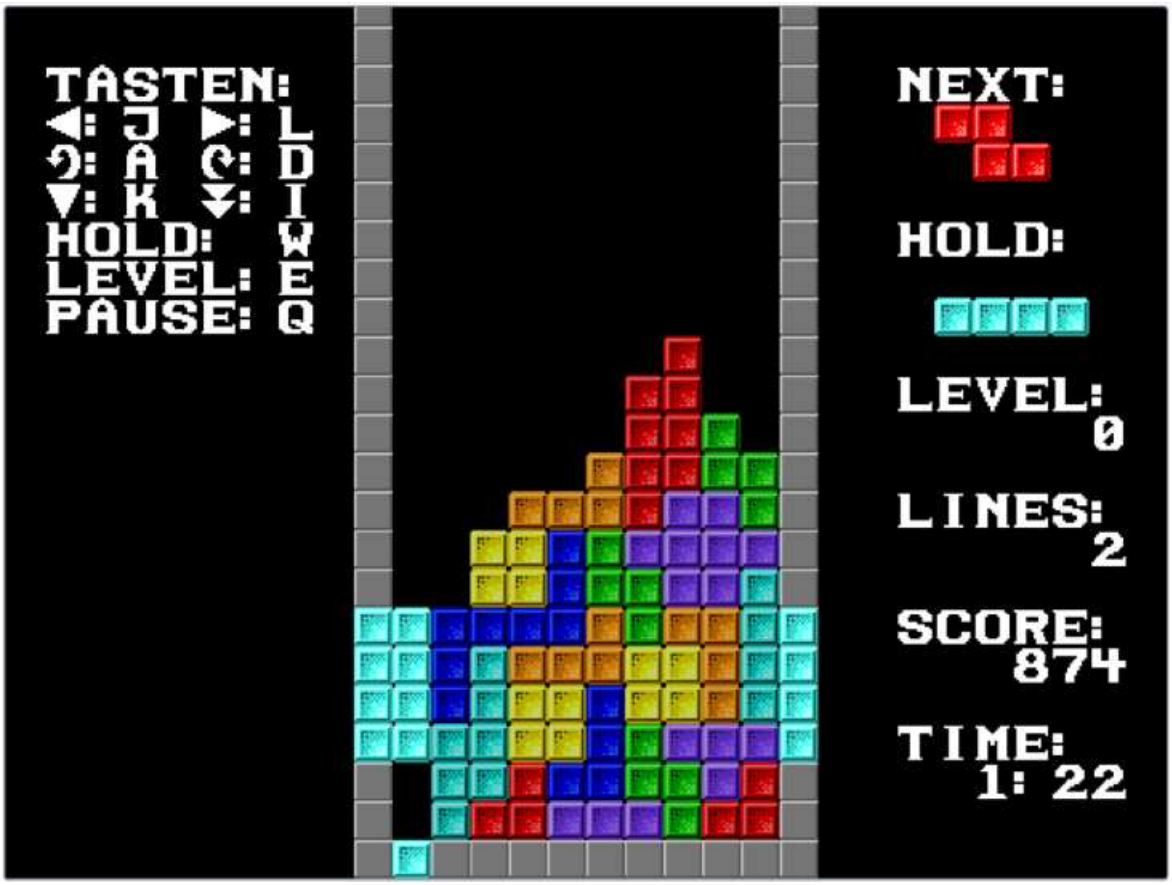
Tetris

Inhalt des Spiels

Tetris ist ein Computerspiel-Klassiker, der 1984 von Alexei Paschitnow entwickelt wurde. Mit fallenden Steinen müssen vollständige Linien gebildet werden.

Dieses Tutorial baut Schritt für Schritt einen Tetris-Klon auf und benötigt dazu Listen, eigene Blöcke, Mathematik und viel Schreibarbeit. Aber: Es lohnt sich!



Erste Überlegungen

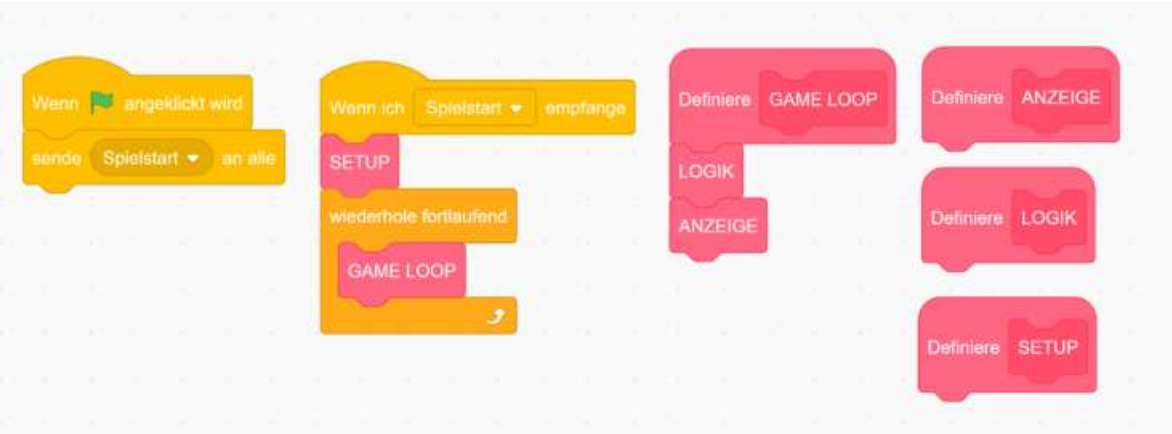
Einfache Scratch-Projekte folgen meist dem Prinzip, dass Spielelemente als eigene Figuren angelegt werden, welche ihre eigenen Codeabschnitte haben und miteinander interagieren. Für Tetris ist das jedoch ungeeignet. Beispielsweise könnten die unterschiedlichen Tetris-Steine nicht als Figuren angelegt werden, da ja später einzelne Zeilen des Spielfeldes gelöscht werden - also Teile der jeweiligen Tetris-Blöcke.

Wir werden deshalb einen anderen Weg gehen, bei dem die Spielelogik und die Darstellung strikt voneinander getrennt werden. Zu Beginn werden in einem **Setup** alle benötigte Variablen festgelegt - und wir werden einige davon haben. Danach wird fortlaufend eine "**Game Loop**" wiederholt, in der zunächst die Spiellogik überprüft wird und anschließend der aktuelle Spielstatus angezeigt. Zur Anzeige wird dann der **Malstift** verwendet, welcher erst bei den **Erweiterungen** aktiviert werden muss.

Lösche die Katze Scratchy, lege stattdessen eine neue Figur "*Feld*" durch malen an und lasse das Kostüm leer.

Lege drei eigene *Blöcke* an namens **GAME LOOP**, **ANZEIGE** und **LOGIK** und achte darauf, dass diese jeweils ohne Bildschirmaktualisierung laufen.

Dann sende beim Klicken der Fahne die Nachricht "*Spielstart*" an alle, und beim empfangen der Nachricht "*Spielstart*" führe einmal den Block "*SETUP*" und anschließend fortlaufend den Block "*GAME LOOP*" aus. **"GAME LOOP"** soll die Blöcke "*LOGIK*" und "*ANZEIGE*" aufrufen.



Wir werden später viele weitere Blöcke anlegen, welche mit "*SETUP*", "*ANZEIGE*" oder "*LOGIK*" beginnen. Diese müssen jeweils im entsprechenden Hauptblock aufgerufen werden - wenn diese Anleitung also sagt, wir sollen "*SETUP Tasten*" anlegen, dann muss dieser Block im Block *SETUP* aufgerufen werden.

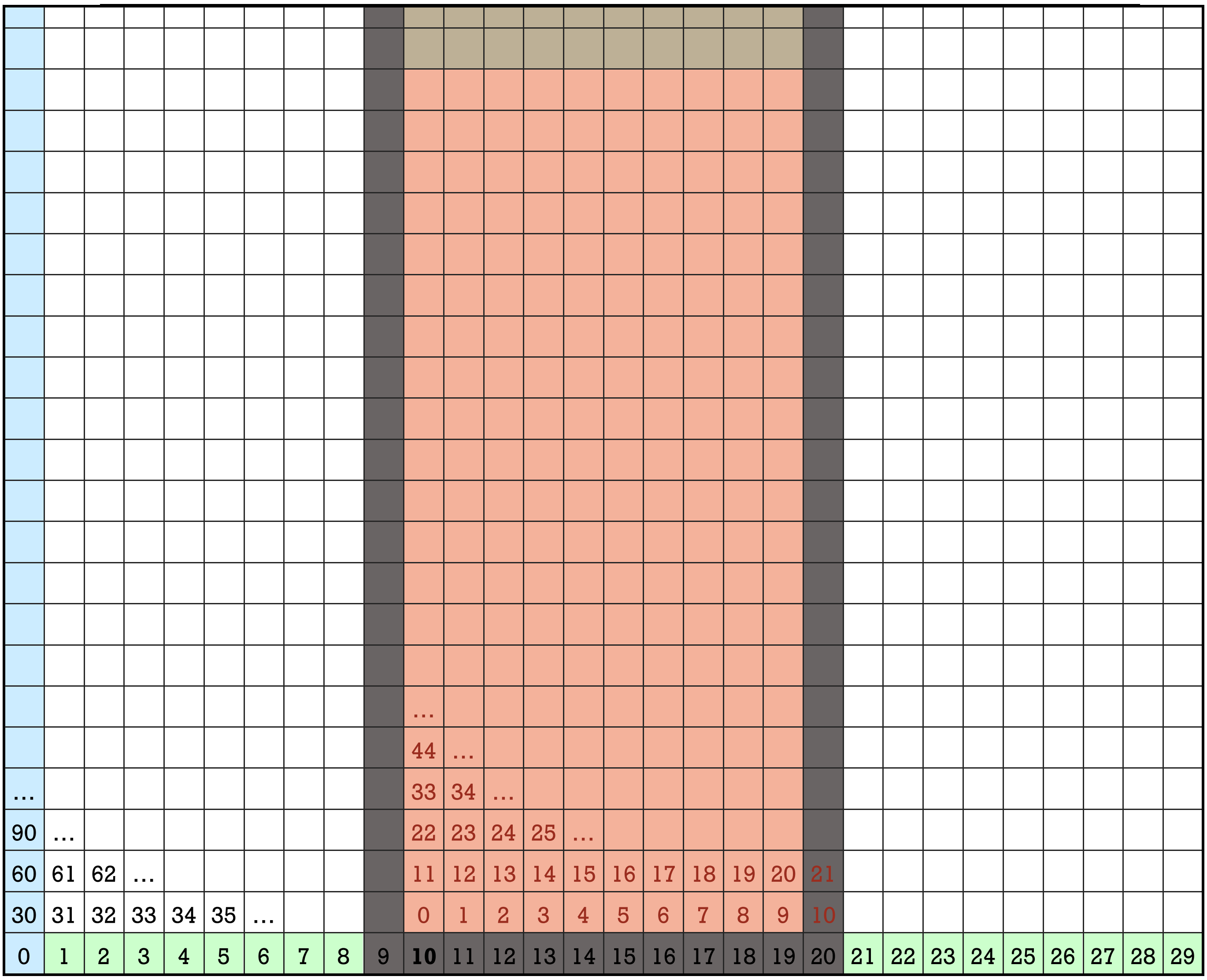
Außerdem gilt für alle eigene Blöcke in dieser Anleitung, dass sie *ohne Bildschirmaktualisierung* laufen sollen.

Das Spielfeld anlegen

Das Spielfeld von Tetris ist 10 Felder breit und 20 Felder hoch, mit 2 extra-Zeilen für das neuerscheinen von Blöcken. Da die Bildschirmfläche von Scratch 360 Pixel hoch ist, nutzen wir eine Feldgröße von 16x16 Pixeln. Denn damit haben wir 22 1/2 Felder Höhe, und können in der Breite 30 Felder unterbringen.

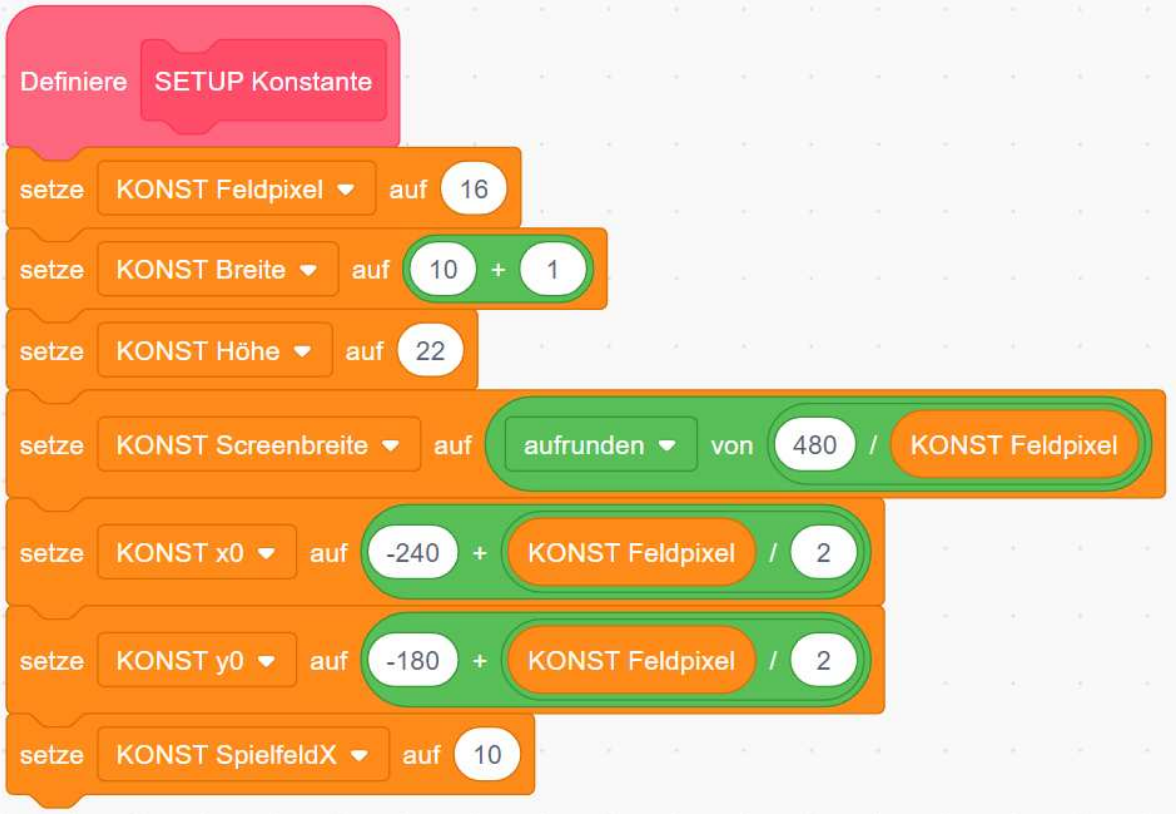
Die Grafik zeigt die Aufteilung des Spielfeldes. Schwarze Zahlen geben Positionswerte auf den Bildschirm bezogen an, rote Zahlen auf das Spielfeld bezogen.

Der rote Bereich ist das Spielfeld, der Rand ist mit grau dargestellt. Ganz oben ist der Spawnbereich.



Diese Werte werden wir häufiger benötigen, daher speichern wir sie in Variablen ab. Da wir sehr viele Variablen haben werden, benennen wir sie alle beginnend mit KONST. Wir legen an: "*KONST x0*", "*KONST y0*", "*KONST Feldpixel*", "*KONST Screenbreite*", "*KONST Breite*", "*KONST Höhe*" und "*KONST SpielfeldX*".

In einem eigenen Block "**SETUP Konstanten**" (welcher im Block "**SETUP**" aufgerufen wird) weisen wir die entsprechenden Werte zu.



"*KONST Feldpixel*" gibt an, wie viele Pixel breit und hoch ein Feld ist, also 16.

"*KONST Höhe*" bezieht sich auf die Spielfeldhöhe, welche 22 Felder hoch ist.

"*KONST Breite"* bezieht sich auf die breite des Spielfeldes, das wären 10 Felder.
Aus technischen Gründen, auf die wir später noch eingehen, benötigen wir aber noch eine zusätzliche Spalte, also 11.

"*KONST Screenbreite*" gibt an, wie viele Felder horizontal am Bildschirm Platz finden. Das sind also 480/Feldpixel, und genau so (aufgerundet) geben wir die Konstante an.

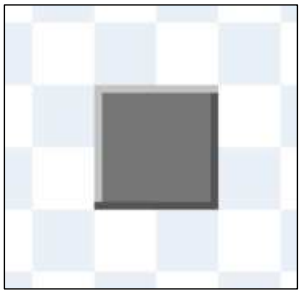
"*KONST x0*" ist die x-Position der linksten Feldspalte. Also um ein halbes Feld gegen den linken Rand verschoben (die blauen Felder in der Grafik): (-240 + Feldpixel/2)

"*KONST y0*" sind die grünen Felder und werden ähnlich bestimmt (-180 + Feldpixel/2)

"*KONST SpielfeldX*" gibt an, auf welcher X-Position das Spielfeld beginnen soll. Für eine mittige Ausrichtung wählen wir hier den Wert 10.

Erste Kostüme anlegen

Unsere Figur "*Feld*" hat ja bereits ein leeres Kostüm, dem wir nun den Namen *" "* geben - ein Leerzeichen. Achte darauf, dass auch tatsächlich ein Leerzeichen eingegeben wird und nicht etwa der Name leer gelassen - das wird hier noch wichtig.



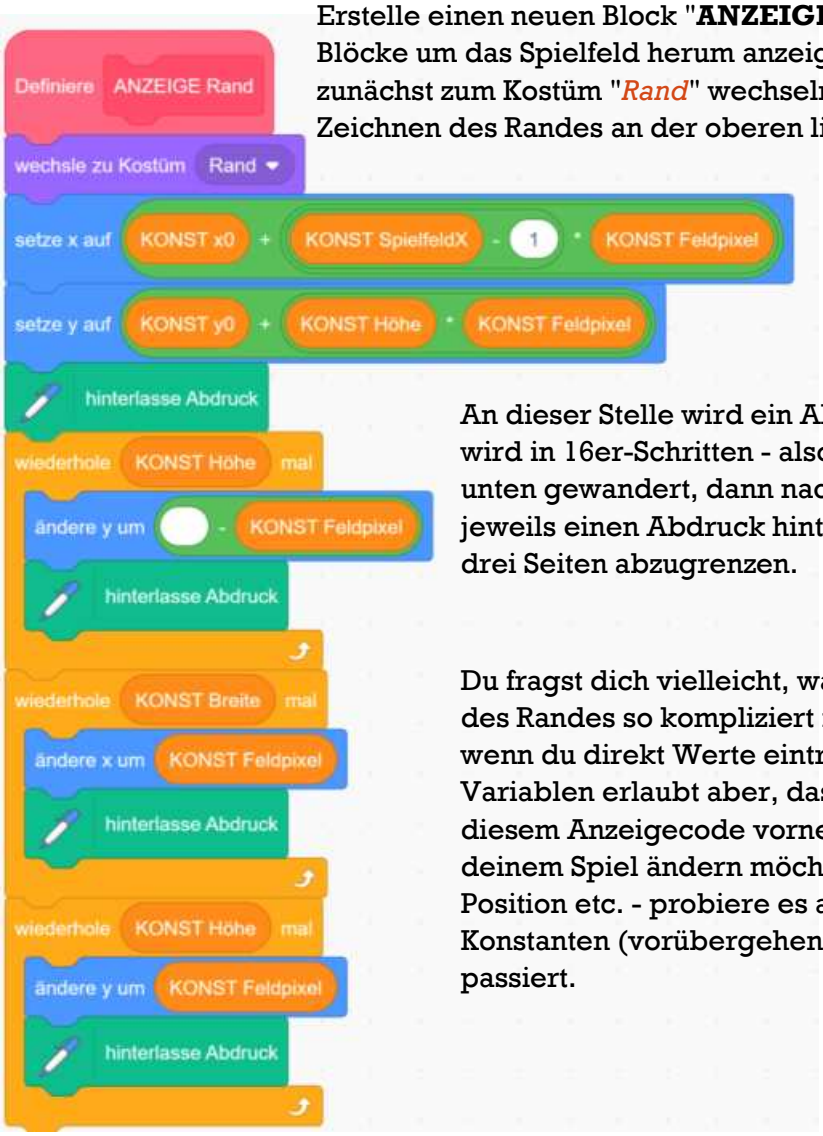
Dann legen wir ein neues Kostüm an und nennen es "*Rand*". Dieses soll einfach ein graues Quadrat sein, welches 16 Pixel hoch und 16 Pixel breit ist.
Wähle dazu am Besten *"In Rastergrafik umwandeln"* am unteren Bildschirmrand und zeichne es so ein, dass 4 der Hintergrundquadrate bedeckt sind. Durch eine Reihe Pixel in hellerem Grau oben und links, sowie dunklerem Grau unten und rechts sieht es aus wie ein Würfel mit abgerundeten Kanten.

Zeichne auch gleich eine Figur "*Block*". Diese ist genauso groß, wird aber später die herunterfallenden Tetris-Blöcke repräsentieren.
Tobe dich gerne aus was die Gestaltung betrifft, aber ein einfaches Quadrat reicht auch.

Damit die Kostüme mit all ihren Pixeln angezeigt werden, setze die *Größe* auf 200.

Anzeige des Randes

Erstelle einen neuen Block "**ANZEIGE Rand**". Wir möchten die Blöcke um das Spielfeld herum anzeigen. Dazu soll unser Feld zunächst zum Kostüm "*Rand*" wechseln. Dann möchten wir mit dem Zeichnen des Randes an der oberen linken Ecke beginnen.



Wir wählen y als
y0 + (Höhe * FeldPixel)
und x als
x0+((SpielfeldX-1)*FeldPixel).

An dieser Stelle wird ein Abdruck hinterlassen. Von dort wird in 16er-Schritten - also *Feldpixel*-Schritten - nach unten gewandert, dann nach rechts und wieder hoch, jeweils einen Abdruck hinterlassend um das Spielfeld an drei Seiten abzugrenzen.

Du fragst dich vielleicht, warum allein schon das zeichnen des Randes so kompliziert ist. Es ginge natürlich auch, wenn du direkt Werte einträgst. Die Nutzung von Variablen erlaubt aber, dass du keine Änderungen an diesem Anzeigecode vornehmen musst, wenn du etwas an deinem Spiel ändern möchtest - die Spielfeldgröße, deren Position etc. - probiere es aus, indem du unsere Konstanten (vorübergehend) veränderst und schaust, was passiert.

Anzeige des Spielfeldes

Im Spielfeld müssen die bereits gefallenen Blöcke angezeigt werden. Damit wir wissen, an welchen Positionen Blöcke liegen, legen wir nun eine *neue Liste* an namens *„Spielfeld“.*

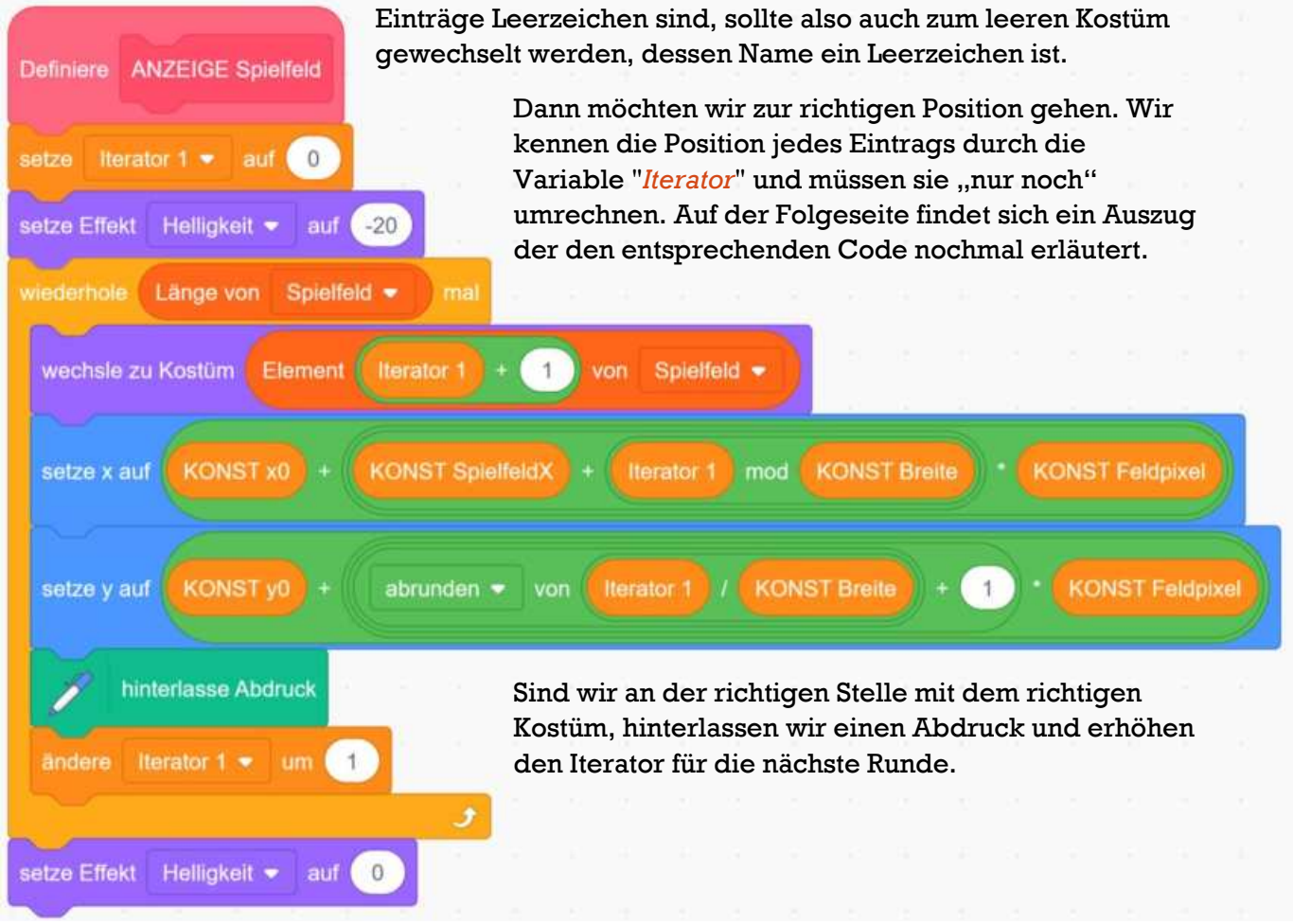
Wir nummerieren die Positionen unseres Spielfeldes einfach durch, wobei wir nach einer vollständigen Reihe von 10 noch eine zusätzliche, elfte Position einfügen, von der wir wissen, dass sie der Rand ist. Betrachte dazu die roten Zahlen in der Grafik vor ein paar Seiten.

Lege einen Block "**SETUP Liste Spielfeld**" an, in dem alles aus dem Spielfeld gelöscht wird, und dann *breite * höhe* mal ein *Leerzeichen* zur Liste hinzugefügt wird. (Wieder gilt: Achtung, es muss tatsächlich genau ein Leerzeichen sein!)

Ein neuer Block **"ANZEIGE Spielfeld"** soll das Spielfeld anzeigen. Für jeden Eintrag in der Liste *Spielfeld* wird ein Abdruck an der entsprechenden Position im entsprechenden Kostüm hinterlassen.

Setzen wir zunächst eine neue Variable "*Iterator*" auf 0. "Iterieren" heißt so viel wie wiederholen, und der "*Iterator*" zählt mit, die wie vielte Wiederholung es ist.

Wir wiederholen "*Länge von Spielfeld*" mal, denn wir wollen jeden Eintrag ansehen. Da die Liste mit dem Eintrag Nummer 1 beginnt, schauen wir uns das Element "*Iterator + 1*" an - bzw. wechseln zum Kostüm mit diesem Namen. Da bislang alle Einträge Leerzeichen sind, sollte also auch zum leeren Kostüm gewechselt werden, dessen Name ein Leerzeichen ist.



Dann möchten wir zur richtigen Position gehen. Wir kennen die Position jedes Eintrags durch die Variable "*Iterator*" und müssen sie „nur noch“ umrechnen. Auf der Folgeseite findet sich ein Auszug der den entsprechenden Code nochmal erläutert.

Sind wir an der richtigen Stelle mit dem richtigen Kostüm, hinterlassen wir einen Abdruck und erhöhen den Iterator für die nächste Runde.

Sehen wir uns diese Formel mal im Detail an:

x0 + (SpielfeldX + (Iterator mod Breite)) * Feldpixel

Anhand der Klammern (bzw. der Verschachtelung der Blöcke) wissen wir, dass wir mit „*Iterator mod Breite*“ beginnen müssen. „*mod*“ steht für Modulo und gibt den Rest einer Division zurück. Beispielsweise 7 mod 5 = 2, denn 7/5 ist 1 mit 2 Rest.

14 mod 11 ergibt 3. Vergleichen wir das mit unserem Spielfeld erkennen wir, dass wir mit dem Modulo die Werte der untersten Zeile bekommen – welche für eine x-Position relevant sind. Dann addieren wir SpielfeldX, welche die Position des Spielfeldes angibt, und bekommen den Wert der untersten Reihe – für 14 also 13.

	…									
	44	…								
	33	34	…							
	22	23	24	25	…					
	11	12	13	**14**	15	16	17	18	19	20	21
	0	1	2	3	4	5	6	7	8	9	10
9	**10**	11	12	13	14	15	16	17	18	19	20

Da das die X-Position in Feldern ist multiplizieren wir noch mit unseren Feldpixeln, um die Position in Pixeln zu erhalten – relativ zu unserem *x0*, weswegen wir den Wert zu diesem addieren.

Diese Formel ist ganz ähnlich:

y0 + (abrunden(Iterator/Breite)+1)*Feldpixel.

Wir beginnen wieder ganz innen und dividieren diesmal. Eine Division sagt uns ja, wie oft eine Zahl in eine andere hineinpasst. Wir sehen in der ersten roten Spalte deutlich, dass die Zahl mit jedem y um 11 steigt. Ensprechend sagt uns die Division, auf welcher Zeile wir uns befinden – Im Falle von 14 etwa in der Zeile 1 (was nicht die erste Zeile ist, welche die Zeile 0 ist).

Durch abrunden entfernen wir den Rest der Division, da dieser ja nur das x betrifft und uns beim y nur stört. Wir ergänzen 1, da unser Spielfeld nicht in der grauen Randzeile beginnt, und müssen wieder mit dem Pixelmaß unserer Felder multiplizieren.

Auch hier gilt, dass wir den y-Wert relativ zu unserem eigenen Nullpunkt erhalten, welcher in y0 gespeichert ist. Deshalb addieren wir erneut zu diesem.

Wenn dir noch nicht klar ist, wie das funktioniert, frag besser nach.
Solche Überlegungen verfolgen uns durch das gesamte Projekt!

Einzelblöcke fallen lassen

Jeder *Tetromino*, wie die Tetrissteine heißen, besteht aus 4 Quadraten. Da wir uns Schritt für Schritt herantasten wollen können wir die Tetrominos vorerst ignorieren und mit diesen einzelnen Quadraten spielen.

Beginnen wir hier von innen nach außen: Für jeden Fallblock müssen wir prüfen können, ob er sich an einer erlaubten Position befindet, die auch Sinn ergibt. In einem Block „**Ist Position erlaubt**“ stellen wir diese Frage, und beantworten sie in einer Variable „*Erlaubt*“ erstmal *ja*. Dann prüfen wir auf 3 Umstände:

1) Ist der Block unten "durchgefallen"?
Eine Position niedriger als 0 würde bedeuten, dass der Block unten außerhalb des Spielfeldes ist - im grauen Randbereich. Dort soll er natürlich nicht hin, deshalb setzen wir erlaubt auf nein.

2) Ist der Block irgendwo, wo schon ein Block liegt?
Wir überprüfen, ob das Element des Spielfeldes, welches der Position entspricht, ein Leerzeichen ist. Wenn dem nicht so ist, ist es nämlich bereits belegt, also kann kein anderer Block dort sein - und wir setzen erlaubt auf nein.

3) Ist der Block links oder rechts "hinausgefallen"?

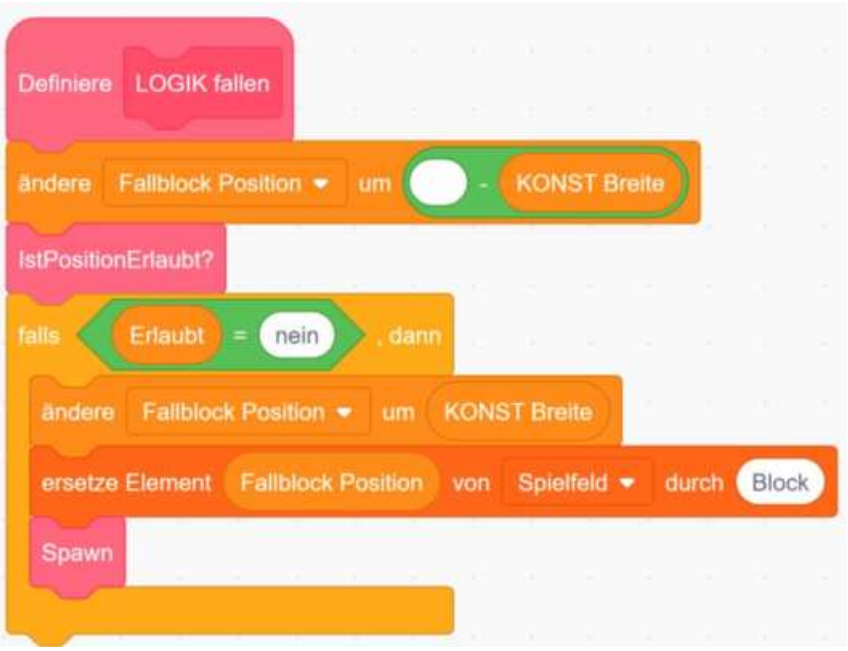
Landet der Block in der elften Spalte, so steckt er in der Wand. Wir können die Modulo-Funktion nutzen um festzustellen, ob der Block in der Randspalte ist, und es ihm ebenfalls verbieten.

Also: Jedes Mal, wenn wir unseren Block "*Ist Position erlaubt*" aufrufen, hat die Variable "*Erlaubt*" hinterher entweder den Wert "*ja*" oder den Wert "*nein*".

Erzeugen wir einen weiteren Block, "**LOGIK fallen**", in dem wir dafür sorgen wollen, dass Fallblöcke ihrem Namen gerecht werden.

In **LOGIK fallen** ändern wir die *Fallblock-Position* um *(- KONST breite)* - wenn du die vorherige Seite verstanden hast weißt du, dass du ihn damit um ein Feld nach unten verschiebst, er "fällt" also.

Dann fragen wir nach, ob er an einer erlaubten Position ist – falls nicht, schieben wir ihn wieder zurück und übertragen ihn in unsere Spielfeld-Liste.
Außerdem *spawn*en wir in diesem Fall einen neuen Block.



Der Block „**Spawn**“ macht vorerst nichts anderes, als die Fallblock-Position zu setzen – nämlich auf 224, in der Mitte der 21. Reihe.

LOGIK fallen hast du gewiss schon in den **LOGIK**-Block eingefügt. Spawn muss auch eingefügt werden, allerdings im Setup. Eh klar: Es muss nur einmal ein Block gespawnt werden, alle Folgeblöcke spawnen dann, wenn der vorhergehende irgendwo liegengeblieben ist.

Damit man einen Fallblock auch sieht, muss er angezeigt werden. Bislang kannst du ihn nur sehen, nachdem er gelandet ist.

Erzeuge einen neuen Block „**Anzeige Fallblock**“ in dem zum Kostüm „Block“ gewechselt wird, dann die x- und y- Position gesetzt und schließlich ein Abdruck hinterlassen. Die Formel zum setzen von x und y dürfte dir bekannt vorkommen.

Bewegung nach links und rechts

Bestimmt kennst du bereits zwei Arten von Bewegungssteuerungen.

Zum Einen kann man mit dem Block „Wenn Taste [] gedrückt“ direkt auf einzelne Tasteneingaben reagieren, mit dem Nachteil, dass man sie immer wieder loslassen muss, bevor man sie erneut drücken kann.

Zum Anderen kann man in einer fortlaufenden Wiederholung abfragen, ob „Taste [] gedrückt?“ ist, womit etwas auch wirklich so lange geschieht, wie man die Taste gedrückt hält.

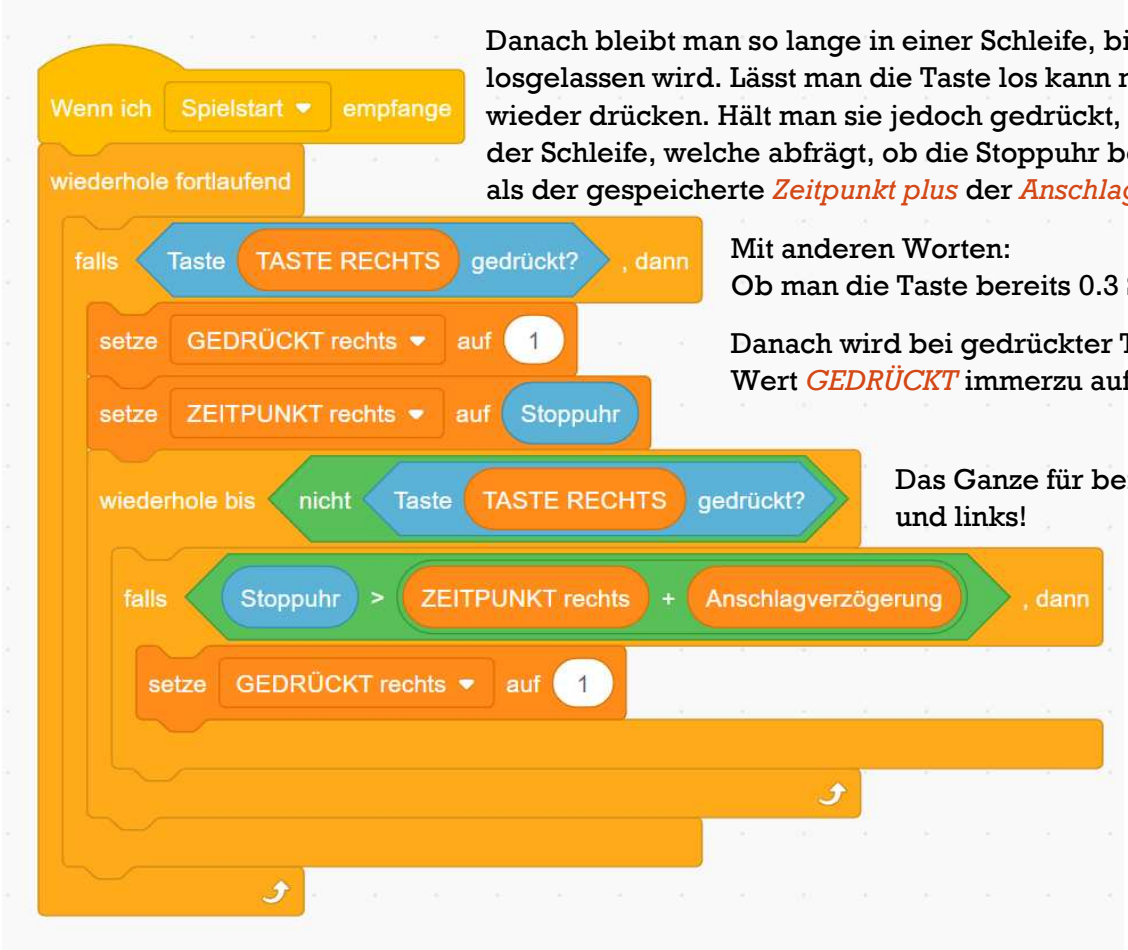
Tetris benötigt ein bisschen von beidem. Man möchte sicherstellen, dass sich ein Fallblock mit einem Tastendruck auch wirklich nur um eine Spalte verschiebt, aber auch ein gedrückt halten möglich ist. Zu diesem Zweck nutzen wir eine „*Anschlagverzögerung*“. Wir benötigen dafür gleich mehrere Variablen:

Anschlagverzögerung, ZEITPUNKT rechts, TASTE rechts, GEDRÜCKT rechts, ZEITPUNKT links, TASTE links, GEDRÜCKT links.

Viele dieser Variablen werden im SETUP gesetzt. Setze die *Taste rechts* in einem eigenen Block **SETUP Tasten** auf „**l**“ und die *Taste links* ebendort auf „**j**“. In einem Block **SETUP Zeiten** werden *Zeitpunkt links* sowie *Zeitpunkt rechts* auf **0** gesetzt, außerdem die *Anschlagverzögerung* auf **0.3**. Am Ende von SETUP Zeiten sollte auch die *Stoppuhr zurückgesetzt* werden.

Der Code, um gedrückte Tasten auszulesen, sieht wie folgt aus:

Bei *Spielstart* beginnt eine *fortlaufende Wiederholung*, in der geprüft wird, ob eine Taste gedrückt wird. Ist dem so wird die zugehörige „*GEDRÜCKT*“-Variable auf **1** gesetzt – diese wird später in der Spiellogik ausgelesen. In der entsprechenden *ZEITPUNKT*-Variable wird der Wert der *Stoppuhr* erfasst.



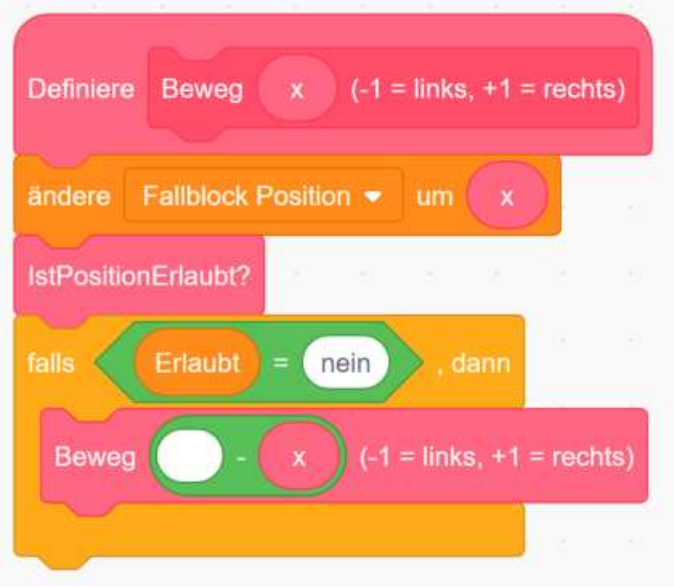
Danach bleibt man so lange in einer Schleife, bis die Taste losgelassen wird. Lässt man die Taste los kann man sie sofort wieder drücken. Hält man sie jedoch gedrückt, verweilt man in der Schleife, welche abfragt, ob die Stoppuhr bereits höher ist als der gespeicherte *Zeitpunkt plus* der *Anschlagverzögerung*.

Mit anderen Worten:
Ob man die Taste bereits 0.3 Sekunden hält.

Danach wird bei gedrückter Taste auch der Wert *GEDRÜCKT* immerzu auf **1** gesetzt.

Das Ganze für beide Tasten, rechts und links!

Was soll geschehen, wenn eine Taste gedrückt wird? Erstelle dazu einen neuen Block „**Beweg**“ mit einem Eingabefeld. In das Eingabefeld kommt später +1 für eine Bewegung nach rechts und -1 für eine Bewegung nach links.



Da wir derzeit nur einen einzelnen Block haben ist es denkbar einfach: Ändere dessen Position um den eingegeben Wert.

Überprüfe dann, ob die neue Position erlaubt ist. Ist sie es nicht, so wird einfach eine Bewegung in die Gegenrichtung aufgerufen. Da der Block so wieder an seinen Ursprungspunkt zurückkehrt muss diese Position erlaubt sein.

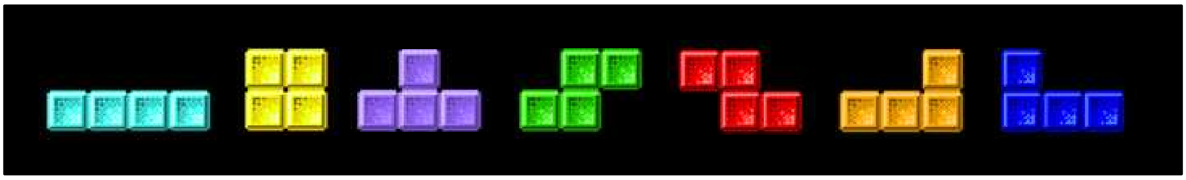
Aufgerufen wird der Block **Beweg** in einem neuen Block **LOGIK Bewegen**.

Vorerst ruft dieser lediglich den Block „**Beweg**“ auf, basierend auf den *GEDRÜCKT*-Variablen, und setzt diese im Anschluss auf null zurück.

Probiere es aus. Wenn du alles richtig gemacht hast, sollten die Steine nun von selber fallen und sich mit den Tasten j und l (oder anderen, wenn es dir lieber ist) nach links und rechts steuern lassen, sowohl mit einzelnen kurzen Drückern als auch mit längerem Druck.

Tetrominos erzeugen

Es gibt 7 verschiedene Tetris-Blöcke oder Tetrominos, welche nach ihnen ähnlichen Buchstaben benannt sind - I, O, T, S, Z, L, und J. Jeder davon besteht aus 4 Quadraten und kann jeweils um 90 Grad gedreht werden.



Überlegen wir zunächst, was es bedeuten würde, wenn ein solcher Stein an einer bestimmten Position wäre, und wie er sich drehen würde.
Mit einem x wird markiert, an welcher Position sich der gesamte Block befindet. Die anderen Felder können entsprechend ihrer relativen Position dazu definiert werden.

+10	+11	+12	+13
-1	**x**	**+1**	**+2**
-12	-11	-10	-9
-23	-22	-21	-20

+10	+11	**+12**	+13
-1	x	**+1**	+2
-12	-11	**-10**	-9
-23	-22	**-21**	-20

+10	+11	+12	+13
-1	x	+1	+2
-12	**-11**	**-10**	**-9**
-23	-22	-21	-20

+10	**+11**	+12	+13
-1	**x**	+1	+2
-12	**-11**	-10	-9
-23	**-22**	-21	-20

+10	**+11**	**+12**
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	**+12**
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	**+12**
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	**+12**
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	+12
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	+12
-1	**x**	**+1**
-12	**-11**	-10

+10	+11	+12
-1	**x**	**+1**
-12	**-11**	-10

+10	**+11**	+12
-1	**x**	+1
-12	**-11**	-10

+10	**+11**	**+12**
-1	**x**	+1
-12	-11	-10

+10	**+11**	+12
-1	**x**	**+1**
-12	-11	**-10**

+10	+11	+12
-1	**x**	**+1**
-12	**-11**	-10

+10	+11	+12
-1	**x**	+1
-12	**-11**	-10

+10	**+11**	+12
-1	**x**	**+1**
-12	-11	-10

+10	+11	**+12**
-1	**x**	**+1**
-12	**-11**	-10

+10	+11	+12
-1	**x**	+1
-12	**-11**	**-10**

+10	**+11**	+12
-1	**x**	+1
-12	-11	-10

+10	+11	**+12**
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	+12
-1	**x**	+1
-12	**-11**	**-10**

+10	+11	+12
-1	**x**	**+1**
-12	-11	-10

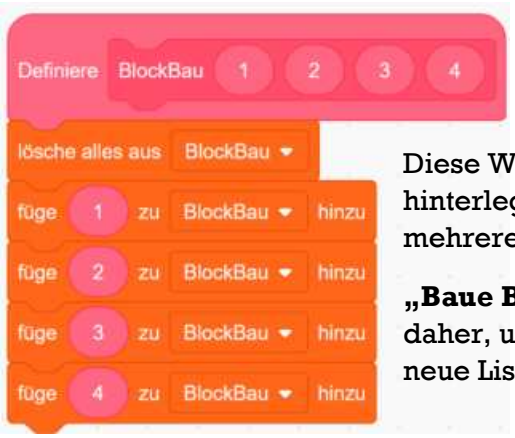
+10	**+11**	+12
-1	**x**	+1
-12	**-11**	-10

+10	+11	+12
-1	**x**	**+1**
-12	-11	-10

+10	**+11**	**+12**
-1	**x**	+1
-12	**-11**	-10

+10	+11	+12
-1	**x**	**+1**
-12	-11	**-10**

+10	**+11**	+12
-1	**x**	+1
-12	**-11**	-10



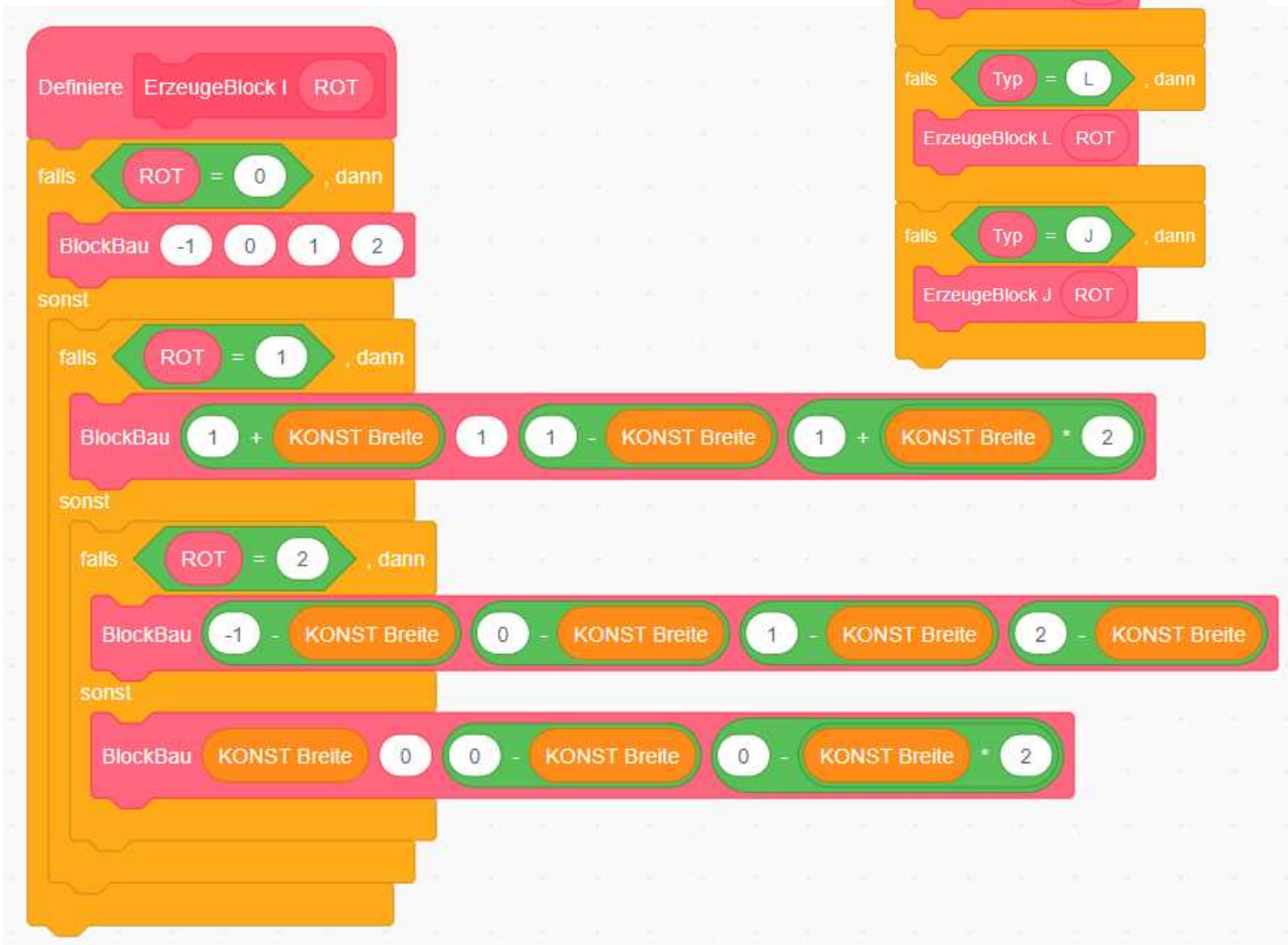
Diese Werte müssen nun in Scratch hinterlegt werden. Dazu werden gleich mehrere eigene Blöcke definiert.

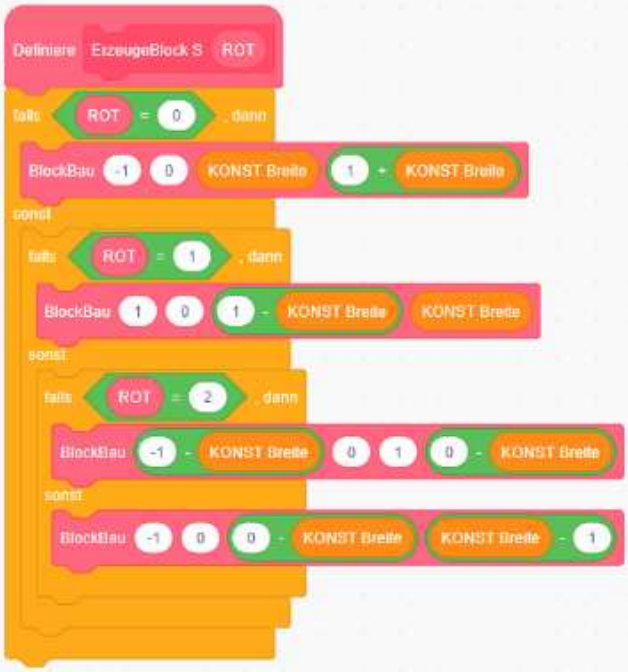
„Baue Block“ kommt mit 4 Eingabefeldern daher, und überträgt diese Werte in eine neue Liste *„BlockBau“.*

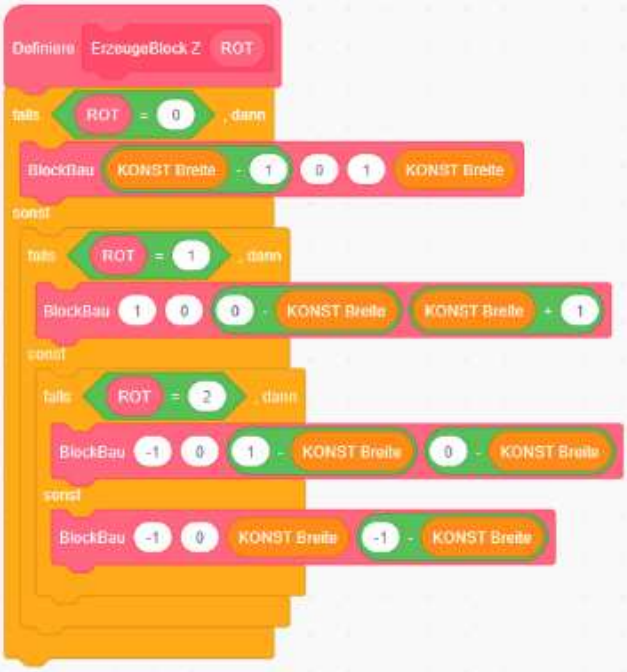
„Erzeuge Block“ benötigt zwei Eingabefelder, Typ und Rotation. Der Typ ist dabei der Buchstabe des Blocks, und es wird je nachdem der entsprechende **ErzeugeBlock X** aufgerufen, den es für jeden möglichen Block geben muss.

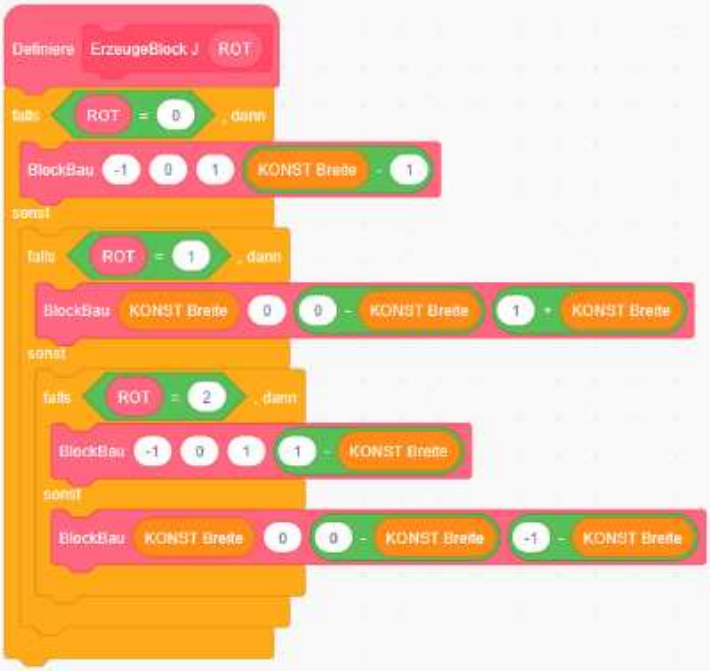
„Erzeuge Block X“ benötigt die Rotation als Eingabe, und ruft **„BlockBau“** mit den Werten auf, die in der Übersicht der Vorseite auch angeführt sind – natürlich unter Nutzung der Variable *„KONST Breite“*. Beispielhaft an **Erzeuge Block I** zu sehen.

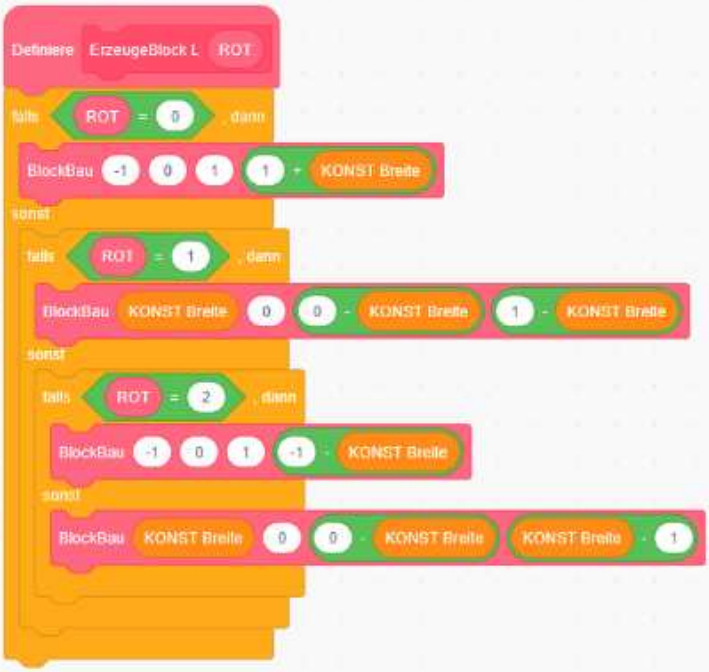
Einzig **Erzeuge Block O** weicht etwas ab, da für diesen Block keine Rotation notwendig ist.




Definiere ErzeugeBlock S ROT
falls ROT = 0 , dann
BlockBau -1 0 KONST Breite 1 + KONST Breite
sonst
falls ROT = 1 , dann
BlockBau 1 0 1 - KONST Breite KONST Breite
sonst
falls ROT = 2 , dann
BlockBau -1 - KONST Breite 0 1 0 - KONST Breite
sonst
BlockBau -1 0 0 - KONST Breite KONST Breite - 1


Definiere ErzeugeBlock Z ROT
falls ROT = 0 , dann
BlockBau KONST Breite - 1 0 1 KONST Breite
sonst
falls ROT = 1 , dann
BlockBau 1 0 0 - KONST Breite KONST Breite + 1
sonst
falls ROT = 2 , dann
BlockBau -1 0 1 - KONST Breite 0 - KONST Breite
sonst
BlockBau -1 0 KONST Breite -1 - KONST Breite


Definiere ErzeugeBlock J ROT
falls ROT = 0 , dann
BlockBau -1 0 1 KONST Breite - 1
sonst
falls ROT = 1 , dann
BlockBau KONST Breite 0 0 - KONST Breite 1 + KONST Breite
sonst
falls ROT = 2 , dann
BlockBau -1 0 1 1 - KONST Breite
sonst
BlockBau KONST Breite 0 0 - KONST Breite -1 - KONST Breite


Definiere ErzeugeBlock L ROT
falls ROT = 0 , dann
BlockBau -1 0 1 1 + KONST Breite
sonst
falls ROT = 1 , dann
BlockBau KONST Breite 0 0 - KONST Breite 1 - KONST Breite
sonst
falls ROT = 2 , dann
BlockBau -1 0 1 -1 - KONST Breite
sonst
BlockBau KONST Breite 0 0 - KONST Breite KONST Breite - 1

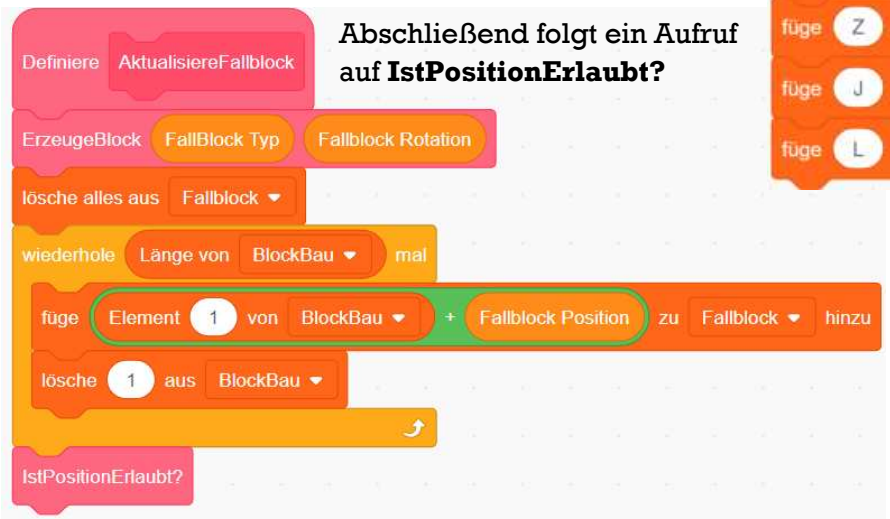

Definiere ErzeugeBlockT ROT
falls ROT = 0 , dann
BlockBau -1 0 1 KONST Breite
sonst
falls ROT = 1 , dann
BlockBau 1 0 0 - KONST Breite KONST Breite
sonst
falls ROT = 2 , dann
BlockBau -1 0 1 0 - KONST Breite
sonst
BlockBau -1 0 0 - KONST Breite KONST Breite


Definiere ErzeugeBlock O
BlockBau 0 1 KONST Breite 1 + KONST Breite

Die Tetrominos verwenden

Damit wir Tetrominos verwenden können müssen wir wissen, welche zur Verfügung stehen. Eine neue Liste „*Mögliche Steine*“ soll diese bereithalten, und wird im **SETUP MöglicheSteine** gefüllt.

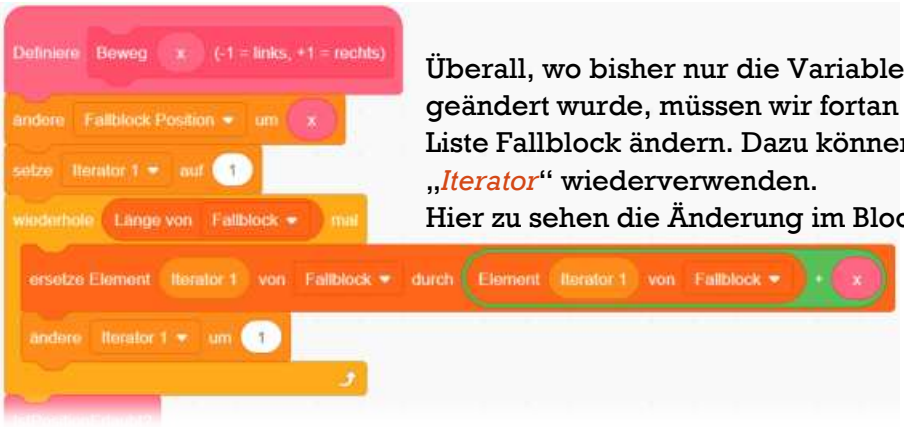
Ein neuer Block mit Namen **Aktualisiere Fallblock** ruft den Block **ErzeugeBlock** mit den Variablen *Fallblock Typ* und *Fallblock Rotation* auf.
Anschließend werden die Einträge der Liste *BlockBau* in eine neue Liste *Fallblock* übertragen.



Abschließend folgt ein Aufruf auf **IstPositionErlaubt?**

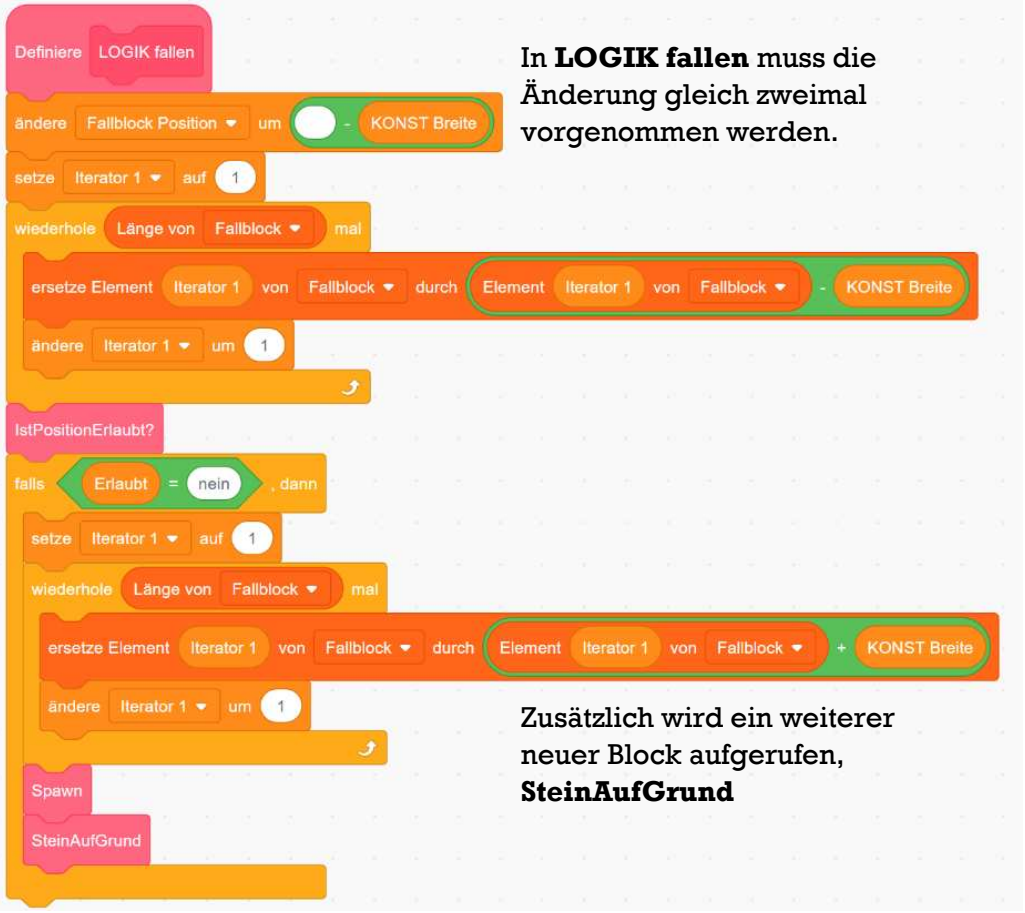
Im Block **Spawn** müssen wir nun ebenfalls ein paar Änderungen vornehmen. Die *Rotation* eines neuen Blockes ist stets auf Null, aber der *Typ* kann zufällig anhand der Liste an Möglichkeiten gewählt werden.

Nach einem Aufruf von **AktualisiereFallblock** prüfen wir das Ergebnis des enthaltenen **IstPositionErlaubt?**. Falls nicht, so konnte kein neuer Block gespawnt werden, und das Spiel ist vorbei. Der entsprechende Block **GameOver** wird zu einem späteren Zeitpunkt beleuchtet.



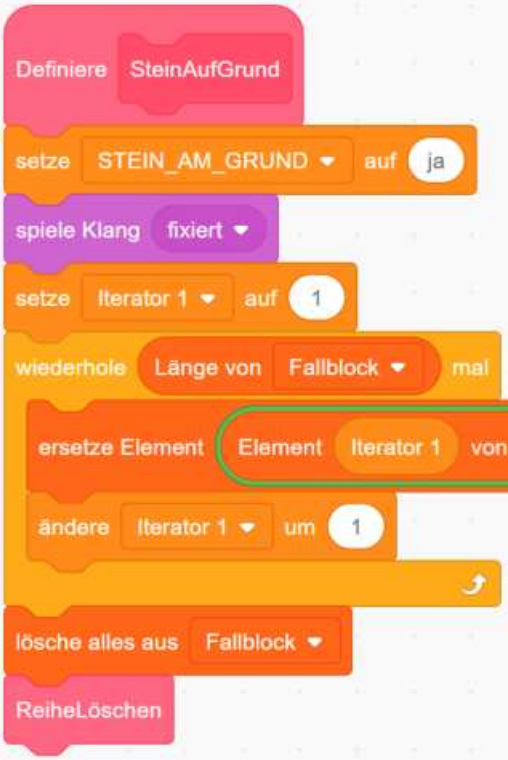
Überall, wo bisher nur die Variable *„Fallblock Position“* geändert wurde, müssen wir fortan auch jedes Element der Liste Fallblock ändern. Dazu können wir unsere Variable „*Iterator*“ wiederverwenden.
Hier zu sehen die Änderung im Block **Beweg ()**

Ganz ähnlich auch in **IstPositionErlaubt?,** wo für jede Prüfung alle Elemente der Liste Fallblock herangezogen werden müssen.



In **LOGIK fallen** muss die Änderung gleich zweimal vorgenommen werden.

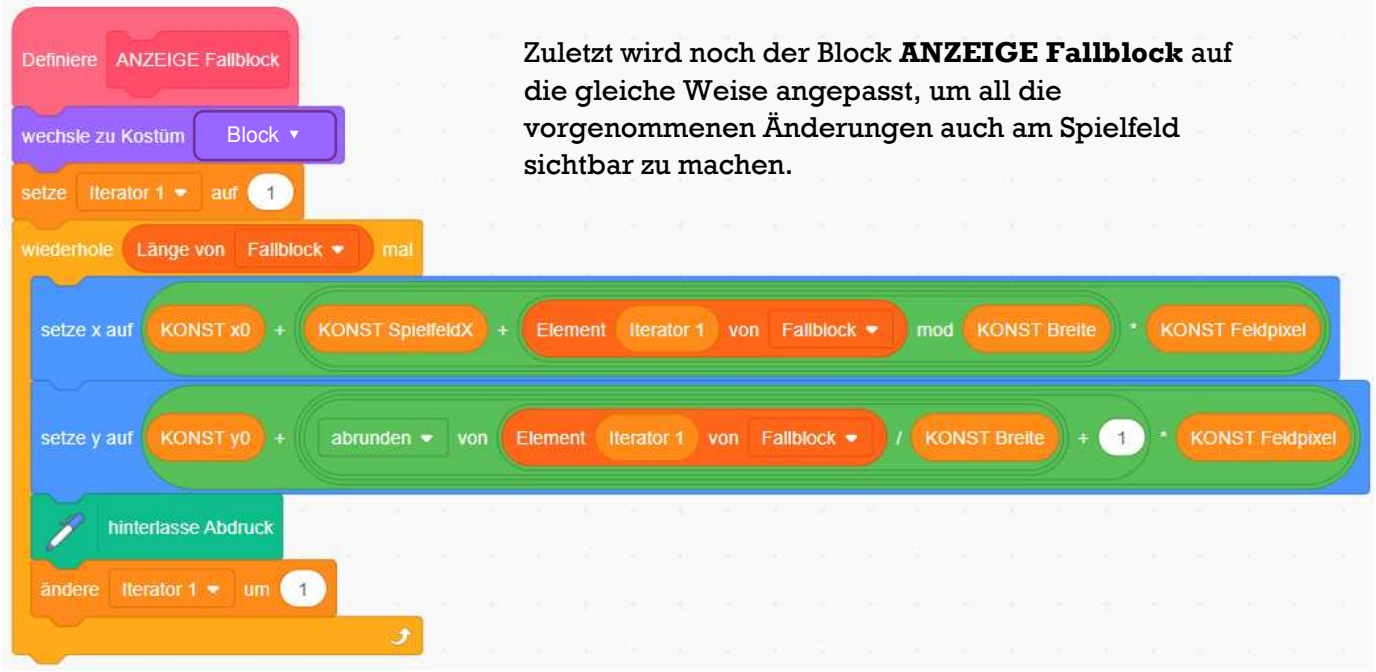
Zusätzlich wird ein weiterer neuer Block aufgerufen, **SteinAufGrund**



SteinAufGrund ist eine Ausgliederung der Logik für die Umwandlung eines Fallblocks in das Spielfeld.

Es kann ein passender Klang abgespielt werden (füge etwas passendes bei den Klängen der Figur hinzu) und anschließend wird über den Iterator jedes Element von *Fallblock* in der Liste Spielfeld durch den Text „Block“ ersetzt.

Der Block „**Reihe Löschen** wird später noch angelegt und kann dann hier eingefügt werden (oder jetzt schon angelegt und hier eingefügt, aber erst später mit Leben befüllt.

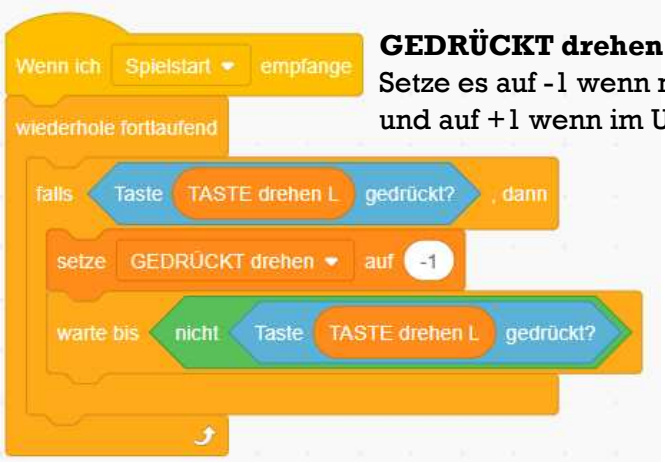


Zuletzt wird noch der Block **ANZEIGE Fallblock** auf die gleiche Weise angepasst, um all die vorgenommenen Änderungen auch am Spielfeld sichtbar zu machen.

Tetrominos drehen

Für die Drehung der Steine beginnen wir mit den entsprechenden Tasteneingaben.

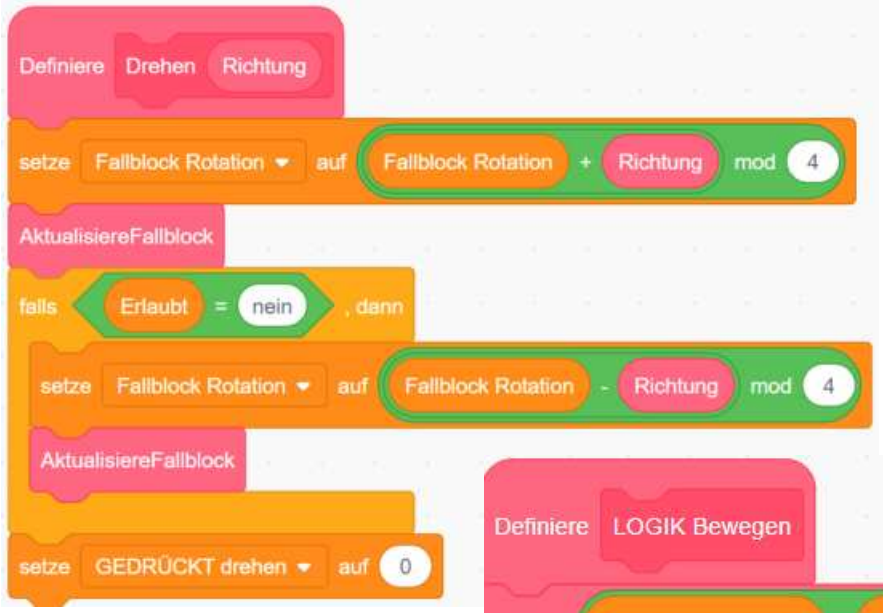
Im Gegensatz zur Bewegung ist es für Drehung nicht nötig, dass die Taste lange gedrückt werden kann. Es reicht daher sicherzustellen, dass jeder Tastendruck genau einmal erfasst wird. Nicht vergessen, die Variablen „*TASTE drehen L*“ und „*TASTE drehen R*“ auch in **SETUP Tasten** zu definieren – beispielsweise mit A und D.



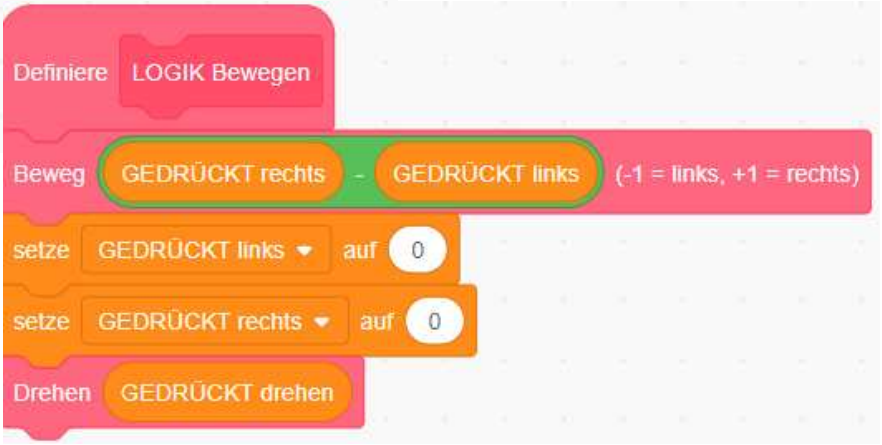
GEDRÜCKT drehen wird in beide Rotationsrichtungen genutzt. Setze es auf -1 wenn nach links gedreht wird (gegen die Uhr) und auf +1 wenn im Uhrzeigersinn nach rechts gedreht wird.

Erstelle einen neuen Block **Drehen()** mit der Eingabe der Richtung. Dieser Block ändert die Rotation um die Richtung und verwendet dabei die bereits bekannte Modulo-Funktion. Durch „**AktualisiereFallblock**“ wird der Block mit dieser Rotation in die Liste *Fallblock* geladen. Wie üblich wird der Vorgang zurückgesetzt, wenn der neue Block nicht erlaubt ist.

Abschließend wird „*GEDRÜCKT drehen*“ auf 0 gesetzt.



Der Aufruf des Blocks **Drehen** erfolgt in **LOGIK Bewegen** mit der zugehörigen Variable.



Reihen entfernen

Ein zentrales Element in Tetris ist das entfernen vollständiger Reihen, daher kommen wir nicht länger um diesen Schritt herum.

Ein neuer Block **ReiheLöschen** soll alle Felder des Spielfeldes Zeilenweise durchlaufen. Wir beginnen daher mit einem *Iterator* auf **1** und begeben uns erst in eine äußere Schleife, welche *KONST Höhe* mal wiederholt wird. Zu Beginn jeder Zeile gehen wir davon aus, dass ein löschen *erlaubt* ist. Wir durchlaufen dann die *KONST Breite* -1 Felder und prüfen, ob eines davon leer ist – mit einem Leerzeichen.

Eine Zeile mit einem leeren Feld dürfen wir nicht löschen, das ist uns nicht erlaubt – also stellen wir *Erlaubt* auf nein.

Falls wir die Zeile löschen dürfen, springen wir zum Zeilenbeginn zurück. Dort wird dann jedes Feld der Zeile (inklusive des 11. Wandfeldes) aus der Liste gelöscht, gleichzeitig aber wieder leere Felder am Listenende hinzugefügt. (Achtung: Nicht leer, sondern Leerzeichen!)

Dadurch rutschen die übrigen Felder der Liste automatisch nach bzw. fallen alle befüllten Felder über dieser Zeile nach unten.

Falls wir aber nicht löschen dürfen, überspringen wir zuletzt noch das Wandfeld, welches wir nicht prüfen wollen, da es ohnehin immer leer ist.

ReiheLöschen muss natürlich noch irgendwo aufgerufen werden. Wie auf einer früheren Seite erwähnt passt er gut in den Block **SteinAufGrund**, da nur auf einen gelandeten Stein hin Reihen entfernt werden können.

Die Basiselemente von Tetris sind nun alle vorhanden. Wir haben Tetrominos, welche von selber fallen und von uns gedreht und gelenkt werden können, und aufgefüllte Zeilen verschwinden von selbst wieder. Dennoch haben wir viel Arbeit für den Feinschliff vor uns.

Level und Geschwindigkeiten

Aktuell rasen die Blöcke mit Höchstgeschwindigkeit nach unten, das macht es schwierig, das Spiel zu spielen. Im Original beginnt das Spiel langsam und wird schneller, wenn man höhere Level erreicht, indem man mehr Reihen entfernt. Also wollen wir das umsetzen.

Wir beginnen, indem wir einige Variablen anlegen: „*ZEITPUNKT Fall*“, „*Takt Fall*“, „*Score*“, „*Level*“ und „*Zeilen gefüllt*“.

Alle müssen im **SETUP** gesetzt werden. Wie du bereits ahnst wird *ZEITPUNKT Fall* in **SETUP Zeiten** auf Null gesetzt, *TAKT Fall* wird ebenfalls in **SETUP Zeiten** gesetzt, allerdings auf **2**. Für *Score*, *Level* und *Zeilen* gefüllt wird ein neuer Block „**SETUP Nullung**“ eingeführt, in dem diese alle auf **0** gesetzt werden.

Du erinnerst dich an die *Anschlagverzögerung*? Wir verwenden nun ein ähnliches Prinzip: Der Aufruf **LOGIK fallen** im Block **LOGIK** wird in einen *Falls-Dann-Block* gepackt, in dem abgefragt wird, ob die Stoppuhr bereits einen höheren Wert hat, als *ZEITPUNKT Fall* und *TAKT Fall* gemeinsam. Falls dem so ist wird *ZEITPUNKT Fall* um *TAKT Fall* erhöht.

Das bedeutet: Zu Beginn des Spiels vergehen 2 Sekunden (*TAKT Fall*), bevor der Block **LOGIK fallen** das erste Mal aufgerufen wird, dann wieder und wieder. Sollte sich TAKT Fall jedoch ändern und kleiner werden, werden sich auch die Abstände zwischen den Aufrufen von **LOGIK fallen** verringern.

Genau dafür ist der Block „**Setze Falltakt**“ gedacht. Dieser setzt den Falltakt auf den Grundwert von 2 zurück, verringert ihn dann aber für jedes Level um 25%. Auf Level 8 haben wir den Takt so bereits auf 0,2 Sekunden geschrumpft, und es wird schon recht hastig.

Das Level zu setzen passt gut in den Block **ReiheLöschen**.

Wir beginnen, indem wir eine Variable *Gedächtnis* anlegen und auf 0 setzen. Ähnlich wie die Variablen *Iterator* und *Erlaubt* werden wir diese noch an anderen Stellen wiederfinden. Grundsätzlich ist es eine Variable, welche einen Wert kurz zwischenspeichert, für den es sich nicht lohnt, eine eigene Variable anzulegen.

Haben wir eine Reihe gefunden, welche gelöscht werden darf, so merken wir uns das, indem wir das *Gedächtnis* um eins erhöhen. Dann erhöhen wir unsere Score um *Level * 100 * Gedächtnis*. Das bedeutet: Je mehr Reihen auf einmal gelöscht werden, um so mehr Punkte gibt es dafür. 100 Punkte für eine einzelne Reihe, 300 Punkte für 2 Reihen, 600 Punkte für 3 Reihen und ganze 1000 Punkte für 4 Reihen. Es lohnt sich also, im Spiel möglichst häufig gleich 4 Reihen mit dem I-Block zu löschen.

Wir erhöhen zudem die Variable „*Zeilen gefüllt*“, welche wir zum errechnen des aktuellen Levels benötigen.

Am Ende des Blocks „**Reihe Löschen**“ prüfen wir nochmals, ob eine Reihe gelöscht wurde, also *Gedächtnis* größer Null ist. Falls dem so ist, können wir einen passenden Klang abspielen (dieser muss bei den Klängen der Figur erst hinzugefügt werden) und den *Level* erhöhen. Dieser soll nie niedriger sein als ein zehntel der *gefüllten Zeilen*. Aktuell kann er zwar auch nicht höher sein, das könnte sich jedoch in Zukunft noch ändern, wenn jemand gleich auf einem höheren Level starten will, um mehr Punkte zu verdienen.

Zum Abschluss wird der zuvor angelegte Block „**Setze Falltakt**“ aufgerufen.

Das Sack-System

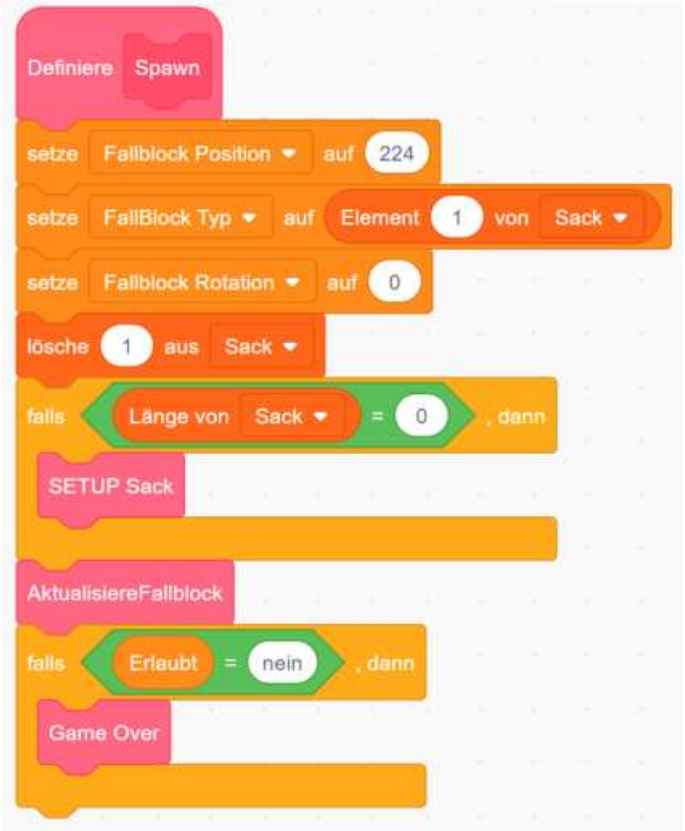
Bisher wählen wir zufällige Tetrominos aus. Zufällig bedeutet: Alles ist möglich, auch, dass mehrmals hintereinander der gleiche Tetromino erscheint, oder man auf einer langen Durststrecke vergeblich auf den I-Tetromino als Lückenfüller wartet. Das ist uns vielleicht etwas zu zufällig.

In Tetris gibt es das Sack-System. Es funktioniert wie folgt: Alle möglichen Steine werden in den Sack geworfen und dann zufällig daraus gezogen, bis der Sack leer ist. Danach wird der Sack wieder aufgefüllt. Somit muss man nie länger als 12 Steine auf den gewünschten Tetromino warten, und hat auch nie mehr als zweimal den gleichen hintereinander.

Für unseren Sack legen wir einen neuen Block **SETUP Sack** an, sowie eine neue Liste *Sack*. Wir rufen das **SETUP Mögliche Steine** auf und nehmen einen zufälligen Stein daraus, den wir aus den Möglichkeiten entfernen, aber unserem Sack hinzufügen – so oft, wie wir Steine haben. Damit haben wir einen Sack Steine in zufälliger Reihenfolge, als hätten wir ihn gut durchgeschüttelt.

Nun ändern wir unseren Block **Spawn**, indem wir den zufälligen Stein durch den ersten aus dem Sack ersetzen und in Folge diesen Stein aus dem Sack löschen. Ist der Sack daraufhin leer, wird er neu angefüllt.

Neben des gleichmäßig wirkenderen Zufalls erreichen wir so auch, dass wir bereits wissen, welcher Stein als nächstes kommen wird. Dies wird später noch wichtig, wenn wir derlei Informationen anzeigen wollen.

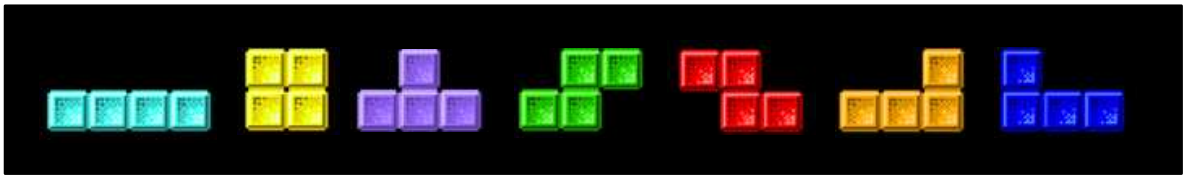


Viele viele bunte Tetrominos

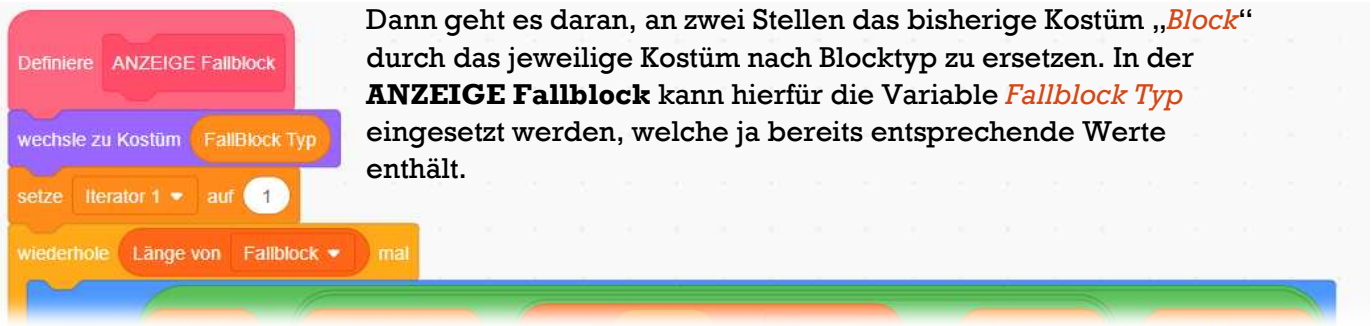
Jeder Tetromino hat eine bestimmte Farbe, und diese ist sogar über die meisten Spiele hinweg gleich. Auch wir wollen unsere Tetrominos farbig gestalten.

Gehe dazu in deine Kostüme und lege für jeden Tetromino-Typ ein neues Kostüm an, welches den Namen des Tetrominos trägt. Also **I**, **O**, **T**, **S**, **Z**, **L** und **J**.

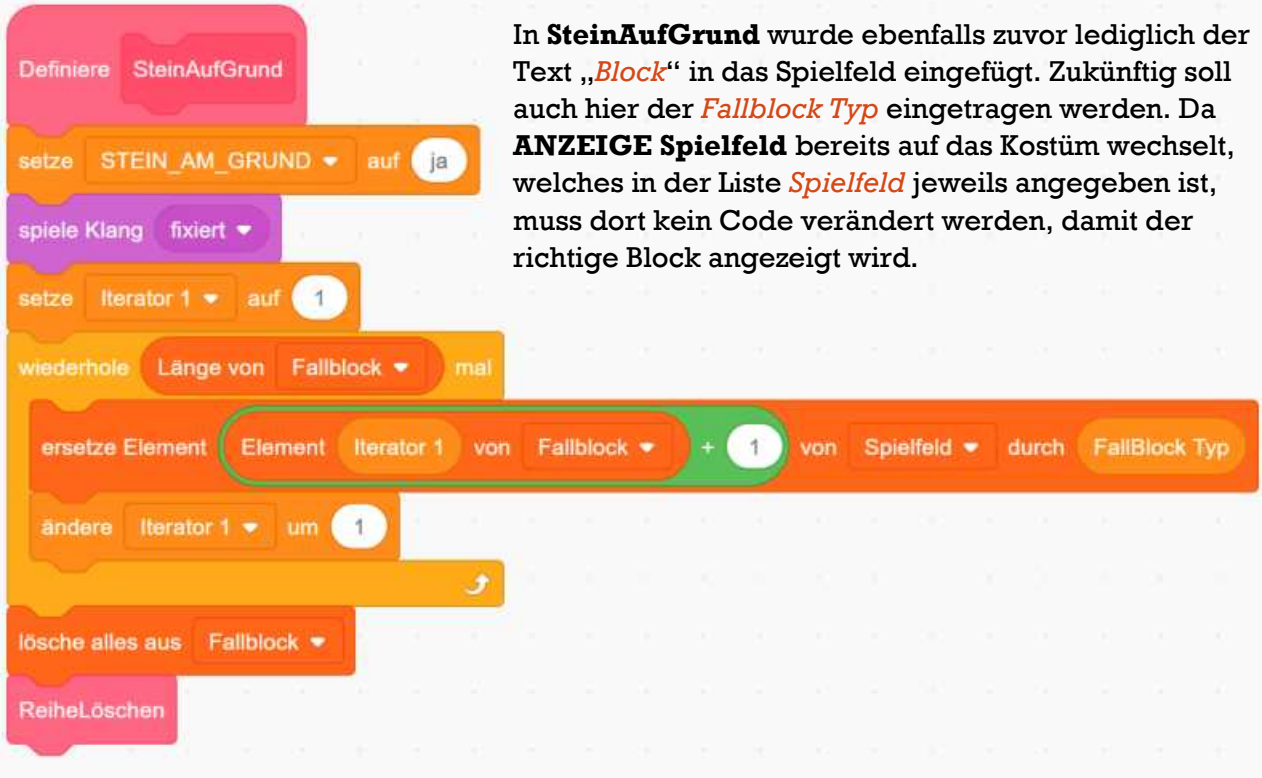
Jedes dieser Kostüme soll natürlich ein 16x16 Pixel großer Block sein, in der passenden Farbe des jeweiligen Tetrominos.



Wieder kannst du dich bei der eigentlichen Gestaltung der Blöcke austoben, wie du möchtest. Oder du lässt sie alle gleich aussehen und änderst wirklich nur die Farbe, das liegt ganz bei dir.



Dann geht es daran, an zwei Stellen das bisherige Kostüm „*Block*“ durch das jeweilige Kostüm nach Blocktyp zu ersetzen. In der **ANZEIGE Fallblock** kann hierfür die Variable *Fallblock Typ* eingesetzt werden, welche ja bereits entsprechende Werte enthält.



In **SteinAufGrund** wurde ebenfalls zuvor lediglich der Text „*Block*“ in das Spielfeld eingefügt. Zukünftig soll auch hier der *Fallblock Typ* eingetragen werden. Da **ANZEIGE Spielfeld** bereits auf das Kostüm wechselt, welches in der Liste *Spielfeld* jeweils angegeben ist, muss dort kein Code verändert werden, damit der richtige Block angezeigt wird.

Buchstaben-Kostüme zur Textanzeige

Im nächsten Schritt möchten wir Text anzeigen, ohne die „sage“-Funktion von Scratch zu verwenden. Der Code dazu ist recht einfach in einem neuen Block „Zeige Text“ anzulegen.

Wie gehabt wird eine Ausgangsposition bestimmt. Von dort wird über einen *Iterator* für jedes Zeichen des Textes ein *Abdruck* hinterlassen.

Tatsächlich ist es so simpel, der eigentliche Aufwand besteht darin, dass für jedes Zeichen auch ein Kostüm angelegt werden muss, welches gezeichnet werden will. Jeweils ein Buchstabe, der möglichst 14*14 Pixel ausfüllen soll.
(Also 16*16, ohne die Ränder zu verwenden)

Es ist nicht nötig, alle Buchstaben sofort anzulegen, sondern immer nur die, die auch benötigt werden.

Für Zahlen bietet sich eine rechtsbündige Darstellung an. Dieser Code ergänzt einen Text um so viele vorangestellte Leerzeichen, dass er vom Zeilenanfang gerade bis zur gewünschten Position reicht.

Zuletzt wollen wir auch Tetrominos anzeigen können, in **Zeige Block.**

Für diese wird ein Block des gewünschten Typs erzeugt, und anschließend für jedes Element aus *BlockBau* die richtige Position gesucht – zunächst werden x und y auf die gewünschte Position gesetzt und diese anschließend den Werten in *Blockbau* entsprechend verschoben

Damit haben wir alles, was wir zur Anzeige einiger Infos am rechten Rand benötigen, und können einen neuen Block **ANZEIGE Infos** anlegen:

Der Game-Over Screen

Wenn das Spiel vorbei ist wird keine Logik mehr benötigt, wir stoppen daher zunächst alle Skripte. Indem wir einmal mit einem extrem dicken, halbtransparenten schwarzen Stift gegen den Screen tippen können wir erreichen, dass alles, was sich darauf befindet, ein wenig dunkler wird. Dadurch wird schneller klar, dass das Spiel vorbei ist.

Mit Hilfe des **Zeige Text** – Blocks können wir die Nachricht „*Spiel Vorbei*“ auch mitten am Bildschirm anzeigen.

Nach einer Sekunde folgt auch der Hinweis, dass man die Leertaste betätigen soll, um durch die Nachricht „Spielstart“ wieder von vorne zu beginnen.

TETRIS! - Nachricht

Wenn vier Zeilen auf einmal entfernt werden, nennt man das einen Tetris. Für diese Leistung soll es noch eine besondere Anzeige geben.

Erzeuge neue Variablen *ZEITPUNKT Nachricht*, *Nachricht* und *TAKT Nachricht*. Du ahnst es bereits: *ZEITPUNKT* und *TAKT* werden in **SETUP Zeiten** gesetzt, die *Nachricht* selbst in **SETUP Nullung** geleert. TAKT bestimmt, wie lange die Nachricht angezeigt werden soll, **2** Sekunden erscheinen angemessen.

In einem neuen Block „**Setze neue Nachricht**“ wird der *ZEITPUNKT* mit der *Stoppuhr* gleichgesetzt und die eigentliche *Nachricht* gesetzt.

ANZEIGE Nachricht sorgt dafür, dass die *Nachricht* angezeigt wird, sofern sie nicht älter als *TAKT Nachricht* ist oder die *Stoppuhr* in der zweiten Hälfte einer Sekunde ist – Letzteres sorgt für ein langsames blinken.

Eingesetzt wird diese Nachricht im Block **Reihe Löschen** gegen Ende. Wurden vier Reihen entfernt, so wird sowohl die Nachricht angezeigt als auch ein passender Klang (wie gehabt selbst auszuwählen) abgespielt.

Der Dreh mit dem Wallkick

Aktuell ist es so: Wenn ein Fallblock zu nahe an einer Wand ist, lässt er sich nicht drehen. Insbesondere beim I-Tetromino ist das ein auffälliges Problem.

Ein ordentliches Tetris, das etwas auf sich hält, benötigt deshalb Wallkicks. Das bedeutet: Wenn sich ein Stein nicht regulär drehen lässt, werden erst noch andere Positionen untersucht und der Stein gegebenenfalls verschoben, bevor er zurückgedreht wird. Welche alternativen Positionen ausprobiert werden hängt vom Typ, der Ausrichtung und der Rotationsrichtung ab.

Die meisten Tetrominos:

Rotation	Richtung	Test 1	Test 2	Test 3	Test 4
0	R	-1|0	-1|-1	0|2	-1|2
1	R	-1|0	-1|1	0|-2	-1|2
2	R	1|0	1|-1	0|2	1|2
3	R	1|0	1|1	0|-2	1|-2
0	L	1|0	1|-1	0|2	1|2
1	L	-1|0	-1|1	0|-2	-1|-2
2	L	-1|0	-1|-1	0|2	-1|2
3	L	1|0	1|1	0|-2	1|-2

Der I-Tetromino:

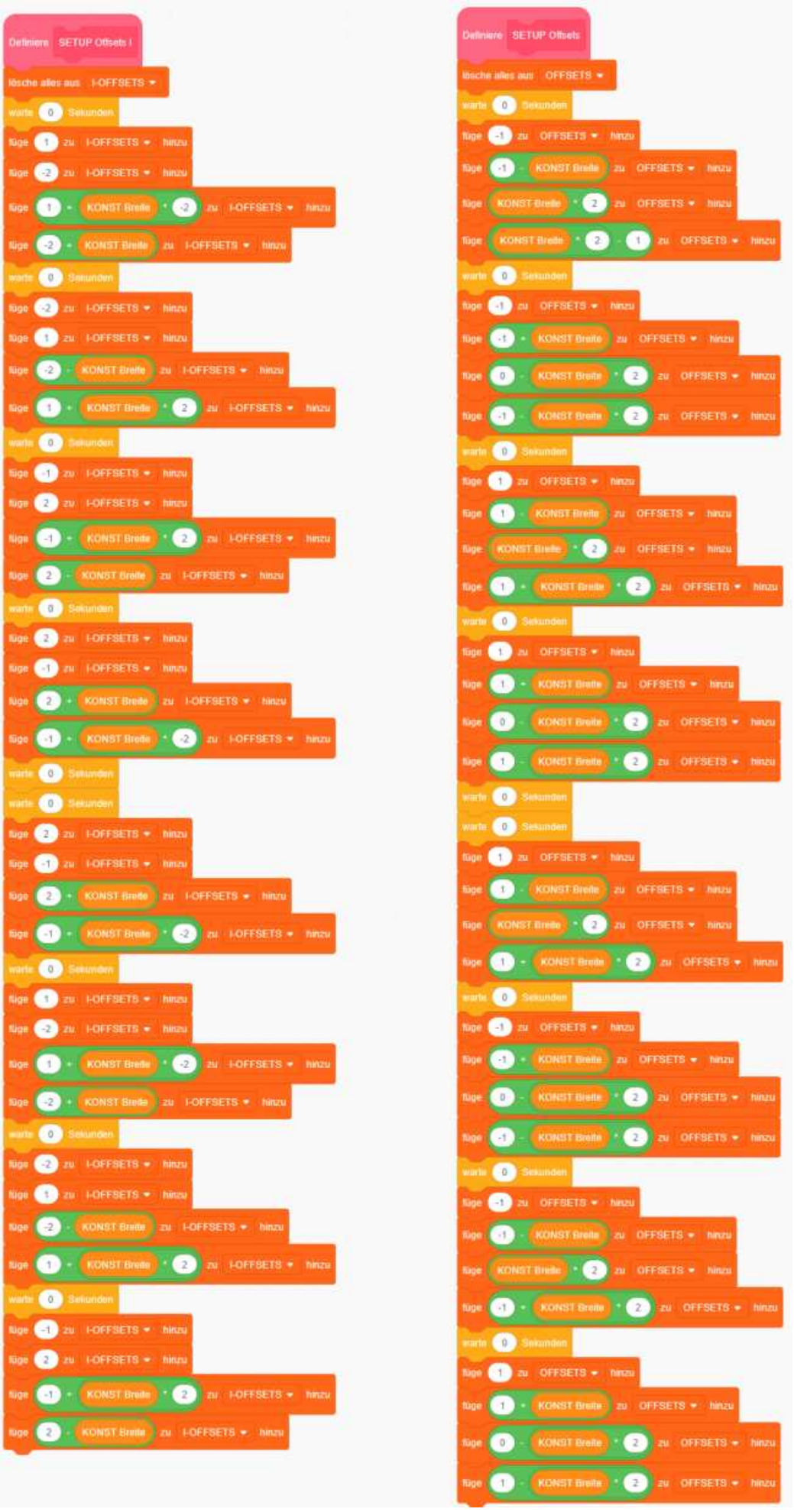
Rotation	Richtung	Test 1	Test 2	Test 3	Test 4
0	R	1|0	-2|0	1|2	-2|1
1	R	-2|0	1|0	-2|-1	1|2
2	R	-1|0	2|0	-1|2	2|-1
3	R	2|0	-1|0	2|1	-1|-2
0	L	2|0	-1|0	2|1	-1|-2
1	L	1|0	-2|0	1|-2	-2|1
2	L	-2|0	1|0	-2|-1	1|2
3	L	-1|0	2|0	-1|2	2|-1

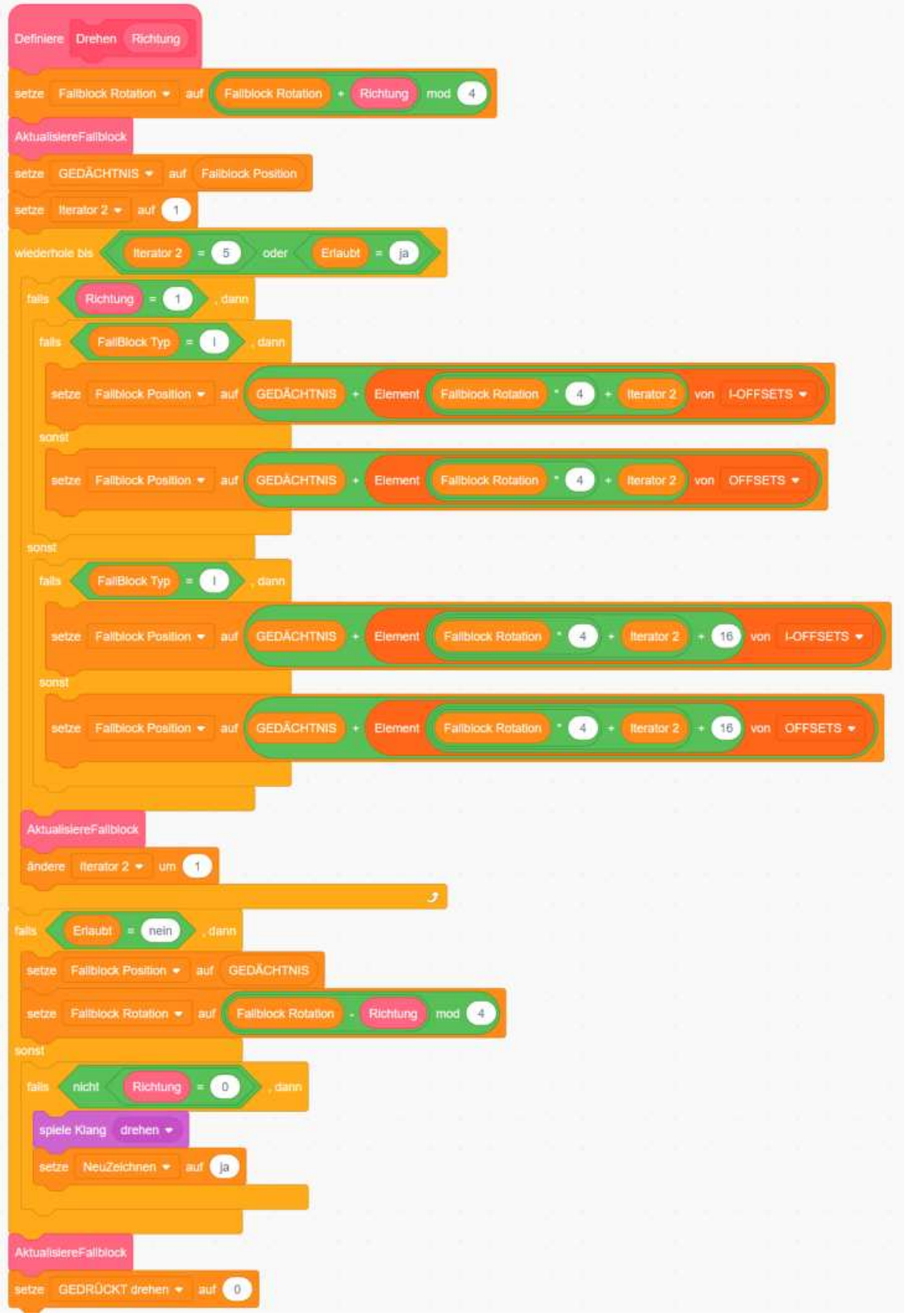
Wir wollen all diese Informationen in zwei Listen packen – „*Offsets*“ und „*I-Offsets*“, allerdings sind diese Angaben zweidimensional und wir benötigen nur einen Wert. Entsprechend wird bei den Einträgen der zweite Wert jeweils mit der *KONST Breite* multipliziert.

Auf der Folgenden Seite sind die beiden Blöcke *SETUP Offsets I* und *SETUP Offset* einsehbar, es ist jedoch empfehlenswerter sich der Daten aus der oberen Tabelle zu bedienen. Die diversen „*warte 0 Sekunden*“ dienen rein der Strukturierung und haben keinen praktischen Mehrwert.

Die Seite danach zeigt die Anpassungen an **Drehen()**, besonders zu beachten ist hierbei dass nicht auf die gleiche Iterator-Variable zurückgegriffen werden kann, da diese in **IstPositionErlaubt?** In **AktualisiereFallblock** genutzt wird. Lege daher eine neue Variable an und nenne sie schlicht Iterator 2.

Im Wesentlichen werden alle Möglichkeiten durchprobiert, bis eine passt oder keine mehr übrig ist. In letzterem Fall muss wieder auf die ursprüngliche Position und die ursprüngliche Rotation zurückgesetzt werden.

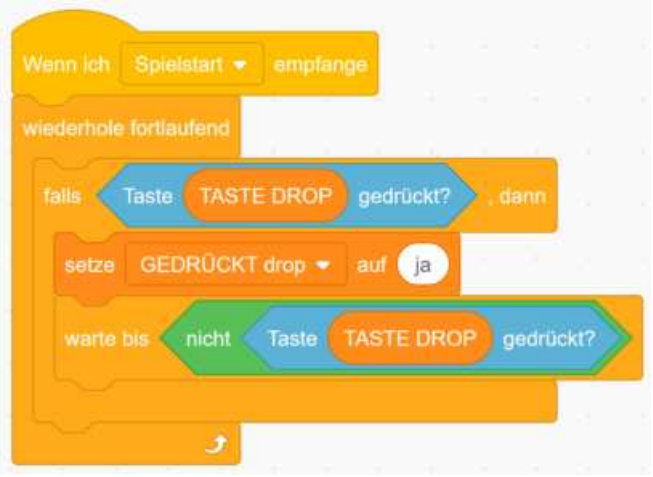

Definiere SETUP Offsets I
lösche alles aus I-OFFSETS
warte 0 Sekunden
füge 1 zu I-OFFSETS hinzu
füge -2 zu I-OFFSETS hinzu
füge 1 - KONST Breite * -2 zu I-OFFSETS hinzu
füge -2 + KONST Breite zu I-OFFSETS hinzu
warte 0 Sekunden
füge -2 zu I-OFFSETS hinzu
füge 1 zu I-OFFSETS hinzu
füge -2 - KONST Breite zu I-OFFSETS hinzu
füge 1 + KONST Breite * 2 zu I-OFFSETS hinzu
warte 0 Sekunden
füge -1 zu I-OFFSETS hinzu
füge 2 zu I-OFFSETS hinzu
füge -1 + KONST Breite * 2 zu I-OFFSETS hinzu
füge 2 - KONST Breite zu I-OFFSETS hinzu
warte 0 Sekunden
füge 2 zu I-OFFSETS hinzu
füge -1 zu I-OFFSETS hinzu
füge 2 + KONST Breite zu I-OFFSETS hinzu
füge -1 + KONST Breite * -2 zu I-OFFSETS hinzu
warte 0 Sekunden
warte 0 Sekunden
füge 2 zu I-OFFSETS hinzu
füge -1 zu I-OFFSETS hinzu
füge 2 + KONST Breite zu I-OFFSETS hinzu
füge -1 + KONST Breite * -2 zu I-OFFSETS hinzu
warte 0 Sekunden
füge 1 zu I-OFFSETS hinzu
füge -2 zu I-OFFSETS hinzu
füge 1 + KONST Breite * -2 zu I-OFFSETS hinzu
füge -2 + KONST Breite zu I-OFFSETS hinzu
warte 0 Sekunden
füge -2 zu I-OFFSETS hinzu
füge 1 zu I-OFFSETS hinzu
füge -2 - KONST Breite zu I-OFFSETS hinzu
füge 1 + KONST Breite * 2 zu I-OFFSETS hinzu
warte 0 Sekunden
füge -1 zu I-OFFSETS hinzu
füge 2 zu I-OFFSETS hinzu
füge -1 + KONST Breite * 2 zu I-OFFSETS hinzu
füge 2 - KONST Breite zu I-OFFSETS hinzu
Definiere SETUP Offsets
lösche alles aus OFFSETS
warte 0 Sekunden
füge -1 zu OFFSETS hinzu
füge -1 - KONST Breite zu OFFSETS hinzu
füge KONST Breite * 2 zu OFFSETS hinzu
füge KONST Breite * 2 - 1 zu OFFSETS hinzu
warte 0 Sekunden
füge -1 zu OFFSETS hinzu
füge -1 + KONST Breite zu OFFSETS hinzu
füge 0 - KONST Breite * 2 zu OFFSETS hinzu
füge -1 - KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
füge 1 zu OFFSETS hinzu
füge 1 - KONST Breite zu OFFSETS hinzu
füge KONST Breite * 2 zu OFFSETS hinzu
füge 1 + KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
füge 1 zu OFFSETS hinzu
füge 1 + KONST Breite zu OFFSETS hinzu
füge 0 - KONST Breite * 2 zu OFFSETS hinzu
füge 1 - KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
warte 0 Sekunden
füge 1 zu OFFSETS hinzu
füge 1 - KONST Breite zu OFFSETS hinzu
füge KONST Breite * 2 zu OFFSETS hinzu
füge 1 + KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
füge -1 zu OFFSETS hinzu
füge -1 + KONST Breite zu OFFSETS hinzu
füge 0 - KONST Breite * 2 zu OFFSETS hinzu
füge -1 - KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
füge -1 zu OFFSETS hinzu
füge -1 - KONST Breite zu OFFSETS hinzu
füge KONST Breite * 2 zu OFFSETS hinzu
füge -1 + KONST Breite * 2 zu OFFSETS hinzu
warte 0 Sekunden
füge 1 zu OFFSETS hinzu
füge 1 + KONST Breite zu OFFSETS hinzu
füge 0 - KONST Breite * 2 zu OFFSETS hinzu
füge 1 - KONST Breite * 2 zu OFFSETS hinzu


Definiere Drehen Richtung
setze Fallblock Rotation auf Fallblock Rotation + Richtung mod 4
AktualisiereFallblock
setze GEDÄCHTNIS auf Fallblock Position
setze Iterator 2 auf 1
wiederhole bis Iterator 2 = 5 oder Erlaubt = ja
falls Richtung = 1 , dann
falls FallBlock Typ = I , dann
setze Fallblock Position auf GEDÄCHTNIS + Element Fallblock Rotation * 4 + Iterator 2 von I-OFFSETS
sonst
setze Fallblock Position auf GEDÄCHTNIS + Element Fallblock Rotation * 4 + Iterator 2 von OFFSETS
sonst
falls FallBlock Typ = I , dann
setze Fallblock Position auf GEDÄCHTNIS + Element Fallblock Rotation * 4 + Iterator 2 + 16 von I-OFFSETS
sonst
setze Fallblock Position auf GEDÄCHTNIS + Element Fallblock Rotation * 4 + Iterator 2 + 16 von OFFSETS
AktualisiereFallblock
ändere Iterator 2 um 1
falls Erlaubt = nein , dann
setze Fallblock Position auf GEDÄCHTNIS
setze Fallblock Rotation auf Fallblock Rotation - Richtung mod 4
sonst
falls nicht Richtung = 0 , dann
spiele Klang drehen
setze NeuZeichnen auf ja
AktualisiereFallblock
setze GEDRÜCKT drehen auf 0

Fallblöcke schneller fallen lassen

Gerade in den frühen Leveln kann es passieren, dass man einschläft, bevor ein Block den Boden erreicht. Abhilfe schaffen zwei zusätzliche Tasten: *Schnell* und *Drop*

Die *Taste Schnell* (welche natürlich auch in SETUP Tasten definiert werden muss) ist passenderweise schnell erledigt: Solange sie gedrückt ist, wird der *TAKT Fall* auf einen sehr niedrigen Wert reduziert, womit die Fallblöcke entsprechend schnell nach unten sausen.

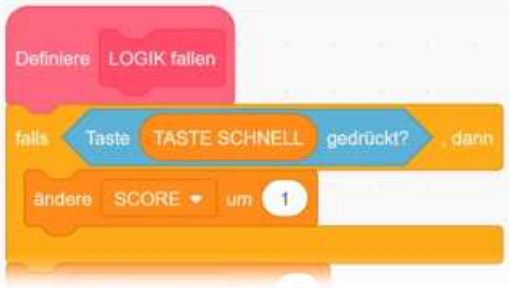


Im Gegensatz dazu soll die *Taste Drop* einen Stein sofort zum Boden befördern. Die Tastenlogik setzt daher nur eine Variable „*Gedrückt drop*“ auf ja, der eigentliche Drop wird im **LOGIK** Block abgehandelt.

Hier wird zunächst abgeprüft, ob die drop gedrückt wurde. Wenn dem so ist, wird immer wieder die **LOGIK fallen** aufgerufen, bis der *Stein am Grund* ist. Diese Variable wird wiederum im ähnlich benannten Block **Stein auf Grund** auf ja gesetzt.

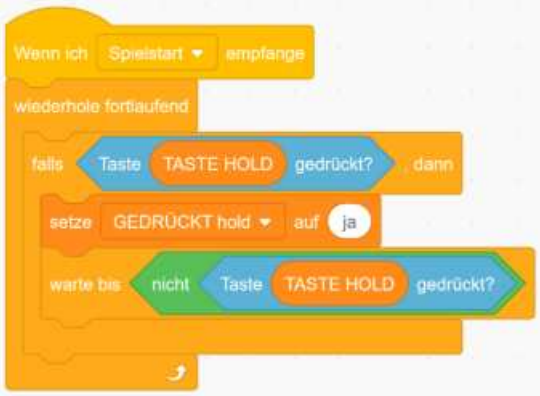
Nebenbei bekommt man ein paar Punkte für den Drop.

Hinterher müssen zwei Variablen resettet werden.



Übrigens: Auch mit der *Taste Schnell* kann man Punkte verdienen, wenn eine entsprechende Abfrage in **LOGIK fallen** eingebaut wird.

Blöcke Zwischenspeichern



Manchmal bekommt man gerade den falschen Tetromino spendiert, der nirgends hinpassen will. Eine Abhilfe schafft ein Block-Zwischenspeicher, auch **Hold** genannt.

Lege zunächst wie gehabt eine entsprechende Tastenlogik an. Zusätzlich wird eine Variable HoldVerwendet benötigt, denn der Zwischenspeicher darf nur einmal pro Block genutzt werden – logisch, sonst könnte man die gleichen zwei Blöcke ewig hin- und hertauschen.

Im Block **LOGIK** wird geprüft, ob *hold* gedrückt wurde und noch nicht verwendet wurde. In diesem Fall wird der gehaltene Block ins *Gedächtnis* verschoben und der aktuelle Block gehalten. Falls das *Gedächtnis* nicht leer ist (also bereits ein Block gehalten wurde) wird dieser an erster Stelle im Sack eingefügt – und wird als nächster Stein ge**spawn**t.

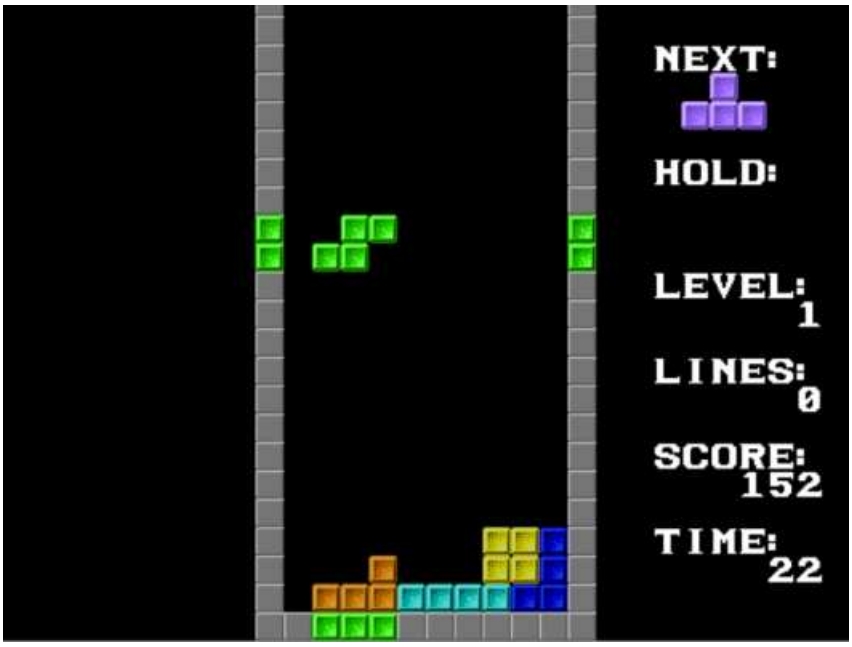
Damit auch wirklich ge**spawn**t wird, muss der entsprechende Aufruf umziehen. Zu Beginn des Logikblockes kann er immer dann aufgerufen werden, wenn nichts mehr in der Liste *Fallblock* steht – ein klarer Indikator dafür, dass ein neuer Block spawnen muss.

HoldVerwendet wird dann in **SteinAufGrund** zurückgesetzt, damit der folgende Stein wieder ausgetauscht werden kann.

Schließlich kann im Block **ANZEIGE Info** nun noch eine Lücke gefüllt werden, indem auch der gehaltene Block angezeigt wird. Genutzt wird dazu natürlich der Block **Zeige Block**

Rand-Indikatoren für den Fallblock

Es kann manchmal schwierig sein zu sehen, wo ein Tetramino landen wird, vor allem im Vollbildmodus. Eine Lösung dafür sind Rand-Indikatoren. Im Rand werden immer jene Felder farbig dargestellt, welche in einer Linie mit dem Fallblock liegen.



Zur Umsetzung wird ein neuer Block **ANZEIGE Fallblock Rand** benötigt. In diesem wird schlicht auf das Kostüm des aktuellen Fallblockes gewechselt und für jedes Element des Fallblockes drei Abdrücke hinterlassen – einmal bei korrekter x-Position und einer y-Position gleich dem unteren Rand, und zweimal bei korrekter y-Position, wobei x auf die beiden Ränder gesetzt wird.