Spiele programmieren mit Godot

Vom kinderleichten Einstieg zum eigenen Game

DIE LESEPROBE

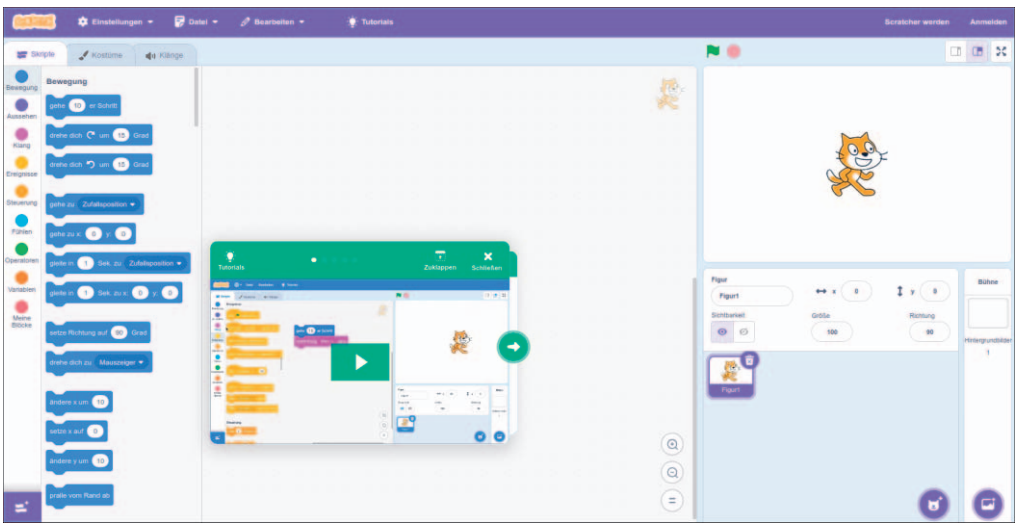
Kapitel 2
Aller Anfang ist leicht

Scratch ist eine orangefarbene Katze, die sich mit Puzzleteilen programmieren lässt – ohne kniffligen Code. Sie tanzt, spricht, stellt Mathe-Aufgaben oder verwandelt sich in andere Tiere. Und die Konzepte, die wir mit ihr lernen, sind dieselben wie in »richtigen« Programmiersprachen und in Godot. Einfacher kann man Coden nicht lernen!

Loslegen mit Scratch

Es ist zum Glück kein großer Aufwand, eine sichere und effiziente Umgebung für das Entwickeln mit Scratch auf die Beine zu stellen. Möglicherweise brauchst du kurz die Hilfe deiner Eltern, um Scratch zu starten. Die einfachste Möglichkeit, Scratch zu benutzen, ist mit dem Browser. Egal, ob Firefox, Chrome, Safari, ob unter Windows, macOS oder Linux, gib einfach in die Adressleiste ein: *https://scratch.mit.edu/*

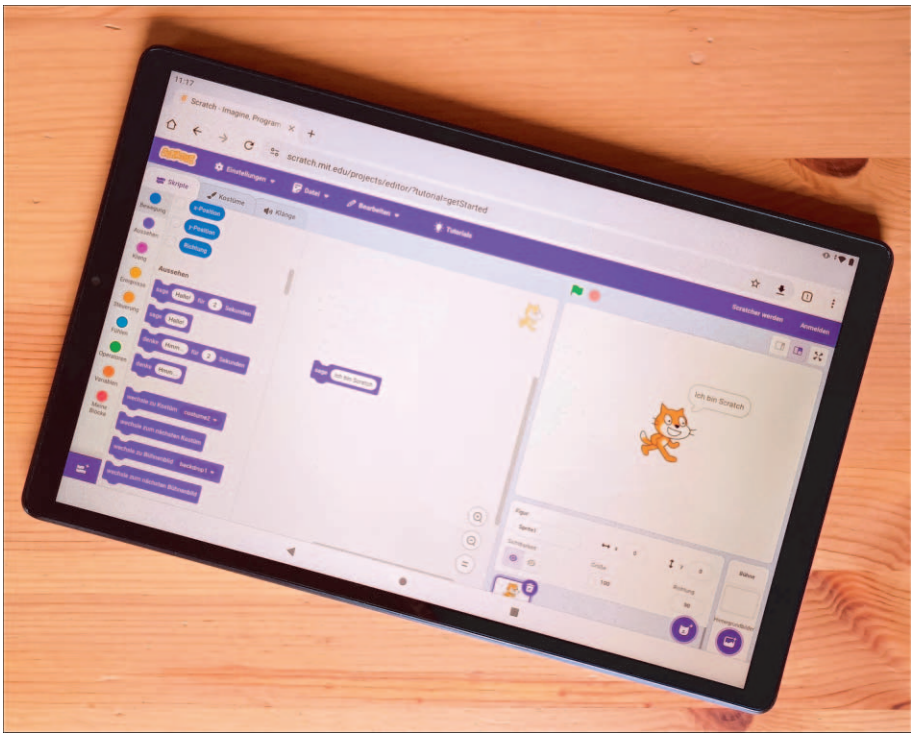
Eine Anmeldung oder Registrierung ist nicht erforderlich. Die eigentliche Programmierumgebung erscheint sofort nach einem Klick auf **Entwickeln**. Das sollte dann so aussehen:



Falls es sich um einen sehr alten Browser handelt, wird Scratch dich darauf hinweisen. Dann hilft ein Update des Browsers.

Sollte Scratch auf Englisch daherkommen, lässt sich das ändern mit einem Klick auf **Settings** und **Language**. Dort kannst du alles auf Deutsch umschalten.

Scratch läuft auch auf Tablets. Es lässt sich im Tabletbrowser aufrufen, wie oben beschrieben, oder als App installieren, aber das macht keinen großen Unterschied.



Auf Tablets ist Scratch nicht besonders komfortabel zu bedienen und außerdem etwas langsam. Abgesehen von den ersten Tests ist ein PC mit Tastatur und Maus auf jeden Fall vorzuziehen. Sobald wir zur ausgewachsenen Spiele-Engine Godot wechseln, führt daran sowieso kein Weg mehr vorbei.

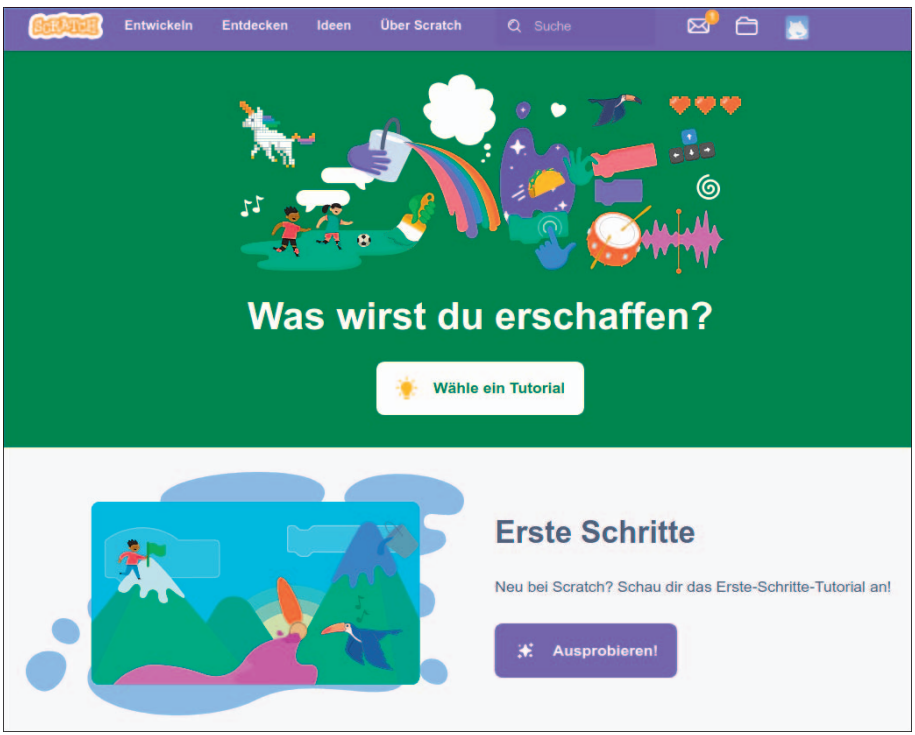
Hinweis zum Datenschutz

Solange du dich nicht als *Scratcher* anmeldest, speichert oder verarbeitet Scratch keinerlei personenbezogene Daten. Der Vorteil einer Anmeldung ist,

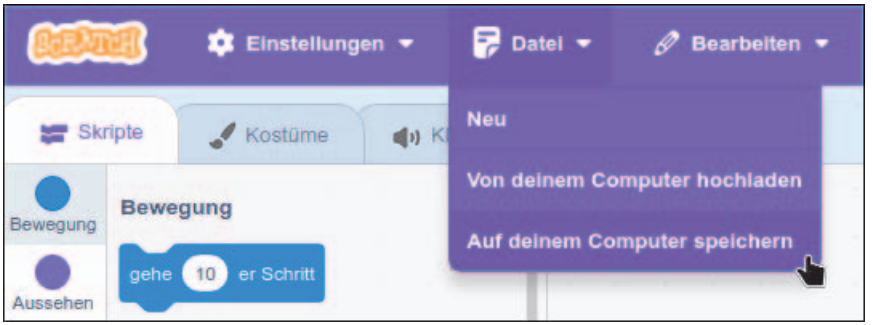
dass Projekte online gespeichert werden. Du kannst also einfach den Rechner ausschalten und später an der gleichen Stelle weitermachen, sogar von einem anderen PC aus. Außerdem kannst du Projekte mit anderen angemeldeten Scratchern in »Studios« teilen oder veröffentlichen, wenn sie besonders gelungen sind. Bis auf einen Nutzernamen und die E-Mail-Adresse müssen keine Daten eingegeben werden.

Auf der Scratch-Website kannst du unzählige von anderen Scratchern veröffentlichte Programme ausprobieren und besichtigen. Das fängt bei simplen Spielchen an und geht bis zu nachprogrammierten Hits wie »Brawl Stars« – was Scratch allerdings an seine Grenzen bringt. Der Programmierer hat die Weiterentwicklung aufgegeben, weil Scratch einfach zu langsam wurde. Aber für große, umfangreiche Spiele ist es auch nicht gedacht – sondern zum Lernen!

Im Bereich **Ideen** warten unzählige Tutorials für Einsteiger und Fortgeschrittene. Wenn du nach der Einführung in diesem Buch noch Lust auf mehr Scratch hast, kannst du dich hier austoben!



Fertige Programme kannst du im Menü **Datei** auf den PC herunterladen. Die Dateien mit Namen *Scratch-Projekt.sb3* landen zunächst im Download-Ordner deines Rechners. Leider haben sie immer den gleichen Namen, du solltest sie also passend umbenennen. Am besten schiebst du sie dann in einen eigenen Ordner. Von dort aus kannst du sie über das gleiche Menü wieder hochladen, um sie weiterzubearbeiten.



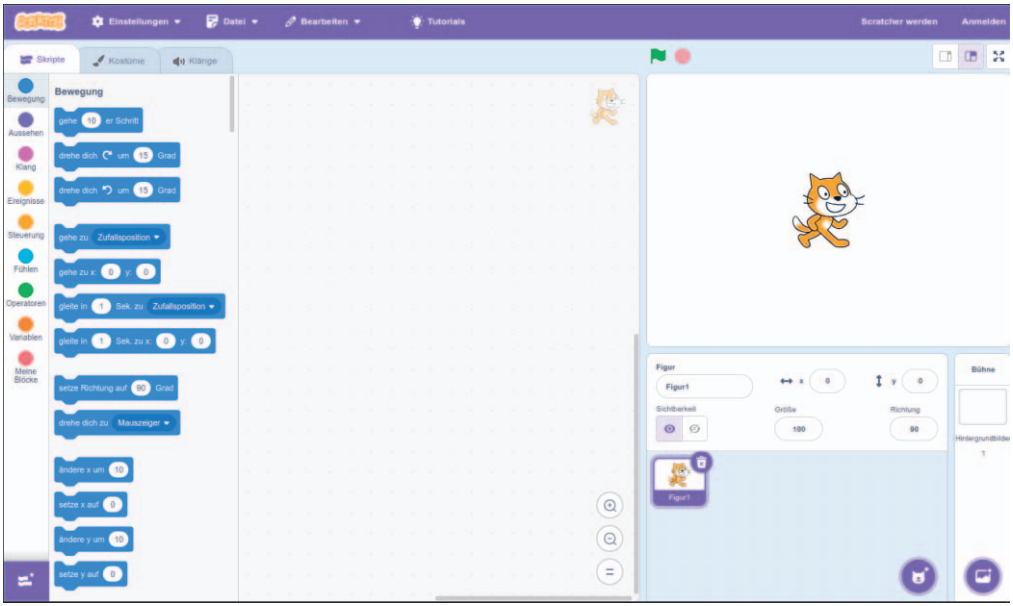
Um Scratch ohne Internetzugang zu verwenden, muss es einmalig als App heruntergeladen werden. Dazu besuchst du mit dem Browser die Seite *https://scratch.mit.edu/download*.

Unter Windows lässt sich die App über den Microsoft Store installieren oder direkt herunterladen. Dasselbe gilt für macOS. Für ChromeOS und Android findet sich die App im Play Store. Einmal installiert, kann die Internetverbindung getrennt werden, und alles läuft wie gewohnt. Nur das Onlinespeichern oder Veröffentlichen von Projekten funktioniert so nicht.

Für Linux-Nutzer gibt es leider keine guten Nachrichten: Es gibt zwar eine inoffizielle Offlineversion von Scratch namens Scratux (*https://github.com/scratux/scratux/releases*). Diese hat aber ein paar Schwierigkeiten mit den mitgelieferten Kostümen und Sounds. Daher empfiehlt sich unter Linux ausschließlich die Onlineversion im Browser.

Dein erstes Scratch-Spiel

Mach dich bereit für den Start! Computer an, Scratch starten bzw. *scratch.mit.edu* aufrufen, und es kann losgehen. Es ist völlig egal, welche Version du verwendest. Du wirst ziemlich genau das hier sehen:

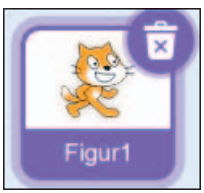


Möglicherweise erscheint ein grünes Tutorial-Fenster. Das klickst du einfach weg. Links siehst du die bunten Puzzleteile, aus denen du alle Scratch-Spiele zusammenbasteln wirst. Rechts oben befindet sich die *Bühne*. Das ist der Bereich, in dem jedes Spiel stattfindet, und dort wartet schon eine *Spielfigur*: die Scratch-Katze. Den Bereich darunter kannst du erst mal völlig ignorieren.

Dein *Skript*, also das Spielprogramm, gehört in den (noch) leeren Bereich in der Mitte.

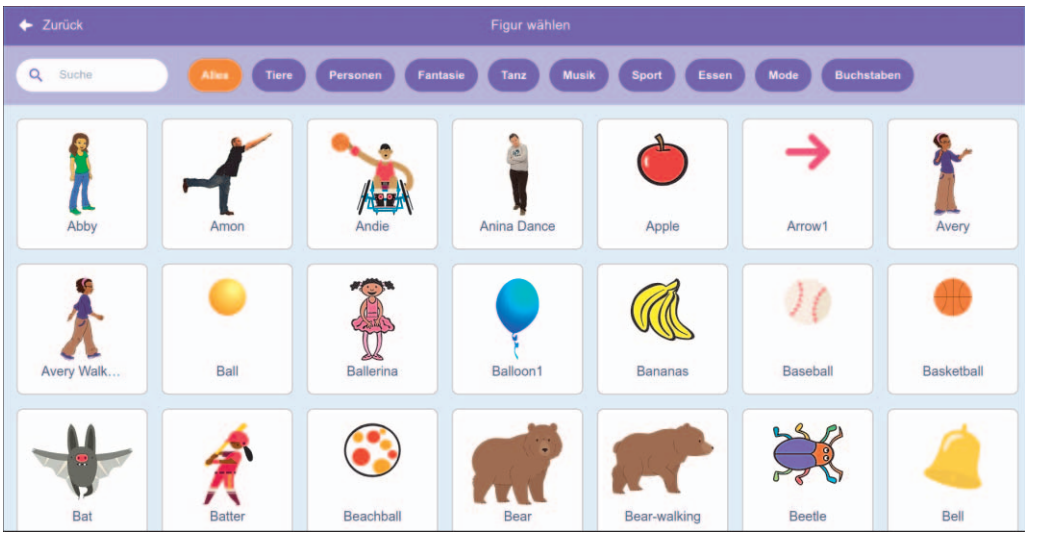
Unser erstes Spiel wird ein Wettbewerb im Platzenlassen von Luftballons, und zwar ganz ohne Gummireste, die dann weggeschmissen werden müssen, aber natürlich mit ordentlich Lärm!

1. Für ein Luftballonspiel brauchen wir keine Katze als Spielfigur, sondern Luftballons. Lösche also die Katze. Dazu klickst du rechts unten unterhalb der Bühne auf den kleinen lila Mülleimer neben der Katze.

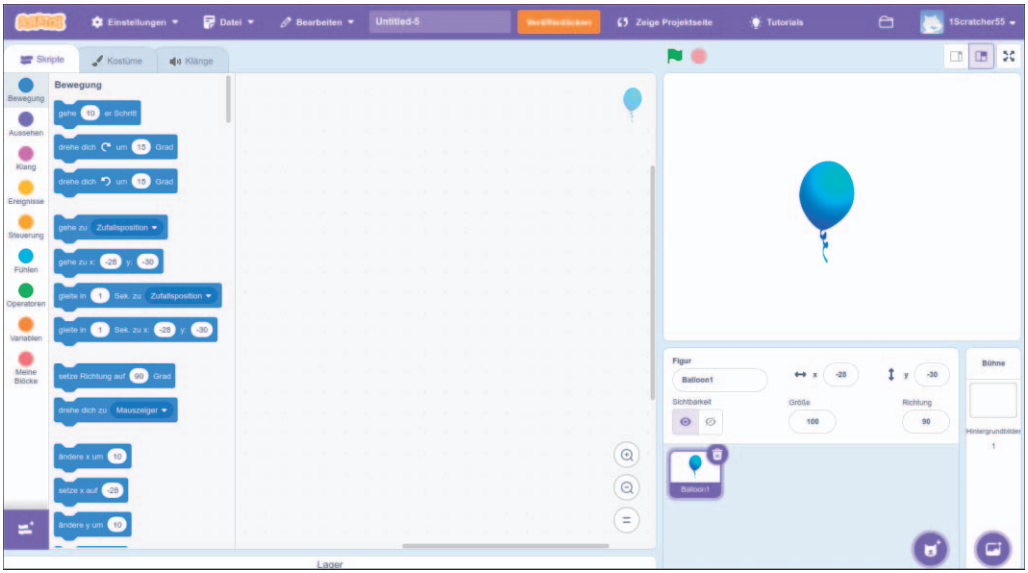


2. Jetzt brauchen wir einen Luftballon als Figur. Etwas weiter unten ist eine lila Taste mit einem Katzengesicht und einem kleinen Pluszeichen zu sehen. Sobald du mit dem Mauszeiger darüberfährst, wird die Taste grün und es erscheint **Figur wählen**. Klick darauf!

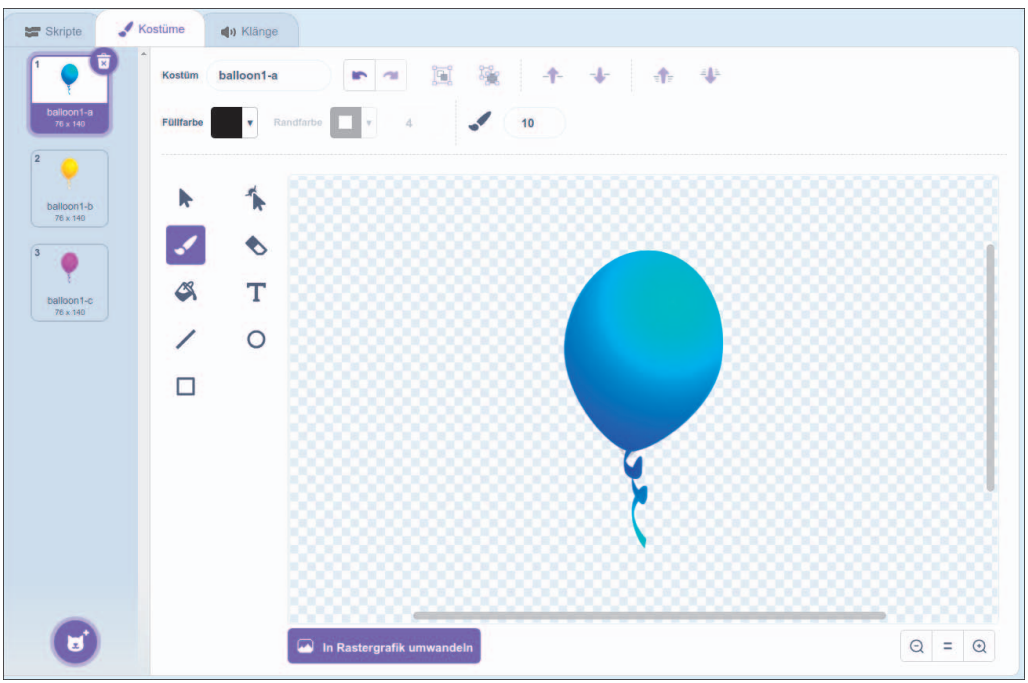
3. Du kannst jetzt eine Figur wählen. Ziemlich genau in der Mitte ist ein Ballon. Klick ihn an!



Wenn du den Mauszeiger über den Ballon hältst, wechselt er die Farbe. Warum, wirst du gleich sehen. Zunächst erscheint der Ballon auf der Bühne, der Skriptbereich bleibt noch leer.



4. Klick einmal oben links auf die Registerkarte **Kostüme**. Dann siehst du, warum der Ballon vorhin die Farbe gewechselt hat: In Wirklichkeit verfügt unsere Figur über drei Kostüme in drei Farben. Das werden wir später ausnutzen.



Wenn du möchtest, kannst du hier mit dem Pinsel ein Gesicht auf die Ballons malen oder sie anderweitig verschönern. Du kannst auch die Reihenfolge der Kostüme in der linken Leiste mit der Maus ändern. Auf der Bühne erscheint die Figur immer zunächst mit dem obersten Kostüm.

Als Nächstes soll der Ballon platzen, sobald er angeklickt wird. Und dafür brauchen wir jetzt ein paar erste Puzzlestücke.

Figuren auf die Bühne!

Scratch-Skripte bestehen aus Puzzleteilen, die ineinanderpassen. Du suchst einfach das gewünschte Teil links in der Liste und ziehst es mit der Maus rüber in den Programmierbereich in der Mitte. Passende Teile fügen sich zusammen, als wären sie magnetisch. Du kannst Teile aber immer auch abziehen und abreißen, wenn sie mal an der falschen Stelle gelandet sind.

Die Puzzleteile sind nach Aufgabenbereichen sortiert und tragen die entsprechende Farbe, so sind sie leicht zu finden. Ganz links siehst du eine Liste mit Farbkreisen, die für diese Aufgabenbereiche stehen. Zum Beispiel Bewegung, Aussehen usw.

Wenn du einen der Kreise anklickst, zeigt der Bereich daneben die Puzzleteile der passenden Farbe. Klick mal auf den gelben Kreis, unter dem Ereignisse steht. Von den gelben Puzzleteilen, die dann auftauchen, wählst du das dritte und ziehst es rüber in den leeren Skriptbereich. Das Teil sieht so aus:

Logisch, oder? Wir wollen ja, dass der Luftballon platzt, wenn er angeklickt wird. Also brauchen wir genau diesen *Skript-Starter*. Denn immer, wenn die Figur angeklickt wird, führt die Figur alle Puzzleteile aus, die unten dranhängen.

Passt oder passt nicht

Der Skript-Starter hat oben eine Rundung und keine Puzzlelücke. Daran siehst du, dass du oben kein Puzzleteil einstecken kannst, das geht nur unten. Es ergäbe ja auch wenig Sinn, wenn die Figur etwas tun würde, bevor

sie angeklickt wird, denn der Klick ist ja der Auslöser des Geschehens. Ohne Schuss kein Tor!

Andere Skript-Starter werden ausgelöst, wenn die grüne Fahne oberhalb der Bühne angeklickt wird oder wenn du eine bestimmte Taste auf der Tastatur drückst. Letzteres brauchen wir später, wenn wir Figuren per Tastatur über den Bildschirm lenken wollen.

Übrigens: Wenn du ein Teil nicht mehr benötigst, schieb es einfach nach links aus dem Skriptbereich hinaus, und es wird verschwinden.

Was genau soll nun passieren, wenn jemand die Figur anklickt? Natürlich soll der Luftballon platzen. Da wir dafür (im Moment) keinen passenden grafischen Effekt haben, lassen wir die Figur einfach nur verschwinden. Und wir spielen ein Geräusch ab. Aber eins nach dem anderen.

Wie lassen wir die Figur verschwinden? Dafür gibt es natürlich ein passendes Puzzleteil. Bevor du lange suchst, erkläre ich dir die neun Funktionsbereiche, dann findest du es sofort.

Die erste Funktion wirst du sicherlich häufig einsetzen. Mit den blauen Puzzleteilen kannst du eine Spielfigur in verschiedene Richtungen bewegen oder drehen.

Im lilafarbenen Bereich findest du Puzzleteile zum Ändern von Kostümen, Bühnenbildern und der Größe einer Figur. Außerdem kannst du mit bestimmten Puzzleteilen Sprechblasen bei einer Figur erscheinen lassen wie in Comics. Ziemlich wichtig sind auch Teile, die die Figur verbergen (unsichtbar machen) oder wieder herbeizaubern. Such diese beiden Teile schon einmal heraus und zieh sie in den Programmbereich.

Du kannst mit den pinken Klangteilen nicht nur vorgefertigte, sondern auch eigene Töne erklingen lassen. Außerdem kannst du die Tonhöhe und Lautstärke ändern und, ganz wichtig für deine Eltern, die Figur verstummen lassen.

Die gelben Puzzleteile hast du gerade schon gesehen: Hier warten mehrere Starter-Teile, die bei Ereignissen, wie zum Beispiel Mausklicks, ein Skript starten. Ebenfalls sind hier Teile untergekommen, mit denen eine Figur Nachrichten senden oder empfangen kann. Das werden wir später oft benutzen.

Die orangefarbenen Teile steuern den Ablauf des Skripts. Damit kannst du Skriptteile wiederholt oder abhängig von bestimmten Bedingungen ausführen. Auch Wartebefehle gibt es hier. Steuerungspuzzleteile sind etwas komplizierter zu verstehen als die anderen, aber keine Sorge: Ich werde nach und nach erklären, wozu man sie braucht.

Mit den hellblauen Puzzleteilen kann deine Spielfigur etwas über ihre Umgebung herausfinden, zum Beispiel ob sie den Mauszeiger oder eine bestimmte Farbe auf der Bühne berührt. So kann die Katze fauchen, wenn sie ins Wasser fällt. Oder ein Pirat kann einen goldenen Schatz einsammeln. Auch die Tastatur lässt sich hier abfragen. Damit werden wir später eine Figur mit den Pfeiltasten über den Bildschirm steuern.

Die grünen Operatorenteile unterscheiden sich nicht nur in ihrer Farbe von den meisten anderen, sondern auch in ihrer Form. Sie haben nämlich links und rechts runde oder spitze Seiten. Dadurch passen sie zum Beispiel in die Ausschnitte von Teilen zur Ablaufsteuerung. Rundliche Operatoren produzieren Zahlen oder Wörter, zum Beispiel für eine Rechenaufgabe. Spitze Operatoren produzieren *Wahrheitswerte*, also »ja« oder »nein«. Sie werden immer dann gebraucht, wenn der Teil eines Skripts abhängig von einem solchen Wert ausgeführt werden soll oder nicht. Natürlich werden wir auch diese Operatoren häufig benötigen.

Variablen sind kleine Speicherelemente, die einen Text oder eine Zahl enthalten können, zum Beispiel für den Punktestand. Du kannst beliebig neue erzeugen und ihnen eigene Namen geben. Die dunkelorangefarbenen Variablenpuzzleteile sind rundlich und passen daher in Operatoren oder andere runde Öffnungen in anderen Teilen. So lässt sich zum Beispiel eine Punktzahl speichern und dann abhängig von deren Wert ein Skript ausführen oder eben nicht.

Der hellrote Bereich ist eine Art leerer Spielplatz. Dort kannst du eigene Skripte als sogenannte *Blöcke* unter einem eigenen Namen speichern. Der Clou: Jeder Block lässt sich dann als einzelnes rotes Puzzleteil verwenden und verbirgt in diesem einen Teil dein ganzes Skript.

Das lilafarbene Puzzleteil **verstecke dich** hast du jetzt ja sicher schon gefunden. Es ist im Aussehen-Bereich versteckt. Hänge es an das gelbe Starter-Teil, dann klick auf der Bühne auf den Luftballon – und natürlich verschwindet er!

Das Mini-Skript sieht jetzt so aus:

Natürlich funktioniert das im Moment nur einmal – der Luftballon ist und bleibt weg. Klarer Fall, wir haben ihm ja auch nirgendwo gesagt, dass er wieder auftauchen soll.

Um das zu ändern, such das Puzzlestück **zeige dich** im **Aussehen**-Bereich und klick einmal darauf. Dann erscheint der Luftballon wieder.

Der Ausprobieren-Klick-Trick

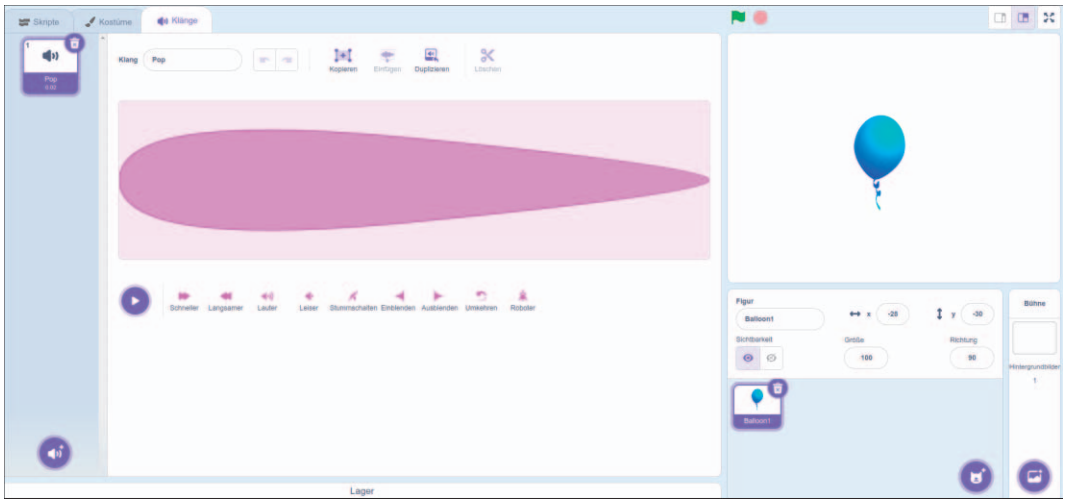
Wie du siehst, kannst du jedes Puzzleteil einfach anklicken, um es sofort auszuführen. Das ist nicht nur hilfreich, um eine verschwundene Figur schnell zurückzubringen, sondern auch, um ein Teil einfach mal auszuprobieren. Klick zum Beispiel einfach mal im **Klang**-Bereich auf das oberste Puzzlestück!

Übrigens funktioniert der Trick nicht nur in der Bibliothek, sondern auch im Programmierbereich. Du kannst zum Beispiel das Zeige-dich-Teil einfach irgendwo hier ablegen, um es immer griffbereit zu haben. Solange du es nicht anklickst und es nirgendwo angebaut ist, bleibt es untätig.

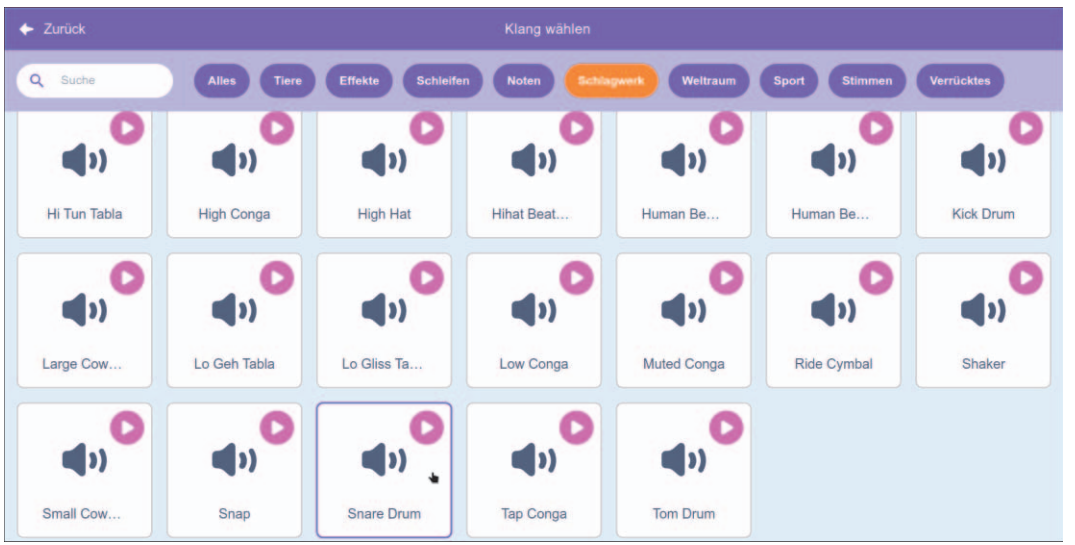
Auch zusammengestöpselte Puzzleteile, also Skripte, kannst du mit einem Klick ausprobieren. Um den Luftballon verschwinden zu lassen, kannst du also auch im Programmbereich auf das kleine Skript klicken.

Am besten ziehst du das pinke Puzzleteil **spiele Klang** direkt mal in den Programmierbereich und stöpselst es an dein Skript, damit der Luftballon geräuschvoll

platzt. Um ein anderes Geräusch abzuspielen, schaltest du zunächst die Ansicht auf **Klänge** um, indem du auf die Registerkarte klickst.



Das sieht jetzt so ähnlich aus wie die Kostümauswahl – bloß eben mit Geräuschen. Unten links findest du eine runde, lilafarbene Taste mit Lautsprecher und Pluszeichen. Du kannst sie anklicken, um ein mitgeliefertes Geräusch auszuwählen. Die Auswahl ist groß und nach Arten von Sounds unterteilt. Natürlich kannst du dir hier jedes Geräusch direkt anhören, dafür musst du nur den Mauszeiger auf das jeweilige Abspielensymbol halten.



Probiere ruhig alle Sounds durch. Einen richtigen Ballonknall gibt es nicht, der Standardplopp klingt etwas harmlos, aber ich finde, dass die **Snare Drum** ganz gut passt. Du findest sie im Bereich **Schlagwerk**. Klick den Klang einfach an, um ihn auszuwählen. Den zuvor vorhandenen Klang kannst du links aus der Leiste löschen. Dein Skript sollte jetzt so aussehen:

Probiere es aus! Der Luftballon verschwindet jetzt mit einem Knall, sobald du ihn anklickst. Um ihn zurückzubekommen, klickst du auf **zeige dich**. Achte darauf, dass du nicht versehentlich das Teil **zeige dich** ans Skript hängst. Dann verschwindet der Ballon nämlich und taucht sofort wieder auf – das wäre ja ein bisschen unrealistisch, oder?

Klang ganz abspielen oder nicht?

Du hast sicher gesehen, dass es zwei verschiedene Puzzleteile gibt, um einen Klang abzuspielen. Das erste spielt den Klang ganz, bevor das Skript fortgesetzt wird. Das zweite startet den Klang und macht sofort weiter, völlig egal, wie lang der Sound ist.

Es kommt immer auf die Situation an, welches der beiden Teile sich besser eignet. Für unser Ballonspiel sind beide gleich sinnvoll.

Natürlich ist das noch längst kein fertiges Spiel. Wir müssen ganz viele Ballons auf die Bühne bringen – so viele, dass es ein richtiges Knallfeuerwerk gibt!

99 Luftballons

Ein einsamer Luftballon ist natürlich deutlich zu wenig für ein richtiges Spiel. Wir müssen unseren Ballon also kopieren. Scratch nennt diese Kopien *Klone*. Das benötigte Puzzleteil findest du im Bereich **Steuerung**.

Klone übernehmen alles vom Original: Kostüm, Position, Skripte. Sie sind identische Kopien, können sich aber unterschiedlich verhalten. Wenn du später einen Ballonklon anklickst, wird nur dieser platzen, die anderen nicht!

Aber wann muss ein Klon erzeugt werden? Sicher nicht, wenn der Ballon angeklickt wird, sondern immer wieder, in regelmäßigen Abständen – aber natürlich erst, wenn das Spiel gestartet wurde. Und dafür ist das grüne Fähnchen über der Bühne da. Sobald es angeklickt wird, soll der Ballon anfangen, Klone zu erzeugen. Lass uns das Skript Schritt für Schritt zusammensetzen:

1. Du brauchst zuerst das Starter-Teil, das beim Anklicken der grünen Fahne ein Skript startet. Ziehe es in den Arbeitsbereich, sodass es neben oder unterhalb des vorhandenen Skripts liegt.

2. Als Nächstes brauchst du eine *Schleife*. So nennt man ein Skriptelement, das andere Skriptteile wiederholt ausführt. Du findest alle Schleifen im Bereich **Steuerung**. Schleifen sind eine wichtige Programmstruktur in der Informatik und kommen immer wieder vor! Nimm für den Anfang die 10-mal-Schleife und hänge sie an den neuen Skript-Starter.

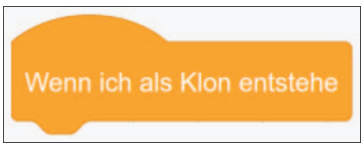
Du kannst die Zahl ändern, indem du sie anklickst. Wichtig ist: Das, was die Schleife wiederholen soll, gehört in die schmale Lücke. Keine Sorge: Die wird automatisch breiter, wenn du mehrere Puzzleteile einsetzt.

3. Genau an diese Stelle gehört jetzt das Puzzleteil **erzeuge Klon von**. Füge außerdem ein **warte**-Teil ein, damit die Ballons nacheinander und nicht alle gleichzeitig entstehen. Das fertige Skript sieht dann so aus:

Aber noch bist du nicht fertig! Denn jetzt würden zwar beim Anklicken der Fahne nacheinander zehn Ballons auftauchen, aber alle an der gleichen Stelle und in der gleichen Farbe – wie langweilig!

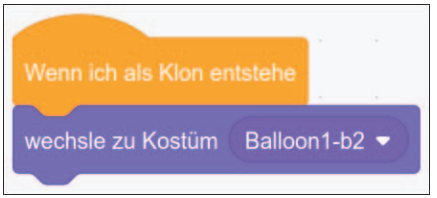
Das ändern wir mit einem weiteren Skript, das neben dem ersten liegt und immer dann startet, wenn der neue Klon entsteht. Beide Skripte funktionieren unabhängig voneinander, solange keine Verbindung zwischen ihnen besteht.

1. Dazu gibt es einen weiteren Skript-Starter:

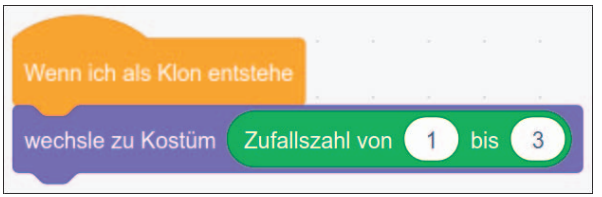


Vorsicht: Dieses Teil ist nicht bei den gelben Ereignissen einsortiert, wo die anderen Starter-Teile liegen, sondern unter **Steuerung**.

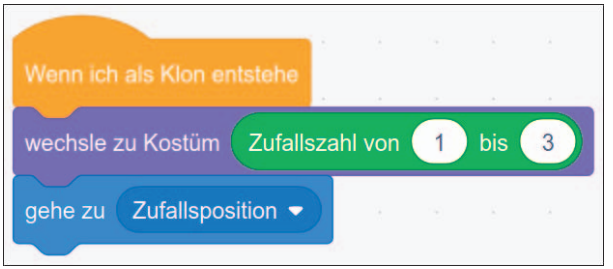
2. Dieses Skript wird also nur für die Klone gestartet, nicht für das Original. Lass uns erst das Kostüm ändern. Das erforderliche lilafarbene Puzzlestück findest du im **Aussehen**-Bereich. Hänge es an den Klon-Starter.



Allerdings soll jeder Ballonklon ein zufälliges Kostüm bekommen. Endlich kommt auch einmal ein grünes Puzzleteil zum Einsatz, nämlich der *Zufallsgenerator*! Schau mal, ob du ihn findest. Er passt genau in den dunklen, rundlichen Bereich des Kostümwechslers. Da wir drei Kostüme haben, musst du eine Zufallszahl von 1 bis 3 erzeugen. Das geht so:



3. Außerdem soll der Ballon an einer zufälligen Stelle der Bühne erscheinen. Wir müssen ihn also *bewegen*. Suche also im blauen Bereich das Puzzleteil **gehe zu Zufallsposition** und hänge es an. Das fertige Skript sieht jetzt so aus:



Wenn Du jetzt die grüne Fahne anklickst, erscheinen nacheinander zehn bunte Ballons auf der Bühne, die du zum Platzen bringen kannst.

Es gibt allerdings einen kleinen Fehler: Sobald du den Originalballon triffst, wird er ja unsichtbar. Alle danach erzeugten Klone sind also auch unsichtbar, weil sie wirklich alle Eigenschaften des Originals kopieren! Du kannst dieses Problem lösen, indem du in das letzte Skript am Anfang ein lila Element **zeige dich** einfügst.

Wenn du möchtest, kannst du die ganze Sache beschleunigen, damit man richtig ins Schwitzen kommt: Ändere die Anzahl der Wiederholungen in der Schleife zum Beispiel auf 100 und die Wartezeit auf 0.1 Sekunden.

Punkt statt Komma

Achtung, in Scratch musst du Kommazahlen mit einem Punkt anstelle eines Kommas als Trennzeichen eingeben. Das liegt daran, dass Scratch ein amerikanisches Programm ist und man vergessen hat, das Komma richtig zu übersetzen – im angloamerikanischen Sprachraum wird bei Kommazahlen immer der Punkt verwendet.

Damit ist das erste Mini-Spiel fertig! Natürlich fehlt diesem »Spiel« noch so einiges. Aber ich will dir hier im Buch nicht alles vorkauen. Du kannst Scratch auf eigene Faust weiter erforschen. Schau dir an, was es noch für Puzzleteile gibt, probiere dies und jenes aus und werde kreativ!

Hier ein paar Anregungen, was das Spiel noch so brauchen könnte:

- ein richtiges »Game over« (Zum Beispiel, wenn zu viele Ballons auf dem Bildschirm sind, dazu brauchst du eine Variable als Zähler.)
- einen Punktezähler (Tipp: Variablen können auf der Bühne sichtbar gemacht werden.)
- einen steigenden Schwierigkeitsgrad (Tipp: Die Wartezeit bis zum nächsten Klon könnte immer kürzer werden.)
- Male ein »Fetzenkostüm«, das du nach dem Knall für einige Sekundenbruchteile anzeigst.
- Fülle Helium in die Ballons, sodass sie langsam aufsteigen (Tipp: Dafür brauchst du eine Schleife und das blaue Puzzleteil **ändere y um**.)
- Ändere den Hintergrund der Bühne. (Tipp: Die Taste dafür findest du ganz rechts unten.)
- Oder denk dir selbst etwas aus!

Natürlich werden in den nächsten Abschnitten viele dieser Elemente sowieso vorkommen. Aber etwas Neugier ist nicht verboten, im Gegenteil: Neugier ist der Antrieb aller Erfinderinnen und Erfinder!

In diesem Kapitel hast du gelernt, dass jede Figur mehrere Skripte besitzen kann, und wie die verschiedenen Puzzleteile zusammenpassen. Viele dieser Konzepte tauchen nicht nur in den nächsten Scratch-Beispielen wieder auf, sondern auch später im Buch, wenn wir Spiele in Godot schreiben.

Kapitel 9
Viel los im Raumschiff

Major Tim wird in Kürze merken, dass er nicht allein in dem seltsamen Raumschiff chillt und dass die anderen Fluggäste seiner Gesundheit nicht gerade zuträglich sind. Jede Berührung kostet ihn Lebenspunkte. Wie bei jeder Hitpoints-Anzeige wartet an deren Null-Ende das unausweichliche *Game over* ...

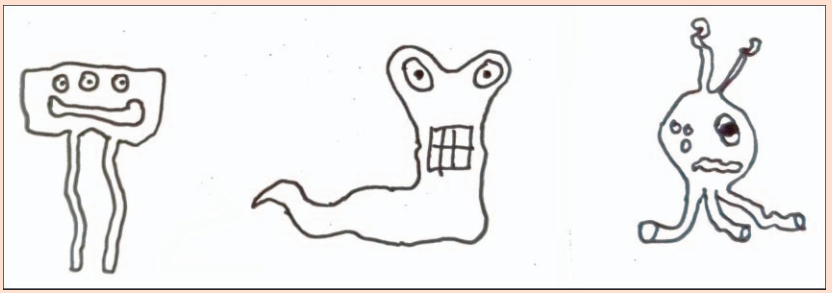
Hungrige Aliens

Um Major Tim mit gefährlichen Begegnungen der dritten Art zu konfrontieren, brauchst du erstens Aliens und zweitens einen Lebenspunktezähler. Die Aliens werden genau wie der Astronaut *CharacterBody2D*-Nodes, die sich allerdings nicht per Tasteneingabe lenken lassen, sondern ihrem eigenen Weg folgen. Die Aliens lassen sich sogar animieren. Das funktioniert wie bei einem Trickfilm, es gibt also mehrere Einzelbilder mit unterschiedlichen Posen. Ich habe im Download-Paket drei Aliens mit je zwei Körperhaltungen zur Verfügung gestellt, die du am besten direkt ins Projektverzeichnis kopierst. Natürlich kannst du dir auch andere Bilder suchen, zum Beispiel bei *opengameart.org*. Oder du malst selbst welche. Achte darauf, dass deine Aliens nach rechts schauen. Für die Bewegung nach links werden wir sie spiegeln, genau wie Major Tim.

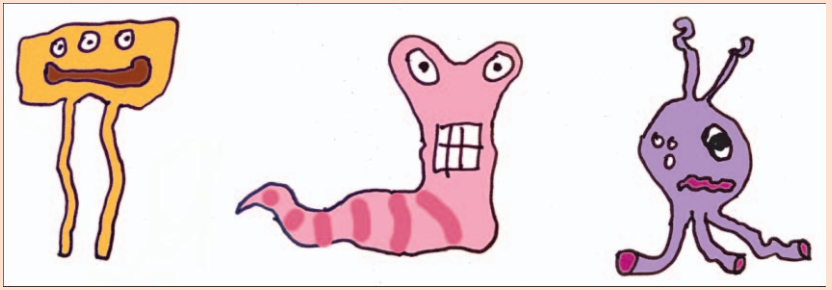
Eigene Aliens dank KI

Du kannst mithilfe von KI-Bildgeneratoren einfach und schnell ansehnliche Game-Grafiken erstellen. Auch meine drei Aliens Frapax, Snakix und Violepix sind so entstanden. Du kannst wie folgt vorgehen:

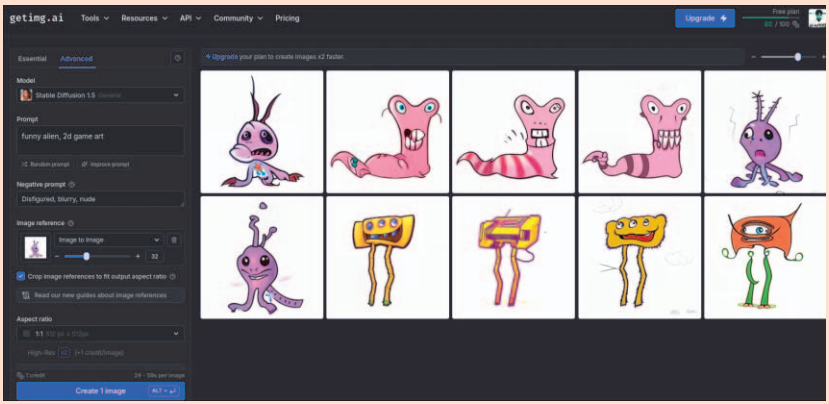
1. Zeichne mit einem schwarzen Stift eine grobe Skizze eines Aliens auf weißem Papier, mach mit dem Handy ein Foto und kopiere es auf deinen PC (am besten eignet sich eine App wie Microsoft Lens, die einen Filter für eine Schwarzweißskizze besitzt). Die Skizze muss wirklich nicht besonders kunstfertig sein, wie du an meiner erkennen kannst:



2. Benutze ein einfaches Grafikprogramm, um die Flächen mit einer Farbe deiner Wahl zu füllen. Wenn nötig, radiere störende Flecken im Hintergrund weg. Natürlich kannst du das auch alles auf Papier mit einem Filzstift machen oder auch die ganze Zeichnung im Grafikprogramm erstellen. Erstelle jedenfalls mehrere farbige Skizzen, die du als Grafikdateien mit 512 × 512 Pixeln Größe auf dem PC speicherst:



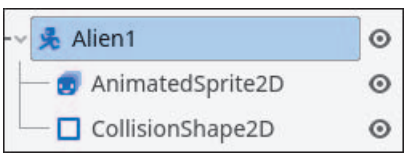
3. Jetzt kommt die KI zum Einsatz. Es gibt mehrere Webseiten, die kostenlose Bild-zu-Bild-Generatoren anbieten. Ich verwende meistens *getimg.ai*. Beachte, dass üblicherweise ein kostenloser Account erforderlich ist. Wenn nötig, lass dir dabei von deinen Eltern helfen.
4. Bild-zu-Bild-Generatoren benötigen neben dem Ausgangsbild einen *Prompt*, der die gewünschte Ausgabe beschreibt. In diesem Fall eignet sich zum Beispiel: »funny alien, 2d style game art« (Englisch versteht diese KI besser). Bei getimg.ai wähle ich als passendes **Model Stable Diffusion 1.5** sowie ein 1:1-Bildformat aus und lade die Skizze hoch. Nach einigen Sekunden bekomme ich brauchbare Ergebnisse:



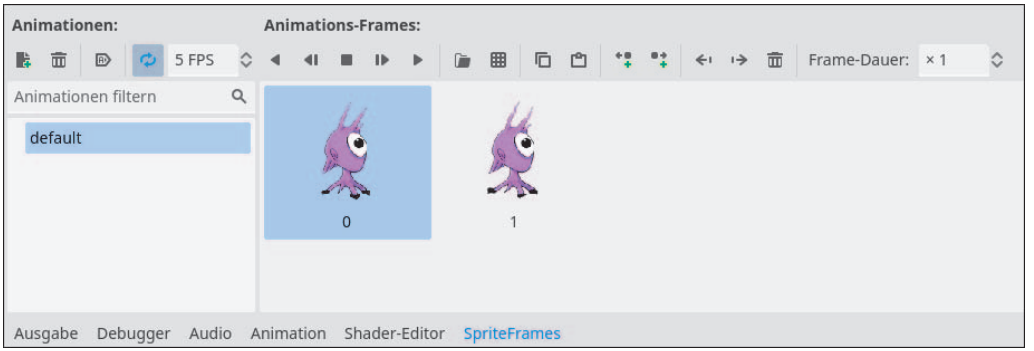
5. Du kannst Prompt und Einstellungen variieren, bis du das gewünschte Ergebnis bekommst. Speichere die fertigen Bilder und entferne mit einem Bildbearbeitungsprogramm den weißen Hintergrund (das nennt man *Freistellen*). Speichere das Bild im PNG-Format mit Transparenz (RGBA).
6. Wenn du verschiedene Körperhaltungen für eine Animation haben möchtest, verschiebe im Bildbearbeitungsprogramm zum Beispiel Beine oder Gliedmaßen etwas und speichere das Bild unter einem anderen Namen. Im kostenlosen Bildbearbeitungsprogramm *GIMP* heißt diese Funktion **Warptransformation**, aber auch in anderen Programmen ist sie leicht zu finden. Mit etwas Geduld und geschicktem Prompt kannst du übrigens auch einen Bild-zu-Bild-Generator dazu überreden, die Körperhaltung auf dem Eingabebild zu ändern oder gleich mehrere Bilder mit verschiedenen Haltungen auf einmal zu erzeugen.

Sobald die Bilder zur Verfügung stehen, kannst du die Aliens ins Spiel einbauen:

1. Lege für das erste Alien in der Spielszene einen neuen *CharacterBody2D*-Node an. Nenne ihn **Alien1** und hänge darunter einen *AnimatedSprite2D*-Node und einen *CollisionShape2D*-Node:



2. Wähle den *AnimatedSprite2D*-Node aus und erzeuge im **Inspektor** einen neuen **SpriteFrames**-Datensatz. Daraufhin erscheint unter der 2D-Ansicht der **SpriteFrames**-Editor. Ziehe die Einzelbilder des Aliens in den freien Bereich:



Dabei erscheint das Alien bereits oben im **2D**-Bereich im Raumschiff. Wenn du im **SpriteFrames**-Editor auf das Abspielen-Icon klickst, kannst du die Animation bereits sehen.

3. Stelle im **Inspektor** das **Scale**-Attribut des Sprites auf eine passende Gesamtgröße ein.

4. Hänge ein neues Skript an den *Alien*-Node. Schreibe dieses einfache Skript, damit die Animation automatisch losläuft:

```
1  extends CharacterBody2D
2
3  @onready var _animated_sprite = $AnimatedSprite2D
4
5  func _ready() -> void:
6      _animated_sprite.play()
```

5. Wähle den *CollisionShape2D*-Node aus und erzeuge im **Inspektor** einen passenden **CollisionShape**. Je nach Form deines Aliens passen verschiedene Shapes unterschiedlich gut, probiere es am besten aus. Beachte, dass sehr

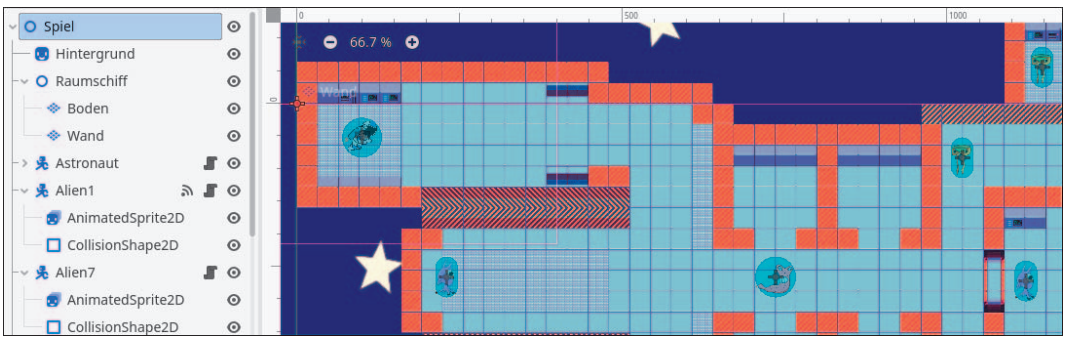
genaue Polygon-Shapes mit vielen Punkten das Spiel verlangsamen können, aber auch das beste Spielerlebnis bieten.

6. Aktiviere in der **CollisionMask** neben Layer 1 auch Layer 2, denn darin befindet sich der Astronaut, mit dem das Alien später zusammentrifft.

7. Positioniere das **Alien1** (nicht das Sprite!) an einer geeigneten Stelle im Raumschiff, etwas entfernt vom Astronauten. Es sollte dort eine gewisse freie Strecke vor sich haben, entlang der es später patrouillieren wird.

8. Kopiere den *Alien1*-Node für jedes weitere Alien, das du im Raumschiff positionieren möchtest. Für anders aussehende Aliens musst du im jeweiligen *AnimatedSprite2D*-Node neue SpriteFrames erzeugen und mit den entsprechenden Grafiken füllen. Denk dran, auch die CollisionShapes passend einzurichten.

   Wenn du Nodes kopierst, kopierst du Datensätze nicht mit, die von ihnen benutzt werden, sondern nur die Referenzen darauf. Deshalb haben alle Aliens den gleichen CollisionShape – änderst du ihn für ein Alien, ändert sich der Kollisionsbereich auch für alle anderen! Vermutlich haben deine Aliens aber unterschiedliche Formen, daher musst du mit der **Neu**-Funktion für jedes unterschiedliche Alien-Sprite auch einen neuen CollisionShape anlegen.



Bevor du die Aliens in Bewegung versetzt, probiere das Spiel ruhig einmal aus. Die Aliens sollten jetzt animiert sein, aber sonst an Ort und Stelle bleiben. Major Tim kommt nicht an ihnen vorbei – und ich fürchte, das wird nicht sein einziges Problem bleiben ...

Aliens in Bewegung

Die Aliens sollen immer geradeaus in die gleiche Richtung laufen, bis sie auf eine Wand stoßen und umkehren. Das kann eine horizontale oder vertikale Richtung sein.

1. Öffne das Skript, das dank Kopierens an allen Aliens hängt.

2. Füge oben zwei Exportvariablen ein:

```
@export var richtung = Vector2(1,0)
@export var tempo = 50
```

Das `@export`-Schlüsselwort sorgt dafür, dass die Werte im **Inspektor** der Aliens einstellbar sind. Dabei kann natürlich jedes Alien andere Werte haben. Die im Code zugewiesenen Werte sind lediglich die Standardvorgaben, die du mit Eingaben im **Inspektor** überschreiben kannst. Die vorgegebene Richtung (1,0) bedeutet von links nach rechts, da x positiv ist und y = 0.

3. Schreibe folgenden Code für die Bewegung:

```
func _physics_process(delta: float) -> void:
    _animated_sprite.flip_h= richtung.x<0
    var kollision = move_and_collide(richtung*tempo*delta)
    if kollision:
        if kollision.get_collider().name=="Wand":
            richtung *= -1
```

Dieser Code »flippt« das Sprite, falls die Bewegungsrichtung nach links führt, also x kleiner als 0 ist.

Die Methode `move_and_collide()` bewegt die Spielfigur und gibt, falls es eine Kollision gab, Infos darüber zurück. Falls das Alien mit der Wand kollidiert ist (das ist der Name des *TileMapLayer*-Nodes mit den Wänden des Raumschiffs), wird die Richtung einfach umgekehrt.

4. Setze jetzt bei jedem Alien in deinem Raumschiff die Anfangsrichtung auf einen passenden Wert. Aliens, die in horizontalen Gängen stehen, sollten

eine Richtung (1,0) oder (–1,0) erhalten. Aliens, die auf dem Bildschirm vertikal unterwegs sind, erhalten (0,1) für die Abwärtsbewegung oder (0,–1) für die Aufwärtsrichtung.

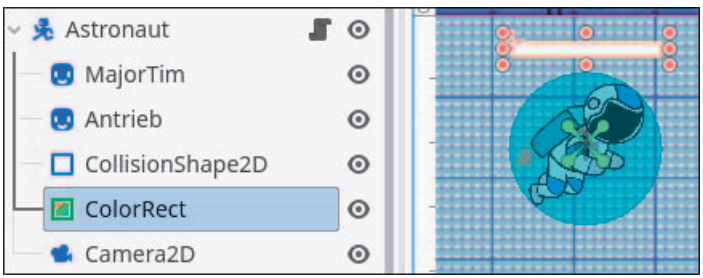
5. Du kannst auch das Tempo der Aliens unterschiedlich wählen, da auch dieses über `@export` im **Inspektor** eingeblendet wird. Aber Vorsicht! Falls ein Alien schneller unterwegs ist als Major Tim, kann er sich nicht mehr retten.

Probiere das einmal aus. Die Aliens sollten jetzt durchs Raumschiff schweben, und zwar immer hin und her. Eine Kollision mit Major Tim hat noch keine besondere Auswirkung – er wird einfach weggeschoben. Aber das ändert sich in Kürze auf ungesunde Weise ...

Hitpoints, zu viel Schaden und zurück ins Hauptmenü

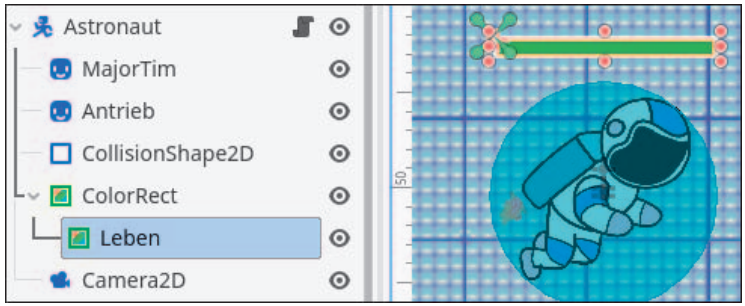
Jede Berührung zwischen Major Tim und einem Alien soll Schaden verursachen. Dieser wird von Tims Lebenspunkten (*Hitpoints*) abgezogen. Um diese anzuzeigen, brauchen wir einen Balken, der die verbleibenden Lebenspunkte darstellt.

1. Füge dem *Astronaut*-Node einen *ColorRect*-Node hinzu. Positioniere ihn über dem Astronauten und forme ihn zu einem länglichen, weißen Balken:



Der Balken wird sich mit der Figur bewegen, weil er deren Child Node ist. Er soll aber nur der Hintergrund für den eigentlichen Balken sein.

2. Kopiere den Balken, hänge ihn im Node Tree unter den ersten, färbe ihn grün und verkleinere ihn so, dass der weiße Balken als Umrandung sichtbar wird. Nenne den Balken *Leben*:



3. Erzeuge wie schon im Asteroidenspiel einen *Node2D* namens *Ereignisse* und sorge in den **Projekteinstellungen** dafür, dass er automatisch erzeugt wird. Hänge ein Skript daran, das ein Schadensereignis definiert. Die Punktzahl ist die Anzahl der verursachten Schadenspunkte:

```
1   extends Node2D
2
3   signal schaden(pkt)
```

4. Im *Alien*-Skript gibt es schon eine Kollisionsabfrage – mit der Wand. Erweitere diese wie folgt, um eine Kollision mit dem Astronauten zu erkennen:

```
func _physics_process(delta: float) -> void:
    _animated_sprite.flip_h= richtung.x<0
    var kollision = move_and_collide(richtung*tempo*delta)
    if kollision:
        if kollision.get_collider().name=="Wand":
            richtung *= -1
        if kollision.get_collider().name=="Astronaut":
            Ereignisse.schaden.emit(schaden)
```

Wie du siehst, genügen hier die beiden letzten Zeilen, die das `schaden`-Signal senden.

5. Der Astronaut muss dieses Ereignis verarbeiten, da er für seine Schadensberechnung selbst zuständig ist. Ergänze also im *Astronaut*-Skript:

```
func _init():
    Ereignisse.connect(Ereignisse.schaden.get_name(),_schaden)
```

6. Ergänze oben im Skript zwei Variablen:

```
@export var startleben=200
@onready var leben=startleben
```

Die Lebenspunkte zu Beginn des Spiels kannst du dank des `@export`-Schlüsselwortes im **Inspektor** ändern. Beim Start sorgt das `@onready`-Schlüsselwort dafür, dass die Variable `leben` auf die anfänglichen Lebenspunkte gesetzt wird. Du brauchst sowohl die anfänglichen als auch die aktuellen Lebenspunkte, um die Balkenlänge berechnen zu können.

7. Es fehlt noch die Methode, die beim Schadensereignis aufgerufen wird. Die sieht so aus:

```
func _schaden(pkt: int):
    leben -= pkt
    $ColorRect/Leben.size.x=($ColorRect.size.x*leben/startleben)-2
```

Diese Methode ist ja bereits mit dem Ereignis verdrahtet. Sie subtrahiert zunächst die übergebene Schadenspunktzahl von den Lebenspunkten, danach setzt sie die Breite (`size.x`) des grünen Lebenspunktebalkens auf eine Breite, die dem Anteil der noch verbleibenden Punkte entspricht.

Die Notation `$ColorRect/Leben` erlaubt es durch den Schrägstrich, direkt auf den Lebenspunktebalken zuzugreifen, der ein Child Node des weißen *ColorRect* ist.

Dir ist sicher aufgefallen, dass noch nichts weiter passiert, wenn Major Tim die Lebenspunkte ausgehen. Bevor du dich um seinen Tod kümmerst, probiere das Spiel einmal aus. Es gibt nämlich noch ein kleines Problem, das dir auffallen wird, wenn du einem Alien von hinten in den Rücken rennst: Es ändert sich nichts am Lebenspunktebalken!

Das liegt daran, dass der `move_and_collide()`-Aufruf im Alien nur prüft, ob es in ein anderes Kollisionsobjekt hineinläuft. Im problematischen Fall aber läuft der Astronaut in das Alien hinein.

Folglich muss auch der Astronaut die Kollision mit den Aliens behandeln:

1. Zum Glück ist das schnell gemacht. Ändere das **Astronaut**-Skript wie folgt:

```
var kollision = move_and_collide(velocity*delta)
if kollision:
    if kollision.get_collider().name.begins_with("Alien"):
        _schaden(kollision.get_collider().schaden)
```

   Natürlich musst du hier nicht das Schadensereignis bemühen, du kannst die Methode `_schaden()` direkt aufrufen.

2. Jetzt fehlt noch die Game-over-Prüfung. Aber was soll dann passieren? Ich schlage vor, du bastelst schnell eine Hauptmenüszene – die brauchst du früher oder später sowieso. Dorthin kehrt das Spiel dann zurück. Bei dieser Gelegenheit zeige ich dir, wie du die aktive Szene wechselst. Ergänze zwei Zeilen am Ende der Methode `_schaden()`:

```
if leben<=0:
    get_tree().change_scene_to_file("res://menu.tscn")
```

3. Natürlich fehlt jetzt noch die entsprechende Menüszene. Wähle im Hauptmenü von Godot **Neue Szene**. Verwende als Root Node **Benutzeroberfläche** und speichere die Szene als *menu.tscn*.

4. Gestalte ein Logo für das Spiel oder verwende mein Beispiellogo aus dem Download-Paket. Lege die Grafikdatei ins Projektverzeichnis und erzeuge dann zwei *Sprite2D*-Nodes, einen für den *Sternenhimmel*-Hintergrund und einen für das Logo:

5. Füge einen *Label*-Node hinzu, nenne ihn *Start* und ändere Text und Schriftart. Wie üblich kannst du eine beliebige Schriftart oder eine aus dem Download-Paket verwenden.

6. Positioniere die Elemente passend auf dem Bildschirm.

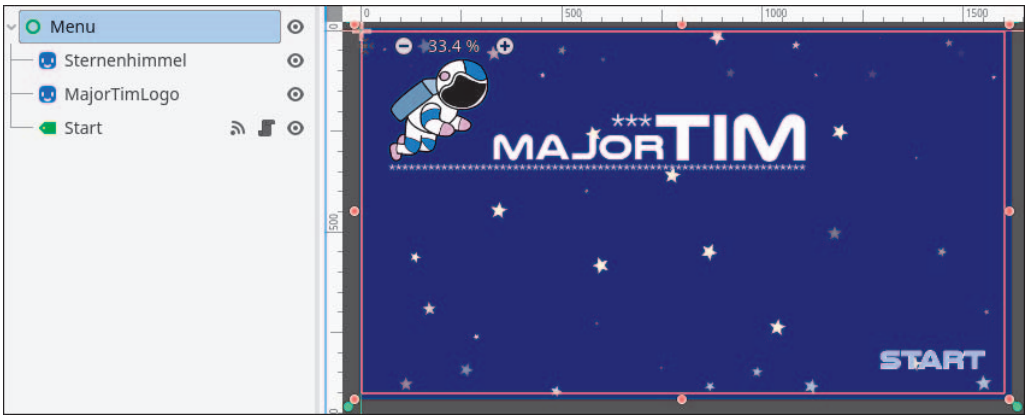
7. Erzeuge ein Skript für das **Start**-Label und programmiere Folgendes:

```
1   extends Label
2
3   func _on_gui_input(event: InputEvent) -> void:
4       if event is InputEventMouseButton and event.pressed and event.button_index==1:
5           get_tree().change_scene_to_file("res://spiel.tscn")
```

Verknüpfe im **Inspektor** das Signal `gui_input` mit der Methode.

Normalerweise können Labels jedoch nicht auf Mausklicks reagieren. Das musst du explizit anschalten, indem du im **Inspektor** unter **Control** und **Mouse** den **Filter** auf **Stop** setzt.

Die Methode prüft, ob die linke Maustaste gedrückt wurde, und wechselt dann zu der Spielszene. So könnte dein Menübildschirm jetzt aussehen:



8. In den **Projekteinstellungen** bestimmst du unter **Anwendung • Ausführen** die Menüszene als Hauptszene.

Jetzt kannst du testen, ob alles funktioniert, einschließlich des Spielstarts und des bedauerlichen und unvermeidlichen Endes unseres tapferen Majors Tim.

Hier noch ein paar Tipps zum Schluss:

- Je nachdem, wo du deine Aliens postiert hast, kann es sein, dass sie miteinander kollidieren. Das könnte ihre Bewegungspfade stören. Um das zu verhindern, ändere den Collision-Layer der Aliens auf 3, dann gleiten sie durchei-

nander durch. Damit Major Tim weiterhin mit ihnen kollidiert, füge bei ihm 3 als Collision-Mask hinzu.

- Im Moment gibt es keine numerische Anzeige der Lebenspunkte. Der zugehörige Balken schrumpft einfach immer weiter, wenn Major Tim nicht aufpasst. Damit er aber wirklich Angst vor dem Ende hat, sollte der Balken rot statt grün sein, wenn nicht mehr viel Leben in ihm steckt. Füge also eine `if`-Abfrage hinzu, um bei gewissen Lebenspunktegrenzen (zum Beispiel `startleben/3` und `startleben/5`) die Balkenfarbe (`$ColorRect/Leben.color`) in Orange (`Color.ORANGE`) und Rot (`Color.RED`) zu ändern.
- Außerdem sollte das Spiel nicht übergangslos zum Hauptmenü wechseln, wenn Major Tim am Ende ist. Füge unterhalb des Astronauten einen *Timer*-Node mit einer Laufzeit von einer halben Sekunde ein. Starte im Game-over-Fall den Timer mit `$Timer.start()` und verdrahte dessen `timeout`-Signal mit einer neuen Methode, die den eigentlichen Wechsel zum Hauptmenü vornimmt.
- Zusätzlich kannst du Major Tim, während der Timer läuft, immer kleiner werden lassen, um sein Dahinscheiden auf einfache Weise zu visualisieren. Das funktioniert am einfachsten über einen *Tween*, den ich später noch ausführlicher erklären werde. An der gleichen Stelle, also nach dem Timer-Start, solltest du außerdem den Lebenspunktebalken ausblenden:

```
$ColorRect.hide()
create_tween().tween_property($MajorTim,"scale",Vector2(0,0),0.5)
```