Tunnelflug

Ein Geschicklichkeitsspiel:
Ein Flieger soll durch einen Tunnel gesteuert werden.
Er darf die Wände des Tunnels dabei nicht berühren.
Ziel ist, das Ende des Tunnels zu erreichen.
Dann wechselt das Spiel zum nächsten Tunnel.
Das Spiel besteht also aus mehreren „**Leveln**“.

In jedem Level geschieht im Wesentlichen das gleiche: Der Flieger bewegt sich vorwärts und wird vom Spieler gesteuert. Wenn er den Rand des Tunnels berührt, verliert der Spieler ein „Leben“ und wird zurück zum Anfang bewegt.

Wenn der Spieler das Ende eines Tunnels erreicht, beginnt das nächste Level, d.h. es wird ein anderer Tunnel gezeigt, und der Flieger wird zum Anfang dieses Tunnels bewegt.

Standbilder

Um dem Spieler durch den Ablauf zu leiten, werden zu Beginn, während des Spiels und am Ende **Standbilder** gezeigt. Diese zeigen z.B. anfangs eine kurze Anleitung, vor Beginn jedes Levels eine Ankündigung, bei einem Crash, dass ein Leben verloren wurde usw.

Figuren und Variablen

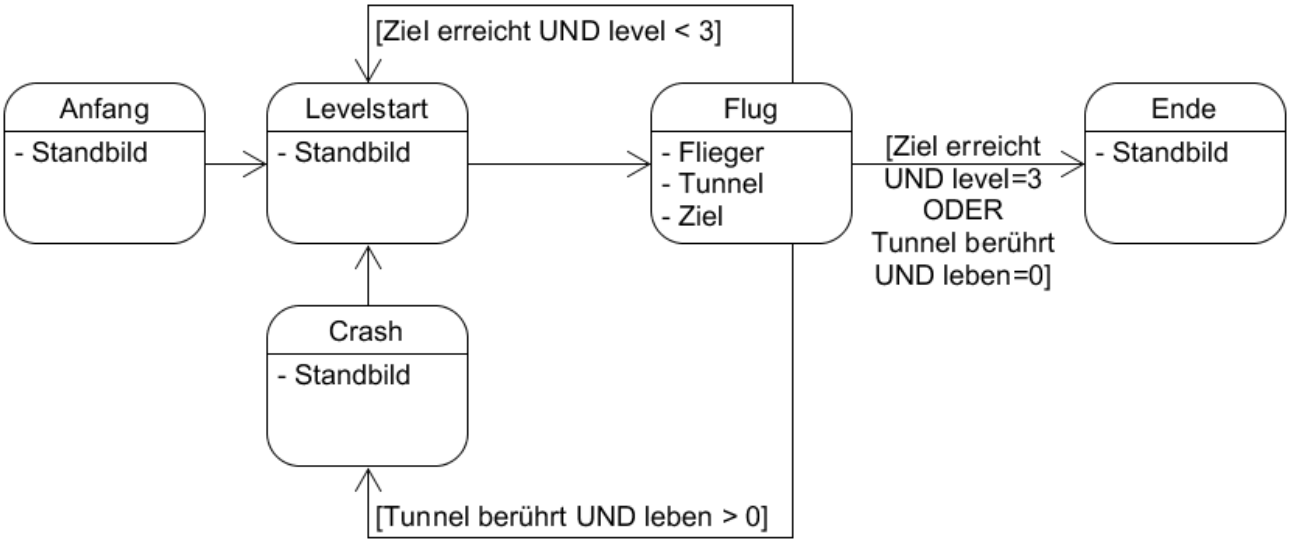
Es gibt mehrere Möglichkeiten, die Figuren für dieses Spiel zu anzuordnen. Man könnte z.B. für jeden Tunnel eine Figur erstellen, oder Klone verwenden. Da aber nie mehrere Tunnel gleichzeitig angezeigt werden, braucht es keine Klone. Mehrere Figuren wären für die Programmierung lästig. Für die Tunnel genügt daher eine einzige Figur mit mehreren Kostümen. Welches davon angezeigt wird, kann man mithilfe einer Variablen bestimmen, die das aktuelle Level speichert.
Auch für die Standbilder braucht es nur eine Figur mit verschiedenen Kostümen.

Es gibt also folgende Figuren und Variablen:

- Flieger (Variable für die Anzahl der Leben, eventuell Kostüme für Animation)
- Tunnel (pro Level ein Kostüm, Variable um das aktuelle Level zu speichern)
- Ziel (um das Tunnelende zu markieren)
- Standbild (pro Standbild ein Kostüm)

Zustandsdiagramm

Da der Ablauf des Spiels aus mehreren Phasen besteht (die man in der Informatik „Zustände“ nennt), und diese nicht einfach nacheinander, sondern je nach Erfolg oder Misserfolg in unterschiedlicher Reihenfolge ablaufen, ist es hilfreich, den Ablauf durch ein **Zustandsdiagramm** darzustellen:



Jeder **Zustand** wird als Rechteck mit abgerundeten Ecken dargestellt. Die Pfeile stellt **Wechsel** zwischen den Zuständen dar, wobei in einigen Fällen die Bedingungen für den Wechsel an die Pfeile geschrieben wurden. Zum Beispiel wechselt das Spiel vom Flug zurück zum Levelstart, wenn das Ziel erreicht wurde und noch mindestens ein Level übrig ist.

Da wir in Scratch mit mehreren Figuren arbeiten, notieren wir für die einzelnen Zustände außerdem, welche Figuren in diesem Zustand sichtbar sind. Beim Levelstart zum Beispiel wird (nur) eins der Standbilder angezeigt, während im Zustand „Flug“ die Standbilder versteckt und dafür Tunnel, Flieger und Ziel angezeigt werden.

Hinweise zu Zustandsdiagrammen:

Wenn du, wie in diesem Spiel, mit mehreren Zuständen und mit Nachrichten (die gleich vorgestellt werden) arbeitest, ist es wichtig, diese durch ein Zustandsdiagramm zu **dokumentieren**. Sonst sind später die vielen Skripte in Scratch kaum noch nachzuvollziehen, und weder die Lehrkraft, noch du selbst, kannst dich darin zurechtfinden.

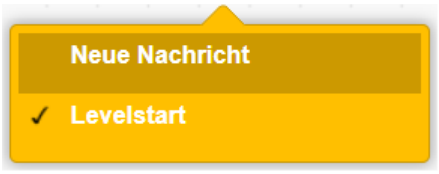
Ein nützliches Werkzeug ist das Zeichenprogramm **UMLet**, mit dem man Zustandsdiagramme zeichnen kann und das noch viele weitere Diagrammarten kennt, die im Informatikunterricht der Oberstufe wichtig werden.
Du kannst es kostenlos von https://umlet.com herunterladen.

Nachrichten

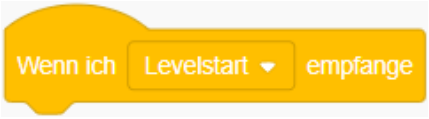
Wenn das Spiel von einem Zustand in einen anderen wechselt, sind meist mehrere oder alle Figuren betroffen. Daher braucht es eine Möglichkeit, alle Figuren über Zustandswechsel zu informieren, damit diese entsprechend reagieren können.
Für diesen Zweck gibt es in Scratch die sogenannten „Nachrichten“.

Jede Figur kann jederzeit eine Nachricht abschicken, und damit den Wechsel in einen anderen Zustand auslösen.
Dazu gibt es den Block **Sende (Nachricht) an alle**.

Mit dem gleichen Block definierst du auch, welche **Nachrichten** es gibt: Wenn du auf „Neue Nachricht“ klickst, kannst du Namen für deine Nachrichten selbst definieren. Verwende die Namen der **Zustände**, die im Zustands-diagramm dokumentiert sind: in diesem Beispiel Levelstart, Flug, Crash und Ende. Für den Anfangs-Zustand braucht es hier keine Nachricht, da das Spiel nie wieder in diesen Zustand wechselt – er wird einfach durch die grüne Fahne dargestellt.



Alle Figuren können auf Nachrichten **reagieren**, und zwar, indem sie bei Empfang der Nachricht ein Skript starten[1].



Senden und empfangen

Ein Spiel mit mehreren Figuren, die jeweils mehrere Skripte enthalten und Nachrichten senden und empfangen, wird schnell unübersichtlich. Um die Sache nicht zu kompliziert zu machen, legen wir einige Regeln für den Gebrauch von Nachrichten fest:

- Es gibt **nur zwei** bis drei Figuren, die Nachrichten **senden** dürfen. Empfangen können natürlich alle Figuren, sie müssen sich ja entsprechend der Zustände des Spiels verhalten.
  In unserem Beispiel senden nur die Hauptfiguren „Standbild“ und „Flieger“.
- Eine Figur sollte nur dann eine Nachricht **senden**, wenn sie in dem jeweiligen Zustand gerade **sichtbar** ist. Das bedeutet: In allen Zuständen außer „Flug“ ist nur das Standbild sichtbar – von hier werden in diesen Zuständen also Nachrichten gesendet. Im Zustand „Flug“ sind Flieger, Tunnel und Ziel sichtbar, dann sendet also der Flieger.

1 Leider können bereits laufende Skripte nicht auf Nachrichten reagieren. Das muss man bei Bedarf anders, z.B. über Variablen lösen. Im Tunnelflug-Spiel ist das nicht notwendig, im Spiel Asteroids (Aufgabe 3) schon.

Beispiel: Skripte mit Nachrichten

Da die **Figur „Standbild“** als einzige am Anfang des Spiels gezeigt wird, reagiert sie auf die grüne Fahne. Sie zeigt das Titelbild an und wartet, bis der Spieler die Leertaste gedrückt hat. Anschließend **sendet** sie die Nachricht „Levelstart“.

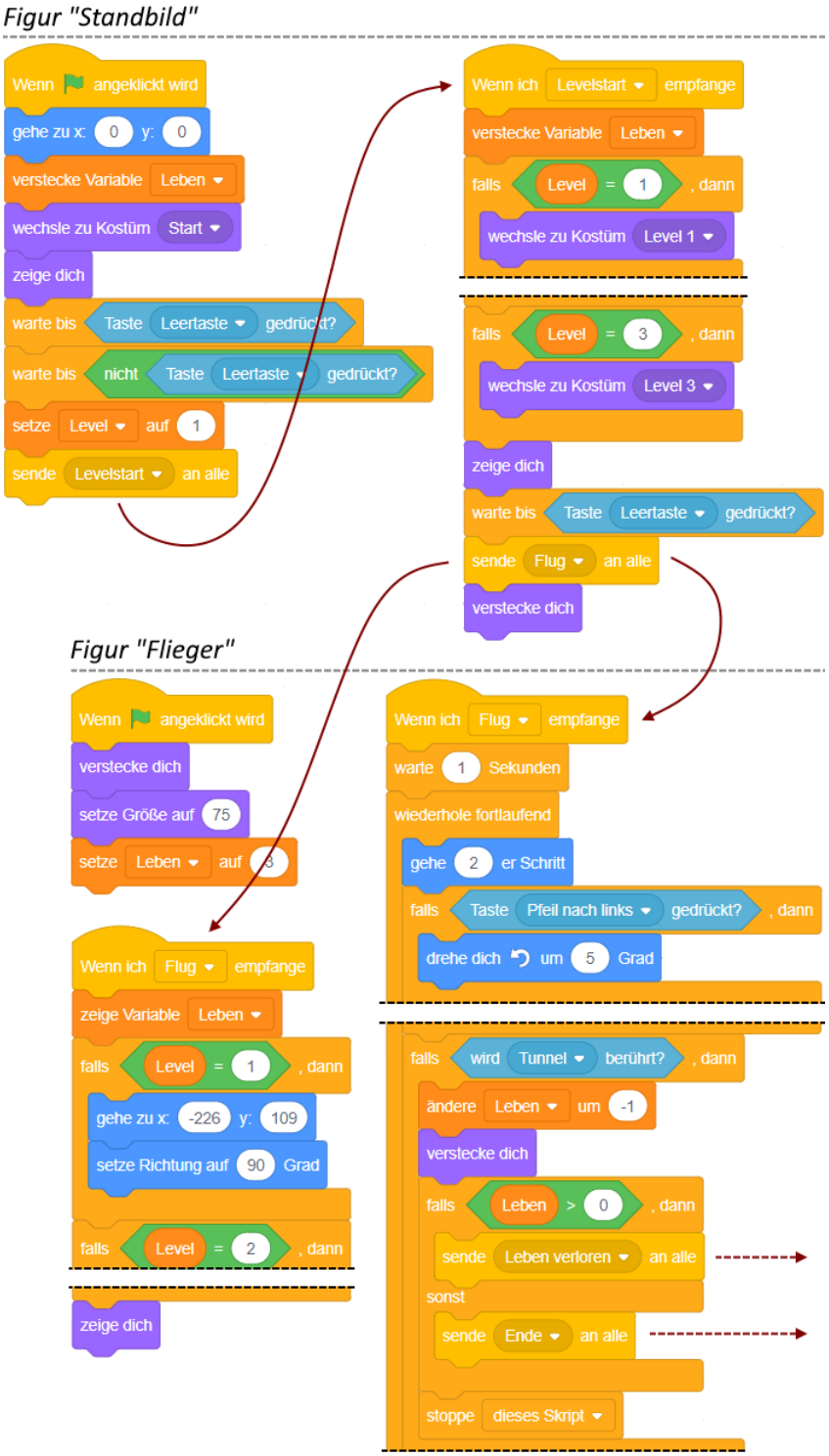
Auch im Zustand „Levelstart“ ist das Standbild die einzige sichtbare Figur. Sie zeigt entsprechend des aktuellen Levels das passende Bild an und wartet nochmals auf die Leertaste. Dann **sendet** sie die Nachricht „Flug“.

Die **Figur „Flieger“** wird erst im Zustand „Flug“ sichtbar, daher versteckt sie sich anfangs

Erst, wenn sie die Nachricht „Flug“ erhält, wird sie sichtbar. Sie bewegt sich entsprechend der Variable „Level“ an die passende Anfangsposition und fängt an, sich zu bewegen.

Wenn der Flieger den Tunnel berührt, muss er sich wieder verstecken, und dann die anderen Figuren durch eine Nachricht über den Zustandswechsel informieren.

Mit dem Block **stoppe dieses Skript** kann das Skript beendet werden. Es wird ja wieder gestartet, wenn das Spiel das nächste Mal in den Zustand „Flug“ übergeht.



Aufgabe 1: Verständnisfragen

Beantworte schriftlich folgende **Fragen** zu den auf Seite 4 abgebildeten Skripten:

a) Im Skript der Figur „Standbild“ stehen die Blöcke **warte bis < Leertaste gedrückt >** und **warte bis < nicht Leertaste gedrückt >** nacheinander. Welchen Zweck hat das? (Falls du nicht auf die Lösung kommst, probiere im Scratch-Projekt, den Block mit **nicht** wegzulassen).

b) Die Anzahl der noch übrigen „Leben“ soll während des Fluges durch den Tunnel, aber nicht auf den Standbildern angezeigt werden. Wir wird das in den Skripten realisiert?

c) Im Skript der Figur „Flieger“ ist zu sehen, dass bei Berührung mit dem Tunnel entweder die Nachricht „Leben verloren“ oder die Nachricht „Ende“ gesendet wird.

   i. Dabei wird der Flieger versteckt, und das Skript endet – warum?

   ii. Welche anderen Figuren des Projekts müssen in dem Moment außerdem versteckt werden? Welche müssen angezeigt werden?

   iii. Es gibt noch einen weiteren Fall, in dem das gleiche Skript Nachrichten versenden muss. Welcher Fall ist das, und welche Nachrichten sollten versendet werden?

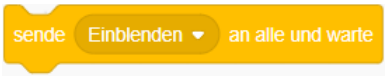
Aufgabe 2: Tunnelflug

a) Bearbeite die ausgeteilte Vorlage für den **Tunnelflug**. Sie enthält die Figuren mit ihren Kostümen und die benötigten Variablen, sowie bereits einige grundlegende Skripte.
Stelle das Spiel entsprechend der Beschreibung auf Seite 1 und des Zustandsdiagramms auf Seite 2 fertig. Verwende Nachrichten, um zwischen den Zuständen zu wechseln.

b) **Zusatzaufgabe** (nach Aufgabe 3): Einblenden und Ausblenden

Ein schöner Effekt ist, wenn bei einem Zustandswechsel nicht einfach das nächste Standbild bzw. ein Tunnel angezeigt wird, sondern diese eingeblendet werden.
Eine einfache Möglichkeit ist, beim Einblenden das nächste Bild langsam heller werden zu lassen. Du kannst das mit einer Figur erreichen, die die Größe der Bühne hat (480 x 360 Pixel) und ganz schwarz ist. Sie sollte auf der vordersten Ebene liegen, damit sie die darunter liegenden Figuren (z.B. Standbild oder Tunnel) verdeckt.
Beim Einblenden wird die Figur sichtbar gemacht, und dann mehrmals der Effekt „Durchsichtigkeit“ angewendet, z.B. 20 Mal um jeweils 5%. Es sieht dann so aus, als würde das dahinter liegende Bild heller werden.
Den gleichen Effekt kannst du umgekehrt zum Ausblenden verwenden.
Das Aus- und Einblenden kannst du durch Nachrichten starten. Die eigentlichen Zustände des Spiels werden davon nicht beeinflusst, insofern brauchst du das Diagramm nicht anzupassen.
Sende die entsprechenden Nachrichten am besten nur von der Figur für die Standbilder, um das Projekt nicht zu kompliziert zu machen.

Nützlich ist hier der Block **sende < > an alle und warte**:
Nach dem Senden wartet der Block, bis alle Skripte, die von dieser Nachricht gestartet werden, beendet sind (hier die Skripte der schwarzen Figur, die langsam durchsichtig wird).



Aufgabe 3: Asteroids

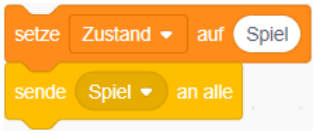
Das Spiel „Asteroids“ soll um einen Shop erweitert werden. Im Spiel erhält man für abgeschossene Asteroiden Geld („credits“). Der Shop bietet dafür Upgrades wie einen verbesserten Laser, Raketen, Schutzschild und zusätzliche Leben.

a) Zeichne mit UMLet ein **Zustandsdiagramm** für das Spiel. Verwende dazu die bereitgestellte Vorlage „asteroids.uxf“.

Zum Beginn und zum Ende des Spiels soll es ein Standbild geben.
Während des Spiels soll man jederzeit zum Shop wechseln können, um einzukaufen. Anschließend wird das Spiel an der gleichen Stelle fortgesetzt.
Falls das Raumschiff durch Kollision mit einem Asteroiden zerstört wird, wird die Animation mit der Explosion abgespielt (währenddessen kann das Raumschiff natürlich nicht weiter schießen). Falls noch Leben übrig sind, beginnt eine neue Runde, in der das Raumschiff wieder in der Mitte startet und nach und nach neue Asteroiden-Klone erzeugt werden.

b) **Programmiere** das Spiel fertig, insbesondere das Senden und Empfangen von Nachrichten.
Es gibt zwei Vorlagen zur Auswahl:
1. (schwieriger): Die Musterlösung aus dem vorigen Abschnitt, bei der du alle Neuerungen programmieren musst, also die Standbilder, den Shop und die neuen Funktionen.
2. (einfacher): Eine erweiterte Vorlage, die bereits die Skripte für die Spielmechanik enthält. Hier musst du nur die Zustandswechsel programmieren.

Für dieses Spiel müssen einige Skripte bei Zustandswechseln beendet oder unterbrochen werden. Dazu ist es sinnvoll, den aktuellen Zustand durch eine **Variable** festzuhalten:
Jedesmal, wenn du mit dem Block **sende ( ) an alle** einen Zustandswechsel auslöst, setzt du vorher die Variable auf den neuen Zustand.



Dann können Wiederholschleifen in anderen Skripten darauf reagieren: mit dem Block **wiederhole bis** < > kannst du eine Schleife beenden, wenn ein bestimmter Zustand erreicht ist. Mit den Blöcken **falls** < > und **warte bis** < > kannst du eine Schleife unterbrechen, solange das Spiel in einem bestimmten Zustand ist.