http://schuelerlabor.informatik.rwth-aachen.de

Leitprogramm

Hello again, B-O-B-3!

[1]

Verfasser: Christina Schramm, Jan Nowak

Kurz-Info:

Informatischer Inhalt: Robotik, Programmierung in Arduino

Jahrgangsstufe: 7 bis EF

Vorwissen: Das Leitprogramm baut auf dem Basis-Workshop zu B-O-B-3 im Rahmen von go4IT! auf.

KURZINFORMATION

[1]

Titel: Hello again, B-O-B-3!

Schulstufe: Mittel- und frühe Oberstufe

Optimale Jahrgangsstufe: 7 bis EF

Themenbereich: Robotik, (textuelle) Programmierung eines Roboters in Arduino

Leitideen: Dieses Leitprogramm kann nach dem Besuch des go4IT!-Basis-Workshops zu B-O-B-3 zur weiteren, eigenständigen Beschäftigung mit dem eigenen Roboter genutzt werden.

EINORDNUNG IN GESETZLICHE RAHMENBEDINGUNGEN

Lehrplan NRW: Kompetenzbereiche: Modellieren und Implementieren; Inhaltsfelder: Algorithmen, Informatiksysteme

Vorkenntnisse: Das Leitprogramm baut auf dem Basis-Workshop zu B-O-B-3 im Rahmen von go4IT! auf.

Inhaltsbeschreibung: Dieses Leitprogramm gibt den Schülerinnen Impulse für die weitere Beschäftigung mit ihrem B-O-B-3. In spannenden Projekten wiederholen und festigen sie ihr Wissen aus dem Workshop und wenden es in neuen Kontexten an. Darüber hinaus lernen sie neue informatische Konzept kennen. Fachbegriffe wie Integer, Bool, For-Schleife, Zufallsfunktion etc. können die Schülerinnen nach der Bearbeitung inhaltlich definieren, denn sie wenden diese im Rahmen der Programmierprojekte an. Ein Disco-Licht, ein Kometendetektor, eine Hitzefrei-Anzeige und ein Merkspiel warten darauf, von den Schülerinnen programmiert und auf dem B-O-B-3 ausgeführt zu werden.

Technische Voraussetzungen: Neben dem B-O-B-3 und dem zugehörigen USB-Kabel sollte ein Computer oder Laptop zur Verfügung stehen, an den/das der B-O-B-3 per USB angeschlossen werden kann. Für die Programmierung verwenden die Schülerinnen wie schon im Basis-Workshop Arduino. Die Software kann kostenlos heruntergeladen werden. Die Bibliotheken für den B-O-B-3 stellt das InfoSphere auf Anfrage gerne zur Verfügung. Installation und Einbindung der Bibliotheken werden zu Beginn angeleitet. Bei Fragen können die Schülerinnen sich an infosphere-support@informatik.rwth-aachen.de wenden.

Zur Testung der Programme ist es gelegentlich hilfreich, wenn man den B-O-B-3 ohne Kabel nutzen kann. Hierfür wird eine Batterie vom Typ CR2032 benötigt.

EINFÜHRUNG

Du hast am go4IT!-Basis-Workshop zum B-O-B-3 teilgenommen und willst deinen kleinen Roboter nun weiterhin nutzen, indem du neue, spannende Projekte programmierst? Dann ist dieses Leitprogramm genau das richtige für dich. Zunächst hilft dir das Leitprogramm, die Software zu installieren, die du bereits im Workshop verwendet hast. Falls du fürchtest, dass du Einzelheiten aus dem Workshop schon wieder vergessen hast, ist das kein Problem: Das Leitprogramm startet mit einer kleinen Wiederholung, damit du dich schnell wieder zurechtfindest. Anschließend wendest du dein schon vorhandenes Programmierwissen in neuen Kontexten an und entdeckst darüber hinaus auch Neues. Fachbegriffe wie Datentypen, Integer, Boolean, For-Schleife oder Zufallsfunktion sind dann kein Problem mehr für dich. Keine Sorge: Es wird zwar immer wieder kleinere, theoretische Exkurse geben, jedoch dienen diese als Basis für deine Hauptaufgabe: das Programmieren spannender Projekte wie Kometendetektor, Hitzefrei-Anzeige, Disco-Licht und die Realisierung eines Merkspiels.

Was du brauchst:

- Bleistift, Füller oder Kugelschreiber
- Computer oder Laptop
- Zusammengelöteter B-O-B-3 (inkl. USB-Kabel, Helm und Batterie)
- Um die Aufgaben auf Papier zu bearbeiten, druckst du dir das Leitprogramm am besten aus. Alternativ kannst du aber auch ein paar Blätter Papier für deine Notizen nehmen.

Arbeite dieses Leitprogramm sorgfältig durch! Überspringe keine Aufgabe, denn diese bauen aufeinander auf. Deshalb ist es auch wichtig, erst weiterzuarbeiten, wenn du die vorherige Aufgabe erfolgreich gelöst hast, oder wenn dein Programm läuft. Arbeite nicht weiter, wenn dein Programm einen Fehler anzeigt. Suche lieber nach diesem Fehler. Auch dies gehört zur Arbeit einer Informatikerin. Manchmal muss man zweimal hinsehen, um das Problem zu finden. Aber lasse dich davon nicht unterkriegen. Du schaffst das! Falls es dennoch irgendwo haken sollte, kannst du die Musterlösungen zu Rate ziehen. Außerdem hilft das InfoSphere-Team dir sehr gerne weiter. Sende deine Fragen an infosphere-support@informatik.rwth-aachen.de.

INHALT

ARBEITSANLEITUNG

Das Leitprogramm kannst du alleine bearbeiten. Die Kapitel bestehen aus unterschiedlichen Teilen, die du an den folgenden Symbolen erkennen kannst:

Lernziel

Was wirst du nach dem Bearbeiten des Kapitels (mehr) können?

Theorie

Lies diesen Teil aufmerksam. Hier lernst du alles, was du für die Aufgaben und die Tests brauchst.

Übungsaufgaben am Computer

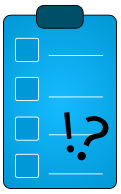
Bearbeite diese Aufgaben am Computer. Arbeite erst weiter, wenn dein Programm läuft. Du darfst diese Aufgaben alleine oder mit einem Partner bearbeiten.

Übungsaufgaben auf Papier

Bearbeite diese Übungsaufgaben alleine. Vergleiche sie dann mit der Musterlösung im Anhang.

Sicherungsphase

Dieser Teil hilft dir beim Einprägen und der Kontrolle der gelernten Themen!



Test

Teste dich selbst. Hast du alles verstanden, oder musst du dir einzelne Aufgaben noch einmal ansehen? Starte erst mit dem nächsten Kapitel, wenn du den Test erfolgreich gemeistert hast.

TECHNISCHE VORBEREITUNG

Bevor es inhaltlich losgeht, musst du deinen Computer vorbereiten, indem du die Arduino-Software installierst, die du für die Programmierung des B-O-B-3 verwenden wirst. Diese Software kennst du schon aus dem Workshop. Damit Arduino für deinen B-O-B-3 funktioniert, musst du Bibliotheken einbinden. Einen **Ordner mit sämtlichen Materialien** und den **Bibliotheken** erhältst du auf Anfrage von uns (schuelerlabor@informatik.rwth-aachen.de). Folge der Anleitung, um die Arduino-Software so zu installieren, dass du deinen B-O-B-3 damit programmieren kannst. (Die Anleitung entspricht den Schritten unter Windows 10. Je nach Betriebssystem sind Unterschiede möglich.) Solltest du Fragen oder Probleme haben, kannst du uns eine Mail schreiben.

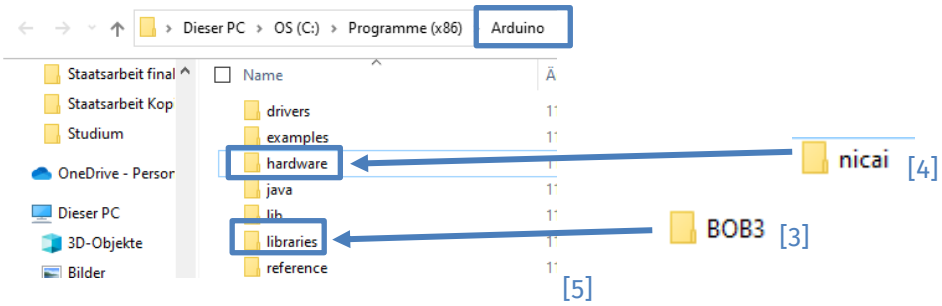
1.) Gehe in dem Ordner mit den **Materialien**, die du von uns erhalten hast, in den Unterordner mit dem Namen **Software**.

2.) Installiere die **Arduino Software:** arduino-1.8.2-windows [2]

- **a.** Starte die Installation durch einen Doppelklick.
- **b.** Es erscheint ein Fenster mit der Frage: „Möchten Sie zulassen, dass durch diese App Änderungen an Ihrem Computer vorgenommen werden?“ → „Ja“ anklicken.
- **c.** Dann musst du der Lizenz zustimmen, indem du „I agree“ anklickst.
- **d.** Es öffnet sich ein Fenster mit Installationsoptionen. Klicke hier auf „Next“.
- **e.** Dann wählst du den Ort, an dem die Software auf deinem PC gespeichert werden soll. Meist ist hier schon Laufwerk C vorgeschlagen. Frage deine Eltern, ob du das Programm hier speichern darfst. Klicke „Install“ und warte, bis die Installation abgeschlossen ist.
- **f.** Es erscheint ein Windows-Sicherheitsfenster. Hier musst du ein paar Mal „Installieren“ anklicken, dann „Close“.

3.) Nach der Installation musst du die Arduino-Software noch auf B-O-B-3 vorbereiten.

- **a.** Navigiere dazu ins **Arduino-Verzeichnis**, also dorthin, wo du Arduino auf deinem PC installiert hast. Am einfachen ist es, hierzu einen Rechtsklick auf die Arduino-Software auf deinem Desktop zu machen und dann „Dateipfad öffnen“ anzuklicken.
- **b.** Hier sind nun zwei Unterordner wichtig: **hardware** und **libraries.** In diese beiden Unterordner musst du nun zwei Ordner aus den Materialien kopieren.
  - i. BOB3 [3] (unter Materialien → Software → libraries) kopierst du in den **libraries-Ordner** der Arduino-Software.
  - ii. nicai [4] (Materialien → Software) in den **hardware-Ordner** der Arduino-Software.



4.) Jetzt musst du noch ein paar kleine Einstellungen in der Arduino-Software vornehmen.

a. Starte die Arduino-Software.

b. Gehe in die **Werkzeuge**. Wähle unter **Board** „nicai systems BOB3 coding bot“ aus.

[6]

Wähle unter **Board** „nicai systems BOB3 coding bot“ aus. Setze den **Programmer** auf „ArduinoISP“.

c. Schließe das Programm, damit die Änderungen übernommen werden.

Da du im Workshop bereits mit Arduino gearbeitet hast, empfehlen wir dir, hiermit weiterzuarbeiten. Falls du Arduino dennoch nicht installieren möchtest, kannst du alternativ online mit **ProgBob** arbeiten. Hier musst du allerdings ein Benutzerkonto anlegen. Außerdem kann der Code teilweise abweichen.

Übersicht Das menschliche Gehirn ist kein Computer ☺. Mache dir also keine Sorgen, wenn du einzelne Inhalte oder Programmierbefehle aus dem Workshop wieder vergessen hast. In diesem Kapitel wiederholst du noch einmal das Wichtigste.

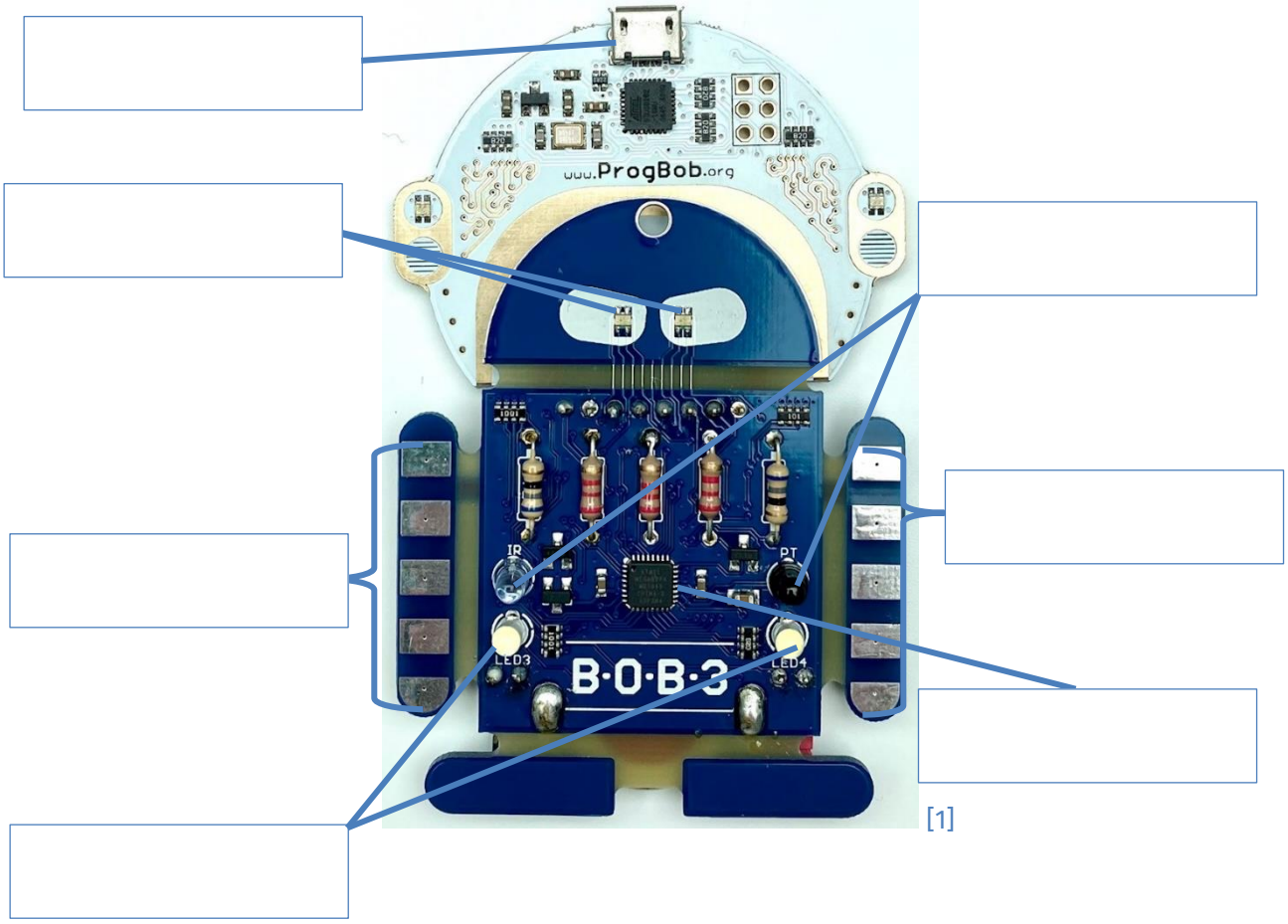
LERNZIEL

In diesem Kapitel wiederholst du die Inhalte des Basis-Workshops, indem…

- du die Bauteile des B-O-B-3 benennst sowie ihre Funktion bestimmst und
- die relevanten Programmierbefehle in bereits aus dem Basis-Workshop bekannten Kontexten anwendest.

Aufgaben 1.1

Schau dir zunächst noch einmal deinen B-O-B-3 (im Folgenden einfach Bob genannt) im Detail an. Kannst du noch alle Bauteile bestimmen? Nutze Abbildung 1, um die einzelnen Bauteile zu beschriften. Du brauchst Tipps? Dann schau in die **Box** auf der nächsten Seite, in der du alle Bauteile aufgelistet findest.



[1]

Tipp-Box:

RGB-LEDs, Weiße LEDs, Mikro-USB-Anschluss, Temperatur-Sensor, Touch-Sensor, Infrarot-Sensor

Aufgabe 1.2

Alle Bauteile haben eine Funktion. Ordne die Bauteile ihrer jeweiligen Funktion zu.

Bauteil	Funktion
	Mit diesem Sensor, der aus einem Sender und einem Empfänger besteht, kann der Roboter erkennen, ob ein Hindernis vor ihm ist und wie weit dieses entfernt ist.
	Mit Hilfe der Sensorfelder kann der Roboter auf Berührung reagieren.
	Versteckt im Mikro-Prozessor („Roboter-Hirn“) kann dieser Sensor die Temperatur messen.
	Diese LEDs können in verschiedenen Farben leuchten.
	Diese LEDs geben weißes Licht ab.
	Hier kann das Kabel angeschlossen werden, das Bob mit dem Computer verbindet.

Bobs Aufbau hast du somit schon einmal wiederholt. Wirf, bevor du weiterarbeitest, einen Blick in die Musterlösungen und überprüfe, ob du in den Aufgaben 1.1 und 1.2 alle Bauteile richtig zugeordnet hast.

Aufgabe 1.3

Da es möglicherweise schon eine ganze Zeit her ist, seit du den Workshop besucht hast, wiederholst du nun noch einmal die Basis-Programmierbefehle, die du benötigst, um die unterschiedlichen Bauteile zu verwenden. Im Folgenden siehst du Code-Schnipsel. Beschreibe in deinen eigenen Worten, was diese bewirken, wenn du sie in ein Programm einbaust.

Beispiel:

```
#include <BOB3.h>
```

Bindet die Befehlsbibliothek für Bob ein und steht am Anfang jedes Programms

```
bob3.setLed(EYE_1, GREEN);
```

```
bob3.setWhiteLeds(ON,ON);
```

```
int wert;
```

```
delay(500);
```

```
if (…)
  {…}
  else {…}
```

```
while(…) {…}
```

```
bob3.getIRSensor()
```

```
bob3.getTemperature()
```

```
bob3.getArm(1)
```

Wirf einen Blick in die Musterlösungen, bevor du weiterarbeitest. Hier findest du eine Befehlsübersicht, die alle Befehle zusammenfasst, die du im Workshop gelernt hast.

THEORIE: DAS GRUNDGERÜST DES PROGRAMMS

```
#include <BOB3.h>

void setup() {

}

void loop() {

}
```

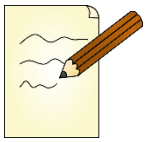
[7]

Gehört immer an den Anfang, um die **Befehle für Bob** einzubinden.

Wird **einmal** beim Programmstart ausgeführt.

Wird immer wieder **wiederholt**.

Dieses Grundgerüst muss in jedem Programm für deinen Bob vorhanden sein. Außerdem solltest du immer auf Klammern, Semikolons (;), Groß- und Kleinschreibung und die korrekte Schreibweise achten. Wenn beim Kompilieren [8] ein Fehler ausgegeben wird, kannst du diese Punkte als Erstes überprüfen.



Aufgabe 1.4

a. Kim hat im Workshop mehrere Programme geschrieben, um die LEDs zu programmieren. Sie bekommt nach dem Kompilieren aber jedes Mal Fehlermeldungen angezeigt. Hilf ihr, die Fehler zu finden und berichtige sie.

Kims erstes Programm zeigt drei Fehlermeldungen:

```
#include BOB3.h
void setup() {
  bob3.setLeds(ON,ON)
}
```

Kims zweites Programm zeigt vier Fehlermeldungen:

```
void setup() {
  bob3.setWhiteLeds(ON,ON);

void loop(){
  bob3.setLed(EYE_1, GRIN);
  bob3.setLed(EYE_2, GRIN);
}
```

Das dritte Programm zeigt keinen Fehler an, funktioniert aber nicht wie gewünscht. Kim möchte, dass die weißen LEDs blinken. Sie sind aber dauerhaft an. Findest du den Fehler?

```
#include <BOB3.h>
void setup() {
{
void loop() {
  bob3.setWhiteLeds(ON,ON);
  bob3.setLeds(OFF,OFF);
}
```

b. Überprüfe, ob dein Feedback an Kim aus **Aufgabe 1.4a** richtig ist und ihre Programme nun funktionieren, indem du deine verbesserten Programme in Arduino schreibst und dann auf deinen Bob überträgst.

- Schalte also erst einmal deinen Computer an und starte die **Arduino**-Software.
- Überprüfe noch einmal, ob die Software für den Bob passend konfiguriert ist (siehe technische Vorbereitung, Schritt 4). Verbinde deinen Bob mit deinem Computer.

- Schreibe nach und nach Kims verbesserte Programme in Arduino. Öffne für jedes Programm einen neuen Sketch [9]. Speichere das jeweilige Programm unter einem passenden Namen (Datei → Speichern unter). Wenn ein Programm fertig ist, dann überprüfe es [8] und lade es dann auf deinen Bob [10].

Probleme beim Hochladen? Dann überprüfe, ob der richtige Port eingestellt ist. Gehe dazu auf **Werkzeuge → Port**. Hier sollte **COM** ausgewählt sein. Hinter dem COM steht eine Zahl, die variieren kann.

Konntest du Kims Programme auf Bob übertragen? Funktionieren sie, wie sie sollen? Das ist super! Dann kannst du sofort weiterarbeiten. Wenn du in Arduino Fehlermeldungen angezeigt bekommst oder Bob nicht ausführt, was er soll, dann wirf einen Blick in die Musterlösungen, um zu überprüfen, ob du alle Fehler in Kims Programmen richtig korrigiert hast. Überprüfe zusätzlich, ob du alle Klammern und Semikolons (;) richtig gesetzt hast.

Test

Hast du dir die grundlegenden Workshopinhalte wieder ins Gedächtnis gerufen? Dann bist du sicher fit, dein Wissen zu testen.

Aufgabe T1: Nachdem Kim und ihre Freundin den Workshop besucht haben, sollen sie für ihre Informatik-AG einen Blogbeitrag schreiben, der Bob vorstellt. Die beiden Mädchen haben gemeinsam einen Textentwurf geschrieben, den Kim nun nur noch digitalisieren muss. Leider kann sie die Handschrift ihrer Freundin an einigen Stellen nicht mehr lesen. Hilf ihr, indem du in die Lücken die fehlenden Begriffe einsetzt.

In dem Workshop haben wir den kleinen Roboter Bob kennengelernt, ihn zusammengelötet und programmiert. Bob hat einiges drauf. So verfügt er über mehrere Lampen. Da es sich bei den Lampen in den Augen um _____ handelt, können sie in allen Farben leuchten. Die LEDs am Körper dagegen leuchten nur in _____.

Am Roboter-Körper befinden sich zwei weitere LEDs – die eine ist hell, die andere dunkel. Zusammen bilden sie den _____, mit dem Bob nah und fern unterscheiden kann. Daneben verfügt Bob über weitere Sensoren. So befindet sich im Mikro-Prozessor, dem „Roboter-Hirn“, das den Programmcode ausführt, der _____. In den Armen sind _____ verbaut. Mit Sensoren lassen sich spannende Programme verwirklichen, denn mit ihnen ist es dem Roboter überhaupt erst möglich, auf seine Umwelt zu reagieren. Jedes Programm hat ein Grundgerüst. Neben dem Setup- und dem Hauptprogramm gehört der Befehl, der die Bibliothek für Bob einbindet, zum Grundgerüst jedes Programms; er lautet _____ und muss am Anfang jedes Programms stehen. Das Setup-Programm wird einmal ausgeführt beim _____. Das Hauptprogramm wird immer wieder _____. Ein wichtiges Konzept in der Programmierung sind bedingte Anweisungen. Bei der _____-Anweisung wird ein Programmabschnitt nur ausgeführt, wenn eine bestimmte Bedingung erfüllt ist. Bei der _____-Schleife wird eine Anweisung solange wiederholt, wie eine bestimmte Bedingung erfüllt ist. Das klingt vielleicht kompliziert, aber es macht wirklich Spaß.

Hast du alle Lücken füllen können? Falls nein, dann such noch einmal nach den passenden Antworten in den Lösungen der Aufgaben, die du bisher bearbeitet hast. Falls ja, dann wirf einen Blick in die Musterlösung und vergleiche deine Antworten.

Übersicht Weihnachten 2021 hat die Netflix-Produktion *Don't Look Up!* Schlagzeilen gemacht. In dem Film entdecken Forschende einen Kometen, der sich auf Kollisionskurs mit der Erde befindet. Aus politischen und wirtschaftlichen Gründen schenkt die amerikanische Staatschefin den Forschenden in dieser schwarzen Komödie keine Aufmerksamkeit. In der Realität wirst du deinen Bob, der mit seinem Helm sowieso wie ein kleiner Astronaut oder Bewohner eines anderen Planeten aussieht, zum Kometendetektor machen.

LERNZIEL

In diesem Kapitel programmierst du den IR-Sensor und die LEDs deines Bobs, die du aus dem Workshop bereits kennst. Du wendest die Programmierbefehle in einem neuen Kontext an, indem du Bob als **Kometendetektor** programmierst.

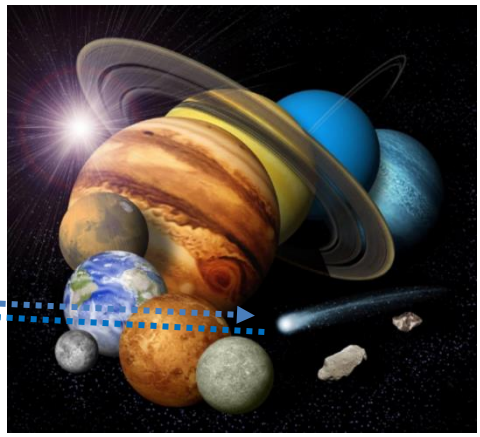
Aufgabe 2.1

Öffne ein neues Sketch. Schreibe wie immer `#include <BOB3.h>` in die erste Zeile dieses Sketches, um die Befehlsbibliothek einzubinden. Benenne dein Programm passend und speichere es. Denke später daran, auch Zwischenschritte immer wieder abzuspeichern.

THEORIE: DER IR-SENSOR

Den IR-Sensor kennst du als Bauteil bereits aus dem Workshop. Trotzdem findest du hier noch einmal die wichtigsten Punkte sowie einige Details, die im Workshop nicht genauer betrachtet wurden und dir bei der Programmierung des Kometendetektors helfen:

[1]


[11]

Der IR-Sensor besteht aus zwei Teilen: der hellvioletten Sende-LED und der schwarzen Empfänger-LED. Die Sende-LED sendet Infrarotlicht aus. Dieses Infrarotlicht trifft dann auf ein Hindernis, beispielsweise einen Kometen, und wird von diesem zurückreflektiert. Das zurückreflektierte Licht kann dann durch die Empfänger-LED detektiert werden. Je näher das Hindernis vor dem IR-Sensor

ist, desto mehr Licht wird zurückgestrahlt und kann durch die Empfänger-LED detektiert werden. Durch dieses Reflektionsverfahren kann Bob erkennen, ob ein Hindernis nah oder fern ist. Je höher also der Sensorwert ist, desto mehr Licht wird detektiert und desto näher ist das Hindernis.

Die gemessenen Sensorwerte können mit der Methode `bob3.getIRSensor` abgefragt und in einer Variablen (z. B. `int sensorwert;`) abgespeichert werden.

[1]

Erinnerst du dich? **Variablen** sind Speicher mit einem eindeutigen Namen. Sie werden zu Beginn eingeführt, also deklariert. Immer wenn der Name des Speichers benutzt wird, wird dessen Inhalt benutzt.

Aufgabe 2.2

Programmiere Bob als Kometendetektor.

a. Deklariere zunächst die Variable für die IR-Sensorwerte.

b. Frage in der loop-Funktion mit der Methode `bob3.getIRSensor` den aktuellen Sensorwert ab und speichere ihn in der Variablen:
`int sensorwert = bob3.getIRSensor`

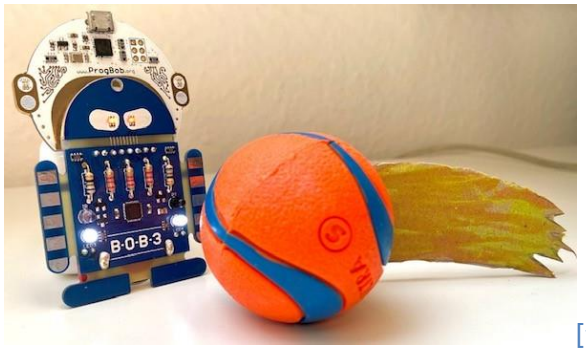
c. Programmiere dann die Reaktion des Bobs auf die gemessenen Sensorwerte. Unterscheide dabei zwei Fälle:

- Wenn der Komet sich nähert, aber noch nicht ganz nahe ist, dann sollen die LED-Augen orange leuchten. Als Sensorwert kannst du hier z. B. 10 ausprobieren.
- Wenn der Komet sehr nahe ist, sollen die Alarmsignale verstärkt werden: Die weißen LEDs sollen zusätzlich blinken.

Du benötigst für die beiden Fälle also zwei **if-Anweisungen**.

d. Programmiere den **else-Teil**: Wenn kein Komet in der Nähe ist, dann soll auch keine LED leuchten.

e. Teste dein Programm und sei dabei ein bisschen kreativ. Einen Kometen kannst du beispielsweise mit einem kleinen Ball simulieren.



[12]

Funktioniert dein Kometendetektor. Falls ja, dann ist das super. Falls nicht: Überprüfe noch einmal, ob du die Operatoren (>, <) und Werte sinnvoll gesetzt hast: Je näher das Hindernis vor dem IR-Sensor ist, umso mehr Infrarotlicht wird zurückgeworfen.

Du kannst auch die Musterlösung zu Rate ziehen, wenn du Hilfe brauchst.

Test

Wenn du deinen Kometendetektor erfolgreich programmiert hast, bist du bereit für den nächsten Kapiteltest.

Aufgabe T2.1: Wahr oder falsch?

a. Variablen sind Speicher für Daten.

☐ wahr ☐ falsch

b. Wenn man Variablen einführt, nennt man das in der Informatik „deklarieren“.

☐ wahr ☐ falsch

c. Um den Abstand zu einem Objekt, wie beispielsweise einem Kometen, messen zu können, nutzt Bob den Temperatursensor.

☐ wahr ☐ falsch

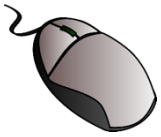
Aufgabe T2.2: Welche Art von Licht nutzt Bob, um den Abstand zu einem Objekt zu messen?

_____

Übersicht Bei jeder Party sorgen bunt blinkende Lichter für die richtige Stimmung. Für deine nächste Party kannst du Bob als Discolicht nutzen. Du verwendest die Touch-Sensoren in den Armen, um mehrere Schalter für die unterschiedlichen Discolichter zu programmieren.

LERNZIEL

In diesem Kapitel wiederholst du Programmierbefehle, die dir aus dem Basisworkshop schon bekannt sind, und wendest sie in einem neuen Kontext an, indem du Bob als **Discolicht** programmierst. Darüber hinaus erweiterst du dein Programmierwissen: Um deine Programme noch effizienter und übersichtlicher gestalten zu können, lernst du die **For-Schleife** kennen.

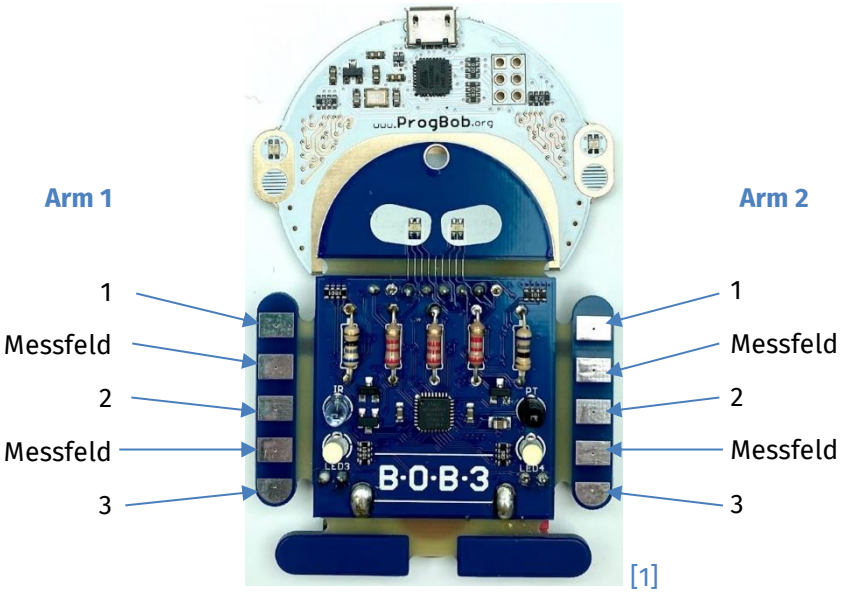


Aufgabe 3.1

Öffne ein neues Sketch. Binde wie immer die Befehlsbibliothek für Bob ein. Benenne dein Programm passend und speichere es. Denke später daran, auch Zwischenschritte immer wieder abzuspeichern.

THEORIE: DIE TOUCH-SENSOREN

Als Schalter für dein Discolicht sollen die **Touch-Sensoren** in den Armen dienen. Hier siehst du noch einmal den genauen Aufbau beider Arme. Damit die Sensoren 1 bis 3 funktionieren, muss immer ein Messfeld mitberührt werden.



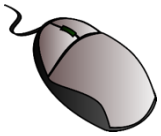
[1]

Mit `bob3.getArm(1)` fragst du den Sensorwert für Arm 1 ab. Wird der Arm berührt, ist der Wert ungleich Null. Das schreibt man so:

```
bob3.getArm(1)!=0
```

Möchtest du nun wissen, an welcher Stelle genau Bob an Arm 1 berührt wird, brauchst du einen Vergleich (==). Dieser sieht für das mittlere Sensorfeld von Arm 1 folgendermaßen aus:

```
bob3.getArm(1)==2
```



Aufgabe 3.2

Programmiere nun den ersten Schalter (Arm 1, Sensorfeld 1) deines Discolichts.

Falls du den ersten Schalter betätigst, **dann** sollen Bobs LED-Augen leuchten oder (in unterschiedlichen Farben) blinken, **sonst** nicht.

a. Schreibe also zunächst in der loop-Funktion eine **if-Abfrage** mit der Bedingung `bob3.getArm(1)==1`

b. Schalte im **dann-Teil** die Augen deines Roboters in der gewünschten Farbe ein. Du kannst auch den **delay**-Befehl einsetzen, um die Augen (in unterschiedlichen Farben) blinken zu lassen.

c. Schalte im **else-Teil** die Augen aus.

d. Teste das Programm für den ersten Schalter, indem du es zunächst kompilierst und dann auf deinen Bob hochlädst.

Tipp-Box:

Denk beim Testen des Programms daran, dass du neben dem eigentlichen Sensorfeld (1, 2, 3) auch immer ein **Messfeld** berühren musst.

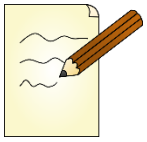
THEORIE: DIE FARBEN

Im Folgenden findest du eine Liste der Farben, die du als Parameter für die RGB-LED-Augen programmieren kannst.

OFF	YELLOW	CYAN	PURPLE
WHITE	ORANGE	AQUAMARINE	VIOLET
GREEN	RED	ROYALBLUE	
FORESTGREEN	COLERED	BLUE	

Aufgabe 3.3

Nutze jetzt alle weiteren Sensorfelder außer das dritte Feld des zweiten Arms, um weitere Modi für dein Discolicht zu programmieren. Es wäre doch cool, wenn du für jede Musikrichtung ein passendes Licht hättest.

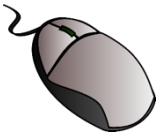


a. Bevor du die unterschiedlichen Touch-Sensoren programmierst, solltest du deine Ideen sammeln:

- Was soll jeweils passieren, wenn du die unterschiedlichen Sensorfelder berührst?
- Welche Farben sollen jeweils eingesetzt werden?
- Sollen die Augen blinken? Wie schnell?

Arm 1	
1	
2	
3	

Arm 2	
1	
2	

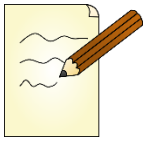


b. **Programmiere** nun die Ideen, die du in der Tabelle gesammelt hast. Teste dein Programm auf deinem Bob.

Tipp-Box:

Bei der Programmierung der übrigen Schalter kannst du dich an deinem Programm für Arm 1, Feld 1 orientieren.

Funktionieren die fünf Sensorfelder, die du bisher als Schalter für dein Discolicht programmiert hast? Falls ja, herzlichen Glückwunsch! Falls nicht, dann vergleiche dein Programm mit der Musterlösung. So kannst du mögliche Fehler entdecken und dann korrigieren.



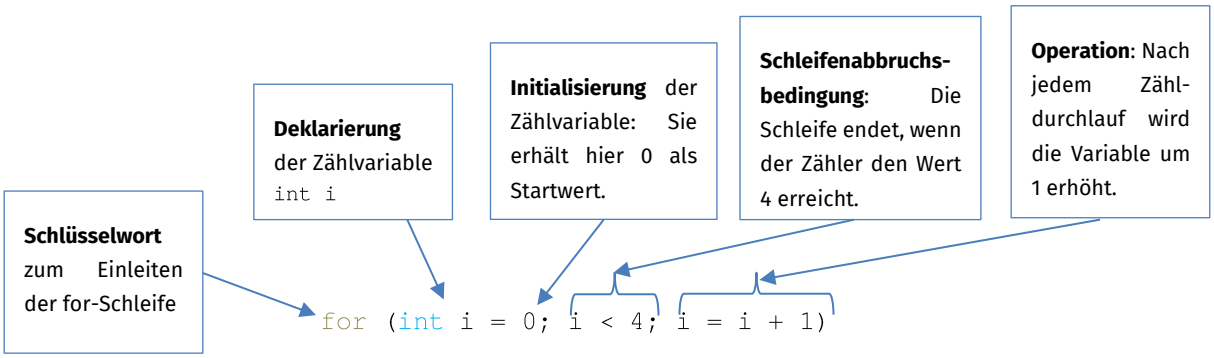
Aufgabe 3.4

Sicherlich hast du dich schon gefragt, warum du bisher nicht das dritte Sensorfeld des zweiten Arms für dein Discolicht nutzen solltest. Dieses Feld wirst du jetzt für eine Licht-Choreographie nutzen. Allerdings ist das Programm für die anderen fünf Felder schon so recht lang, oder? Lies den folgenden Theorieteil zur **for-Schleife**, damit du dein Programm für die Lichtchoreographie effizient gestalten kannst.

THEORIE: DIE FOR-SCHLEIFE

Wenn die Anzahl, die ein Befehl (oder mehrere Befehle) durchlaufen werden soll, bekannt ist, kann man die **for-Schleife** verwenden. Die for-Schleife ist nämlich zählergesteuert; sie wird daher Zählvariable genannt. Man benötigt immer eine **Variable**, die verwendet wird, um die Durchläufe zu zählen. Meist wird diese Zähl-Variable mit einem Buchstaben des Alphabets benannt. Im folgenden Beispiel ist dies das i.

Eingeleitet wird die Schleife mit dem Schlüsselwort *for*. Es folgt eine runde Klammer () mit drei Bereichen und dann eine geschweifte Klammer {}, in der der Befehl (oder die Befehle) steht, die von der for-Schleife wiederholt werden sollen.




```
    //die Befehle, die von der for-Schleife wiederholt
    werden sollen
```

}

Aufgabe 3.5

a. Überlege dir eine Lichtchoreographie, in der die for-Schleife genutzt wird. Sammle zunächst deine Ideen:

Arm 2	
3	

b. Programmiere deine **Licht-Choreographie** für Arm 2, Sensorfeld 3. Nutze hierzu die **for-Schleife**.


[13]

Funktionieren nun alle sechs Sensorfelder, inklusive der Lichtchoreographie? Falls ja, kann die nächste Party kommen! Sollten dir noch Fehler angezeigt werden oder du Probleme hast, dann wirf einen Blick in die Musterlösung.

Test

Wenn du dein Discolicht erfolgreich programmiert hast, bist du bereit für den nächsten Kapiteltest.

Aufgabe T3: Kim hat für ihre Lichtchoreographie schon Ideen aufgeschrieben. Übertrage diese in einen Code. Speichere diesen in einem neuen Programm, damit du auf deinem PC weiterhin dein komplettes Discolicht-Projekt hast und es, wenn du möchtest, immer wieder auf deinen Bob laden kannst.

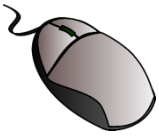
Kims Ideen:

Arm 2	
3	• Beide Augen leuchten für 0,5 Sekunden lila. • Dann leuchtet das linke Auge orange und das rechte gelb für eine Sekunde. • Anschließend soll ein schnelles, weißes Blinken 10-mal wiederholt werden. • Zum Abschluss sollen beide Augen zwei Sekunden grün leuchten.

Übersicht In diesem Kapitel programmierst du Bob zu einer Hitzefrei-Anzeige, mit der du dann feststellen kannst, ob es warm genug ist, um Hitzefrei zu bekommen.

LERNZIEL

In diesem Kapitel lernst du, den Temperatursensor zu programmieren, indem du Bob zu einer Hitzefrei-Anzeige machst. Hierbei lernst du, Werte des Temperatursensors abzufragen und das Ergebnis in einer Variablen zu speichern. Außerdem lernst du einen Operator kennen, mit dem du beispielsweise prüfen kannst, ob mehrere Bedingungen wahr sind: das logische Und.



Aufgabe 4.1

Beginne mit deinem Hitzefrei-Anzeige-Projekt:

a. Öffne ein neues Sketch. Gib ihm einen passenden Namen, indem du es speicherst. Denk wie immer daran, im Verlauf auch Zwischenschritte abzuspeichern.

b. Binde die Bibliothek für Bob ein.

c. Deklariere direkt dahinter eine **Variable**, in der die gemessenen Werte des Temperatursensors gespeichert werden:

```
int sensorwert
```

THEORIE: DER TEMPERATURSENSOR

Damit du die gemessenen Temperaturwerte jederzeit in deinem Programm nutzen kannst, ist es sinnvoll, diese in einer Variablen zu speichern. Wie schon in deinen anderen Projekten kannst du auch in diesem Fall eine `int`eger-Variable nutzen, die ganze Zahlen (1, 2, 3 ...) speichert. Am Programmanfang deklarierst du die Variable, führst sie also ein.

Mit `bob3.getTemperatur()` kannst du dann den Sensorwert der Temperatur abfragen. Der Sensor nimmt Werte in einem Bereich zwischen 0 und 255 wahr. Du kannst dir sicher schon denken, dass dies keine Angabe in Grad Celsius ist. Die Werte müssen also zunächst umgerechnet werden.

Mit Blick auf die Temperaturschwellen für Hitzefrei, interessieren dich Temperaturen zwischen 25°C und 27°C, denn unter 25°C darf es kein Hitzefrei geben und ab 27°C wird Hitzefrei empfohlen. Zwischen 25°C und 27°C kann die Schulleitung entscheiden.

25°C entsprechen dem Sensorwert 126.

27°C entsprechen dem Sensorwert 128.

Aufgabe 4.2

Deine Hitzefrei-Anzeige soll drei Bereiche unterscheiden und diese über die Farbe der Augen anzeigen. Du benötigst im loop-Teil somit drei **if-Abfragen**. Starte mit der Programmierung für den ersten Temperaturbereich, also wenn die Temperatur unter 25°C ist und es somit kein Hitzefrei gibt.

a. Speichere zuerst innerhalb der **loop-Funktion** die gemessene Temperatur in der Variablen: `sensorwert = bob3.getTemperature ();`

b. Schreibe dann die **erste if-Abfrage** für den Temperaturbereich unter 25°C mit der Bedingung: `sensorwert<126`

c. Ergänze die if-Abfrage so, dass für diesen Temperaturbereich die Augen des Bobs **grün** leuchten.

d. Teste das Programm. Sollte es an deinem aktuellen Aufenthaltsort wärmer als 25°C sein, suche einen Ort auf, für den dies nicht gilt (Keller, Schlafzimmer, Kühlschrank etc.).

Hast du an einem Ort, an dem die Temperatur unter 25°C liegt, dein Programm getestet? Leuchten die Augen in diesem Fall grün? Falls ja, dann ist das super, und du kannst direkt weiterarbeiten. Solltest du noch Probleme haben, kann das vielleicht hieran liegen:

- Ist dein Bob nicht mehr am PC angeschlossen, sondern läuft nur über die Batterie? Dann funktioniert der Temperatursensor möglicherweise nicht zuverlässig. Das liegt nicht an deinem Programm, sondern an der Konstruktion des Roboters. Teste dein Programm daher am besten, wenn Bob per Kabel mit dem PC oder Laptop verbunden ist.
- Berührst du deinen Bob in der Nähe der Batterie oder des Temperatursensors? Dann nimmt dieser möglicherweise deine Körpertemperatur wahr.
- Wenn die Augen zwischen zwei Farben flackern, dann liegt die Temperatur möglicherweise in einem Grenzbereich.

Im Anhang findest du eine Musterlösung des Programms.

Aufgabe 4.3

Programmiere nun die if-Abfrage für den Temperaturbereich zwischen 25°C und 27°C, in dem Hitzefrei möglich, aber nicht zwingend ist. In diesem Fall sollen die Augen gelb leuchten. Teste dein Programm.

Achtung: Da hier zwei Bedingungen miteinander verknüpft werden, brauchst du ein **logisches Und**. Achte außerdem darauf, dass der Temperaturbereich die Temperaturen 25°C, 26°C und 27°C umfassen soll. Du benötigst also die Operatoren <= und >=.

THEORIE: DAS LOGISCHE UND

Mit einem logischen Und kann man zwei Bedingungen verknüpfen. Die Bedingung als Ganze ist dann nur erfüllt, wenn beide Teil-Bedingungen erfüllt sind. Ein logisches Und wird im Code mit `&&` angegeben.

Aufgabe 4.4

Programmiere nun die letzte if-Abfrage für den Temperaturbereich über 27°C, in dem Hitzefrei empfohlen wird. In diesem Fall sollen die Augen rot leuchten. Teste dann alle drei Temperaturbereiche.

Möglicherweise musst du, um die unterschiedlichen Temperaturbereiche testen zu können, ein wenig kreativ werden. So ist es im Badezimmer häufig wärmer als im Schlafzimmer. Im Kühlschrank ist es kalt, direkt neben der Heizung oder in der Sonne warm.

Funktioniert deine Hitzefrei-Anzeige? Dann können die heißen Temperaturen ja kommen. Aber nicht vergessen: Die endgültige Entscheidung, ob es Hitzefrei gibt, liegt nicht bei Bob, sondern bei der Schulleitung 😉. Im Anhang findest du wie immer eine Musterlösung, die du zu Rate ziehen kannst, wenn noch nicht alles funktioniert, wie es soll.

Test

Wenn du die Hitzefrei-Anzeige erfolgreich programmiert hast, bist du bereit für den nächsten Kapiteltest.

Aufgabe T4: Wahr oder falsch?

a. Die Bezeichnung „Variable“ verdeutlicht die Bedeutung dieses Programmierwerkzeugs, denn die gespeicherten Werte sind veränderbar.

☐ wahr ☐ falsch

b. Mit `int` gibt man eine Variable vom Typ Integer an.

☐ wahr ☐ falsch

c. Die Integer-Variable benutzt man für Kommazahlen.

☐ wahr ☐ falsch

d. Der Befehl, um den Sensorwert der Temperatur abzufragen lautet: `bob3.getTemperatur{}`

☐ wahr ☐ falsch

e. Wenn zwei Bedingungen erfüllt werden müssen, damit ein bestimmter Fall eintritt, benutzt man das logische Und.

☐ wahr ☐ falsch

f. In der Programmierung wird das logische Und so angegeben: `<=`

☐ wahr ☐ falsch

Übersicht

Jetzt wird es knifflig. Du wirst das *Merk's-dir*-Spiel programmieren. Dabei wirst du feststellen, dass diese Programmieraufgabe deutlich komplexer und die Anleitungen offener sind. Verzweifle nicht, wenn dein Code nicht direkt funktioniert. Zur Arbeit von Informatikerinnen gehört, dass man ab und an ein wenig experimentieren und tüfteln muss. Wenn du bei einem Schritt gar nicht weiter kommst, kann dir die Musterlösung helfen. Vermeide jedoch, einfach nur den Code abzuschreiben. Schau dir stattdessen gezielt die Lösung der Stelle an, die dir Schwierigkeiten bereitet.

Nun aber zum eigentlichen Spiel. Dieses soll nach den folgenden **Regeln** funktionieren:

- Zu Beginn soll zunächst das linke, dann das rechte Auge jeweils 1- bis 10-mal aufleuchten.
- Die Anzahl, in der das jeweilige Auge aufgeleuchtet hat, muss der Spieler sich merken.
- Nach der Darbietung soll der Spieler nämlich über die beiden Arme eingeben, wie oft die beiden Augen jeweils geblinkt haben. Für Auge 1 wird Arm 1, für Auge 2 Arm 2 gedrückt.
- Wurde für beide Augen jeweils richtig gedrückt, leuchten Bobs Augen grün und das Spiel beginnt von vorn.

In diesem Kapitel programmierst du deinen Bob also zu einem Merkspiel.

LERNZIEL

In diesem Kapitel wirst du dein Wissen über Variablen vertiefen. Dazu wirst du zunächst unterschiedliche Datentypen kennenlernen. Dann wirst du Variablen unterschiedlichen Datentyps verwenden, um das Spiel zu programmieren. Außerdem lernst du, wie man in Arduino eine Methode für den Neustart und eine Zufallsfunktion verwendet.

THEORIE: DATENTYPEN

Variablen können unterschiedliche Datentypen aufweisen. Den Datentyp Integer für ganze Zahlen kennst du bereits. Im Folgenden findest du eine Übersicht über ein paar weitere nützliche Datentypen, ihre Bedeutung sowie ihre Schreibung im Code.

Typ	Bedeutung	Code
Integer	Ganze Zahlen (0, 1, 2, ...)	`int deineZahl = 1;`
Float	Reelle Zahlen mit einfacher Genauigkeit, d. h. auch gebrochene Zahlen (3/4, 0,75, 2,3 ...)	`float deinWert = 0.5;`

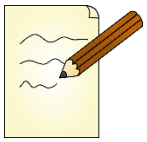
Charakter	alphanumerische Zeichen, also Buchstaben, Zahlen und Sonderzeichen	`char anfangsbuchstabe = 'a';`
String	Zeichenkette, bestehend aus Buchstaben, Zahlen, Sonderzeichen, Steuerzeichen	string stringEins = "Hallo!";
Boolean	Wahrheitswert (wahr/true oder falsch/false)	`bool spielaktivierung = true;`

Allgemein werden Variablen also immer folgendermaßen eingeführt, bzw. **deklariert**:

```
typ variablenName;
```

Der aktuelle **Wert** der Variablen wird dann nach dem folgenden Schema angegeben:

```
variablenName = wert;
```



Aufgabe 5.1

Bestimme den Datentyp.

a. _____ `abstand;`
`abstand = 5.5;`

b. _____ `maennlich;`
`maennlich = true;`

c. _____ `anfangsbuchstabe;`
`anfangsbuchstabe = 'f';`

d. _____ `klasse;`
`klasse = "10c";`

e. _____ `zielzustand;`
`zielzustand = false;`

Aufgabe 5.2

Beginne mit der **Programmierung** des *Merk's-dir*-Spiels, indem du **Startvariablen** initialisierst. Du brauchst folgende Variablen:

- Du bespielst beide Seiten des Bobs und wirst später u. a. eine Zufallszahl zwischen 1 und 10 generieren, die angibt, wie oft das linke und das rechte Auge jeweils aufleuchten. Deklariere daher für die beiden Seiten `links` und `rechts` eine Variable vom Typ **Integer** und setze sie auf **0**.
- Später wirst du zwischen **zwei Spielmodi** unterscheiden müssen.
  - **Präsentationsmodus**: Hier blinken die Augen des Bobs.
  - **Spielmodus**: Hier reagiert der Spieler auf das Blinken, indem er eine Eingabe über die Arme vornimmt.

  Welcher Variablentyp bietet sich also zur Unterscheidung dieser zwei Zustände an?

Deklariere die Variable für den Spielmodus.

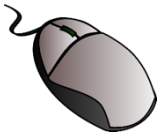
- Außerdem benötigst du zwei **temporäre Variablen**, um später zu überprüfen, ob die Eingabe über die Arme für beide Augen richtig erfolgt ist. Nur wenn in einem Durchgang, also temporär, diese Zustände erfüllt (`true`) sind, ist das Spiel gewonnen und kann neugestartet werden. Lies den folgenden Theorietext und deklariere dann die temporären Variablen, die du zunächst auf `false` setzt.

THEORIE: TEMPORÄRE VARIABLEN

Temporäre Variablen haben – wie ihr Name schon sagt – eine kurze Lebensdauer. Man braucht sie nur eine gewisse Zeit lang. Im konkreten Fall brauchst du sie später nur, um zu überprüfen, ob in einem Durchgang deines Spiels die Eingaben links und rechts abgeschlossen sind.

Eine temporäre Variable im Rahmen der Eingabe über den linken Arm (Arm 1) könnte z. B. `tempL`, eine für den rechten Arm `tempR` heißen.

Zu Beginn kannst du diese beiden temporären Variablen auf `false` setzen. Wenn die Eingabe jeweils abgeschlossen ist, kannst du sie dann auf `true` setzen.



Aufgabe 5.3

Widme dich jetzt dem **Programmstart**, also dem `setup`-Teil.

- Damit der Spieler weiß, dass es losgeht, sollen beim Programmstart die Augen kurz weiß aufblinken.
- Damit die Augen im Präsentationsmodus nicht immer gleich oft aufblinken, musst du eine **Zufallsfunktion** nutzen. Lies zunächst den folgenden Theorietext zur Zufallsfunktion. Nutze dann dein neues theoretische Wissen, um den `setup`-Teil zu programmieren, in dem in den Variablen `links` und `rechts` ein zufälliger Wert zwischen 1 und 10 abgespeichert werden soll.

THEORIE: DIE ZUFALLSFUNKTION

Um Zufallszahlen zu generieren, kannst du eine Zufallsfunktion nutzen. Der Code dieser Zufallsfunktion sieht so aus:

```
random(minimalwert,maximalwert);
```

Möchtest du einer Variablen einen Zufallswert zuordnen, schreibst du Folgendes:

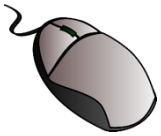
```
variablenname = random(minimalwert,maximalwert);
```

Wenn du also beispielsweise wie in **Aufgabe 5.3** der Variablen `links` eine Zufallszahl zwischen 1 und 10 zuweisen möchtest, dann sieht dein Code so aus:

```
links = random(1,10);
```

Damit nach einem Spieldurchlauf der **Algorithmus komplett neugestartet** wird und auch die Zufallszahl neu generiert wird, solltest du **vor den** `random`**-Befehl**, in dem du Minimal- und Maximalwert festlegst, noch folgenden Befehl verwenden:

```
randomSeed(analogRead(0));
```



Aufgabe 5.4

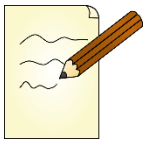
Programmiere nun, was passieren soll, wenn du eine Spielrunde beendet hast und das Spiel neugestartet wird. Dieser Programmteil für den **Neustart** ist genauso strukturiert wie der `setup`- und der `loop`-Teil, die du bereits kennst:

```
void restart () {
// Befehle, die bei Neustart ausgeführt werden sollen
}
```

Nach dem Neustart soll Folgendes passieren:

- Generierung einer (neuen) Zufallszahl zwischen 1 und 10, die in der Variablen `links` gespeichert wird.
- Generierung einer (neuen) Zufallszahl zwischen 1 und 10, die in der Variablen `rechts` gespeichert wird.
- Kurzes Aufblinken der Augen in Weiß.

Übertrage dies als Code in dein Programm.



Aufgabe 5.5

Als Nächstes widmest du dich dem `loop`. Im `loop` startest du mit der Programmierung des ersten Spielmodus. In diesem Modus wird die zufällige LED-Reihenfolge präsentiert. Betrachte den Code-Ausschnitt, der für das linke Auge (EYE 1) zuständig ist. Notiere in den Kästen, was der jeweilige Code-Ausschnitt bewirkt.

```
void loop() {

  if (spielmodus == false){

    for(int i=links;i>0; i-=1){

      bob3.setLed(EYE_1, BLUE);
      delay(200);
      bob3.setLed(EYE_1, OFF);
      delay(200);
    }
```

Aufgabe 5.4

a. Übertrage den Code-Ausschnitt für das linke Auge in dein Programm. In welcher Farbe du das aufblinken lassen möchtest, ist natürlich dir überlassen.

b. Schreibe dann den Code für das rechte Auge. **Achtung**: Hier solltest du den Buchstaben für die **Zählvariable** in der `for`-Schleife ändern. Statt `i` kannst du z. B `j` nehmen.

Aufgabe 5.6

Programmiere jetzt den zweiten Spielmodus, also den Teil des Spiels, in dem der Spieler auf die LED-Reihenfolge **reagiert**. Hier noch ein paar **Tipps**:

- Du musst zunächst den **Spielmodus** ändern.
- Nutze den `else`-Teil deiner Bedingung für die **Reaktion des Spielers**.
- Die **`while`-Schleife** könnte dir hier nützlich sein.
- In eine **`while`-Schleife** können **`if`-Abfragen** integriert werden:

  Solange die Variable ungleich 0 ist, soll der Wert dieser Variablen um 1 reduziert werden, **wenn** der Arm gedrückt wird.

  Für `links` sieht das als Code so aus:
  ```
  while (links!=0) {
    if(bob3.getArm(1)!=0) {
      links-=1;
    }
  ```
  Um zu überprüfen, ob Bob deine Eingabe über den Arm auch wirklich registriert hat, ist es sinnvoll, die Eingabe über kurzes Aufleuchten der Augen anzeigen zu lassen. Bob könnte als Rückmeldung z. B. einmal gelb aufblinken, wenn du seinen Arm gedrückt hast.
- Du brauchst außerdem noch **zwei `if`-Abfragen**, um zu überprüfen, ob die **Eingaben** über die beiden Arme **vollständig** erfolgt sind. Hier kommen jetzt auch die **temporären Variablen** ins Spiel. Diese setzt du, wenn die Bedingung, dass die Eingabe vollständig ist, erfüllt ist, auf `true`. Das sieht für `links` als Code so aus:
  ```
  if (links<=0) {
  tempL=true
  }
  ```
  Auch hier kannst du wieder einen Befehl ergänzen, um dir anzeigen zu lassen, wenn die Bedingung erfüllt ist.
- Eine letzte **`if`-Abfrage** benötigst du, um abzufragen, ob sowohl `links` als auch `rechts` die Eingaben vollständig erfolgt sind. Du brauchst also sowohl die **temporären Variablen** als auch das **logische Und** 😉: Wenn sowohl links als auch rechts die Eingaben vollständig erfolgt sind, dann soll der Startzustand (der Variablen) wiederhergestellt und das Spiel neugestartet (`restart`) werden.

Das war schon ganz schön knifflig, oder? Sollte es an manchen Stellen noch haken, ist das kein Grund zu verzweifeln. Einen funktionierenden Code zu erhalten ist häufig mit Tüftelei verbunden. Suche zunächst noch einmal selbst nach einer Lösung, indem du dein Programm Zeile für Zeile durchgehst. Betrachte besonders die Zeilen, in denen dir die Arduino-Software Fehler anzeigt. Wenn du weitere Hilfe benötigst, kannst du außerdem die Musterlösung zu Rate ziehen. Sollte dein Spiel so

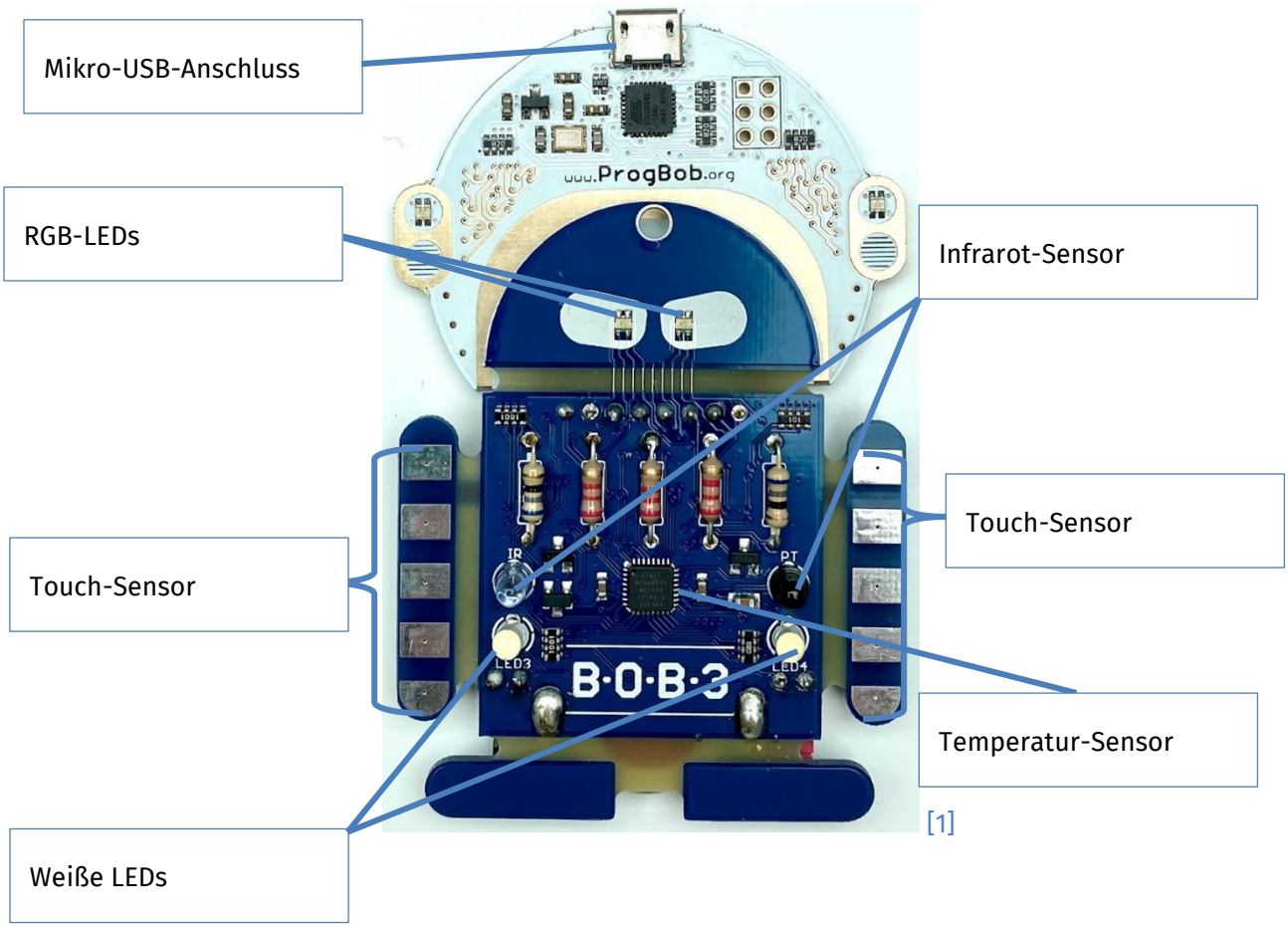
funktionieren, wie in den Regeln in der Übersicht beschrieben, dann: Herzlichen Glückwunsch! Damit hast du nun die Grundlagen des Spiels erstellt. Du bist nun ein echter Profi und am Ende dieses Leitprogramms angekommen, das dir hoffentlich Spaß gemacht hat!

Du kannst dein Spiel, wenn du magst, noch erweitern. Beispielsweise könntest du programmieren, dass zu häufiges Drücken eines Arms als Fehler gewertet wird. Oder du könntest die Reihenfolge, in der die beiden Seiten blinken, im Präsentationsmodus variieren lassen. Bestimmt hast du auch noch weitere Ideen.

Falls du weitere Programmier-Ideen für deinen Bob hast, würden wir uns freuen, wenn du uns diese schreibst. Gerne kannst du auch direkt deinen Code mitsenden. Auch über Feedback zu diesem Leitprogramm würden wir uns sehr freuen. Nutze dazu die go4IT!-Mailadresse: go4it@informatik.rwth-aachen.de

KAPITEL 1: WIE GING DAS NOCHMAL?

Aufgabe 1.1



[1]

Aufgabe 1.2

Bauteil	Funktion
Infrarot-Sensor	Mit diesem Sensor, der aus einem Sender und einem Empfänger besteht, kann der Roboter erkennen, ob ein Hindernis vor ihm ist und wie weit dieses entfernt ist.
Touch-Sensor	Mit Hilfe der Sensorfelder kann der Roboter auf Berührung reagieren.
Temperatur-Sensor	Versteckt im Mikro-Prozessor („Roboter-Hirn“) kann dieser Sensor die Temperatur messen.
RGB-LEDs	Diese LEDs können in verschiedenen Farben leuchten.

Weiße LEDs	Diese LEDs geben weißes Licht ab.
Mikro-USB-Anschluss	Hier kann das Kabel angeschlossen werden, das deinen Bob mit dem Computer verbindet.

Aufgabe 1.3 – Befehlsübersicht

Statt einer Musterlösung folgt eine Befehlsübersicht. Sie beinhaltet alle Befehle, die du im Workshop schon gelernt hast. Du kannst sie nicht nur zur Kontrolle von Aufgabe 1.3, sondern auch zum Nachschlagen verwenden.

Das Grundgerüst

`#include <BOB3.h>`	Einbindung der Befehlsbibliothek für B-O-B-3
`void setup() {` `}`	Der {Befehl} wird einmal zu Programmbeginn ausgeführt.
`void loop() {` `}`	Das Hauptprogramm, das immer wieder wiederholt wird.

Allgemein

`int wert;`	Deklaration der Variable wert
`delay(500);`	Pause; hier: 500 ms (1 s = 1000 ms)

`if (Bedingung)`
`{…}`
`else {…}`
	Wenn die (Bedingung) erfüllt ist,
	dann wird der {Befehl X} ausgeführt,
	sonst wird der {Befehl Y} ausgeführt.

`while(Bedingung) {…}`	Solange die (Bedingung) erfüllt ist, wird der {Befehl} ausgeführt.

LEDs

`bob3.setLed(EYE_1, GREEN);`	Einschalten von LED-Auge 1 in Grün

`bob3.setWhiteLeds(ON,ON);`	Einschalten der beiden weißen LEDs

Sensoren

`bob3.getIRSensor()`	Abfrage des Infrarot-Sensorwertes
`bob3.getTemperature()`	Abfrage des Temperatur-Sensorwertes
`bob3.getArm(1)`	Abfrage des Touch-Sensorwertes, hier von Arm 1

Aufgabe 1.4

Kims erstes Programm

```
#include BOB3.h
void setup() {
  bob3.setLeds(ON,ON)
}
```

```
#include <BOB3.h>
void setup() {
  bob3.setWhiteLeds(ON,ON);
}
```

Kims zweites Programm

```
void setup() {
  bob3.setWhiteLeds(ON,ON);

void loop(){
  bob3.setLed(EYE_1, GRIN);
  bob3.setLed(EYE_2, GRIN);
}
```

```
#include <BOB3.h>
void setup() {
  bob3.setWhiteLeds(ON,ON);
}
void loop(){
  bob3.setLed(EYE_1, GREEN);
  bob3.setLed(EYE_2, GREEN);
}
```

Kims drittes Programm

```
#include <BOB3.h>
void setup() {
{
void loop() {
  bob3.setWhiteLeds(ON,ON);
  bob3.setLeds(OFF,OFF);
}
```

```
#include <BOB3.h>
void setup() {
{
void loop() {
  bob3.setWhiteLeds(ON,ON);
  delay(500);
  bob3.setLeds(OFF,OFF);
}
```

Ohne die Pause (`delay`) kann das menschliche Auge den Unterschied zwischen den Zuständen An und Aus nicht so schnell erfassen. Manche LEDs können den Unterschied der Zustände ohne Pause auch gar nicht anzeigen.

Aufgabe T1

In dem Workshop haben wir den kleinen Roboter Bob kennengelernt, ihn zusammengelötet und programmiert. Bob hat einiges drauf. So verfügt er über mehrere Lampen. Da es sich bei den Lampen in den Augen um RGB-LEDs handelt, könne sie in allen Farben leuchten. Die LEDs am Körper dagegen leuchten nur in Weiß.

Am Roboter-Körper befinden sich zwei weitere LEDS – die eine ist hell, die andere dunkel. Zusammen bilden sie den Infrarot-Sensor, mit dem Bob nah und fern unterscheiden kann. Daneben verfügt Bob über weitere Sensoren. So befindet sich im Mikro-Prozessor, dem „Roboter-Hirn“, das den Programmcode ausführt, der Temperatursensor. In den Armen sind Touch-Sensoren verbaut. Mit Sensoren lassen sich spannende Programme verwirklichen, denn mit ihnen ist es dem Roboter überhaupt erst möglich, auf seine Umwelt zu reagieren. Jedes Programm hat ein Grundgerüst. Neben dem Setup- und dem Hauptprogramm gehört der Befehl, der die Bibliothek für Bob einbindet, zum Grundgerüst jedes Programms; er lautet `#include <BOB3.h>` und muss am Anfang jedes Programms stehen. Das Setup-Programm wird einmal ausgeführt beim Programmstart. Das Hauptprogramm wird immer wieder wiederholt. Ein wichtiges Konzept in der Programmierung sind bedingte Anweisungen. Bei der if-else-Anweisung wird ein Programmabschnitt nur ausgeführt, wenn eine bestimmte Bedingung erfüllt ist. Bei der while-Schleife wird eine Anweisung solange wiederholt, wie eine bestimmte Bedingung erfüllt ist. Das klingt vielleicht kompliziert, macht aber wirklich Spaß.

Aufgabe 2.1 und 2.2

```
#include <BOB3.h>
int sensorwert;

void setup() {

}

void loop() {
  int sensorwert = bob3.getIRSensor();
  if (sensorwert > 5)
  {
    bob3.setLed(EYE_1, ORANGE);
    bob3.setLed(EYE_2, ORANGE);
  }
  if (sensorwert > 10)
  {
     bob3.setWhiteLeds(ON, ON);
     delay (100);
     bob3.setWhiteLeds(OFF, OFF);
     delay (100);
  }
  else
  {
   bob3.setLed(EYE_1, OFF);
   bob3.setLed(EYE_2, OFF);
   bob3.setWhiteLeds(OFF, OFF);
  }
}
```

Deklarierung der Variablen

Abfrage des aktuellen IR-Sensorwertes, wird in der Variablen gespeichert

Zwei if-Abfragen für die Reaktionen:

1.) Komet ist noch nicht ganz nah → Augen orange
2.) Komet ist ganz nah → zusätzlich blinken die weißen LEDs

else-Teil: alle LEDs aus

Aufgabe T2.1: Wahr oder falsch?

a. Variablen sind Speicher für Daten.

☒ wahr ☐ falsch

b. Wenn man Variablen einführt, nennt man das in der Informatik „deklarieren“.

☒ wahr ☐ falsch

c. Um den Abstand zu einem Objekt, wie beispielsweise einem Kometen, messen zu können, nutzt der B-O-B-3 den Temperatursensor.

☐ wahr ☒ falsch

Der B-O-B-3 nutzt hierzu den IR-Sensor.

Aufgabe T2.2: Welche Art von Licht nutzt der B-O-B-3, um den Abstand zu einem Objekt zu messen?

Der B-O-B-3 nutzt Infrarot-Licht.

Aufgabe 3.2

Im folgenden Programmbeispiel werden durch das Berühren des **ersten Sensorfeldes des ersten Armes** die LED-Augen eingeschaltet. Sie blinken dann in **Blau** (AQUAMARINE) und **Gelb**. Wird das Feld nicht berührt, sind die Augen aus.

```
#include <BOB3.h>

void setup () {
}

void loop (){
  if (bob3.getArm(1)==1)
    {
    bob3.setLed(EYE_1,AQUAMARINE);
    bob3.setLed(EYE_2,AQUAMARINE);
    delay (300);
    bob3.setLed(EYE_1,YELLOW);
    bob3.setLed(EYE_2,YELLOW);
    delay (300);
    }else{
      bob3.setLed(EYE_1,OFF);
      bob3.setLed(EYE_2,OFF);
    }
}
```

Aufgabe 3.3

a. Hier gibt es keine Musterlösung, da du die unterschiedlichen Schalter individuell gestalten kannst. Ein Beispiel für den ersten Schalter aus **Aufgabe 2.2**:

Arm 1	
1	Beide LED-Augen blinken relativ schnell (300 ms) in den Farben AQUAMARINE und YELLOW. Dieser Schalter kann z. B. genutzt werden, wenn Party-Musik von schwedischen Interpreten wie ABBA oder Avicii gespielt wird.

b. Das Programm für deine Discolichter setzt sich aus sechs if-else-Anweisungen zusammen – je eine für jedes Sensorfeld. Daher findest du hier beispielhaft die **if-else-Anweisung für Arm 1, Feld 1** inklusive **Markierungen** an den Stellen, die du jeweils anpassen musst.

```
void loop (){
  if (bob3.getArm(1)==1)
    {
    bob3.setLed(EYE_1,AQUAMARINE);
```

Alle if-else-Anweisungen müssen innerhalb des loops stehen.

Achte darauf, jeweils den richtigen Arm, das richtige Sensorfeld und den Vergleich mit zwei Gleichzeichen anzugeben.

Die Augen kannst du separat ansteuern.

```
    bob3.setLed(EYE_2,AQUAMARINE);
    delay (300);
    bob3.setLed(EYE_1,YELLOW);
    bob3.setLed(EYE_2,YELLOW);
    delay (300);
  }else{
    bob3.setLed(EYE_1,OFF);
    bob3.setLed(EYE_2,OFF);
  }
//Es folgen die if-else-Abfragen für die übrigen Schalter nach demselben Schema.
}
```

Die Farben kannst du individuell gestalten.

Wenn du ein Blinken erzeugen möchtest, musst du Pausen einbauen. Die Geschwindigkeit des Blinkens kannst du ebenso anpassen.

Im else-Teil musst du die Augen ausschalten, damit dein Discolicht nur aktiv bleibt, solange du das Sensorfeld gedrückt hältst.

Einen kompletten Beispielcode für diese Aufgabe findest du im Unterordner **Musterlösungen**. Die Datei heißt **Discolicht**.

Aufgabe 3.4

Hier gibt es keine Musterlösung, da du ja nur den Theorie-Teil zur for-Schleife aufmerksam lesen solltest.

Aufgabe 3.5

a. Hier gibt es keine Musterlösung, denn du konntest frei deine Ideen notieren.

b. Die folgende Musterlösung bildet nur einen Ausschnitt des Discolicht-Programms: die if-else-Anweisung für Arm 2, Sensorfeld 3 mit der Lichtchoreographie. Den kompletten Beispielcode findest du im Unterordner **Musterlösungen.** Natürlich kann deine Lichtchoreographie auch anders gestaltet sein. Es soll nur beispielhaft gezeigt werden, wie die for-Schleife in einer Choreographie eingebaut werden kann.

```
if (bob3.getArm(2)==3
  {
  bob3.setLed(EYE_1, BLUE);
  bob3.setLed(EYE_2, BLUE);
  delay (500);
  bob3.setLed(EYE_1, OFF);
  bob3.setLed(EYE_2, OFF);
  delay (200);
  bob3.setLed(EYE_1, BLUE);
  bob3.setLed(EYE_2, BLUE);
  delay (1000);

  for (int i = 0; i < 4; i = i + 1)
  {
  bob3.setLed(EYE_1, WHITE);
  bob3.setLed(EYE_2, WHITE);
  delay (250);
  bob3.setLed(EYE_1, OFF);
  bob3.setLed(EYE_2, OFF);
  delay (250);
```

```
bob3.setLed(EYE_1, GREEN);
bob3.setLed(EYE_2, PURPLE);
delay (250);
bob3.setLed(EYE_1, OFF);
bob3.setLed(EYE_2, OFF);
delay (250);
}
}
else
{
bob3.setLed(EYE_1, OFF);
bob3.setLed(EYE_2, OFF);
}
```

Aufgabe T3

```
#include <BOB3.h>

void setup() {

}

void loop() {

    if (bob3.getArm(2)==3)
    {
    bob3.setLed(EYE_1, VIOLET);
    bob3.setLed(EYE_2, VIOLET);
    delay (500);
    bob3.setLed(EYE_1, ORANGE);
    bob3.setLed(EYE_2, YELLOW);
    delay (1000);

    for (int i = 0; i < 10; i = i + 1)   //for-Schleife für weißes Blinklicht
    {
    bob3.setLed(EYE_1, WHITE);
    bob3.setLed(EYE_2, WHITE);
    delay (250);
    bob3.setLed(EYE_1, OFF);
```

```
    bob3.setLed(EYE_2, OFF);
    delay (250);
    }
    bob3.setLed(EYE_1, GREEN);
    bob3.setLed(EYE_2, GREEN);
    delay (500);
    }
    else
    {
    bob3.setLed(EYE_1, OFF);
    bob3.setLed(EYE_2, OFF);
    }
}
```

Aufgabe 4.1

```
#include <BOB3.h>
int sensorwert;

void setup() {

}

void loop() {
}
```

Aufgabe 4.2

```
#include <BOB3.h>
int sensorwert;

void setup() {

}

void loop() {

  sensorwert = bob3.getTemperature();

  if (sensorwert < 126)
  {
  bob3.setLed(EYE_1, GREEN);
  bob3.setLed(EYE_2, GREEN);
  }
}
```

Aufgabe 4.3

Folgende if-Abfrage muss in der loop-Funktion für den Temperaturbereich zwischen 25°C und 27°C, in dem die Augen gelb leuchten sollen, ergänzt werden:

```
if (sensorwert >= 126 && sensorwert <= 128)
{
bob3.setLed(EYE_1, YELLOW);
bob3.setLed(EYE_2, YELLOW);
}
```

Aufgabe 4.4

Folgende if-Abfrage muss in der loop-Abfrage für den Temperaturbereich über 27°C, in dem die Augen rot leuchten sollen, ergänzt werden:

```
if (sensorwert > 128)
{
bob3.setLed(EYE_1, RED);
bob3.setLed(EYE_2, RED);
}
```

Der komplette Code für die Hitzefrei-Anzeige:

```
#include <BOB3.h>
int sensorwert;

void setup() {

}

void loop() {

  sensorwert = bob3.getTemperature();

  if (sensorwert < 126)
  {
  bob3.setLed(EYE_1, GREEN);
  bob3.setLed(EYE_2, GREEN);
  }
```

unter 25°C → grün → kein Hitzefrei

```
  if (sensorwert >= 126 && sensorwert <= 128)
  {
  bob3.setLed(EYE_1, YELLOW);
  bob3.setLed(EYE_2, YELLOW);
  }
  if (sensorwert > 128)
  {
  bob3.setLed(EYE_1, RED);
  bob3.setLed(EYE_2, RED);
  }
}
```

zwischen 25°C und 27°C (Damit nicht nur genau 26°C für diesen Bereich gelten, werden die Operatoren >= und <= verwendet.)

→ gelb → Hitzefrei möglich

über 27°C → rot

→ Hitzefrei empfohlen

Den else-Teil benötigst du hier nicht, da mit den drei Bereichen alle messbaren Temperaturbereiche abgedeckt sind.

Aufgabe T4:

a. Die Bezeichnung Variable verdeutlicht die Bedeutung dieses Programmierwerkzeugs, denn die gespeicherten Werte sind veränderbar.

☒ wahr ☐ falsch

b. Mit `int` gibt man eine Variable vom Typ Integer an.

☒ wahr ☐ falsch

c. Die Integer-Variable benutzt man für Kommazahlen.

☐ wahr ☒ falsch

Kommazahlen werden mit dem Variablentypen Float oder Double angegeben.

d. Der Befehl, um den Sensorwert der Temperatur abzufragen lautet: `bob3.getTemperatur{}`

☐ wahr ☒ falsch

Statt der geschweiften {} müssen hier runde () Klammern stehen.

e. Wenn zwei Bedingungen erfüllt werden müssen, damit ein bestimmter Fall eintritt, benutzt man das logische Und.

☒ wahr ☐ falsch

f. In der Programmierung wird das logische Und so angegeben: `<=`

☐ wahr ☒ falsch

Das logische Und wird im Code so angegeben: `&&`

Aufgabe 5.1

a.
```
float abstand;
abstand = 5.5;
```

b.
```
bool maennlich;
maennlich = true;
```

c.
```
char anfangsbuchstabe;
anfangsbuchstabe = 'f';
```

d.
```
string klasse;
klasse = "10c";
```

e.
```
bool zielzustand;
zielzustand = false;
```

Aufgabe 5.2

Zur Unterscheidung der beiden Spielzustände bietet sich eine Variable vom Datentyp Boolean an. Auch in der Musterlösung des *Merk's-Dir*-Spiels wirst du diesen Datentyp später finden. Alternativ könnte jedoch auch eine Integer-Variable verwendet werden.

Aufgabe 5.5

```
void loop() {

  if (spielmodus == false){

    for(int i=links;i>0; i--=1){

      bob3.setLed(EYE_1, BLUE);
      delay(200);
      bob3.setLed(EYE_1, OFF);
      delay(200);
    }
```

Bedingung: Präsentationssequenz,

d. h. `spielmodus == false`

if-(else-)Abfrage: Ist die Bedingung erfüllt, wird die for-Schleife ausgeführt.

for/Zählschleife für die Ausgabe über das linke Auge:

- In Blau wird über Auge 1 die in der Variablen `links` gespeicherte, zufällige Zahl an Spielzügen ausgegeben.
- Beim Erreichen des Wertes 0 wird die Schleife abgebrochen.
- Ausgeführte Operation: Zählvariable wird auf Zufallswert gesetzt; Zählvariable wird reduziert.

Musterlösung für das Grundgerüst des *Merk's-Dir*-Spiels

Im Folgenden findest du eine Möglichkeit, wie die Spielidee (Aufgabe 5.2 bis 5.6) in einen Programmcode überführt werden kann. Das bedeutet nicht, dass dein Code falsch ist, wenn er sich an manchen Stellen von der Musterlösung unterscheidet. Wichtig ist, dass das Programm für das Spiel so funktioniert, wie es soll.

```
#include <BOB3.h>

int links = 0;

int rechts = 0;

bool tempL = false;

bool tempR = false;

bool spielmodus = false;
```

Initialisierung der **Startvariablen**:

- Die integer-Variablen für `links` und `rechts` werden auf 0 gesetzt.
- Temporäre boolsche Variablen für die Zustände links (`tempL`) und rechts (`tempR`) werden gesetzt: Wurde `links` bzw. `rechts` fertig gezählt oder nicht?
- Boolsche Variable für den Spielmodus wird auf `false` gesetzt (`false`: Präsentations- vs. `true`: Spielmodus).

```
void restart() {

  links = random(1,10);

  rechts = random(1,10);


  bob. setLed(EYE_1, WHITE);

  bob setLed (EYE_2, WHITE);

  delay (500);

  bob3.setLed(EYE_1, OFF);

  bob3.setLed(EYE_2, OFF);

}
```

Methode zum **Neustarten** des Programms:

- Eine zufällige Anzahl an Spielzügen zwischen 1 und 10 wird erzeugt und in den Variablen `rechts` und `links` gespeichert. In dieser zufälligen Anzahl leuchtet das jeweilige Auge dann auf.
- Als Startsignal blinken beide Augen einmal weiß auf (gehen dann aber wieder nach 500 ms aus).
- **Hinweis**: Diese Methode wird erst relevant, nachdem das Spiel schon einmal gespielt wurde, das Programm also schonmal durchlaufen wurde.

```
void setup() {

  randomSeed(analogRead(0));

  links = random(1,10);

  rechts = random(1,10);
```

Programmstart:

- Damit nicht immer dieselbe Zufallszahl zwischen 1 und 10 generiert wird, wird `randomSeed` verwendet.
- Auch hier blinken beide Augen als Startsignal einmal weiß auf (gehen dann aber wieder nach 500 ms aus).

```
  bob. setLed(EYE_1, WHITE);
  bob setLed (EYE_2, WHITE);
  delay (500);
  bob3.setLed(EYE_1, OFF);
  bob3.setLed(EYE_2, OFF);

void loop() {
  if (spielmodus == false){
    for(int i=links;i>0; i-=1){
      bob3.setLed(EYE_1, BLUE);
      delay(200);
      bob3.setLed(EYE_1, OFF);
      delay(200);
    }
    for(int j=rechts;j>0; j-=1){
      bob3.setLed(EYE_1, RED);
      delay(200);
      bob3.setLed(EYE_1, OFF);
      delay(200);
    }
 }
   spielmodus = true;

  }else{
    while(links!=0) {
      if(bob3.getArm(1)!=0) {
        links-=1;
        bob3.setLed(EYE_1; YELLOW);
        delay (400);
```

Der Loop:

if-else-Abfrage:

- Bedingung: Präsentationsmodus, d. h. `spielmodus == false`
- **Zählschleifen** für die Ausgabe über das linke Auge (EYE_1)
  - In Blau wird über EYE_1 die in der Variablen `links` gespeicherte, zufällige Zahl an Spielzügen ausgegeben.
  - Beim Erreichen des Wertes 0 wird die Schleife abgebrochen.
  - Ausgeführte Operation: Zählvariable wird auf Zufallswert gesetzt; Zählvariable wird reduziert.
- **Zählschleife** für die Ausgabe über das rechte Auge (EYE_2)

Wechsel in den Spielmodus (`spielmodus=true`) → `if`-Teil wird verlassen, Übergang in den `else`-Teil

`while`-Schleife: Der linke Arm muss solange gedrückt werden, bis der Wert 0 erreicht ist, d. h. solange der Wert ungleich Null ist.

Bedingung: Wenn Arm 1 berührt wird, dann wird der Wert der Variablen `links` um 1 reduziert. Damit die Eingabe sichtbar ist, wird das `EYE_1` kurz (400 ms) auf Gelb gesetzt.

```
        bob3.setLed(EYE_1; OFF);
      }
    while(rechts!=0) {
      if(bob3.getArm(2)!=0){
        rechts-=1;
        bob3.setLed(EYE_2; YELLOW);
        delay (400);
        bob3.setLed(EYE_2; OFF);
      }
    }
    if(links<=0){
      bob3.setLed(EYE_1, GREEN);
      delay(200);
      tempL = true
    }
    if(rechts<=0){
      bob3.setLed(EYE_2, GREEN);
      delay(200);
      tempR = true
    }
    if (tempL==true && tempR==true){
      tempL = false
      tempR = false
      spielmodus = false;
      restart();
      }
    }
}
```

`while`-Schleife: Der rechte Arm muss solange gedrückt werden, bis der Wert 0 erreicht ist, d. h. solange der Wert ungleich Null ist.

Bedingung: Wenn Arm 2 berührt wird, dann wird der Wert der Variablen `rechts` um 1 reduziert. Damit die Eingabe sichtbar ist, wird das `EYE_2` kurz (400 ms) auf Gelb gesetzt.

Zwei **`if`-Abfragen**, ob die Eingabe `links` und `rechts` jeweils fertig (<= 0) ist. Ist dies der Fall, wird das Auge der jeweiligen Seite kurz (200 ms) grün und die jeweilige temporäre Variable (`tempL` bzw. `tempR`) auf `true` gesetzt.

Abfragen, ob sowohl die Eingabe `links` als auch `rechts` fertig (`true`) ist. Ist dies der Fall, wird der Startzustand wiederhergestellt und das Spiel neugestartet (`restart`).

ANHANG B: ABBILDUNGSVERZEICHNIS

- **Abb. 1, 3 bis 5, 12** – *Quelle: InfoSphere*
- **Abb. 2, 6 bis 10** – *Quelle: Screenshot der Arduino-Software (https://www.arduino.cc/en/software), GNU Lesser General Public License (https://www.gnu.org/licenses/lgpl-3.0.de.html), abgerufen am 09.05.2022.*
- **Abb. 11** – *Quelle: pxhere (https://pxhere.com/de/photo/1162603), CC0 1.0 Universal Public Domain (https://creativecommons.org/publicdomain/zero/1.0/), abgerufen am: 10.05.2022*
- **Abb. 13** – *Quelle: pxhere.com (https://pxhere.com/de/photo/970908), CC0 1.0 Universal Public Domain (https://creativecommons.org/publicdomain/zero/1.0/), abgerufen am: 09.05.2022*
- **Arbeitssymbole** 🏁, 📖, 🖱️, 📝, 🔒, 📘, – *Quelle: InfoSphere*

ANHANG C: MEDIOTHEK

- *Arduino (Software)*, https://www.arduino.cc/en/software, abgerufen am: 25.5.2022.
- *ProgBob*, https://www.progbob.org/, abgerufen am: 25.5.2022.

ANHANG D: LITERATURANGABEN

- Langer, Andrea (2018): *Mädchen für MINT begeistern – zwei Tage mit B-O-B-3, dem Roboter zum Löten und Programmieren* (Schriftliche Hausarbeit im Rahmen der Ersten Staatsprüfung, dem Landesprüfungsamt für Erste Staatsprüfungen für Lehrämter an Schulen vorgelegt).
- Ministerium für Schule und Bildung des Landes Nordrhein-Westfalen (2021): *Kernlehrplan für die Sekundarstufe I – Klassen 5 und 6 in Nordrhein-Westfalen.* Abgerufen von: https://www.schulentwicklung.nrw.de/lehrplaene/lehrplan/256/si_kl5u6_if_klp_2021_07_01.pdf (07.03.2022).
- Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen (Hrsg.) (2014): *Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen - Informatik.* Abgerufen von: https://www.schulentwicklung.nrw.de/lehrplaene/lehrplan/75/KLP_GOSt_Informatik.pdf (eingesehen: 26.04.2022).