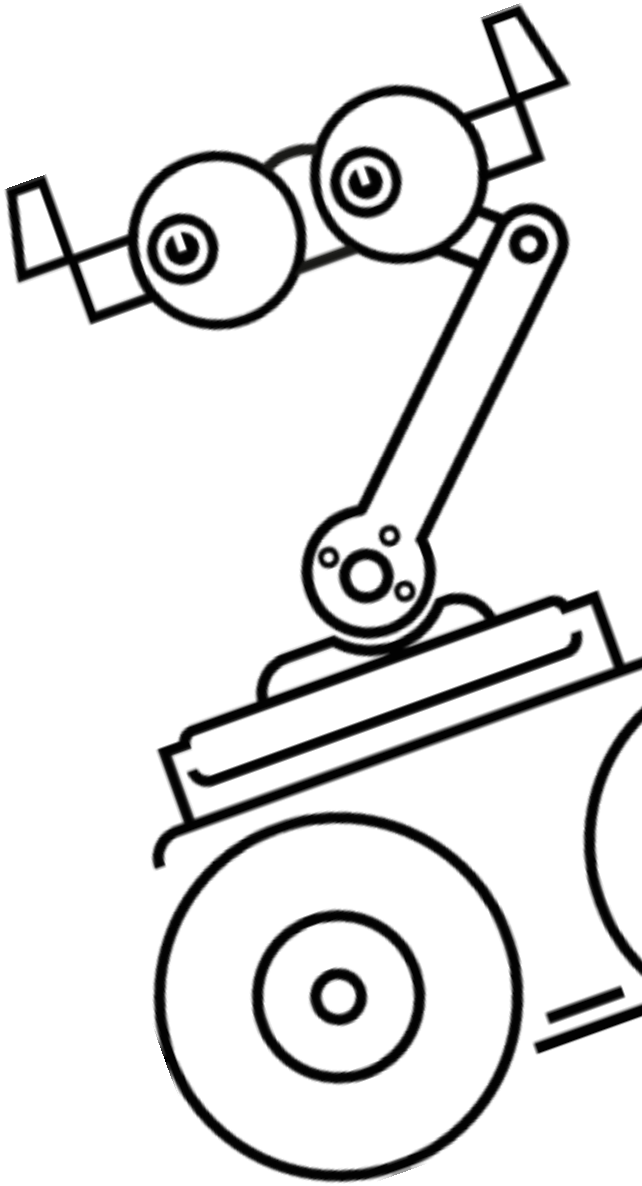
Experiment: Colorsorter

Bearbeitet von Sebastian Dittkrist

Stand: August 2018



ROBERTA INITIATIVE

Überblick

In diesem Experiment geht es um die Programmierung des EV3 Colorsorters. Der Colorsorter (Farbsortierer) soll die eingelegten und verschieden farbigen Teile auf Knopfdruck eigenständig erkennen, zum dafür vorgesehenen Behälter fahren und dort auswerfen.

Die Bauanleitung zum entsprechenden Modell findet man auf https://www.roberta-home.de/ unter den Roberta-Materialien.

Das fertige Programm findet man in der Galerie des Open Roberta Labs (https://lab.open-roberta.org/#gallery) unter dem Namen „NepoColorsorter“.

Es wird von folgender Roboterkonfiguration ausgegangen:

Berührungssensor	Sensorport 1
Farbsensor	Sensorport 3
Großer Motor	Motorport A
Mittlerer Motor	Motorport B

(Bei beiden Motoren ist die Regulierung eingeschaltet und die Drehrichtung >vorwärts< eingestellt)

Vorgehensweise

Bei der Programmierung einer komplexen Anlage ist es meist sinnvoll nicht gleich drauf los zu programmieren, sondern sich zunächst zu überlegen, in welche Teilprobleme man die Aufgabe aufgliedern könnte.

Aufgliederung in Teilprobleme

In unserem Fall könnte die Gliederung für eine Sortierung folgendermaßen aussehen:

- Identifikation
- Positionierung
- Auswerfen

Identifikation

Die Identifikation funktioniert über den Farbsensor an Port 3, der fünf verschiedene Zustände erkennen muss.

Am Fuß der Rutsche befindet sich:

1. Ein rotes Teil
2. Ein blaues Teil
3. Ein grünes Teil
4. Ein gelbes Teil
5. Kein Teil

Positionierung

Die Positionierung läuft über den großen Motor an Port A. Er steuert das Laufband und somit die Position der darauf montierten Teilerutsche.

Für eine sichere Positionierung ist im vorliegenden Modell ein sogenannter Endlagenschalter eingebaut. Dieser wird durch den Berührungssensor an Port 1 repräsentiert.
Wann immer dieser Sensor betätigt wird, wissen wir genau an welcher Position sich das Laufband befindet: Der Startposition.

Auswerfen

Das Auswerfen wird mit dem mittleren Motor an Port B realisiert. Durch seine Ansteuerung wird der Kolben am Fuß der Teilerutsche ausgefahren und das unterste Teil ausgeworfen.

Variablen

Bevor wir mit dem Programmieren beginnen, erstellen wir uns ein paar Variablen, die uns die Einstellung unseres Programms erleichtern.

Variablenname	Typ	Initialisierungswert	Beschreibung
speedConveyor	Zahl	20	Geschwindigkeit, mit der sich das Laufband hin und her bewegt
speedEjection	Zahl	50	Geschwindigkeit, mit der der Kolben am Fuß der Teilerutsche aus- und wieder eingefahren wird
rotationsRed	Zahl	0	Anzahl der Rotationen, die der große Motor durchführen muss, um die Rutsche von der Startposition über den roten Auffangbehälter zu bewegen
rotationsBlue	Zahl	0,5	Anzahl der Rotationen, die der große Motor durchführen muss, um die Rutsche von der Startposition über den blauen Auffangbehälter zu bewegen
rotationsGreen	Zahl	1	Anzahl der Rotationen, die der große Motor durchführen muss, um die Rutsche von der Startposition über den grünen Auffangbehälter zu bewegen
rotationsYellow	Zahl	1,5	Anzahl der Rotationen, die der große Motor durchführen muss, um die Rutsche von der Startposition über den gelben Auffangbehälter zu bewegen
rotationsEjection	Zahl	0,5	Anzahl der Rotationen, die der mittlere Motor durchführen muss, um den Kolben komplett auszufahren
currentPosition	Zahl	0	Beinhaltet während des Programmablaufs die aktuelle Position der Teilerutsche

Programmierung der Teilfunktionen

Auswerfen

Wir erstellen uns nun eine Funktion, die Teile von der Teilerutsche auswirft. Hierzu schauen wir uns den entsprechenden Mechanismus im realen Modell an und überlegen wie man den Auswurf programmtechnisch umsetzen könnte.

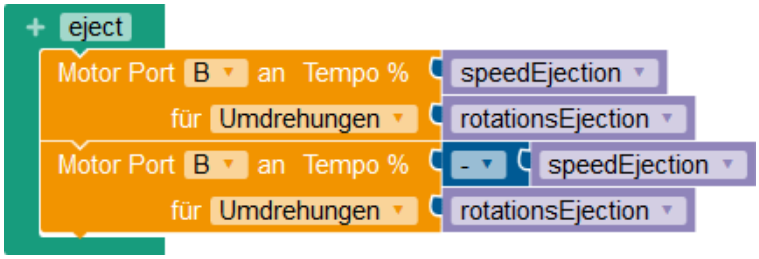


Abbildung 1: Funktion »eject«

In Abbildung 1 ist die Umsetzung einer solchen Funktion zu sehen. Der Motor bewegt sich mit festgelegter Geschwindigkeit (»speedEjection«) und Rotationsanzahl (»rotationsEjection«) zunächst in die eine und dann in die andere Richtung.

Wichtig ist hierbei, dass als Ausgangsposition der ausgefahrene Kolben angesehen wird!

Positionierung

Die Positionierung der Teilerutsche ist hingegen schon etwas aufwendiger. Zunächst sollten wir sicherstellen, dass wir uns in einer definierten Ausgangsposition befinden. Dazu erstellen wir eine Funktion, die die Teilerutsche in die Startposition am Endlagenschalter bringt.

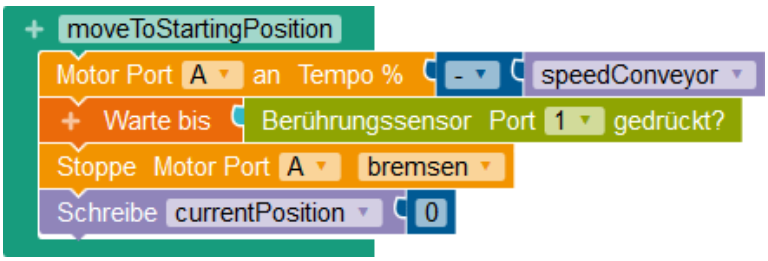


Abbildung 2: Funktion »moveToStartingPosition«

Die in Abbildung 2 dargestellte Funktion schaltet den großen Motor an Port A an und lässt ihn mit der in der Variable »speedConveyor« hinterlegten Geschwindigkeit zurückfahren. Dies wird ausgeführt bis der Endlagenschalter (Berührungssensor an Port 1) gedrückt wird. Nun wissen wir genau, wo sich das Laufband und die Teilerutsche befinden. Jetzt wird der Motor gebremst und wir

halten in der Variable »currentPosition« fest, dass wir am Start angekommen sind (nur für effiziente Variante der Positionierung notwendig).

Für die eigentliche Positionierung werden im folgenden zwei Varianten der Funktion »moveTo« vorgestellt: Die einfache und die effiziente.

Die Funktion erhält über die lokale Variable »colourPosition« die benötigte Umdrehungszahl für die Farbe, die es anzufahren gilt.

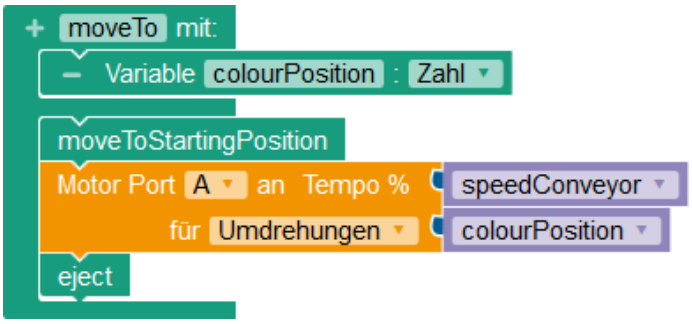


Abbildung 3: Die einfache Variante der Funktion »moveTo«

Die Variante in Abbildung 3 ruft nun unsere zuvor erstellte Funktion »moveToStartingPosition« auf. Anschließend wird mit der festgelegten Geschwindigkeit »speedConveyor« die Anzahl der Umdrehungen ausgeführt, die uns in der lokalen Variable übergeben worden ist.

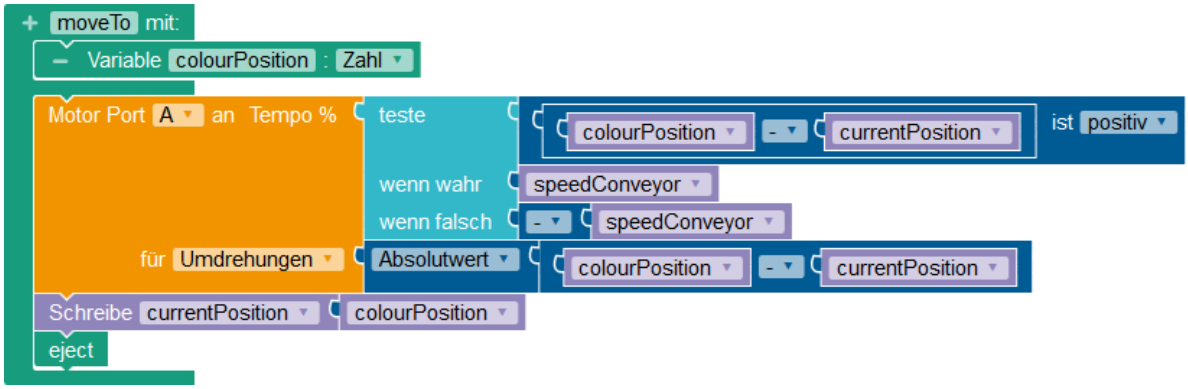
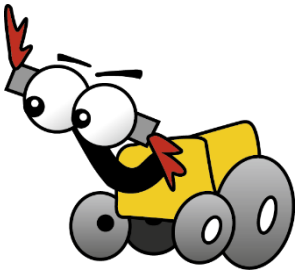


Abbildung 4: Die effiziente Variante der Funktion »moveTo“

Abbildung 4 zeigt nun die effizientere, aber auch etwas komplexere Variante. Hier wird nun nicht mehr nach jedem Teil die Startposition angefahren, sondern von Behälterposition zu Behälterposition verfahren.



Warum wir zu Beginn nicht die Funktion »moveToStartingPosition« aufrufen, wird später unter der Initialisierung erläutert.

Die Bewegungsrichtung des Motors wird über das Vorzeichen des Tempos bestimmt. Daher benutzten wir den »teste«- Block und übergeben entsprechend die Variable »speedConveyor« mit oder ohne Vorzeichenwechsel. Getestet wird hierbei, ob das Ergebnis der Subtraktion von »colourPosition« – »currentPosition« positiv ist.

Beispiel:

Nach unserer Variablenliste sind die Farbbehälterpositionen wie folgt hinterlegt:

Rot = 0
Blau = 0,5
Grün = 1
Gelb = 1,5

Wollen wir also aus der Startposition (»currentPosition« = 0) zum grünen Farbbehälter (»colourPosition« = 1) ist das Ergebnis der Subtraktion »colourPosition« – »currentPosition« (1 – 0 = 1) positiv. Das heißt der Motor dreht vorwärts.

Wollen wir nun vom grünen (»currentPosition« = 1) zum blauen Farbbehälter (»colourPosition« = 0,5) ist das Ergebnis der Subtraktion »colourPosition« – »currentPosition« (0,5 – 1 = -0,5) negativ. Das heißt der Motor dreht rückwärts.

Die benötigte Umdrehungsanzahl wird mit der gleichen Subtraktion bestimmt. Da aber für die Umdrehungen nur positive Zahlen zulässig sind, wird der Absolutwert des Subtraktionsergebnisses genommen.
Anschließend wird in der Variable »currentPosition« die aktuelle Position gespeichert.

Da wir in jedem Fall, in dem wir eine Farbbehälterposition anfahren, auch das entsprechende Teil dort auswerfen wollen, können wir die vorbereitete Funktion »eject« in beiden Varianten am Ende einfügen.

Im Folgenden wird mit der effizienten Variante der Positionierung weitergearbeitet.

Identifikation

Der Farbsensor am Fuß der Rutsche übernimmt für uns in Kombination mit einem erweiterten »Warte bis«- Block die Identifikation.

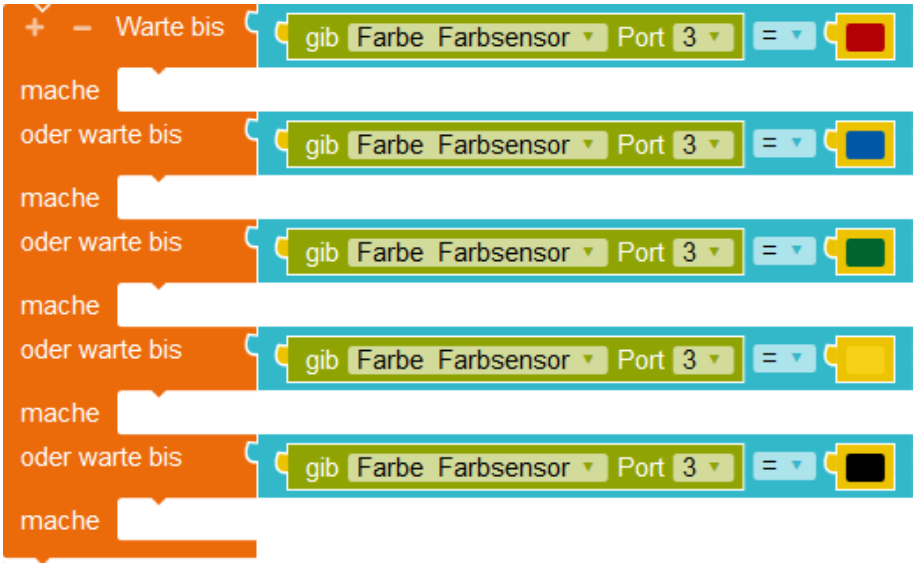


Abbildung 5: Identifikation und Fallunterscheidung

Wie in Abbildung 5 zu sehen ist, unterscheiden wir fünf verschiedene Fälle und führen für jeden eine eigene Aktion aus.

Sortierung

Da wir nun alle Bestandteile der Aufgabe erstellt haben, gilt es diese zu einem Sortieralgorithmus zusammenzusetzen. Hierzu erstellen wir eine neue Funktion »sort« (Abbildung 5).

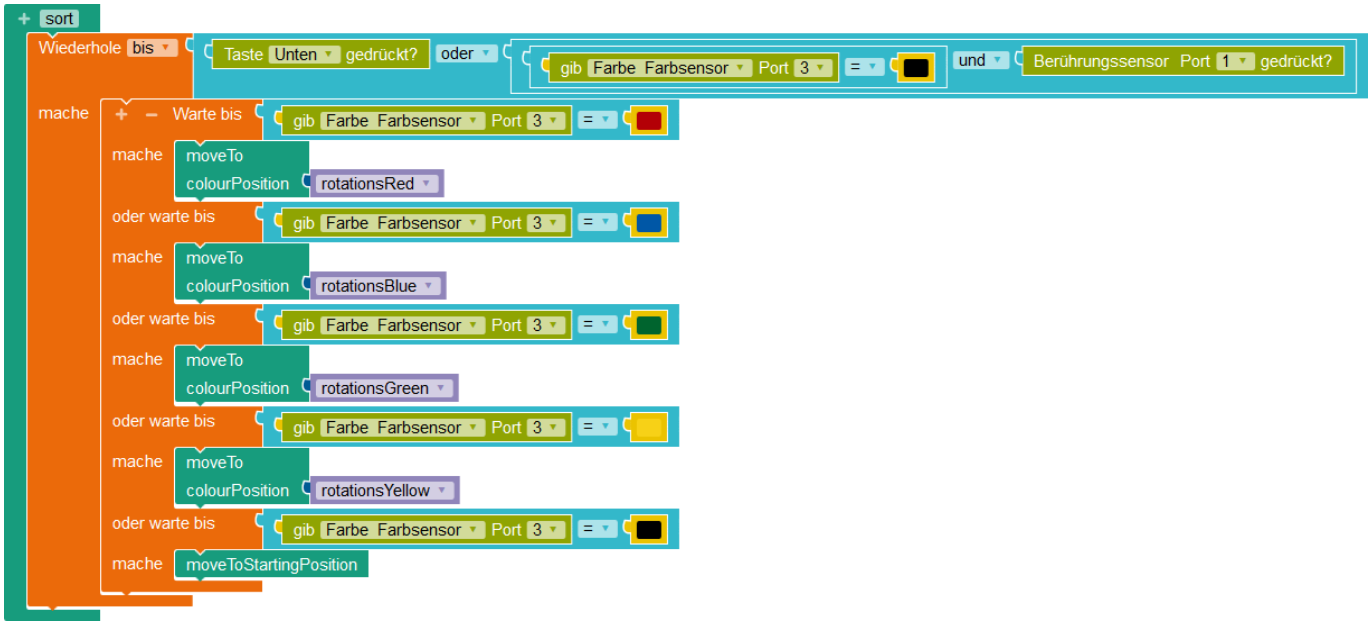


Abbildung 6: Funktion »sort«

In die einzelnen Fälle der Identifikation wurde die Positionierung (enthält das Auswerfen) eingesetzt. Erkennt der Farbsensor eine der vier Farben, wird jeweils unsere präparierte Funktion »moveTo« aufgerufen und die in den Variablen entsprechend gespeicherten Rotationszahlen übergeben. Erkennt der Farbsensor die Farbe »Schwarz« bedeutet dies, dass die Rutsche leer ist und wir fahren mit der Funktion »moveToStartingPosition« zurück zur Ausgangslage.

Dies wird so lange ausgeführt, bis der Benutzer die Sortierung bewusst durch einen Druck auf die untere Taste abbricht, oder wir uns am Endlagenschalter befinden und kein Teil mehr auf der Rutsche ist.

Initialisierung

Die Funktion »initialize« in Abbildung 7 dient dazu, bei jedem Programmstart gleiche Ausgansbedingungen zu schaffen. Daher brauchen wir bei der effizienten Positionierung auch nicht zu Beginn die Funktion »moveToStartingPosition« aufzurufen.

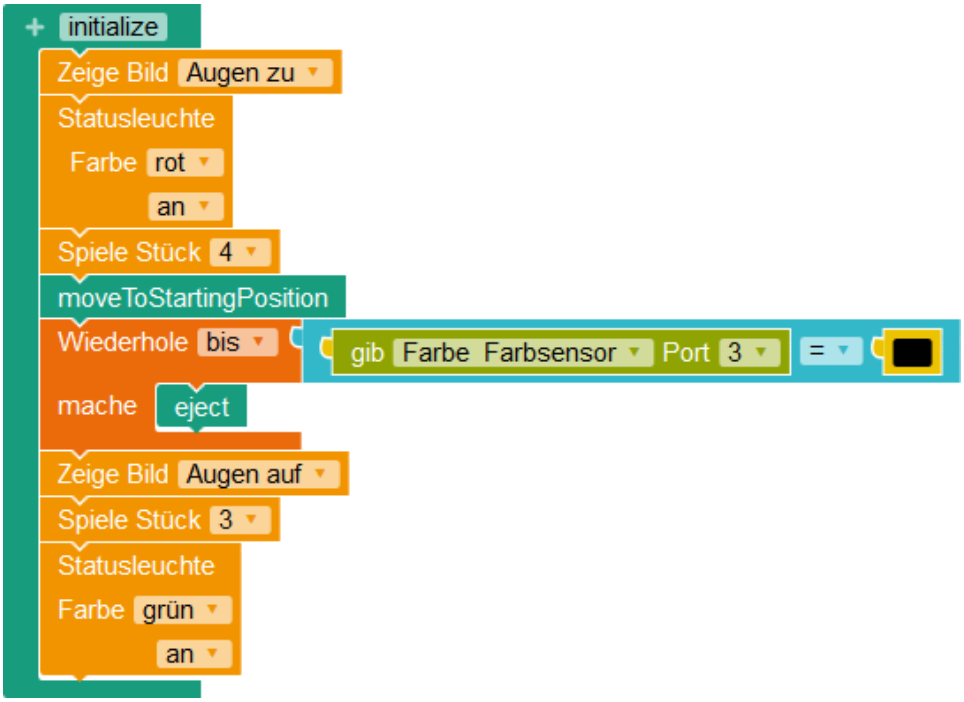


Abbildung 7: Funktion »initialize«

Sie bewegt das Laufband zur Startposition und wirft dann alle auf der Rutsche verbliebenen Teile aus. Umrahmt wurde dies noch mit Bild, Ton und Statusleuchte, um das Ganze optisch und akustisch etwas aufzupeppen.

Hauptprogramm

Nachdem wir die grundlegende Funktionalität hergestellt haben, müssen wir uns nun um den Ablauf unseres Programms kümmern. Die Frage, die wir uns stellen müssen, lautet hierbei:

Was soll wann ausgeführt werden?

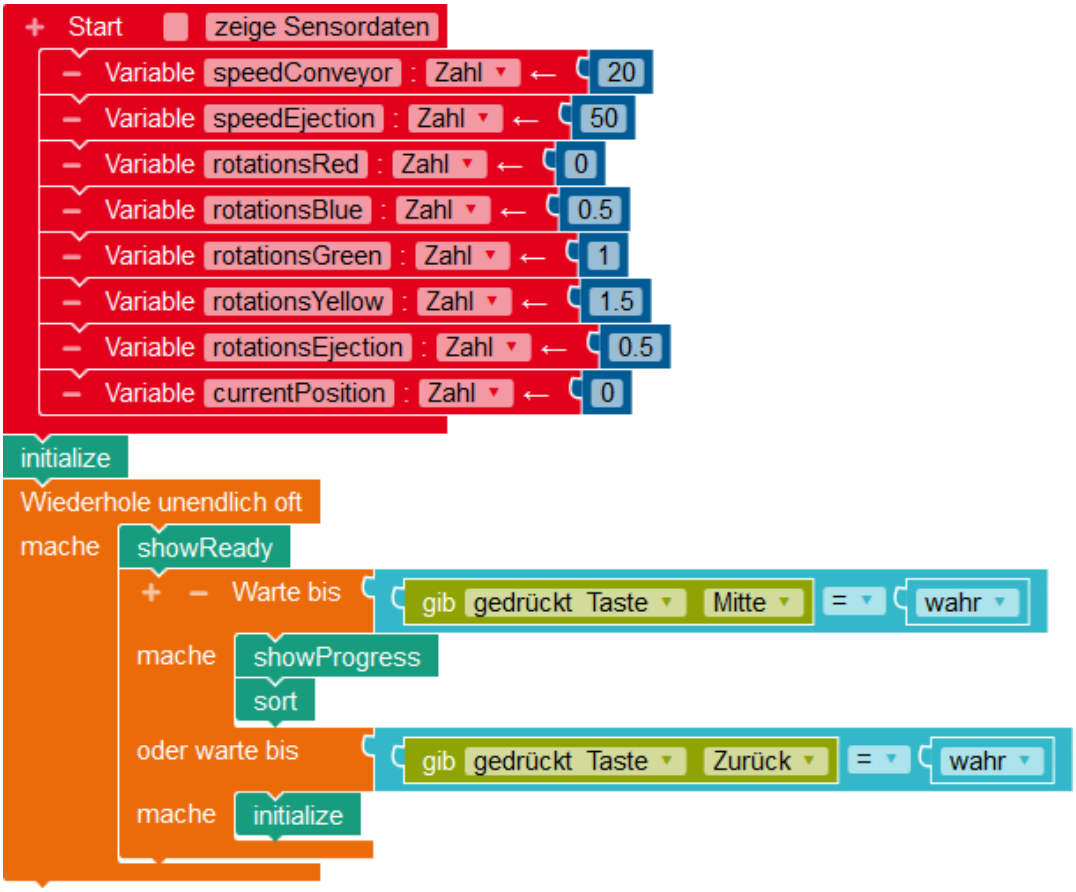


Abbildung 8: Hauptprogramm

Dadurch, dass wir im Vorfeld die einzelnen Arbeitsschritte in Funktionen gekapselt haben, ist unser Hauptprogramm (Abbildung 8) nun ziemlich übersichtlich geworden.

Zu Beginn führen wir die Funktion »initialize« für einen definierten Startzustand aus. Dann gehen wir in eine Endlosschleife über, in der wir dann auf eine Eingabe des Benutzers warten. An dieser Stelle können wir nun beispielsweise die Rutsche mit Teilen füllen und anschließend die mittlere Taste drücken, um den Sortiervorgang zu starten. Eine erneute Initialisierung kann durch einen Druck auf die Zurück-Taste (oben links) ausgeführt werden.

Abschließende Hinweise

Benutzerfreundlichkeit

Wie schon bei der Funktion »initialize« ist auch im Hauptprogramm Wert auf Optik gelegt worden. Dies ist in Form der beiden Funktionen in Abbildung 9 geschehen.

Abbildung 9: Funktionen »showReady« & »showProgress«

Diese sind für Funktionalität der Anlage nicht notwendig, geben dem Benutzer aber Aufschluss an welchem Punkt im Programm er sich gerade befindet.

Funktionsreihenfolge

Programmiert man in NEPO Funktionen, die auf andere Funktionen zugreifen oder diese aufrufen, muss man auf ihre Position auf der Arbeitsfläche achten. Die Funktion, auf die zugegriffen werden soll, **muss** sich oberhalb der zugreifenden Funktion befinden.

Beispiel:

In unserer Funktion »moveTo« rufen wir die Funktion »eject« auf. Daher muss die Programmierung der Funktion »eject« (Abbildung 1) oberhalb der Programmierung von »moveTo« (Abbildung 4) platziert werden.

Die Funktion »sort« ruft wiederum die Funktionen »moveTo« und »moveToStartingPosition« auf. Daher muss sie unterhalb dieser beiden Funktionen platziert werden.

Hintergrund ist der, dass NEPO die Definition der Funktionen in der Reihenfolge generiert, in der die entsprechenden Blöcke auf der Arbeitsfläche platziert sind (von oben nach unten). Wenn man eine Funktion aufrufen möchte, die aber erst danach definiert wird, kann der Computer diese nicht finden und gibt einen Fehler aus.