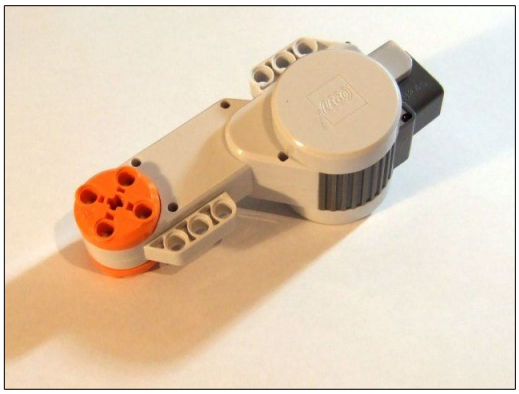
Der NXT Baustein mit Aktoren und Sensoren

Der NXT-Baustein ist ein programmierbarer Kleincomputer mit

- drei (3) **Ausgängen** und
- vier (4) **Eingängen**

Im LEGO Mindstorms NXT Baukasten für Schulen befinden sich

- drei (3) **Motoren** (**Aktoren**)
- und die folgenden **Sensoren**

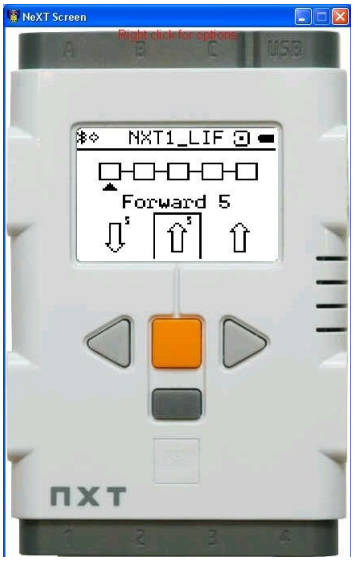


 Berührungssensor (2 x)	 Geräuschsensor	 Lichtsensor	 Ultraschallsensor

Programmierlösungen

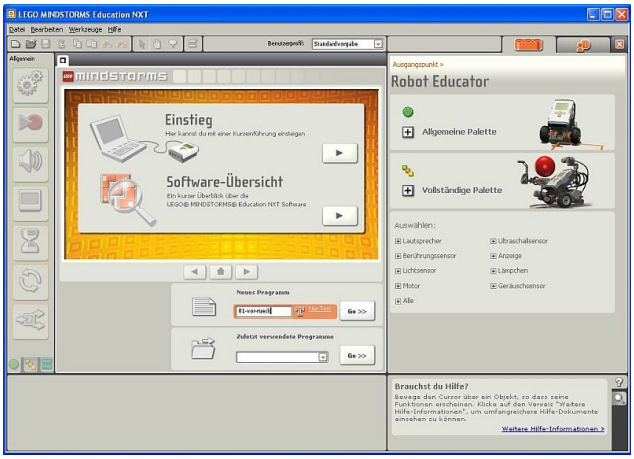
Die NXT-Bausteine können auf verschiedene Arten programmiert werden. Im Schulbetrieb bieten sich folgende besonders an:

...**direkt** am Gerät:



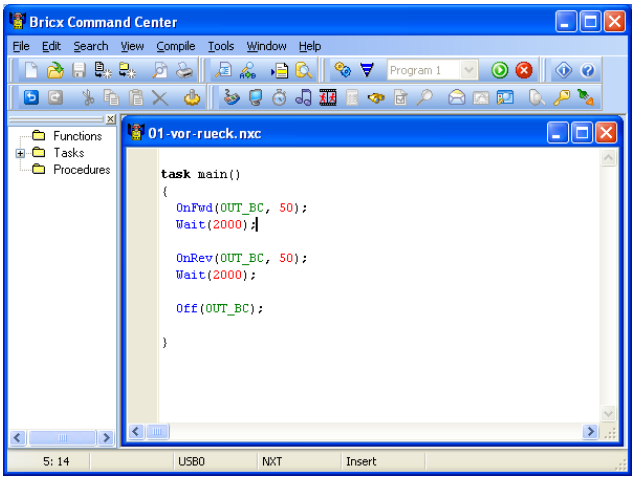
(max. 5 Programmschritte)

...**grafisch** mit der LEGO-Mindstorms-Education-NXT-Software:



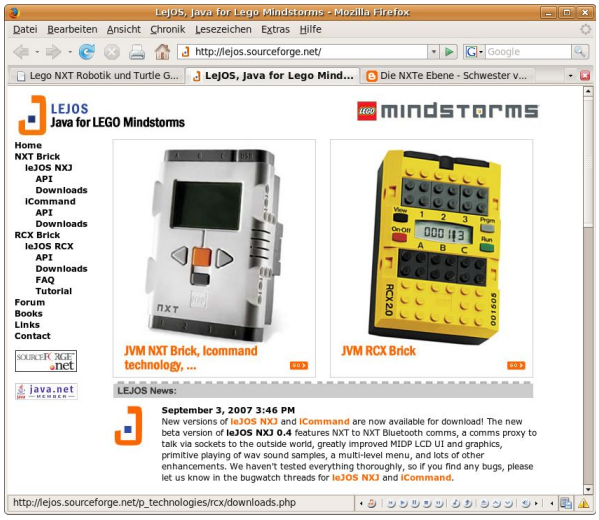
http://nxt-in-der-schule-de/

...**textuell** z.B. mit Bricxcc und NXC:



http://bricxcc.sourceforge.net/

...**objektorientiert** z.B. mit JAVA:



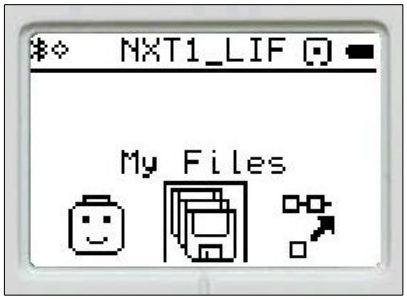
http://lejos.sourceforge.net/

Die direkte und die grafische Programmierung sollten SchülerInnen ab 5. Klasse durchführen können. Für die NXC-Programmierung sollten es mindestens 9.-Klässler sein, JAVA ist eher was für die Sekundarstufe II.

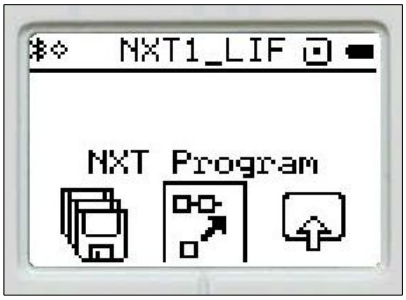
Auf den folgenden Seiten soll eine Programmieraufgabe auf verschiedenen Arten gelöst werden.

Direkte Programmierung am NXT-Baustein

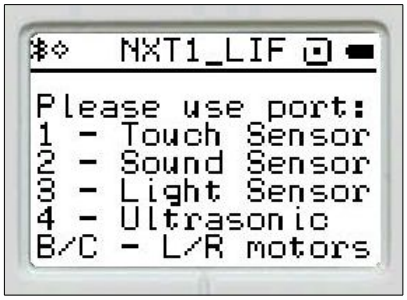
Jede Auswahl muss durch einen Druck auf die große **orange Taste** bestätigt werden.



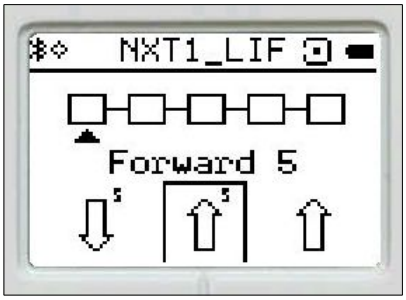
(1) So meldet sich der NXT-Baustein nach dem Einschalten



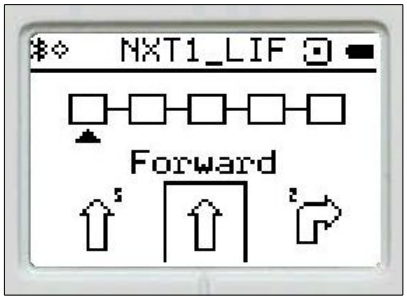
(2) Direktprogrammiermodus (NXT Program) auswählen



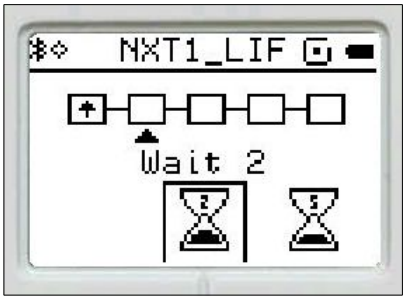
(3) Hier werden die Belegungen für die Eingänge und Ausgänge angezeigt



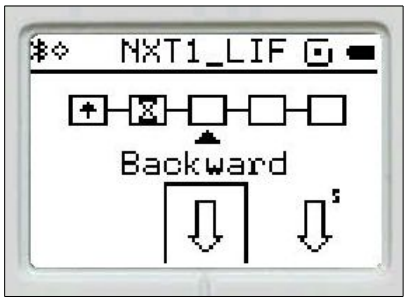
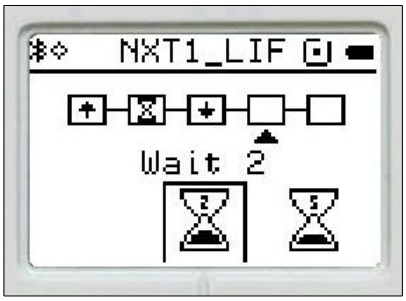
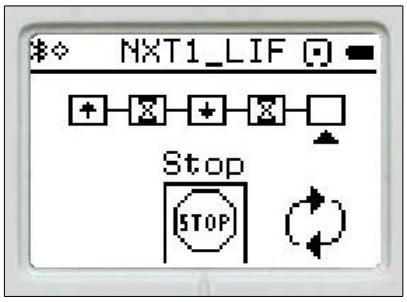
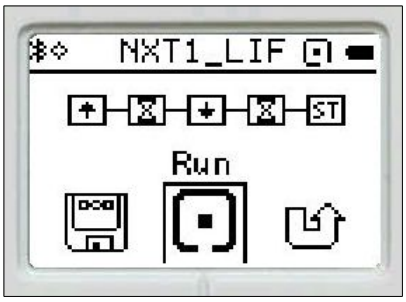
(4) Mit den Pfeiltasten (<-- und -->) kann man sich für 5 Programmplätze jeweils eine Funktion aussuchen



(5) Vorwärts fahren (Forward) auswählen



(6) 2 Sekunden warten (Wait 2) auswählen

	
(7) Rückwarts fahren (Backward) auswählen	(8) 2 Sekunden warten (Wait 2) auswählen
	
(9) Motoren stoppen (Stop) auswählen	(10) Programmstart (Run) auswählen

Voraussetzungen zur weiteren Programmierung

Die NXT-Bausteine können auch mit entsprechender Software mit einem PC programmiert werden.

Drei Stufen durchläuft ein NXT-Programm. Es muss

- zuerst **erstellt**,
- dann **übersetzt** (Fachbegriff: compiliert) und
- zuletzt vom PC auf den NXT-Baustein **übertragen**

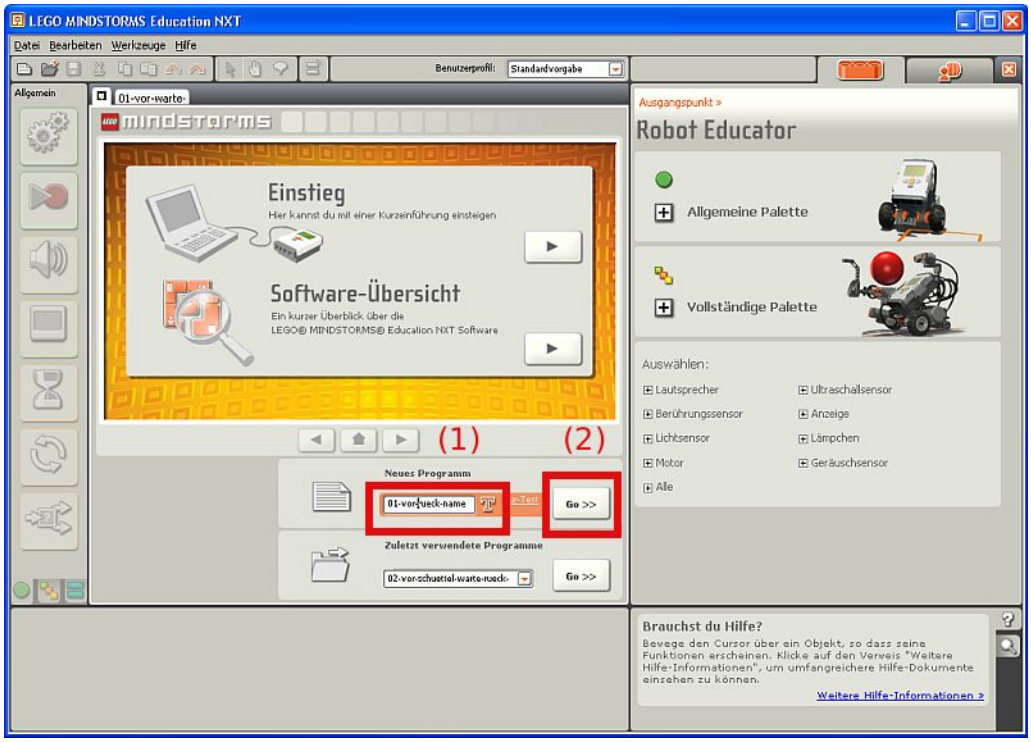
werden.

Zur Übertragung gibt es prinzipiell **zwei Wege**:

- ...über ein **USB-Kabel** (geringer Installationsaufwand, sichere und schnelle Übertragung, NXT muss zur Übertragung angeschlossen sein)...

- ...oder über eine **Bluetooth-Vernetzung** (kabellos, großer Installationsaufwand, tlws. langsame und nicht immer stabile Übertragung)

Grafische Programmierung



Die LEGO-Mindstorms-Education-NXT-Software befindet sich **im Programmmenü** unter ...

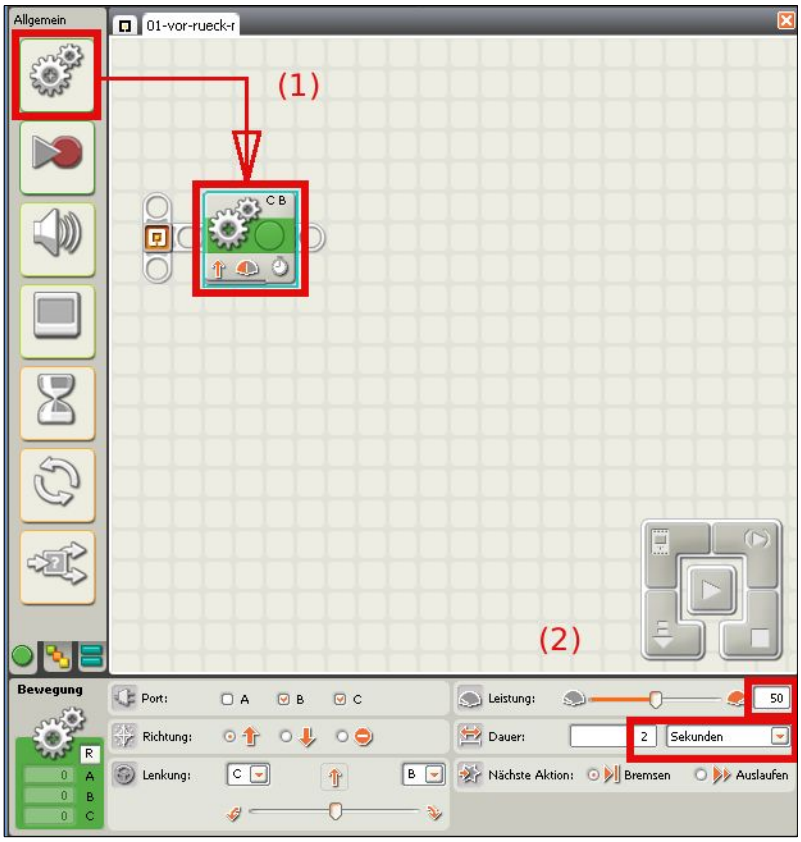
Alle Programme - _____

Starte sie und erstelle ein **neues Programm**, indem du

- einen **Programmnamen (***01-vr-name.rbt)* **eingibst** (1) und
- die **Eingabe** mit einem Klick auf den **Go**-Knopf **bestätigst** (2).

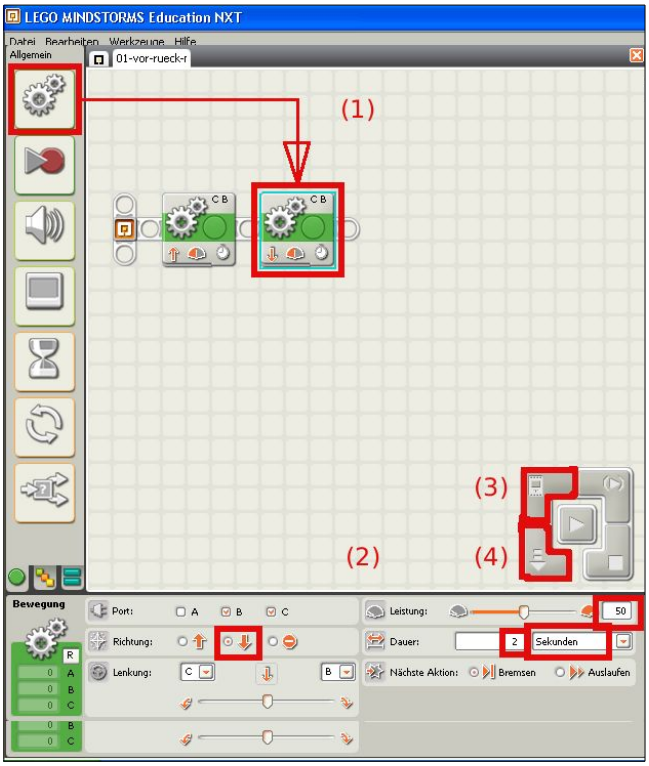
Um eine **Vorwärtsbewegung** zu programmieren, musst du

- aus der **Programmierpalette** (links) einen **Bewegungsblock anklicken** und mit gedrückter linker Maustaste in den **Arbeitsbereich** (rechts) **ziehen** (1)



Jeder **Programmierblock** hat einen eigenen **Konfigurationsbereich** (2). Bestimme hier

- die **Leistung** (50%) und
- die **Dauer** (2 Sekunden)



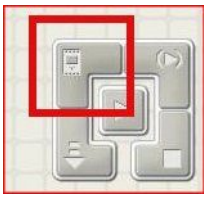
Ergänze das Programm mit einem zweiten Programmierschritt. Wir wollen den Roboter wieder in die Ausgangsstellung zurückfahren lassen. Hierzu

- ziehst du einen weiteren **Bewegungsblock** in den **Arbeitsbereich** (rechts) (1) und
- bestimmst im **Konfigurations-bereich** (unten) wieder
  - die **Leistung** (50%) und
  - die **Dauer** (2 Sekunden) (2)

Den **Kontakt** zum (angeschalteten) **NXT-Baustein** (oben unter 3) stellt man in drei Schritten her:

- **suchen** (1)
- **verbinden** (2)
- Dialog **schließen** (3)

Mit der linken unteren Schaltfläche im Controllerbereich (links und oben unter 4) startet man



- die Übersetzung und
- die Übertragung auf den NXT-Baustein.

Nun kann das erstellte Programm am Gerät gestartet werden.

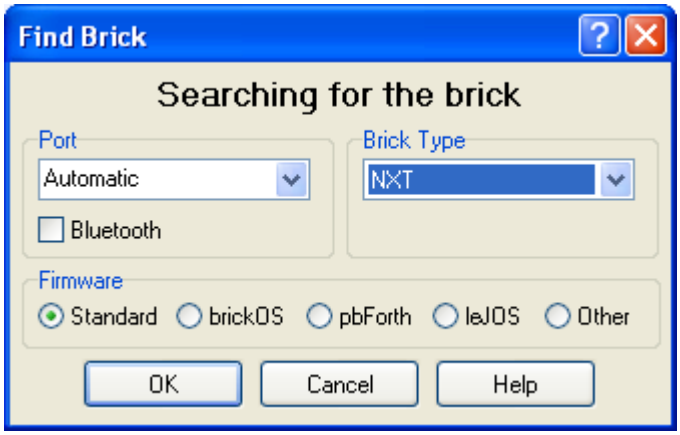
Textuelle Programmierung

Der NXC-Baustein kann mit vielen Programmen und den unterschiedlichsten Programmiersprachen angesteuert und programmiert werden. Wir verwenden in der nächsten Zeit das Programm „BricxCC“ (http://bricxcc.sourceforge.net/) dazu, was die Programmiersprache Not eXactly C (NXC - http://bricxcc.sourceforge.net/nbc/) benutzt. Beide Programme sind Freeware.

BricxCC befindet sich **im Programmmenü** unter ...

Alle Programme - _____

Nach dem **Start** von **BricxCC** zeigt sich folgende Dialogbox:



Hier muss man **auswählen**, wie BricxCC welchen Baustein ansteuert:

- Wenn man den NXT-Baustein über **USB-Kabel** anschließt, dann ist

  - Port: **Automatic** sowie
  - Brick Type: **NXT (WICHTIG:** NXT und **nicht NQC,** sonst geht funktioniert das Übersetzen nicht!)

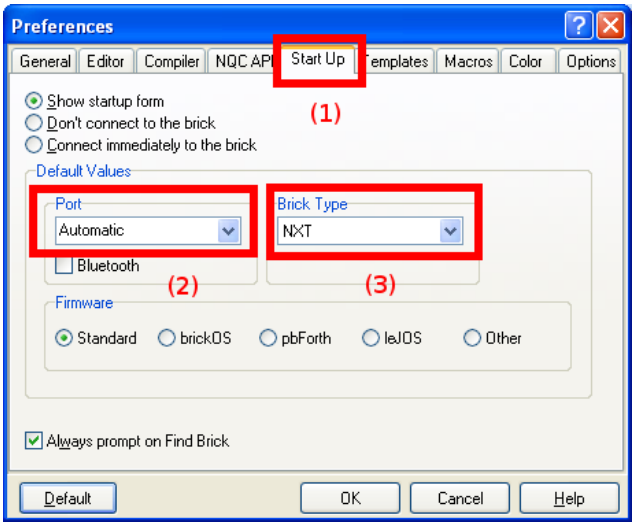
  eine sinnvolle Einstellung. Bei **Bluetooth**-Ansteuerung müsste das entsprechende **Häkchen gesetzt** werden.

- Wenn man den **Baustein nicht** beim Programmstart **angeschlossen** und **angeschaltet** hat, muss man den obigen Dialog über den Menüpunkt **„Tools – Find Brick“** nachträglich aufrufen.

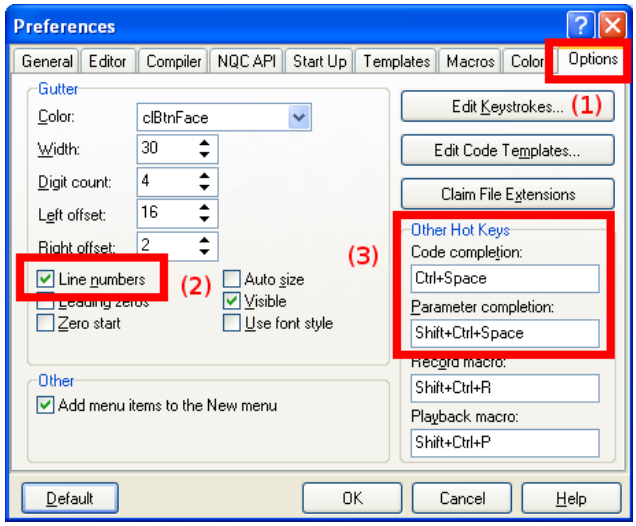
Datum: ***Name:***

Mit einigen Voreinstellungen macht man sich das Leben mit dem Programm BricxCC leichter. Unter „Edit – Preferences“ kann man sie aufrufen.

Damit bei jedem Start von Bricxcc die **Voreinstellung** auf den NXT eingestellt ist, sollte man die entsprechenden Einstellungen im Register „Start-Up“ machen.

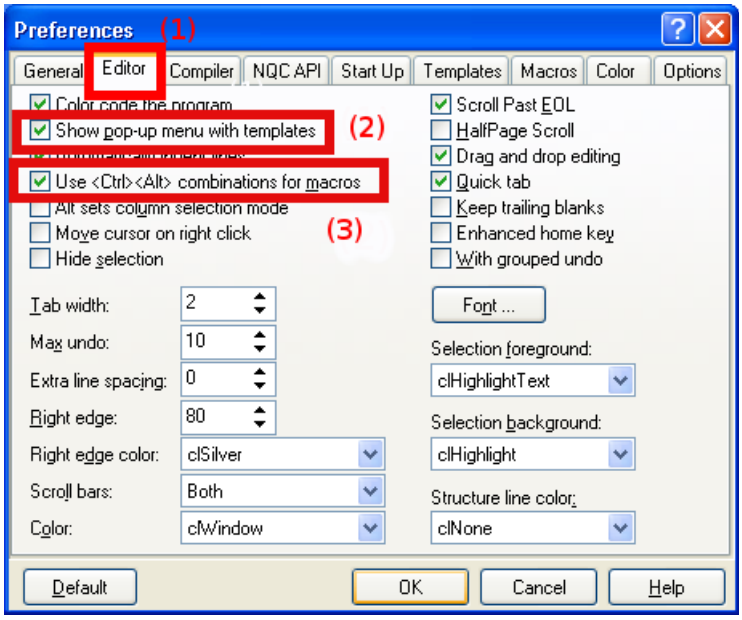


Mit **aktivierten Zeilennummern** (line numbers) wird die Fehlersuche einfacher.



Wichtige Arbeitserleichterung bei Bricxcc ist die sogenannte **„Code-Ergänzung“** (Code completion), die man mit **Strg – Leertaste** aufrufen kann. Man braucht nur den Anfang eines Befehles eingeben und dann die obige Tastenkombination. Danach zeigt sich ein Auswahlmenü der vorhandenen Befehle und Konstanten.

Die **Tastaturkürzel** und das Template-Fenster sind leider nicht in der Voreinstellung von BricxCC aktiviert. Das muss noch nachgeholt werden, indem man unter **„Edit – Preferences – Editor“** die **Häkchen** für „show pop-up menu with templates“ (1) und „USE <CTRL> <ALT> combinations for macro“ (2) **setzt.**



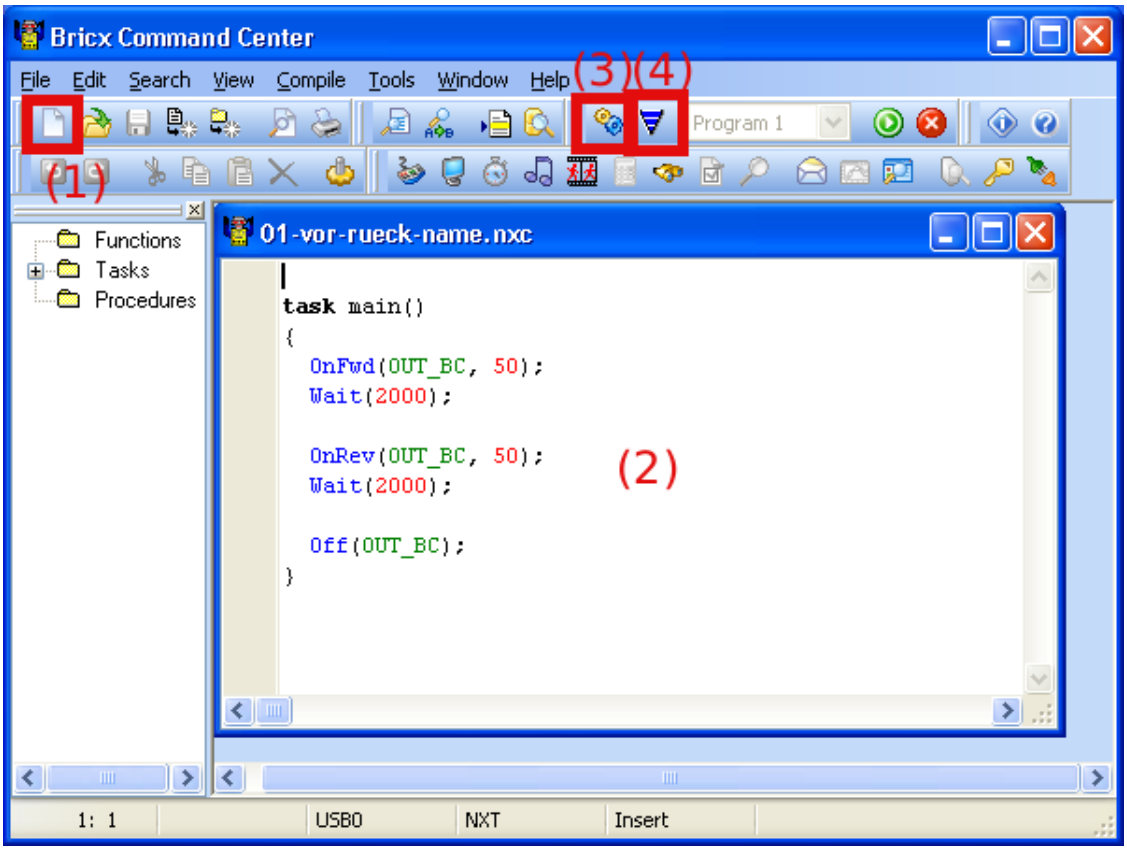
Ein Programm hat unter NXC hat immer folgende Grundstruktur:

```
-
task main()
{
  "Befehle";
}
```

Anstatt der Zeile `"Befehle"` können die gewünschten Befehle eingegeben werden. **WICHTIG:** Jede **Befehlszeile** ist mit einem **Semikolon abzuschließen!** Hilfreich ist hier auch das sogenannte „Syntaxhighlighting“, das farbliche Hervorheben einzelner Programmteile, denn nur korrekt geschriebene Befehle haben die richtige Farbe!

Folgende **Schritte** müssen abgearbeitet werden, um ein neues **Programm** zu **schreiben** und zu übertragen:

1. Über das **„File – New“-Symbol** (1) wird ein **neues Programmfenster** erstellt.
2. Im Programmfenster **schreibt** man das **Programm.**
3. Über den **Knopf (3)** in der Werkzeugleiste (oder auch mit **F5**) **übersetzt (compiliert)** man das **Programm.**
4. Über **Knopf (4)** (oder auch mit **F6**) **überträgt** man das **Programm** auf den LEGO-NXT-Baustein. Dieser muss dazu angeschaltet und angeschlossen sein.



Datum: ***Name:***

Aufgabe 1:

1. **Tippe** das nebenstehende **Programm ab.** Benutze dazu auch die **Tastaturkürzel** bzw. das **Vorlagenfenster.** 2. **Schreibe** neben jede Zeile, **was** der **Befehl** jeweils **macht** (hier auf dem Blatt sowie im Programm). 3. **Beseitige** alle **Fehler** im Programm. 4. **Übersetze** und **übertrage** das **Programm.** 5. **Speichere** es unter ***01-vor-name.nxc.*** 6. **Drucke** das **Programm** aus.	*//* *// **01-vr-name.nxc*** *// J. Stolze - 30.9.07* *//* **task** main() { **OnFwd**(OUT_BC, 50); // _____ **Wait**(2000); // ____ _____ **OnRev**(OUT_BC, 50); // _____ **Wait**(2000); // _____ **Off**(OUT_BC); // _____ }

Bewerte die Arten der Programmierung nach folgenden Kriterien:

	Direkt programmierung	**LEGO-NXT-Software**	**NXC mit BricxCC**
Aufgabe lösbar?			
Hardware-voraussetzungen			
Software-voraussetzungen			
Erlernbarkeit			
Programmierzeit			
Programmfehler sind schnell erkennbar			
Dokumentier-barkeit			

Datum: ***Name:*** 

Aufgabe 2

Unser LEGO-Roboter soll nacheinander

- 2 Sekunden vorwärts fahren
- 2 Sekunden warten und
- wieder 2 Sekunden zurückfahren

Alle für diese Aufgabe benötigten Befehle findest du auf Seite 10 und 12.

Speichere dein Arbeitsergebnis unter `02-v2r-name.nxc`.

Bewerte wieder die Arten der Programmierung nach folgenden Kriterien:

	Direkt programmierung	LEGO-NXT-Software	NXC mit BricxCC
Aufgabe lösbar?			
Hardware-voraussetzungen			
Software-voraussetzungen			
Erlernbarkeit			
Programmierzeit			
Programmfehler sind schnell erkennbar			
Dokumentier-barkeit			

Fazit:

Datum: ***Name:*** 

Weitere Aufgaben:

3.) Dein LEGO-Roboter soll 1s vorwärts fahren, sich um **90° drehen** und wieder 1s vorwärts fahren Speichere dieses Arbeitsergebnis unter `03-90-`*`name`*`.nxc`.

4.) Nun soll er ein **Quadrat fahren**. Benutze dazu eine Schleife. Infos dazu findest du in der Programmhilfe. Speichere dieses Arbeitsergebnis unter `04-quad-`*`name`*`.nxc`.

5.) Dein LEGO-Roboter soll 3 **mal ein Quadrat fahren**. Benutze dazu die Repeat-Schleife (siehe Vorlagenfenster). Speichere dieses Arbeitsergebnis unter `05-3x4-`*`name`*`.nxc.`

6.) Nun soll er eine **Spirale fahren**. Entwickle zuerst eine Vorgehensweise auf Papier und diskutiere sie mit deinen Nachbarn, bevor du dein Programm schreibst. Speichere das Arbeitsergebnis unter `06-spiral-`*`name`*`.nxc`.

7.) Rüste deinen LEGO-Roboter mit dem **Ultraschallsensor** aus. Er soll

- solange geradeaus fahren, bis er direkt vor einem Gegenstand steht,
- dann stoppen und
- einen Ton von sich geben.

Speichere das Arbeitsergebnis unter `07-stop-`*`name`*`.nxc`.

8.) Jetzt brauchen wir den **Geräuschsensor.** Programmiere deinen Roboter so, dass er

- zunächst nach vorne fährt,
- bei einem lauten Geräusch aber immer
  - stoppt,
  - 1 Sekunde rückwärts fährt,
  - sich zufallsgesteuert nach links oder rechts dreht und
  - dann wieder nach vorne fährt und
- wieder diese Prozedur von vorne beginnt (Dauerschleife)

Speichere das Arbeitsergebnis unter `08-laut-`*`name`*`.nxc`.

9.) Lasse deinen LEGO-Roboter mindestens **10 Sekunden** lang **tanzen** und **Musik spielen.** Hier darfst du deiner Kreativität freien Lauf lassen... ;-). Programmiere die Musk und die Bewegung in 2 tasks. Speichere das Arbeitsergebnis unter `09-tanz-`*`name`*`.nxc`.

MERKE: Leider können die **Dateinamen** beim **NXT-Baustein** nur maximal **15 Zeichen** plus **3 Zeichen** für die **Endung** lang sein!

NXC-Anweisungen – Teil 1

task main() { ...Anweisungen... }	Grundgerüst für NXC-Programme
OnFwd(OUT_BC, 50);	Die Motoren B und C laufen mit 50% Kraft vorwärts
OnRev(OUT_C, 50);	Der Motor C läuft mit 50% Kraft rückwärts
Wait(200);	Wartet im Programm 200/1000tel Sekunde
Off(OUT_BC);	Schaltet die Motoren B und C aus
OnFwdReg(OUT_BC,50,OUT_REGMODE_SYNC);	Die Motoren B und C laufen mit 50% Kraft syncron vorwärts
RotateMotor(OUT_B, 50, 360);	Rotiert Motor B mit 50% Kraft 360° vorwärts
RotateMotor(OUT_C, 75, -180);	Rotiert Motor C mit 75% Kraft 180° rückwärts
SetSensorTouch(IN_1);	Schaltet den Eingang 1 zum Berührungssensor
SetSensorSound(IN_2);	Schaltet den Eingang 2 zum Geräuschsensor
SetSensorLight(IN_3);	Schaltet den Eingang 3 zum Lichtsensor
SetSensorLowspeed(IN_4);	Schaltet den Eingang 4 zum Ultraschallsensor
#define FAHREN 50	Definiert FAHREN auf den Wert 50
int licht = 0;	Legt eine Variable mit dem Wert 0 fest
licht = Sensor(IN_3);	Weist der Variablen den Sensorwert an Eingang 3 zu
repeat (4) { ...Anweisungen... }	Zählschleife - die Anweisungen werden hier 4 mal wiederholt
while (true) { ...Anweisungen... }	Dauerschleife – die Anweisungen werden immer wiederholt
while (SensorUS(IN_4) > NEAR) { ...Anweisungen... }	Führt die Anweisungen durch, solange die Bedingung, dass der Wert vom Ultraschallsensor an Eingang 4 größer der Konstanten „NEAR“ erfüllt ist
if (Sensor(IN_1) == 1) { ...Anweisungen... }	führt die Anweisungen durch, wenn die Bedingung, der Wert vom Berührungssensor an Eingang 1 hat den Wert 1 (Taster gedrückt), erfüllt ist
if (licht < GRENZE) { ...JA-Anweisungen... } **else** { ...NEIN-Anweisungen... }	Führt die JA-Anweisungen durch, solange die Bedingung Wert der Variablen „licht“ < Wert von der Konstanten „GRENZE“ erfüllt ist Sonstigenfalls werden die NEIN-Anweisungen ausgeführt

NXC-Anweisungen – Teil 2

`int drehzeit = 0;` `long zeit;`	Die Variablen `drehzeit` bekommt den Wert 0 zugewiesen, `zeit` wird als doppeltgenaue Variable bestimmt.
`fahrzeit = fahrzeit + 10;`	Der Wert für `fahrzeit` wird um 10 erhöht.
`j -= 1;`	Der Wert für `j` wird um 1 erniedrigt.
`wartezeit = (i * j) / 100;`	Auch komplexere Berechnungen mit anderen Grundrechenarten sind möglich.
`PlayTone(440, 500);`	Ein Ton mit einer Frequenz von 440 Hz und einer Dauer von 500/1000 (½) Sekunde wird abgespielt.
`Float(OUT_BC);`	Stoppt die Motoren an den Ausgängen B und C sanft.
`turn_time = Random(400);`	Weist der Variablen turn_time einen zufällig ermittelten Wert zwischen 0 und 400 zu.
`sub display()` `{` `  TextOut(0, 20, "Sensor 3:");` `  NumOut(60, 20, licht);` `  return;` `}` `display();`	Ein Unterprogramm `display` wird definiert und eine Textausgabe auf dem NXT-Display festgelegt: - mit `TextOut` wird einen Zeichenkette, - mit `NumOut` der Inhalt einer Variablen (hier `licht`) ausgegeben. Das Unterprogramm wird aufgerufen.
`task musik()` `{` `  while (true)` `  {` `  PlayTone(440,500); Wait(600);` `  ...` `}`	Definiert eine Aufgabe (task) `musik`. In einer Dauerschleife wird Musik abgespielt.
`Precedes (musik, bewegung); //oder` `start musik;` `start bewegung;`	Ruft zwei Aufgaben (tasks) `musik` und `bewegung` auf, die parallel abgearbeitet werden. Sie müssen vorher mit dem task-Befehle definiert sein.
`Acquire(moveMutex);` `OnFwd(OUT_AC, 75); Wait(1000);` `OnRev(OUT_C, 75); Wait(500);` `Release(moveMutex);`	Innerhalb einer **`task`**-Definition kann mit dem `Acquire`- und dem `Release`-Befehl ein exclusiver Zugriff auf die Motoren geregelt werden.
`PlayFileEx("!Startup.rso", MINVOL, FALSE);` `Wait(2000);`	Spielt die Startmeldung vom NXT-Baustein ab.

Weitere Informationen

- zu NXC:
  - (engl.): http://bricxcc.sourceforge.net/nbc/nxcdoc/index.html
  - (dts.): http://lukas.internet-freaks.net/nxt.php
- zum Vorgänger von NXC (NQC) für die alten LEGO-RCXs:
  - (dts.): http://www.pns-berlin.de/projekte/lego/
  - (engl.): http://bricxcc.sourceforge.net/nqc/

Messungen mit dem Lichtsensor

Für die Programmierung eines Roboters, der einer Linie folgt, brauchen wir für die drei Farben die Helligkeitswerte, die dein Lichtsensor jeweils ausgibt. Arbeite dazu folgende Arbeitsanweisungen ab:

- **Baue** den **Lichtsensor** für den NXT nach den Anweisungen im Handbuch Seite 32 bis 34 z**usammen** (unter http://www.nxt-in-der-schule.de/downloads sind sie auch zu finden).
- **Tippe** das unten stehende **Programm** zur Lichtmessung **ab** und speichere es unter dem Namen `10-licht-name.nxc`.
- Führe das Programm aus und **ermittle** damit die **Sensorwerte** für den **schwarzen**, den **weißen** und den **silbernen** Untergrund.
- **Trage** die **Werte** in die unten stehende Tabelle **ein**.

```
// 10-licht-name.nxc

int licht;

sub display()
{
   TextOut(0, 20, "Sensor 3:");
   NumOut(60, 20, licht);
   return;
}

task main()
{
   SetSensorLight(IN_3);

   while (true)
   {
      licht = Sensor(IN_3);
      display();
   }
}
```

Sensorwerte

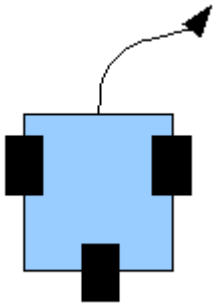
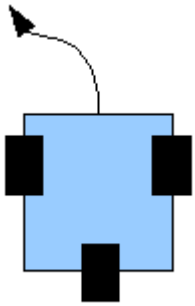
Farbe	schwarz	weiß	silber
Wert			

Merke: Nicht jeder Sensor gibt an gleicher Stelle die gleichen Werte aus. Schreibe dir deshalb hier zur Sicherheit die Bezeichnung von deinem Sensor (bzw. NXT-Bausatz) auf.

Bezeichnung vom Sensor bzw. NXT-Bausatz: _____

Steuerung von Robotern mit zwei Motoren

Damit man gezielt einen Roboter nach links oder nach rechts steuern kann, muss man sich bewusst machen, wie jeweils die beiden Motoren an- bzw. ausgeschaltet werden müssen, damit der Roboter in die gewünschte Richtung fährt.

nach **rechts...**	nach **links...**
	

Roboter haben i.d.R. einen **linken** und einen **rechten Motor**, die jeweils direkt mit den linken und rechten Antriebsrädern verbunden sind.

Drei mögliche **Zustände** kann ein Motor haben:

- **vorwärts** drehend (↑)
- **ausgeschaltet (--)**
- **rückwärts** drehend (↓)

Aufgabe: Bestimme in der unten stehenden **Tabelle**, auf welche Arten ein Roboter jeweils nach links und rechts gesteuert werden kann. **Trage** dazu die passenden **Zeichen (↑, --, ↓) ein.**

Steuerung nach **rechts...**			**Steuerung** nach **links...**		
Nr.	**Motor links (Ausgang C)**	**Motor rechts (Ausgang B)**	**Motor links (Ausgang C)**	**Motor rechts (Ausgang B)**	**Nr.**
1					1
2					2
3					3

Merke: Eine **schnelle Drehung** haben wir bei der Vorgehensweise unter ____.

Langsame Drehungen werden bei den Wegen ____ und ____ durchgeführt.

Nur bei den **Wegen** ____ und ____ bewegt sich der Roboter **vorwärts.**

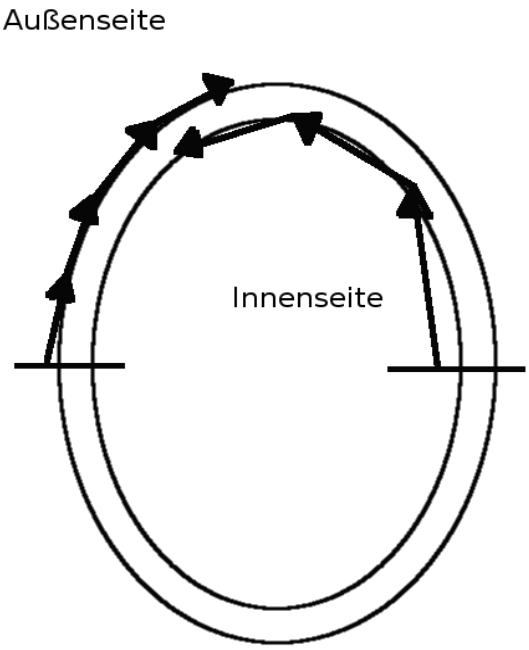
Datum: ***Name:***

Linienfolger für einen Rundkurs programmieren

Ein einfacher Linienfolger hat einige Besonderheiten, die ohne genauere Betrachtung nicht sofort offensichtlich sind:

- Wenn man nur **einen** Sensor hat, dann **fährt** man auf einen der **beiden Übergänge** zwischen der **Linie** und **Umgebungsfläche** und somit
  - entweder auf der **Innenseite** oder
  - auf der **Außenseite** des Kreises

  und nicht direkt auf der Linie.
- Als **Grenzwert** muss ein Wert **Zwischenwert** zwischen dem für Schwarz und Weiß gewählt werden.
- Hätte man zwei Sensoren im optimalen Abstand, wäre es auch anders möglich. Der Aufwand an Hardware ist dabei aber ungleich größer (... und weitere Sensoren sind nicht vorhanden... ;-)).



Merke: Der Weg auf der Innenseite des Kreise ist _____, folglich sind _____ Rundenzeiten zu erwarten.

Aufgaben:

- **Erstelle** ein **Programm** für einen **Linienfolger** nach der nebenstehenden Beschreibung.
- Setze dazu die umgangssprachlichen Beschreibungen in **NXC-Programmtext** um.
- **Hilfestellung** bekommst du durch die NQC-Anweisungen von **Seite 15.**
- Speichere dein Programm unter dem Dateinamen `11-linie-`*`name`*`.nxc` ab.

Umgangssprachliche Beschreibung:

Definiere Grenzwert (Wert zwischen Schwarz- und Weißwert)

Eingang 3 als Lichtsensor schalten

Dauerschleife

  wenn weiß...

    geradeaus fahren

  sonst

    nach links fahren

Optimierung eines Linienfolgers für einen Rundkurs

Ein **Linienfolger** für einen Rundkurs soll sich beim **Überfahren** eines **silbernen Feldes ausschalten** und die **Fahrtzeit anzeigen**. Die **Datei** `11-linie-`*`name`*`.nxc` ist mit folgenden Schritten zu **ergänzen** bzw. zu **bearbeiten**:

- Datei unter angepasstem Dateinamen `12-rund-name.nxc` abspeichern
- " `FILENAME`" und " `AUTHOR`" im Dateikopf anpassen
- Variablen, Konstanten und Anzeigeunterprogramm für die Auswertung vor "task main()" einfügen

```
// FILENAME: 12-rund-name.nxc
// AUTHOR: J.Stolze - 7.10.07 - 14:00

// Konstanten
#define FAHREN 2
#define GRENZE 45
#define ZUSILBER 70
#define SPEED_LOW 30
#define SPEED_FAST 50

//  Variablen
long startzeit, fahrzeit;
int licht = 0;

sub display()
{
   fahrzeit =  CurrentTick() - startzeit;
   TextOut(0, 40, "Fahrzeit:");
   NumOut(60, 40, fahrzeit);
   TextOut(0, 20, "Sensor 3:");
   NumOut(60, 20, licht);
   return;
}
```

Nach **task** und der {-Klammer müssen noch die Befehle zum Festlegen des Eingangs 3 als Lichtsensor, die Abfrage- und die Anzeigeroutine eingefügt bzw. angepasst werden...

```
   SetSensorLight(IN_3);
   startzeit = CurrentTick();
   while (licht < ZUSILBER)
   {
      licht = Sensor(IN_3);       // Lichtsensor an Eingang 3
      display();                  // Anzeige
```

und am Programmende die Ausschalt- und Auswertungsbefehle eingefügt werden.

```
   Off(OUT_BC);
   display();
   Wait(10000);
```

Auftrag: Wer schafft den Rundkurs in der kürzesten Zeit? Optimiere dein Programm!

Veränderung eines Linienfolgers für einen Zickzackkurs

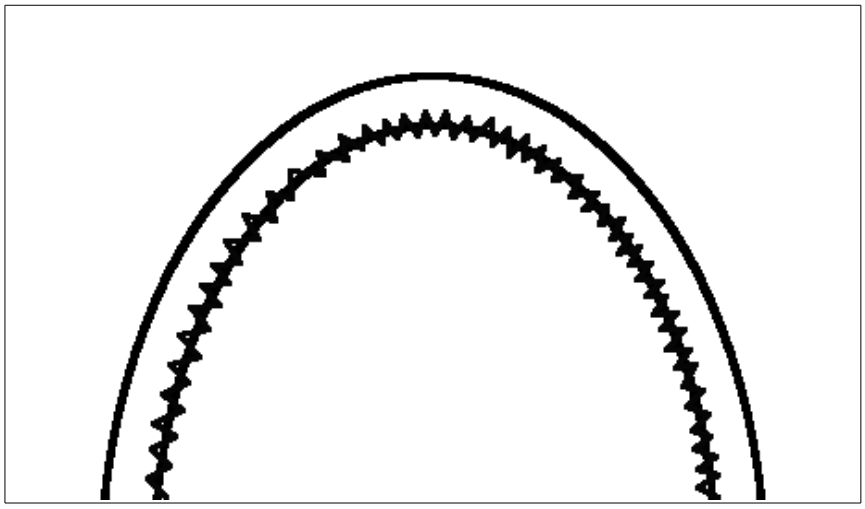
Der Linienfolger für den Rundkurs ist in der while-Schleife so abzuändern, dass er auch auf einem Zick-Zackkurs optimal läuft. Dabei ist folgendes zu bedenken:

Beim **Rundkurs** kann man muss man nur dann die Richtung eines Roboters ändern, wenn beim Befahren

- des Innenkreises man in die schwarze Fläche kommt und
- beim Außenkreis auf die weiße Fläche kommt (siehe Zeichnung Seite 18)

Eine **einfache Programmierung** wird möglich, **weil** die **Außen**- bzw. **Innenseite** der **Kurve** immer die **gleiche Seite** ist.

Beim **Zickzackkurs** ist dies nicht mehr der Fall, dort **ändert sich** diese **Zuordnung immer** wieder, die Programmierung wird aufwändiger. Folgendes Bild soll es veranschaulichen:



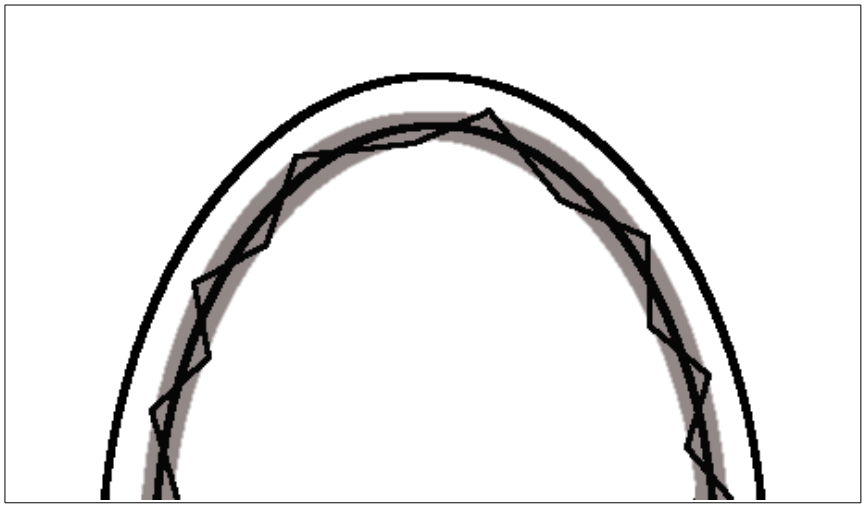
Aufgabe:

- Die Vorgehensweise in der Schleife ist in deutscher Sprache wie unten zu sehen. Setze sie in NXC-Programmcode um.
- Speichere das Ergebnis als `13-zick-`*`name`*`.nxc`.
- Wer schafft den Zickzackkurs in der kürzesten Zeit? Optimiere dein Programm!

Schleife – wenn nicht silber
wenn schwarz
  nach rechts fahren
sonst
  nach links fahren
ende-wenn

Optimierung eines Linienfolgers für einen Zickzackkurs

Der **Linienfolger** für den **Zickzackkurs** fährt ungleich langsamer als der für den Rundkurs. Er **„schwänzelt“**, d.h. er ändert seine Richtung immer sehr schnell. Dies gilt es zu optimieren, indem man eine „**Grauzone“** einführt, in der die Regelung nicht aktiv wird. Dadurch fährt er schon ruhiger. Folgendes Bild soll es veranschaulichen:



Aufgabe:

- Die Vorgehensweise in der Schleife ist in deutscher Sprache wie unten zu sehen. Setze sie in NXC-Programmcode um.
- Speichere das Ergebnis als `14-zack-name.nxc`.
- Wer schafft den Zickzackkurs in der kürzesten Zeit? Optimiere dein Programm!

Schleife – wenn nicht silber
wenn schwarz - GRAUZONE
  nach rechts fahren
sonst
  wenn weiß + GRAUZONE
    nach links fahren
  sonst
    nach vorne fahren