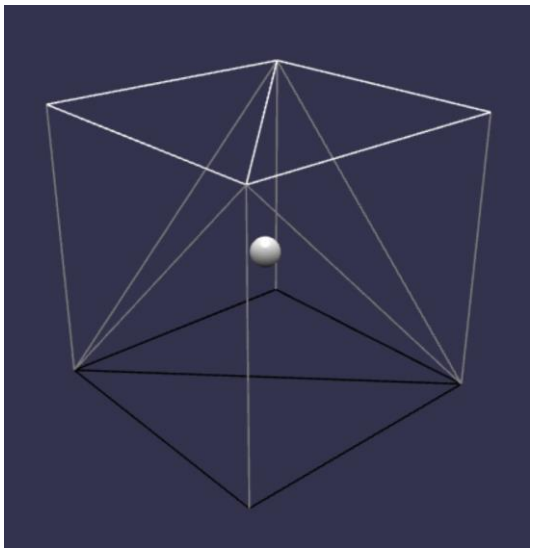
Machen wir die Kugel sichtbar. Dafür verwenden wir das *Gitternetzmodell*. 3D-Welten im Computer bestehen nämlich aus Gitternetzen. Stell dir vor, du würdest eine Figur aus Draht formen und anschließend Stoff darüber spannen. So funktionieren 3D-Modelle im Computer.

Zum Umschalten auf das Gittermodell aktivere den *Inspector*.

Im *Inspector* aktivierst du als erstes mit einem Klick die Szene (siehe Abbildung rechts). Danach kannst du mit *Wireframe* auf das Gitternetz umschalten.



Nach dem Umschalten siehst du die Kugel in der Box:



Tipp: Verwende die Maus, um die Kamera zu bewegen. Dadurch kannst du noch leichter erkennen, dass die Kugel in der Box ist. Experimentiere ruhig noch ein wenig mit dem *Inspector*.

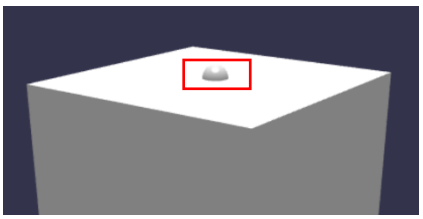
Vergiss nicht, am Ende deiner Experimente wieder auf *Solid* umzustellen, um die Box wieder vollständig anzuzeigen.

Position der Kugel ändern

Wir wollen die Kugel sichtbar machen, indem wir sie an eine andere Position verschieben. Füge nach dem Anlegen der Kugel die rot dargestellte Codezeile hinzu. Sie ändert die Position der Kugel auf der Y-Achse.

```
const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 0.1});
sphere.position.y = 0.5;
```

Die gesamte Box hat eine Größe von 1. Gerechnet vom Mittelpunkt (Null) geht sie 0.5 nach oben und 0.5 nach unten. Wenn wir den Mittelpunkt der Kugel daher um 0.5 nach oben schieben, sehen wir genau den oberen Teil der Kugel (siehe Abbildung rechts).



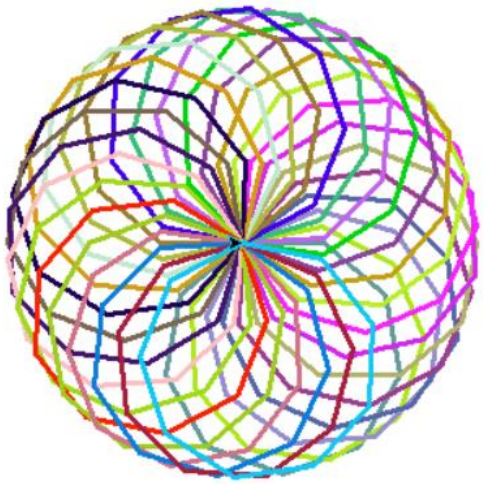
Jetzt bist du an der Reihe:

- Ändere die Position um mehr als 0.5.
- Ändere statt der Y-Position einmal die X- oder die Z-Position. Wo ist die Kugel dann sichtbar?
- Was passiert, wenn du X, Y und Z-Position auf 0.5 setzt? Kannst du erklären, warum die Ausgabe so ist wie sie ist?

Gratulation! Du hast deine erste 3D-Welt programmiert. Wenn du magst, kannst du deine Szene noch um weitere Boxen oder Kugeln ergänzen.

Von Scratch zu Python

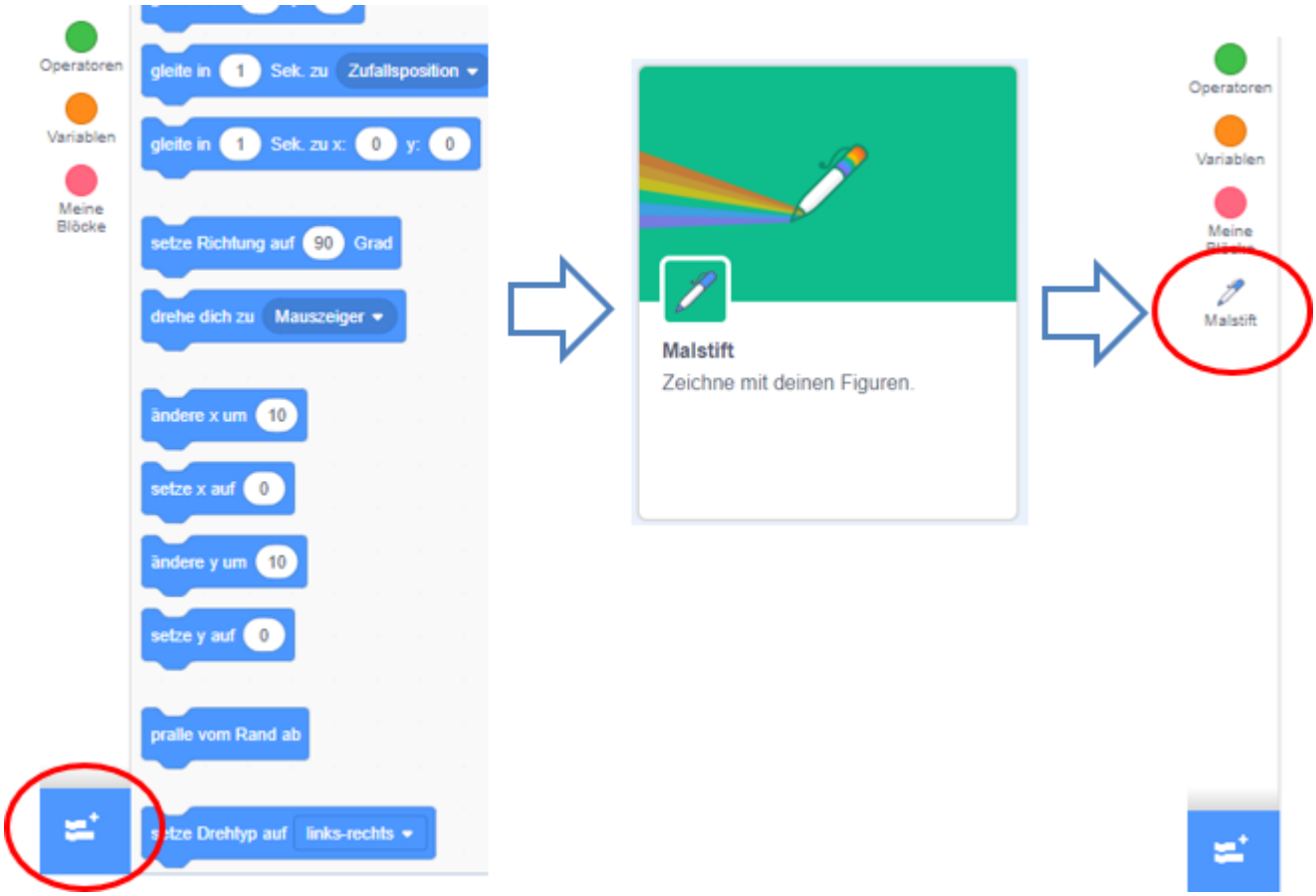
Du zeichnest eine geometrische Figur in Scratch und dann übersetzt du das Programm in die Programmiersprache Python.



Scratch

Laden des Malstifts

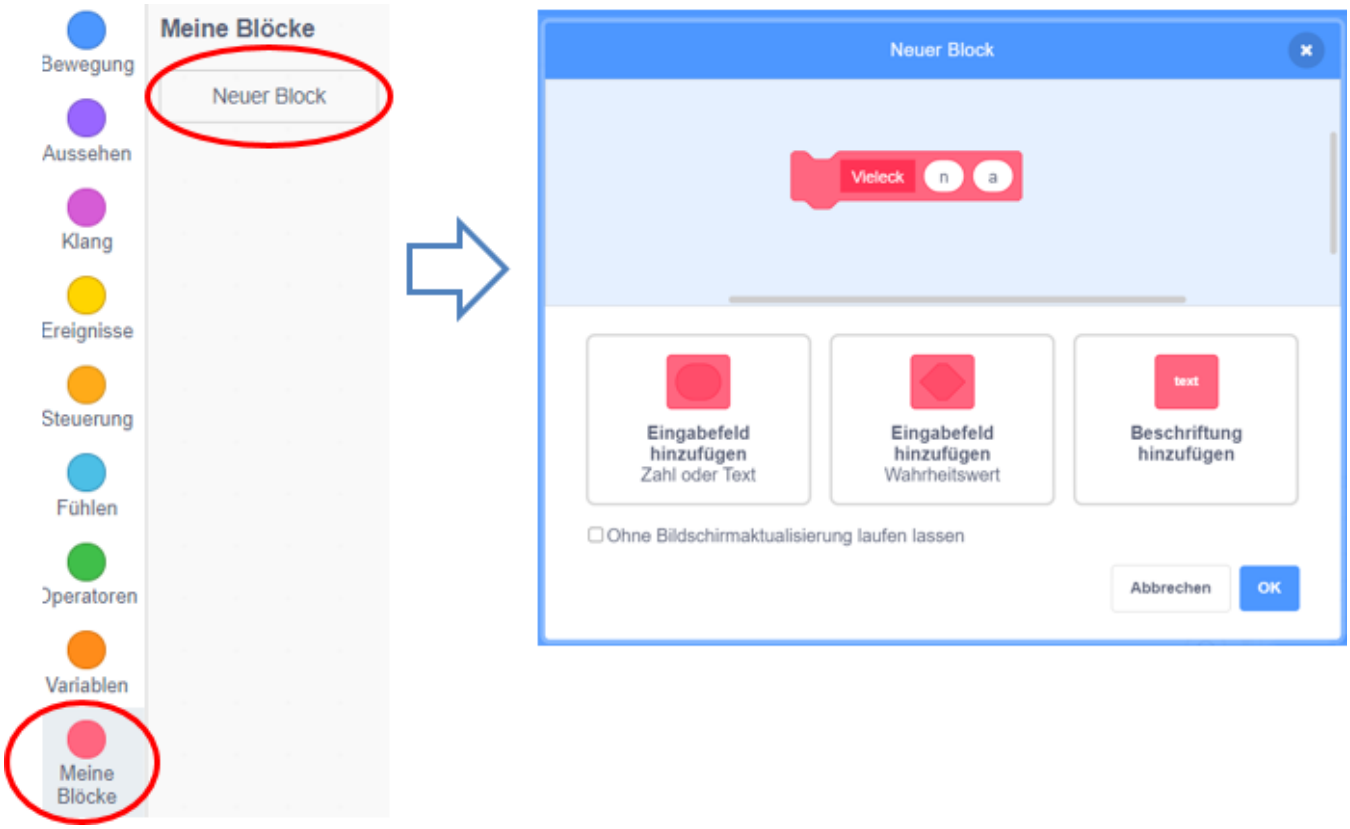
Um in Scratch zu zeichnen, muss man zuerst die Erweiterung „Malstift“ laden. Klicke dazu auf das Symbol in der linken unteren Ecke und wähle dann den Malstift aus. Damit bekommst du einen neuen Eintrag „Malstift“ im Menü am linken Bildschirmrand und eine Menge neuer Blöcke, die du zum Zeichnen verwenden kannst.



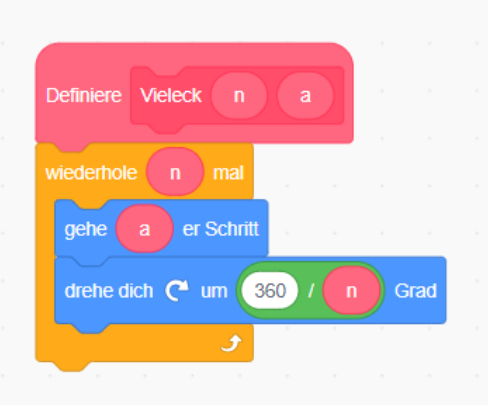
Neuer Block zum Zeichnen von Vielecken

Um ein beliebiges Vieleck (Fünfeck, Achteck, Siebzehneck, …) zu zeichnen, definieren wir einen eigenen Block, wo wir uns aussuchen können, wie viele Ecken und welche Seitenlänge es haben soll.

Gehe dazu unter dem Menüpunkt „Meine Blöcke“ auf „Neuer Block“. Nenne den Block Vieleck und füge zwei Eingabefelder für Zahl hinzu und nenne die Eingabefelder n (für die Anzahl der Ecken und a (für die Seitenlänge des Vielecks).



Um ein n-Eck zu zeichnen, müssen wir einen Schritt mit der Seitenlänge a gehen, dann eine Drehung um 360/n machen und das Ganze n-Mal wieder wiederholen. (Für ein regelmäßiges Viereck, also ein Quadrat, wären das n Wiederholungen und dazwischen eine Drehung um 90 Grad.) Die roten Felder mit den Variablen n und a, die du hier brauchst, holst du einfach aus dem ersten Block „Definiere Vieleck“, indem du die gewünschte Variable einfach mit der Maus nach unten ziehst.



Festlegen von Variablen

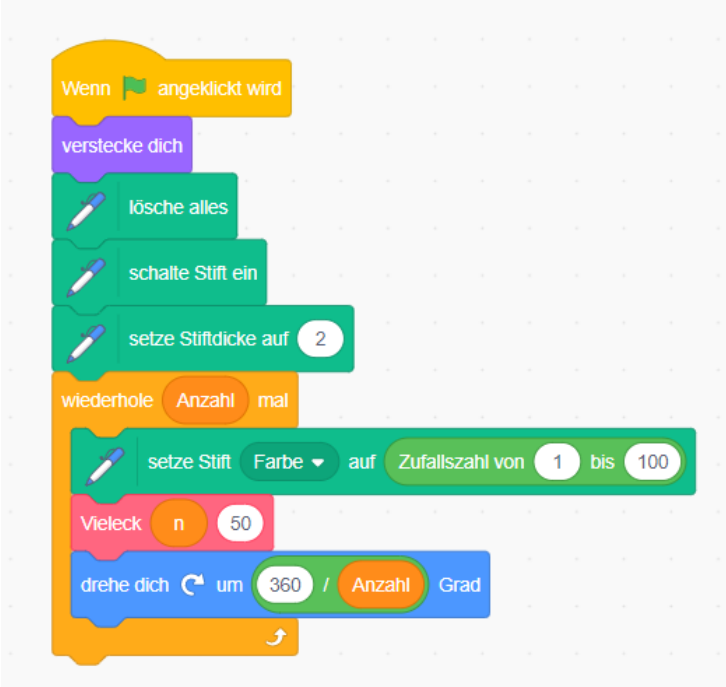
Wir definieren nun zwei Variablen Anzahl (für die Anzahl der Vielecke) und n (für die Anzahl der Ecken im Vieleck). Gehe dazu unter „Variablen“ auf „Neue Variable“ und gib die passenden Namen ein. Wenn du mit der rechen Maustaste auf die angezeigte Variable klickst, kannst du auch einen verstellbaren Schieberegler daraus machen.

Mit vielen Vielecken ein buntes Muster zeichnen

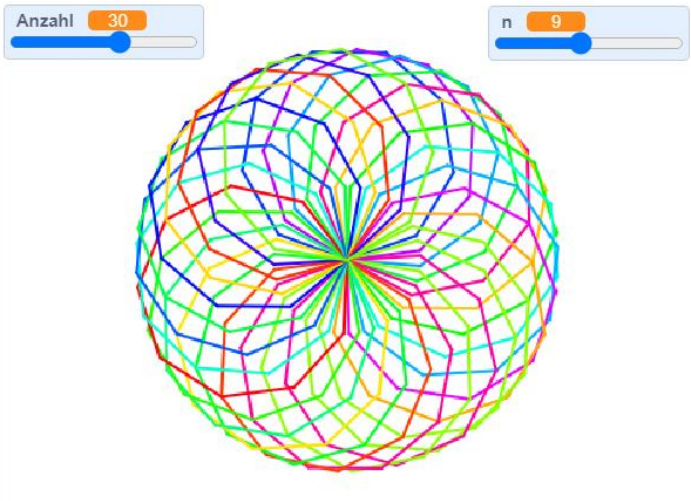
Mit dem selbst definierten Block zum Zeichnen von Vielecken können wir nun ein buntes geometrisches Muster mit vielen Vielecken zeichnen.

Wenn das Spiel gestartet wird, dann führen wir folgende Schritte durch:

- Wir verstecken unsere Figur (Katze), da wir nur unsere Zeichnung haben wollen.
- Wir löschen alles, um immer wieder mit einem leeren Fenster zu starten.
- Wir schalten den Malstift ein.
- Die Stiftdicke setzten wir auf 2.
- Dann zeichnen wir so viele Vielecke, wie wir bei Anzahl ausgewählt haben. Dazu verwenden wir eine Schleife mit „Wiederhole Anzahl mal“,
- Die Farbe ändern wir bei jedem Vieleck zufällig.
- Zum Zeichnen des Vielecks verwenden wir unseren definierten Block, wobei wir als n die erstellte Variable verwenden und als Seitenlänge a = 50.
- Nachdem ein Vieleck gezeichnet wurde, drehen wir uns um 360/Anzahl Grad, um dann das nächste Vieleck zu zeichnen.



So kannst du zum Beispiel 30 verschiedenfärbige 9-Ecke zeichnen.



Python

Starte für Python die Programmierumgebung und öffne eine neue Datei und speichere die Datei auf deinem Computer.

Laden von Packages

So wie du auch in Scratch eine eigene Erweiterung für den Malstift geladen hast, musst du auch in Python eine Erweiterung (Package) für das Zeichnen mit der Schildkröte (turtle) laden.

Auch für das Zufallszahlen (die wir dann für die zufällige Auswahl der Farben brauchen) ist es notwendig eine Funktion aus dem Package random zu laden.

```
from turtle import *
from random import random
```

Da wir aus dem Package *turtle* mehrere Funktionen benötigen, laden wir hier alle Funktionen (mit dem Symbol *). Aus dem Package *random* laden wir nur die Funktion *random*.

Neue Funktion zum Zeichnen von Vielecken

So wie wir in Scratch einen eigenen Block zum Zeichnen von Vielecken erstellt haben, definieren wir hier in Python eine eigene Funktion vieleck ebenfalls mit den Eingabewerten n und a.

```
def vieleck(n, a):
    for i in range(n):
        forward(a)
        right(360/n)
```

Das n-malige Wiederholen geschieht mit einer for-Schleife, wobei die Variable i mitzählt.

Achtung: In Python beginnt man mit 0 beim Zählen und nicht bei 1 und die Zahl, die in range angegeben ist, zählt nicht mehr dazu. Also, wenn n = 10 ist, dann zählt i in range(n)von 0, 1, 2, … bis 9. Das sind aber dann trotzdem 10 Zahlen, du kannst gerne nachzählen.

Alles was in zu dieser for-Schleife gehört muss um vier Leerzeichen eingerückt werden (so wie du in Scratch die Blöcke in den Wiederholen-Block hineinpackst). forward(a) lässt die Turtle um a Schritte vorwärts gehen, right(360/n) machte eine Rechtsdrehung um 360/n Grad.

Festlegen von Variablen

Wir definieren, so wie Scratch, Variablen für die Anzahl der Vielecke und die Anzahl der Ecken eines Vielecks. Auch die Seitenlänge des Vielecks legen wir gleich fest. Um deinen Python-Code auch später noch gut zu verstehen, ist es hilfreich, wenn du Kommentare einfügst. Alles was du hinter einem # schreibst, dient als Erklärung für dich, hat aber keinen Einfluss auf das Programm selbst.

```
anzahl = 30 # Anzahl der Vielecke
n = 9       # Anzahl der Ecken des Vielecks
a = 50      # Seitenlänge des Vielecks
```

Mit vielen Vielecken ein buntes Muster zeichnen

Nun können wir mit dem eigentlichen Programm beginnen:

- Wir löschen das Fenster zu Beginn.
- Dann schalten wir den Turbo unserer Turtle ein und erhöhen die Geschwindigkeit auf 100.

- Die Stiftdicke setzen wir auf 3.
- In einer Schleife machen wir wieder so viele Wiederholungen, wie wir als anzahl vorgegeben haben.
- Die Stiftfarbe wählen wir zufällig. Eine Zufallszahl bekommst du mit dem Befehl random(). Eine Farbe kann man über ihren Rot-, Grün- und Blau-Anteil festlegen, daher verwenden wir hier 3 Zufallszahlen.
- Das Vieleck zeichnen wir mit der zuvor definierten Funktion.
- Nach jedem Vieleck machen wir eine Drehung um 360/anzahl Grad.

```
clear()
speed(1000)
pensize(3)

for i in range(anzahl):
pencolor(random(), random(), random())
vieleck(n, a)
left(360/anzahl)
```

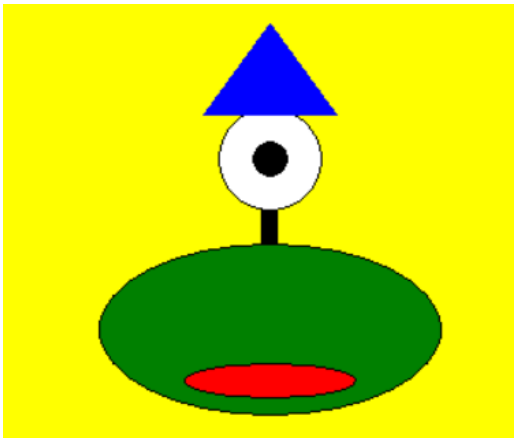
Wenn du diesen Code fertig geschrieben hast, speichere die Datei nochmals mit File -> Save (oder drücke STRG+S). Um dann das Programm zu starten kannst du im Menü Run -> Run Module auswählen oder schneller einfach F5 drücken.

Du erhältst dann das Bild, das zu Beginn dieser Anleitung zu sehen ist (vermutlich mit etwas anderen Farben).

Erweiterungsmöglichkeiten

- Wie könntest du (zumindest annähernd) Kreise zeichnen?
- Du könntest im Programm abfragen, wie viele Ecken das n-Eck haben soll, wie viele Vielecke gezeichnet werden sollen oder wie groß sie sein sollen. (Sieh dir dazu die Übungsanleitung „Zahlen raten mit Python“ an.)

Wir zeichnen einen Alien



Mit dem Package Tkinter zeichnen wir einen Alien und lassen ihn auf Tastendruck reagieren. Die Beschreibung erfolgt in Anlehnung an Beispiele aus dem Buch Programmieren supereasy.

Starte für Python die Programmierumgebung und öffne eine neue Datei und speichere die Datei als Dateityp auf deinem Computer. Wähle als Dateityp *.pyw*, wenn du mit Windows arbeitest.

Laden von Packages

Für dieses Beispiel verwenden wir das Package tkinter. Damit kann man Fenster und Schaltflächen für eine grafische Benutzeroberfläche erstellen, aber auch Formen zeichnen.

```
from tkinter import *
```

Ein Fenster mit einer Leinwand erstellen

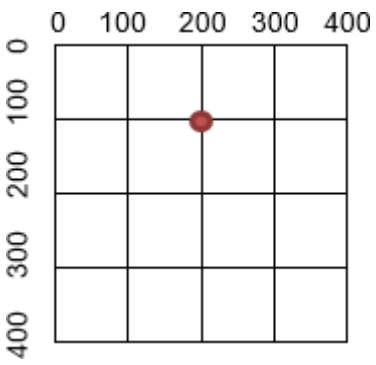
Zuerst erzeugen wir ein neues Fenster, das wir window nennen. Diesem Fenster kann man auch einen Namen geben (z. B. „Alien“). Dieses Fenster erscheint, wenn man das Programm mit F5 laufen lässt. Damit dieses Fenster immer gleich im Vordergrund ist, kann man das bei den Attributen einstellen.

```
window = Tk()
window.title("Alien")
window.attributes("-topmost", 1)
c = Canvas(window, height=400, width=400, bg="yellow")
c.pack()
```

Um auch etwas zeichnen zu können, benötigen wir eine Leinwand (auf Englisch Canvas). Diese Leinwand kommt in unser Fenster, wir legen die Höhe (height), die Breite (width) und die Farbe des Hintergrunds (bg für background) fest. Die Leinwand nennen wir kurz c und packen sie in unser Fenster.

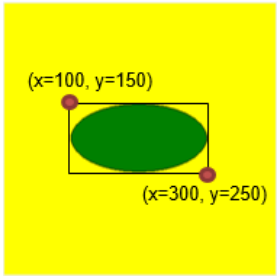
Das Koordinatensystem in Tkinter

In Tkinter ist der Koordinatenursprung in der linken oberen Ecke des Fensters. Wenn unser Fenster 400 Pixel breit und 400 Pixel hoch ist, dann sieht das in etwa so aus wie im Bild rehts. Der rot eingezeichnete Punkt wird dann durch seine waagrechte Entfernung x = 200 und seine Entfernung nach unten y = 100 beschrieben.



Einen Alien zeichnen

Nun können wir einfache Formen erzeugen und damit einen Alien auf unsere Leinwand c zeichnen:



- Der Körper ist oval und grün. Die Zahlen in c.create_oval sind immer abwechselnd x und y Koordinaten und geben die linke obere Ecke und die rechte untere Ecke an.
- Das Auge ist ein weißer Kreis, der ebenfalls durch zwei Punkte festgelegt wird.
- Die Pupille ist ein schwarzer Kreis in der Mitte des Auges.
- Der Mund ist oval und rot.
- Der Hals ist eine Linie mit Dicke (width) 10, die das Auge und den Körper verbindet.
- Der Hut ist ein blau gefülltes Dreieck, das durch drei Eckpunkte beschrieben wird: A(160, 75), B(240, 75), C(200, 20).

```
# Wir zeichnen einen Alien
körper = c.create_oval(100, 150, 300, 250, fill="green")
auge = c.create_oval(170, 70, 230, 130, fill="white")
pupille = c.create_oval(190, 90, 210, 110, fill="black")
mund = c.create_oval(150, 220, 250, 240, fill="red")
hals = c.create_line(200, 150, 200, 130, width=10)
hut = c.create_polygon(160, 75, 240, 75, 200, 20, fill="blue")
```

Den Mund und die Augen animieren

Wir definieren Funktionen, die das Öffnen und Schließen von Mund und Auge simulieren. Achte beim Schreiben der Funktionen auf den Doppelpunkt in der ersten Zeile und die Einrückungen in den folgenden Zeilen.

- Wenn der Mund offen ist, wird er schwarz dargestellt, wenn er zu ist, dann ist er wieder rot.
- Wenn das Auge geschlossen ist, dann ist es grün, so wie der Körper und die Pupille verstecken wir.
- Wenn das Auge offen ist, wird es wieder weiß und die Pupille ist sichtbar.

```
# Mund auf und zu
def mund_auf(event):
    c.itemconfig(mund, fill="black")

def mund_zu(event):
    c.itemconfig(mund, fill="red")

# Auge zu und auf
def auge_zu(event):
    c.itemconfig(auge, fill="green")
    c.itemconfig(pupille, state=HIDDEN)

def auge_auf(event):
    c.itemconfig(auge, fill="white")
    c.itemconfig(pupille, state=NORMAL)
```

Diese Funktionen kombinieren wir nun mit dem Drücken von bestimmten Tasten:

```
c.bind_all("<KeyPress-a>", auge_auf)
c.bind_all("<KeyPress-z>", auge_zu)
c.bind_all("<KeyPress-b>", mund_zu)
c.bind_all("<KeyPress-o>", mund_auf)
```

Text hinzufügen

Wir schreiben noch einen erklärenden Text in das Fenster. Hier geben die beiden Zahlen wieder die x und y Koordinaten der Textposition an. Mit dem Symbol \ kann man einen Zeilenumbruch im Code machen, wenn dieser zu lang für eine Zeile wird.

```
# Text
text1 = c.create_text(200, 280, text = "Drücke a und z \
um meine Augen auf und zu zu machen.")
text2 = c.create_text(200, 300, text = "Drücke b und o, \
um meinen Mund auf und zu zu machen.")
```

Aktivierung des Fensters

Zu guter Letzt gibt es noch einen Befehl, um das Fenster zu aktivieren:

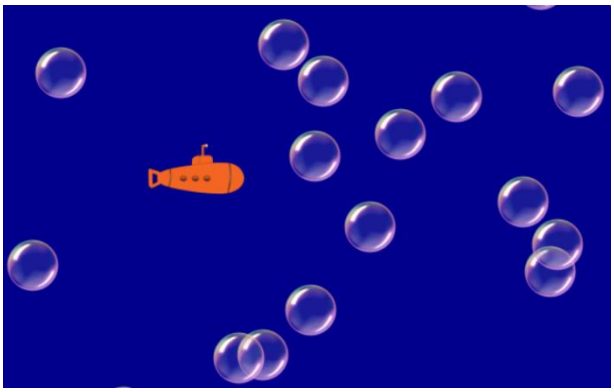
```
# Aktivierung des Fensters
window.mainloop()
```

Wenn du nun das Programm speicherst (STRG + S), kannst du es ausführen (F5). Probiere, ob dein Alien auf die Tastendrücke richtig reagiert!

Erweiterungsmöglichkeiten

- Ändere die Farben und Formen deines Aliens.
- Kannst du einen Alien mit zwei Augen zeichnen? Passe auch die Funktionen zum Öffnen und Schließen der Augen an.

Bubble Blaster



Mit dem Package Tkinter programmieren wir ein erstes Spiel in Python. Ein U-Boot, das mit Pfeiltasten gesteuert wird, soll in der vorgegebenen Zeit so viel Blasen wie möglich zum Platzen bringen. (Die Beschreibung erfolgt in Anlehnung an Beispiele aus dem Buch Programmieren supereasy).

Starte für Python die Programmierumgebung und öffne eine neue Datei und speichere die Datei als Dateityp auf deinem Computer. Wähle als Dateityp *.pyw*, wenn du mit Windows arbeitest.

Laden von Packages

Für dieses Beispiel verwenden wir das Package tkinter. Um Zufallszahlen zu generieren, brauchen wir auch das Package random. Für die Anzeige der Zeit und kurzen Pausen zwischen den Bewegungen der Bubbles brauchen wir die Funktionen sleep und time aus dem Package time.

```
from tkinter import *
from random import *
from time import sleep, time
```

Ein Fenster mit einer Leinwand erstellen

Zuerst erzeugen wir ein neues Fenster, das wir window nennen. Als Titel wählen wir „Bubble Blaster“). Die Leinwand wird 500 Pixel hoch und 800 Pixel breit, der Hintergrund ist dunkelblau.

```
# Leinwand erstellen
H = 500
B = 800
window = Tk()
window.title("Bubble Blaster")
c = Canvas(window, width=B, height = H, bg = "darkblue")
c.pack()
```

Das U-Boot laden

Für das U-Boot laden wir ein Bild, das du hier herunterladen kannst. Speichere dieses Bild in das gleiche Verzeichnis wie dein Python-Programm. Das geladene Bild positionieren wir dann in der Mitte der Leinwand (als x = B/2 und y = H/2). Den U-Boot Radius brauchen wir dann etwas später.

```
# U-Boot zeichnen
img = PhotoImage(file="uboot.png")
uboot = c.create_image(B/2, H/2, image=img)
UBOOT_R = 45
```

Das U-Boot steuern

Das U-Boot soll mit den Pfeiltasten bewegt werden. Bei jedem Tastendruck ändert sich die Position des U-Bootes um die U-Boot Geschwindigkeit, die wir hier mit 10 festlegen.

- Wird die Pfeiltaste „rechts“ gedrückt, wird die x -Koordinate um 10 größer, bei „links“ um 10 kleiner. Die y-Koordinate ändert sich nicht.
- Bei Pfeiltaste „unten“ wird die y-Koordinate größer, bei „oben“ wird die y-Koordinate kleiner. (Der Koordinatenursprung ist links oben!) die x-Koordinate bleibt gleich.
- Damit das U-Boot sich nicht über den Rand der Leinwand hinausbewegen kann, fragen wir zuerst die Koordinaten x, y des U-Boots ab. Die Bewegung des U-Boots wird nur dann durchgeführt, wenn das U-Boot auch nach der Bewegung innerhalb der Leinwand bleibt.

```
# U-Boot steuern
UBOOT_GESCHW = 10
def uboot_bewegen(event):
    x, y = c.coords(uboot)
    if event.keysym == "Up":
        if y >= UBOOT_GESCHW:
            c.move(uboot, 0, -UBOOT_GESCHW)
    elif event.keysym == "Down":
        if y <= H - UBOOT_GESCHW:
            c.move(uboot, 0, UBOOT_GESCHW)
    elif event.keysym == "Left":
        if x >= UBOOT_GESCHW:
            c.move(uboot, -UBOOT_GESCHW, 0)
    elif event.keysym == "Right":
        if x <= B - UBOOT_GESCHW:
            c.move(uboot, UBOOT_GESCHW, 0)
c.bind_all("<Key>", uboot_bewegen)
```

Bubbles erzeugen

Für die Bubbles laden wir auch ein Bild, das du hier herunterladen kannst. Den Bubble Radius benötigen wir später. Im Gegensatz zum U-Boot, brauchen wir mehrere Bubbles, die von rechts nach links mit unterschiedlichen Geschwindigkeiten über den Bildschirm wandern und dann wieder verschwinden. Es entstehen auch laufend neue Bubbles. Um immer einen Überblick zu haben welche Bubbles mit welchen Geschwindigkeiten es gibt, verwenden wir zwei Listen:

- *bub_id:* In dieser Liste merken wir uns den Namen (ID, eindeutige Nummer) der Bubbles. In eine Liste kann man mehrere Einträge mit Beistrichen getrennt hineinschreiben, zusammengefasst werden diese Einträge in Python mit einer eckigen Klammer. Zu Beginn haben wir noch keine Bubbles, also ist diese Liste noch leer.
- *bub_geschw:* Die Geschwindigkeiten schreiben wir auch in eine Liste, die ebenfalls zu Beginn noch leer ist.

Die Geschwindigkeit der Bubbles beschränken wir durch die Konstante MAX_BUB_GESCHW, die wir auf 10 setzen und dann definieren wir noch eine Konstante GAP, die wir für die Anfangsposition der Bubbles brauchen.

Nun schreiben wir uns eine Funktion, die genau eine Bubble erzeugt. Die Bubble entsteht rechts etwas außerhalb unseres Fensters. Die x-Koordinate setzen wir auf die Fensterbreite plus unseren definierten GAP. Die y-Koordinate soll eine Zufallszahl zwischen 0 und H sein (dafür verwenden wir den Befehl randint) und auch die Geschwindigkeit v wählen wir zufällig zwischen 1 und der maximal möglichen Geschwindigkeit.

Nun erzeugen wir die Bubble und hängen die ID (id_num) an unsere ID-Liste an und die Geschwindigkeit hängen wir an unsere Geschwindigkeits-Liste an (mit der Methode append).

```
# Bubbles erzeugen
bubble_img = PhotoImage(file="bubble.png")
BUB_R = 25
bub_id = []
bub_geschw = []
MAX_BUB_GESCHW = 10
GAP = 100

def erzeuge_bubble():
    x = B + GAP
    y = randint(0, H)
    v = randint(1, MAX_BUB_GESCHW)
    id_num = c.create_image(x, y, image = bubble_img)
    bub_id.append(id_num)
    bub_geschw.append(v)
```

Bubbles bewegen

Für die Bewegung der Bubbles über den Bildschirm verwenden wir auch eine Funktion. Diese Funktion geht jede Bubble der Reihe nach durch. Die Anzahl der aktuell vorhandenen Bubbles bekommen wir über die Länge unserer ID-Liste: len(bub_id). Bei jeder Bubble wird die x-Koordinate um ihre Geschwindigkeit verringert und die y-Koordinate gleich gelassen. Damit wandern die Bubbles horizontal nach links.

```
# Bubbles bewegen
def bewege_bubbles():
    for i in range(len(bub_id)):
        c.move(bub_id[i], -bub_geschw[i], 0)
```

Hauptschleife – 1. Test

Nun ist es an der Zeit die, bis jetzt geschriebenen, Funktionen zu testen. In einer ersten Version unserer Hauptschleife verwenden wir eine Endlosschleife (while True), in der wir laufend Blasen erzeugen, die Bubbles bewegen, das Fenster aktualisieren und dann kurz warten.

Damit nicht zu viele Blasen erzeugt werden, sondern nur durchschnittlich in einem Zehntel der Zeitschritte, erzeugen wir eine Zufallszahl zwischen 1 und 10. Nur wenn diese Zufallszahl 1 ist, dann erzeugen wir eine Bubble und sonst nicht.

```
# HAUPTSCHLEIFE
while True:
    if randint(1, 10) == 1:
        erzeuge_bubble()
    bewege_bubbles()
    window.update()
    sleep(0.01)
```

Bubbles entfernen

Jetzt beschäftigen wir uns mit dem Entfernen der Bubbles. Bubbles verschwinden, wenn sie komplett über den Bildschirm gewandert sind und auch wenn das U-Boot die Bubble zum Platzen gebracht hat. Für beides brauchen wir eine Funktion, die eine Bubble löschen kann.

Mit der Methode delete können wir das Bild einer Bubble entfernen, indem wir die ID angeben. Wir müssen aber auch unsere beiden Listen aktuell halten und dort die entsprechenden Einträge löschen.

```
# Bubble löschen
def lösche_bubble(i):
    c.delete(bub_id[i])
    del bub_id[i]
    del bub_geschw[i]
```

Für das Entfernen der Bubbles, wenn sie über den linken Bildschirmrand hinausgewandert sind, verwenden wir folgende Funktion:

```
# Bubbles entfernen (wenn sie über den Bildschirm gewandert sind)
def entferne_bubbles():
    for i in range(len(bub_id)-1, -1, -1):
        x, y = c.coords(bub_id[i])
        if x < -GAP:
            lösche_bubble(i)
```

Auch hier gehen wir wieder jede Bubble durch, holen und die aktuellen Koordinaten mit der Methode coords und löschen die Bubble, wenn die x-Koordinate mehr als unser GAP hinter dem linken Fensterrand liegt. Aber Achtung: Hier müssen wir die Liste unserer Bubbles von hinten durcharbeiten, sonst kommt unserer Nummerierung der Bubbles durcheinander. Würden wir z. B. die zweite Bubble löschen, dann wäre Bubble Nr. 3 plötzlich Bubble Nr. 2. Wir beginnen also bei der letzten Bubble (diese hat die Nr. len(bub_id)-1, weil ja Python bei 0 zu zählen beginnt). Dann ändern wir die Nummer immer um -1 und lassen den range bis -1 laufen, da das range-Objekt den letzten Eintrag nicht mehr dazu nimmt, also ist 0 der letzte Wert, den wir für i bekommen.

Hauptschleife – 2. Test

Das Entfernen der Bubbles können wir gleich in unsere Hauptschleife einbauen:

```
# HAUPTSCHLEIFE
while true:
    if randint(1, 10) == 1:
    erzeuge_bubble()
    bewege_bubbles()
    entferne_bubbles()
    window.update()
    sleep(0.01)
```

Bubbles platzen lassen

Wenn eine Bubble vom U-Boot getroffen wird, soll sie platzen und vom Bildschirm verschwinden. Um feststellen zu können, ob das U-Boot eine Bubble trifft, brauchen wir eine Funktion, die den Abstand zwischen dem U-Boot und einer Bubble berechnen kann. Dazu holen wir uns die aktuellen Koordinaten von U-Boot und Bubble und wenden den Satz von Pythagoras an. Die dazu notwendige Wurzelfunktion (sqrt) importieren wir aus dem math-Package. Der Abstand a wird als Wert der Funktion ausgegeben.

```
# Entfernung zwischen Punkten
from math import sqrt
def abstand(uboot, id_bubble):
    x1, y1 = c.coords(uboot)
    x2, y2 = c.coords(id_bubble)
    a = sqrt((x2-x1)**2 + (y2-y1)**2)
    return a
```

In einer Schleife gehen wir die Liste der Bubbles wieder von hinten durch. Wenn der Abstand zwischen U-Boot und Bubble kleiner ist als die Summe von U-Boot-Radius und Bubble-Radius, dann hat das U-Boot die Bubble getroffen. Wir erhöhen die Punkteanzahl, die wir zu Beginn auf 0 gesetzt haben um den Bubble-Radius + Bubble-Geschwindigkeit. Damit erreichen wir mehr Punkte, wenn die Bubble schneller ist. Die getroffene Bubble wird gelöscht und die erreichte Punkteanzahl wird ausgegeben.

```
# Bubbles platzen lassen (wenn sie vom U-Boot getroffen werden)
def treffer():
    punkte = 0
    for i in range(len(bub_id)-1, -1, -1):
        if abstand(uboot, bub_id[i]) < (UBOOT_R + BUB_R):
            punkte += (BUB_R + bub_geschw[i])
            lösche_bubble(i)
    return punkte
```

Hauptschleife – 3. Test

Um diese Funktion zu testen, bauen wir eine Variable score, die die Gesamtpunkteanzahl abspeichert, in die Hauptschleife ein. Zu Beginn ist die Punkteanzahl 0 und in jedem Zeitschritt wird die Punkteanzahl um die erreichten Treffer erhöht. Um die Punkteanzahl mitzuverfolgen, lassen wir sie im Shell-Fenster ausgeben.

```
# HAUPTSCHLEIFE
score = 0
while true:
    if randint(1, 10) == 1:
    erzeuge_bubble()
    bewege_bubbles()
    entferne_bubbles()
    score += treffer()
    print (score)
    window.update()
    sleep(0.01)
```

Zeit und Punkte anzeigen

In der Endversion des Spiels soll die Zeit und die Punktanzahl in unserem Fenster angezeigt werden. Wir schreiben dazu create_text einen passenden Text in die linke obere Ecke unserer Leinwand. Die beiden Zahlen geben die x und y Koordinaten an, mit der Option fill wird die Textfarbe festgelegt. Für die Anzeige der aktuellen Zeit und Punkteanzahl erzeugen wir die Objekte time_text und score_text. Mit den Funktionen zeige_punkte und zeige_zeit, ändern wir diese Objekte so, dass immer der aktuelle Stand zu sehen ist. Achtung: Der Text muss immer vom Datentyp string sein. Da die Zeit und die Punkte Zahlen sind, müssen wir sie mit der Funktion str in einen string umwandeln.

```
# Zeit und Punkte anzeigen
c.create_text(50, 30, text = "ZEIT", fill="white")
c.create_text(150, 30, text = "PUNKTE", fill="white")
time_text = c.create_text(50, 50, fill="white")
score_text = c.create_text(150, 50, fill="white")
def zeige_punkte(score):
    c.itemconfig(score_text, text = str(score))
def zeige_zeit(time_left):
    c.itemconfig(time_text, text=str(time_left))
```

Hauptschleife - komplett Nun können wir die Hauptschleife fertigstellen. Wir geben ein Zeitlimit von 30 Sekunden für das Spiel vor. In der Variable ende speichern wir den Endzeitpunkt des Spiels. Das ist die aktuelle Zeit, wenn das Spiel gestartet wird plus unser Zeitlimit. Die aktuelle Zeit bekommen wir mit dem Befehl time(). Anstelle der Endlosschleife überprüfen wir, ob unsere Spielzeit noch nicht abgelaufen ist. In der Schleife ergänzen wir die Anzeige der aktuellen Zeit und der Punkte.

Ist das Spiel aus, zeigen wir den Text „GAME OVER“ und die erreichte Punkteanzahl an.

```
# HAUPTSCHLEIFE
score = 0
TIME_LIMIT = 30
ende = time() + TIME_LIMIT
while time() < ende:
    if randint(1, 10) == 1:
        erzeuge_bubble()
    bewege_bubbles()
    entferne_bubbles()
    score += treffer()
    zeige_punkte(score)
    zeige_zeit(int(ende-time()))
    window.update()
    sleep(0.01)

c.create_text(B/2, H/2, text = "GAME OVER", \
              fill = "white", font=("Helvetica", 30))
c.create_text(B/2, H/2 + 30, \
              text = "Punkte: " + str(score), fill = "white")
```

Aktivierung des Fensters

Zu guter Letzt gibt es noch einen Befehl, um das Fenster zu aktivieren:

```
# Aktivierung des Fensters
window.mainloop()
```

Wenn du nun das Programm speicherst (STRG + S), kannst du es ausführen (F5). Wie viele Punkte kannst du erreichen?

Erweiterungsmöglichkeiten

- Zeichne, anstelle des Bubble-Bildes, Kreise als Bubbles. Die Kreise können unterschiedliche Größen und Farben haben. Wenn Bubbles mit einer bestimmten Farbe zum Platzen gebracht werden, gibt es die doppelte Punkteanzahl.
- Erstelle eine Highscore-Liste, auf der die höchsten Punktezahlen gespeichert werden.
- Gib 30 Sekunden Bonus-Zeit zum Spielen, wenn in der vorgegebenen Zeit mindestens 2000 Punkte erreicht werden.


CODING
CLUB
LINZ