Robot Karol

Eine Programmiersprache
für Schülerinnen und Schüler
Version 2.2

Beschreibung der Sprache
Beschreibung der Programmierumgebung
kommentierte Beispiele

Ulli Freiberger
freiberger@schule.bayern.de

Ondrej Krško
krsko@gjh.sk

Inhaltsverzeichnis

Einleitung

Robot Karol

Dem Programm „Robot Karol“ liegt die Idee von „Karel, the Robot“ zugrunde, wie sie zum ersten Mal in „Pattis, Richard E; Karel the Robot: A Gentle Introduction to the Art of Programming; John Wiley & Sons, 1981“ veröffentlicht wurde.

Das Grundkonzept ist, einen Roboter zu programmieren, der in einer „Bildschirmwelt“ lebt. Wenn die Programme laufen, sehen die Schülerinnen und Schüler an der Reaktion des Roboters sofort, was sie programmiert haben und ob ihr Programm die Aufgabenstellung erfüllt. Diese unmittelbare Rückmeldung ist gerade für Einsteiger extrem wertvoll.

Bei Pattis ist die Roboterwelt zweidimensional und besteht aus einem quadratischen Straßengitter in dem sich Karel, dargestellt als Pfeil, entlang der Straßen bewegen kann. In dieser Welt können Mauern aufgebaut werden, die Karel nicht überwinden kann und „Beeper“ verteilt werden, die Karel einsammelt und ablegt. Karel ist mit einer bewußt einfachen Programmiersprache, die nur einen kleinen Satz an Befehlen kennt steuerbar.
Diese Konzeption zur Einführung in die Informatik wurde in vielen Programmen und Publikationen unter verschiedenen Gesichtspunkten aufgegriffen (siehe Dokument KarelÜbersicht.doc).

Die meisten Realisierungen verwenden die zweidimensionale Welt, wie sie von Pattis vorgesehen wurde. Wesentlich mehr Möglichkeiten bieten dreidimensionale Welten und trotzdem bleiben diese einfach genug um sie zur Einführung in die Algorithmik einsetzen zu können. Die dreidimensionale Welt hat einen Boden aus quadratischen Grundpflastern und eine wählbare Höhe. In dieser Welt kann sich Karel von Quadrat zu Quadrat bewegen, Ziegelstapel aufbauen und abbauen, auf die Ziegelstapel klettern und Markierungen setzen. Ein Karel-Programm dieser Art wurde in dem MSDOS-Grafikprogramm „Robot Karel“ von Blaho, Chlebikova und Vitek 1993 umgesetzt und ins Deutsche übertragen. Die Bedienung der Programmieroberfläche erfolgt dabei mit Funktionstasten.

Das Programm „Robot Karol“ greift diese Idee auf, setzt das Konzept in eine zeitgemäße Form unter der Bedienoberfläche Windows um und erweitert die Einsatzmöglichkeiten wesentlich.

Robot Karol ist:

Eine Programmiersprache, die für Schülerinnen und Schüler zum Erlernen des Programmierens und zur Einführung in die Algorithmik gedacht ist.

Eine Programmierumgebung mit:

- einem Editor mit Syntaxhervorhebung und Schlüsselwort-Ergänzung,
- einem Interpreter der schrittweises Abarbeiten von Programmen ermöglicht und diese in einer Code-Übersicht zeigt,
- einer grafischen Darstellung der Welt, die den Roboter Karol als Figur im Raum zeigt und ihn je nach Anweisungen bewegt, ...

Programmdateien

Das Programm „Robot Karol“ wird mit dem Setup-Programm SETUPD.EXE ausgeliefert. Beim Setup werden keine Einträge in der Registrierungsdatei vorgenommen und keine Dateien in das Windows-Verzeichnis kopiert.
Im Zielverzeichnis von Robot Karol werden folgende Dateien installiert:

karol.exe	Programmierumgebung Robot Karol
karol.ini	Programmeinstellungen, diese Datei benötigt Schreibrechte
license.txt	der Lizenzvertrag
ton.wav	Ton für Karol
karol.hlp karol.cnt	Hilfedatei zu Robot Karol
tipps.txt	Text der bei Tipps und Tricks eingeblendet wird
imgs\marke.bmp	Bild für Marke (46x31)
imgs\quader.bmp	Bild für Quader (46x45)
imgs\ziegel.bmp	Bild für Ziegel (46x31)
imgs\robot0.bmp, robot1.bmp, robot2.bmp, robot3.bmp	Bilder für Karol aus 4 Richtungen (40x71)
imgs\karel\ imgs\karelfarbe\ imgs\kind1\ imgs\kind2\ imgs\figur\ imgs\figurgelb\ imgs\mann\	weitere Roboterbilder, diese können mit robot0.bmp - robot3.bmp ausgetauscht werden
beispiele\	einige Karol-Programme *.kdp und Karol-Welten *.kdw; mit zwei Ziffern beginnende gehören zu den Beispielen in diesem Handbuch
doku\karolhandbuch.doc	dieses Handbuch

Gestartet wird das Programm Robot Karol über KAROL.EXE

Es sind drei verschiedene Programm-AufrufParameter möglich. Die entsprechenden Angaben müssen direkt nach dem Parameterkenner erfolgen. Enthalten die Dateinamen Leerzeichen, so ist der Parameter mit " Zeichen einzufassen.

- /I Pfad zur Inidatei Karol.ini z.B. /IC:\MeineKarolBeispiele\
  falls der Parameter fehlt wird der Pfad des Programms karol.exe genommen
- /P Dateiname eines Karol-Programms, das zu Beginn geladen wird
- /W Dateiname einer Karol-Welt, die zu Beginn geladen wird

Die Sprache Karol

Die Sprache Karol umfasst:
vordefinierte Anweisungen; eigene, selbstdefinierte Anweisungen;
vordefinierte Bedingungen; eigene, selbstdefinierte Bedingungen;
Kontrollstrukturen: bedingte Wiederholung; Wiederholung mit fester Anzahl; bedingte Anweisung;
schnelle, langsame Ausführung; diverse Erweiterungen

Roboter Karol und seine Welt

Der Roboter Karol kann aus **objektorientierter Sicht** als ein **Objekt** der Klasse **ROBOTER** betrachtet werden, mit folgender Struktur:

Eigenschaften: Position (mit X und Y); Blickrichtung; Geschwindigkeit; maximaleSprunghöhe; ZiegelImRucksack; RucksackGröße;

Die **Methoden** (Fähigkeiten) des Roboter Karol werden, aufgrund ihrer Verwendung in zwei Gruppen unterteilt.

Karol hat vordefinierte Methoden mit denen er bestimmte Vorgänge ausführen kann. Ohne äußere Aktionen geschieht noch gar nichts. Erst wenn man eine Botschaft an Karol schickt reagiert dieser mit der entsprechenden Methode. Man erteilt also Karol einen Befehl etwas zu tun, in dem man die jeweilige Methode aufruft (zum Beispiel „einen Schritt nach vorne gehen“). Bei dieser Sichtweise bedeutet eine **Anweisungen** im Programm das Senden einer Botschaft an das Objekt Karol, der mit der zugehörigen Methode darauf reagiert.
Eine neue Anweisung in der Sprache Karol festlegen (definieren) heißt, dem Roboter Karol eine neue Methoden beibringen, mit der er auf diese Anweisung reagieren kann.

Karol hat aber auch Methoden, mit denen er auf eine Anfrage mit WAHR oder FALSCH antwortet (zum Beispiel „stehst du vor der Wand?“). Der Aufruf einer Methode dieser Art heißt **Bedingung**. Neue Bedingungen in der Sprache Karol festlegen heißt, dem Roboter Karol neue Methoden beibringen, mit denen er auf eine bestimmte Anfrage mit WAHR oder FALSCH antworten kann.

Um diese Vorstellung zu unterstützen können in Karol-Programmen alle Methoden (Anweisungen und Bedingungen) von Karol auch mit der Angabe des Objekts und Parameterklammern aufgerufen werden. Das heißt, es ist zum Beispiel neben der Schreibweise `schritt` auch die Schreibweise `karol.schritt()` möglich.

Der Roboter Karol kann durch ein Programm in einer Welt mit Quadratmuster bewegt werden. Es gibt ein Objekt der Klasse **WELT** und dieses hat die Eigenschaften Breite, Länge und Höhe. Eingerahmt ist die Welt an allen vier Seiten von Wänden in der entsprechenden Höhe. Die Welt kann verschiedene Objekte aus den folgenden Klassen enthalten.

ZIEGEL: Diese Objekte kann Karol vor sich hinlegen und später wieder aufheben. An einer Stelle sind mehrere Ziegel aufeinander möglich, jedoch maximal bis zur Höhe der Welt. Karol ist ein kräftiger Bursche und kann beliebig viele (oder begrenzte Anzahl, je nach Einstellung der RucksackGröße) Ziegel mit sich herumschleppen. Er kann auch auf Ziegelstapel hinaufspringen und herabspringen, jedoch maximal so hoch wie seine Eigenschaft „maximaleSprunghöhe“ zuläßt.

MARKE: Diese Objekte kann Karol an der Stelle anbringen, an der er sich gerade befindet. An jeder Stelle ist höchstens eine Marke möglich. Eine Stelle kann markiert sein oder nicht.

QUADER: Mit Quader kann die Welt zusätzlich gestaltet werden. An einer Stelle kann höchstens ein Quader stehen und Karol kann nicht auf einen Quader springen. Quader können nur im Direktmodus aufgestellt und entfernt werden, nicht im Programm. Im Programm verhalten sich Quader wie eine Wand.

Aus **zustandsorientierter Sicht** kann das Karol-System eine feste Anzahl von verschiedene **Zuständen** annehmen. Ein Zustand des Karol-Systems wird beschrieben durch:

- die Breite, Länge und Höhe der Welt
- die Anzahl und Position der Ziegel, Quader und Marken
- die Position und Blickrichtung von Karol
- die weiteren Eigenschaften von Karol. Diese können, bei entsprechender Einstellung, auch die Anzahl der Ziegel im Rucksack und die Rucksackgröße umfassen.

Vor dem Programmstart befindet sich die Karol-Welt in einem frei festlegbaren **Ausgangszustand**. Ein Karol-Programm beschreibt einen **Zustandsübergang** bei dem dieser Ausgangszustand in einen **Endzustand** übergeführt wird, dabei werden meist viele Zwischenzustände eingenommen. Ausgelöst wird der Zustandsübergang durch den Programmstart.
Unterschiedliche Ausgangszustände können zu unterschiedlichen Endzuständen führen. Man sieht das sofort, wenn das selbe Programm auf unterschiedliche Ausgangszustände angewandt wird.
Meist wird deshalb zu einem Karol-Programm ein passender Ausgangszustand mit abgespeichert (beide haben den gleichen Namen) und durch Einstellungen festgelegt, dass bei jedem Programmstart auf diesen Ausgangszustand zurückgegriffen wird.

Anweisungen

Die Ausführung einer Anweisung in einem Karol-Programm bedeutet eine Botschaft an Karol zu schicken, der mit der entsprechenden Methode reagiert und dabei eine gewisse Aktion ausführt.

Vordefinierte Anweisungen

Anweisung	Aktion die Karol ausführt	mögliche Ablauffehler
Schritt	macht einen Schritt in die Blickrichtung	steht vor der Wand; steht vor einem Quader; kann nicht so hoch springen;
Schritt(Anzahl**)**	macht „Anzahl“-viele Schritte	wie bei Schritt
LinksDrehen	dreht sich nach links (um 90°)	
RechtsDrehen	dreht sich nach rechts (um 90°)	
Hinlegen	legt vor sich einen Ziegel hin	steht vor der Wand; steht vor einem Quader; maximale Stapelhöhe erreicht; hat nichts zum Hinlegen(*);
Hinlegen(Anzahl**)**	legt „Anzahl“-viele Ziegel vor sich hin	wie bei Hinlegen
Aufheben	hebt einen Ziegel auf, der vor ihm liegt	steht vor der Wand; steht vor einem Quader; kein Ziegel vor Karol; maximale Tragfähigkeit erreicht(*);
Aufheben(Anzahl**)**	hebt „Anzahl“-viele Ziegel auf, die vor ihm liegen	wie bei Aufheben
MarkeSetzen	setzt an seiner Position eine Marke	
MarkeLöschen	löscht an seiner Position eine Marke	
Warten	wartet eine Sekunde	
Warten(Anzahl**)**	wartet „Anzahl“-viele Millisekunden	
Ton	gibt einen Ton von sich	
Beenden	stoppt den Programmablauf	

(*) nur bei eingeschalteter Kontrolle der Tragfähigkeit

Eigene, selbstdefinierte Anweisungen

Es ist möglich eigene Anweisungen festzulegen. Diese können direkt im Programm definiert werden, so wie es auch in anderen Programmiersprachen üblich ist. Die Bezeichner der Anweisungen können Buchstaben (auch Umlaute), Ziffern und _ enthalten. Eine neue Anweisung muss erst definiert werden, bevor sie verwendet werden kann.

Die Verwendung von Parametern bei Anweisungen wird in einem eigenen Abschnitt beschrieben.

```
{ Anfang der Anweisung }
Anweisung Lege3Ziegel
  wiederhole 3 mal
    Hinlegen
  *wiederhole
*Anweisung
{ Ende der Anweisung }

{ Anfang des Programms }
wiederhole solange NichtIstWand
  Lege3Ziegel
  Schritt
*wiederhole
{ Ende des Programms }
```

Bedingungen

Die Ausführung einer Bedingung in einem Karol-Programm bedeutet eine Anfrage an Karol zu schicken, der mit der entsprechenden Methode reagiert, die Situation seiner Umgebung begutachtet und mit WAHR oder FALSCH antwortet.

Vordefinierte Bedingungen

Bedingung	Karol meldet WAHR,
IstWand	wenn er vor der Wand oder vor einem Quader steht und in diese Richtung schaut
NichtIstWand	wenn IstWand nicht zutrifft
IstZiegel	wenn er vor einem Ziegel oder Ziegelstapel steht und zu diesem schaut
IstZiegel(Anzahl**)**	wenn er vor einem Ziegelstapel mit „Anzahl“-vielen Ziegeln steht und zu diesem schaut
NichtIstZiegel	wenn IstZiegel nicht zutrifft
NichtIstZiegel(Anzahl**)**	wenn IstZiegel(Anzahl) nicht zutrifft
IstMarke	wenn er auf einer Marke steht
NichtIstMarke	wenn IstMarke nicht zutrifft
IstSüden, IstNorden, IstWesten, IstOsten	wenn Karol in diese Richtung schaut

Diese Bedingungen sind nur möglich, wenn die Überwachung der Tragfähigkeit von Karol eingeschalten ist (siehe Programmierumgebung-Einstellungen):

Bedingung	Karol meldet WAHR,
IstVoll	wenn er seine maximale Tragfähigkeit erreicht hat
NichtIstVoll	wenn IstVoll nicht zutrifft
IstLeer	wenn er keinen Ziegel mit sich trägt
NichtIstLeer	wenn IstLeer nicht zutrifft
HatZiegel	wenn er mindestens einen Ziegel mit sich trägt
HatZiegel(Anzahl**)**	wenn er genau „Anzahl“-viele Ziegel mit sich trägt

Eigene, selbstdefinierte Bedingungen

Neben den vordefinierten Bedingungen ist es möglich eigene, selbstdefinierte Bedingungen festzulegen. In der Definition der Bedingung müssen die Wörter WAHR bzw. FALSCH vorkommen, die festlegen welchen Wert die Bedingung zurückgibt. Die Bezeichner der Bedingungen können Buchstaben (auch Umlaute), Ziffern und _ enthalten. Eine neue Bedingung muss erst definiert werden, bevor sie verwendet werden kann.

Die Verwendung von Parametern wird in einem eigenen Abschnitt beschrieben.

```
{ Prüft ob rechts von Karol Ziegel sind }
Bedingung IstZiegelRechts
  schnell
  falsch
  Rechtsdrehen
  wenn IstZiegel dann wahr *wenn
  Linksdrehen
  langsam
*Bedingung

{ Anfang des Programms }
wiederhole solange IstZiegelRechts
  Schritt
*wiederhole
```

Kontrollstrukturen

In der Sprache Karol stehen eine Reihe von Kontrollstrukturen zur Verfügung. Für den Anfängerunterricht ist es sinnvoll nur die bedingte Anweisung, die Wiederholung mit Anfangsbedingung und die Wiederholung mit fester Anzahl zu verwenden. Erst im Fortgeschrittenenunterricht sollten die weiteren Arten der Wiederholung zum Einsatz kommen. Für die Wiederholung mit Anfangsbedingung gibt es aus didaktischen Überlegungen (alle Wiederholungen beginnen mit dem Schlüsselwort wiederhole) eine zweite Form. (siehe Programmierumgebung-Einstellungen)

Bedingte Anweisung

Die einseitige bedingte Anweisung hat folgende Syntax:

```
wenn [nicht] <bedingung> dann
  <anweisung>
  ...
  <anweisung>
*wenn
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;

Wenn die Bedingung (bzw. ihre Verneinung) zutrifft werden die Anweisungen im „dann-Block“ ausgeführt, sonst wird mit der nächsten Anweisung nach „*wenn“ fortgefahren.

Die zweiseitige bedingte Anweisung hat folgende Syntax:

```
wenn [nicht] <bedingung> dann
  <anweisung>
  ...
  <anweisung>
sonst
  <anweisung>
  ...
  <anweisung>
*wenn
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Wenn die Bedingung (bzw. ihre Verneinung) zutrifft werden die Anweisungen im „dann-Block“ ausgeführt, sonst die Anweisungen im „sonst-Block“.

Wiederholung mit fester Anzahl

Die Wiederholung mit fester Anzahl hat folgende Syntax:

```
wiederhole <x> mal
  <anweisung>
  ...
  <anweisung>
*wiederhole
```

<x> - ganze Zahl, die die Anzahl der Wiederholungen festlegt;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „wiederhole-Block“ werden so oft wiederholt wie die angegebene Anzahl festlegt.

Bedingte Wiederholung

Die Wiederholung mit **Anfangsbedingung** hat folgende Syntax:

```
solange [nicht] <bedingung> tue
  <anweisung>
  ...
  <anweisung>
*solange
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „solange-Block“ werden solange wiederholt wie die Bedingung (bzw. ihre Verneinung) zutrifft. Die Überprüfung der Bedingung erfolgt am Anfang der Wiederholung und kann dazu führen, dass die Anweisungen ggf. gar nicht ausgeführt werden.
Für den unterrichtlichen Einsatz empfiehlt sich die andere Schreibweise der Wiederholung mit Anfangsbedingung.

Für die Wiederholung mit **Anfangsbedingung** gibt es eine weitere Syntax:

```
wiederhole solange [nicht] <bedingung>
  <anweisung>
  ...
  <anweisung>
*wiederhole
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „wiederhole-Block“ werden solange wiederholt wie die Bedingung (bzw. ihre Verneinung) zutrifft. Die Überprüfung der Bedingung erfolgt am Anfang der Wiederholung und kann dazu führen, dass die Anweisungen ggf. gar nicht ausgeführt werden. Diese Wiederholung ist vom Verhalten identisch zu „solange tue“ und wird aus didaktischen Überlegungen angeboten, damit alle Wiederholungen mit dem Schlüsselwort „wiederhole“ beginnen.

Die Wiederholung mit **Endbedingung** hat folgende Syntax:

```
wiederhole
  <anweisung>
  ...
  <anweisung>
*wiederhole solange [nicht] <bedingung>
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „wiederhole-Block“ werden solange wiederholt wie die Bedingung (bzw. ihre Verneinung) zutrifft. Die Überprüfung der Bedingung erfolgt am Ende der Wiederholung und führt dazu, dass die Anweisungen mindestens einmal ausgeführt werden.

Zusätzlich ist eine Wiederholung mit **Endbedingung** mit folgender Syntax möglich:

```
wiederhole
  <anweisung>
  ...
  <anweisung>
*wiederhole bis [nicht] <bedingung>
```

<bedingung> - vordefinierte oder selbstdefinierte Bedingung;
<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „wiederhole-Block“ werden solange wiederholt wie die Bedingung (bzw. ihre Verneinung) nicht zutrifft. Wenn die Bedingung zutrifft wird die Wiederholung beendet. Die Überprüfung der Bedingung erfolgt am Ende der Wiederholung und führt dazu, dass die Anweisungen mindestens einmal ausgeführt werden.
Diese Wiederholung ist identisch zu „wiederhole … *wiederhole solange nicht“. Sie wird nur aus der Überlegungen heraus angeboten, dass in vielen Programmiersprachen diese Form der Wiederholung mit Endbedingung gebräuchlich ist. Im Anfängerunterricht sollte, um Irritationen zu vermeiden, diese Kontrollstruktur nicht zum Einsatz kommen.

Endlos-Wiederholung

Die endlose Wiederholung hat folgende Syntax:

```
wiederhole immer
  <anweisung>
  ...
  <anweisung>
*wiederhole
```

<anweisung> - vordefinierte oder selbstdefinierte Anweisung; Wiederholung; bedingte Anweisung;
Die Anweisungen im „wiederhole-Block“ werden immer wiederholt, ein Abbruch ist nur über die Schaltfläche „Abbrechen“ bzw. dem Menüpunkt „Ablauf - Stopp“ möglich.

Ausführung - schnell/langsam

Der Roboter Karol kann die Befehle auf zwei Arten ausführen: langsam oder schnell.
Bei langsamer Ausführung legt Roboter Karol (je nach Wert der Eigenschaft Geschwindigkeit) nach jedem Befehl eine „kleine Verschnaufpause“ ein, damit man den Ablauf des Befehls besser am Bildschirm verfolgen kann (bei Programmausführung mittels Schnelldurchlauf tritt diese Verzögerung nicht auf).
Das Wort schnell schaltet in den schnellen Modus, hierbei unterbleibt jegliche Verzögerung und die Veränderungen in der Karol-Welt werden zur Beschleunigung nicht am Bildschirm angezeigt. Das Wort langsam schaltet wieder in den normalen Modus mit Verzögerung und Bildschirmausgabe um. Innerhalb von selbstdefinierten Anweisungen/Bedingungen ist die Anweisung schnell nur gültig bis zum nächsten langsam bzw. bis zum Ende des selbstdefinierten Blocks.

```
{ Ausschnitt aus Programmbeispiel 07 }

// Anweisung Becken bauen
Anweisung BaueBecken
   schnell
   wiederhole 12 mal
     solange NichtIstWand tue
       Hinlegen
       Schritt
     *solange
     LinksDrehen
   *wiederhole
*Anweisung

// Anweisung Becken abreissen
Anweisung AbbauenBecken
   schnell
   wiederhole 12 mal
     solange NichtIstWand tue
       Aufheben
       Schritt
     *solange
```

```
     RechtsDrehen
     schnell
   *wiederhole
*Anweisung
```

Erweiterungen

Trenner - Als Trenner zwischen den Anweisungen kann außer Lücke und neue Zeile auch ein Strichpunkt verwendet werden, ähnlich den Sprachen Pascal, C, C++.

```
wiederhole 10 mal
  LinksDrehen; Hinlegen; RechtsDrehen;
  RechtsDrehen; Hinlegen; LinksDrehen;
  Schritt;
*wiederhole
```

Kommentare - Programmteile, die nicht ausgeführt werden. Sie dienen zur Erklärung des Programms. Möglich sind einzeilige und mehrzeilige Kommentare.

```
wiederhole 10 mal
  LinksDrehen;
  Hinlegen;
  RechtsDrehen; // Karol legt links Ziegel
  RechtsDrehen; Hinlegen; LinksDrehen;
  { jetzt hat Karol neben sich zwei Ziegel
    aufgebaut - einen zur Linken und einen zur
    Rechten }
  Schritt;
*wiederhole
```

Objekt Karol - um zu verdeutlichen, dass die Ausführung einer Anweisung der Aufruf der entsprechenden Methode des Objekts bedeutet, kann man in Karol auch die „Punkt-Schreibweise“ für den Objektbezug verwenden. Ebenso ist, aus didaktischen Überlegungen die Verwendung von Klammern möglich.

```
Karol.Schritt()
Karol.LinksDrehen()
Karol.Hinlegen()
```

Programm - Das Schlüsselwort Programm dient zur deutlichen Hervorhebung des Hauptprogramms. Die ist gerade dann günstig, wenn mehrere selbstdefinierte Anweisungen und Bedingungen zum Einsatz kommen.

```
Anweisung PfeilerSetzen
  wenn IstZiegel dann
    Aufheben
    PfeilerSetzen
  sonst
    LinksDrehen
    LinksDrehen
  *wenn
  Hinlegen
*Anweisung

Programm
  PfeilerSetzen
  Aufheben
*Programm
```

Bibliothek - Selbstdefinierte Anweisungen und Bedingungen können in gesonderten Karol-Programmen (Bibliotheken) abgelegt werden. Mit der Anweisung **Einfügen** können diese Anweisungen bzw. Bedingungen zu einem beliebigen Karol-Programm hinzugefügt werden. Für die Bibliothek ist der Pfad und Dateiname anzugeben. Liegt die Bibliothek im selben Verzeichnis wie das Karol-Hauptprogramm reicht der Dateiname alleine.

```
Einfügen
  C:\Beispiele\Bibliothek.kdp
*Einfügen

Programm
  wiederhole 10 mal
    Umdrehen
    LegeFünf
    Umdrehen
    Schritt
  *wiederhole
*Programm
```

Verwendung von Parametern

Parameter bei vordef. Anweisungen

Es gibt einige vordefinierte Anweisungen und Bedingungen, die durch einen Parameterwert als Zusatzinformation erweitert werden können. Es sind dies die Anweisungen: Schritt, Hinlegen, Aufheben und die Bedingungen: IstZiegel, NichtIstZiegel und HatZiegel. Als Parameterwerte sind nur positive Ganzzahlwerte erlaubt, bei den Bedingungen zusätzlich die Zahl 0.

Nähere Beschreibung über die Wirkung der Parameter findet man in der Übersicht der Anweisungen bzw. Bedingungen.

```
Schritt(3)
Hinlegen(4)
LinksDrehen
Aufheben(2)
```

```
wenn IstZiegel(10) dann
  wiederhole 10 mal
    Aufheben
  *wiederhole
*wenn
```

Formaler Parameter in selbstdef. Anweisungen

In selbstdefinierten Anweisungen und Bedingungen steht ein formaler Parameter zur Verfügung. Dieser hat die festvorgegebene Bezeichnung X . Innerhalb der selbstdefinierten Anweisung/Bedingung kann der formale Parameter bei allen Anweisungen/Bedingungen verwendet werden, die einen Parameter unterstützen (siehe oberhalb: Schritt, Hinlegen, Aufheben, IstZiegel, NichtIstZiegel und HatZiegel). Eine Angabe des formalen Parameters in der Definitionszeile der Anweisung ist nicht nötig, aber aus didaktischen Gründen empfehlenswert.
Beim Aufruf der selbstdefinierten Anweisung/Bedingung im Programm muss dann ein aktueller Parameter angegeben werden, der eine positive Ganzzahl sein darf (bei Bedingungen auch 0).

```
Anweisung Versteck(X)
  Hinlegen(X)
  LinksDrehen
  Hinlegen(X)
  LinksDrehen
  Hinlegen(X)
  LinksDrehen
  Hinlegen(X)
*Anweisung

Programm
  Versteck(5)
*Programm
```

Bei der Kontrollstruktur der Wiederholung mit fester Anzahl kann innerhalb einer selbstdefinierten Anweisung/Bedingung statt der Wiederholungszahl der formale Parameter X verwendet werden.

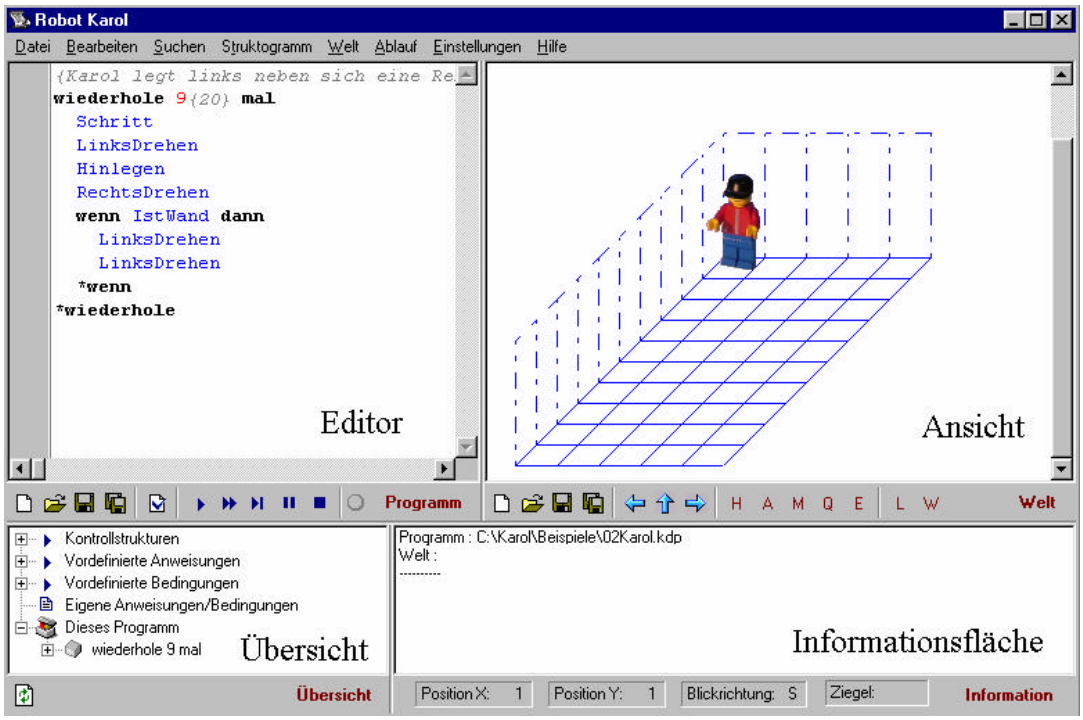
```
Anweisung ReiheLegen(X)
  wiederhole X mal
    Hinlegen
    Schritt
  *wiederhole
*Anweisung

Programm
  ReiheLegen(4)
  Schritt
  ReiheLegen(3)
*Programm
```

Mit dieser einfachen Art der Verwendung von Parametern kann sogar im Anfängerunterricht eine erste Einführung in das Parameterkonzept behandelt werden.

Programmierumgebung

Die Programmieroberfläche „Robot Karol“ läuft unter dem Betriebssystem Windows (in der 32-Bit Version), d.h. Windows95, Windows98, WindowsME, WindowsNT, Windows2000 und WindowsXP. Sie wird in der üblichen Art und Weise wie andere Windows-Programme bedient.



Das Hauptfenster von Robot Karol umfasst die vier Teile Editor, Ansicht, Übersicht und Informationsfläche. Die Größe der Bereiche kann durch Ziehen mit der Maus an den Trennlinien eingestellt werden.

Editor

Texterfassung
Im Editor wird der Programmtext erfasst. Der Editor hebt den Programmtext, entsprechend der Syntax, farblich hervor. Durch Einrückungen mit <Tab> kann der Programmtext sinnvoll strukturiert werden. Die Einrückbreite von <Tab> wird über den Menüpunkt „Einstellungen - Editor“ festgelegt. Der Editor verfügt über die üblichen Fähigkeiten zum Kopieren, Ausschneiden, Einfügen, Suchen und Ersetzen von Text (siehe Menüpunkt „Bearbeiten“).
Für die vordefinierten Anweisungen und Bedingungen besteht, neben der Eingabe durch Eintippen, eine Eingabe durch Auswahl aus einer Aufklappliste, die man mit <Strg>+<Leer> öffnen kann. Zusätzlich kann man eingegebene Anfangsteile der reservierten Wörter durch <Shift>+<Leer> automatisch vervollständigen lassen (z.B. aus „wie“ wird „wiederhole mal“).
Mit RechteMausKlick öffnet sich ein Aufklappmenü aus dem man ein reserviertes Wort auswählen kann, das dann an der Klickstelle eingefügt wird.

Mit „Bearbeiten - Formatieren“ kann eine Formatierung des Textes vorgenommen werden (Der Umfang der Formatierung wird über „Einstellungen - Editor“ festgelegt).
Eine Einblendung von Zeilennummern am Rand hilft bei der Besprechung der Programme (ein- und ausschalten über den Menüpunkt „Einstellungen - Editor“)

Wichtige Steuertasten im Editor:

Tasten bzw. Tastenkombination	
<Pfeil rechts>	Cursor ein Zeichen rechts
<Strg>+<Pfeil rechts>	Cursor ein Wort rechts
<Pfeil links>	Cursor ein Zeichen links
<Strg>+< Pfeil links>	Cursor ein Wort links
<Pfeil oben>	Cursor eine Zeile nach oben
<Strg>+<Pfeil oben>	rollt Text um eine Zeile nach oben
<Pfeil unten>	Cursor eine Zeile nach unten
<Strg>+<Pfeil unten>	rollt Text um eine Zeile nach unten
<Bild auf>	rollt Text um eine Seite nach oben
<Bild ab>	rollt Text um eine Seite nach unten
<Pos1>	Cursor zum Zeilenanfang
<Strg>+<Pos1>	Cursor zum Textanfang
<Ende>	Cursor zum Zeilenende
<Strg>+<Ende>	Cursor zum Textende
<Umschalt>+ <Steuertaste(n)>	markiert Text ab Cursorposition bis zur neuen Cursorposition; diese ergibt sich aus der obigen Übersicht
<Strg>+<a>	markiert gesamten Text
<Rück>	löscht Zeichen links vom Cursor
<Alt>+<Rück>	widerruft die letzte Aktion (Undo); mehrere Stufen möglich
<Umschalt>+<Rück>	führt die letzte widerrufene Aktion wieder durch (Redo)
<Strg>+<y>	löscht die ganze Zeile in der sich der Cursor befindet
<Strg>+<Umschalt>+ <ziffer>	setzt/löscht am Rand eine Markierung mit der Ziffer
<Strg>+<Ziffer>	setzt den Cursor an die markierte Position
<Umschalt>+<Leertaste>	ergänzt das Wort an der Cursorposition, sofern es sich um ein definiertes Schlüsselwort handelt
<Strg>+<Leertaste>	zeigt in einem Aufklappfeld alle zu dem Wortanfang passenden Schlüsselwörter
rechte Mausklick	öffnet AufklappMenü mit allen vordefinierten Kontrollstrukturen, Anweisungen und Bedingungen; fügt ausgewählte an der Cursorposition ein

Der Programmtext kann gespeichert, später wieder geöffnet und ausgedruckt werden (siehe Menü „Datei“). Die übliche Endung für Programmtexte ist *.kdp („Karol deutsch Programm“).

Programmablauf
Mit den Schaltflächen unterhalb des Editors oder über das Menü „Ablauf“ wird die Programmausführung gesteuert. Karol kann das Programm schrittweise (normaler „Programmstart“) und schnell ausführen („Schnelllauf“). Bei der schrittweisen Abarbeitung hält Karol nach jeder Anweisung eine bestimmte Zeit inne, deren Dauer über den Menüpunkt „Einstellungen - Karol“ festgelegt werden kann. Dies ermöglicht eine bessere Betrachtung des Programmablaufes. Zusätzlich ist auch „Einzelschritt“ möglich. Bei jedem Klick auf diese Schaltfläche führt Karol eine einzelne Anweisung aus und wartet bei der nächsten auf einen weiteren Klick. Durch „Pause“ kann das Programm angehalten und durch „Abbruch“ abgebrochen werden. Die „Lampe“ neben den Schaltflächen zeigt den Programmstatus (grün Programmablauf, gelb Einzelschritt).

Über den Menüpunkt „Ablauf - Stopppunkt“ bzw. <Strg>+<b> wird an der aktuellen Cursorstelle ein Stopppunkt gesetzt. Der Programmablauf stoppt bei Erreichen dieser Zeile. Über „Programmstart“, „Schnelllauf“ oder „Einzelschritt“ kann das Programm fortgesetzt werden.

Vor jedem Programmablauf wird der Programmtext einer Syntaxprüfung unterzogen. Liegt ein Fehler vor, so wird dieser in der Informationsfläche beschrieben und die betroffene Zeile durch ein Warnsymbol im Editorrand markiert. Ein Programm kann erst ablaufen wenn die Syntaxprüfung keine Fehler ergab.
Die Syntaxprüfung kann auch unabhängig von einem Programmstart über den Menüpunkt „Ablauf - Syntaxprüfung“ bzw. die Schaltfläche mit „Häckchen“ aufgerufen werden.

Ansicht / Karol-Welt

Karol-Welt
Im Bereich der Ansicht sieht man Roboter Karol in seiner Welt. Eine Welt umfasst neben ihrer Ausdehnung (Breite, Länge, Höhe) auch die Ziegel, Quader und Marken die in ihr liegen und die Startposition von Karol.
Über die Schaltflächen unterhalb der Ansicht bzw. dem Menü „Welt“ kann man eine neue Welt anlegen, eine Welt speichern und eine gespeicherte Welt öffnen. Die übliche Endung für Karol-Welten ist *.kdw (Karol deutsch Welt).
Die Welt kann auch als Grafik gespeichert werden und steht damit als Bild zur Verwendung in anderen Programmen zur Verfügung. Die 3D-Darstellung wird als Bitmap mit der Endung *.bmp (Windows Bitmap), die 2D-Darstellung als Grafik im Windows-Metafile-Format (*.emf) abgespeichert.
Für die Welt gibt es zwei Darstellungsmodi. Standard ist die 3D-Darstellung, in der Karol, die Ziegel und die Welt räumlich in einem Schrägbild gezeichnet werden. Zusätzlich gibt es eine 2D-Darstellung (Menü „Welt - 2D Darstellung“ oder Schaltfläche „2D“) in der die Welt als Grundriss dargestellt wird. Karol wird durch ein Dreieck repräsentiert und die Ziegel durch rote Quadrate. Die Anzahl der Ziegel in einem Stapel wird durch eine entsprechende Zahl wiedergegeben.

Festlegung der Karol-Welt
Die Karol-Welt kann auf zwei Arten festgelegt werden. Entweder durch Steuerung von Karol im Direktmodus und damit Anbringung von Ziegel, Quader oder Marken an bestimmten Stellen. Oder durch direktes Setzen/Löschen der Objekte mit Mausklick (nur in der 2D-Darstellung möglich).

Festlegung durch Karol im Direktmodus
Mit den Schaltflächen „Pfeile“ und „H“, „A“, „M“, „Q“, „E“ kann Roboter Karol direkt gesteuert und zur Arbeit (Ziegel hinlegen, aufheben; Marken setzen, löschen; Quader aufstellen, entfernen) aufgefordert werden.

„H“: Hinlegen - Karol legt auf das Feld vor sich einen Ziegel
„A“: Aufheben - Karol hebt einen Ziegel vom Feld vor sich auf
„M“: Marke - Karol setzt bzw. löscht eine Marke auf dem Feld auf dem er steht
„Q“: Quader - Karol stellt in dem Feld vor sich einen Quader auf
„E“: Entfernen - Karol entfernt einen Quader, der vor ihm steht

Damit kann eine Welt sowohl in der 3D-Darstellung als auch in der 2D-Darstellung aufgebaut werden.
Durch Tastendruck kann der Roboter Karol in seiner Welt ebenfalls direkt gesteuert werden. Die Tastensteuerung wird durch einen Klick auf die Welt aktiviert. Als Tasten sind möglich: <Pfeil links> für LinksDrehen, <Pfeil rechts> für RechtsDrehen, <Pfeil oben> für Schritt vorwärts, <Pfeil unten> für Schritt rückwärts, <h> für Ziegel Hinlegen, <a> für Aufheben, <m> für Marke setzen/löschen, <q> für Quader aufstellen und <e> für Quader entfernen.

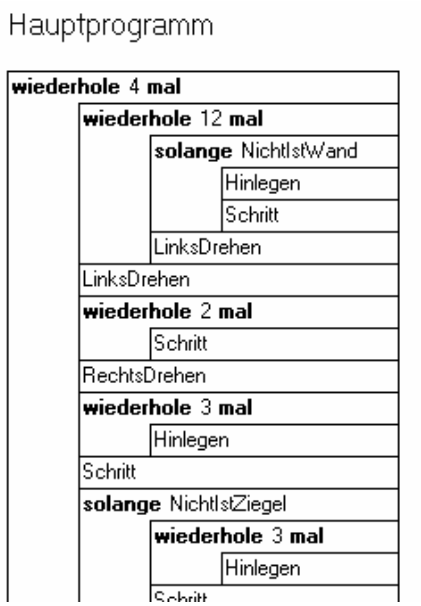
Festlegung durch direktes Setzen/Löschen
In der 2D-Darstellung können die Objekte der Welt durch Mausklick direkt gesetzt bzw. gelöscht werden. Hierzu ist das Werkzeugfenster über das Menü „Welt - Welt direkt festlegen“ aufzurufen.

Im Werkzeugfenster wird durch Klick der entsprechende Arbeitsmodus ausgewählt (von links nach rechts, von oben nach unten: Ziegel setzen, Ziegel entfernen, Marke setzen, Marke entfernen, Quader setzen, Quader entfernen, Karol setzen, Werkzeugfenster schließen). Ein Klick mit der Maus auf die entsprechende Stelle in der Karol-Welt führt die Aktion aus.

Wiederherstellen/Löschen
Während des Programmablaufs (siehe oben) kann man die Bewegungen und Arbeiten von Karol direkt in der Welt betrachten.
Durch die Schaltfläche „W“ kann die Welt wieder hergestellt werden. Wurde die Welt schon gespeichert, so wird sie entsprechend des gespeicherten Zustands wieder hergestellt, andernfalls nimmt die Welt den Zustand vor dem letzten Programmstart an.
„L“ löscht die ganze Welt und stellt Karol in die Ecke ganz hinten links (Ursprung).

Struktogramm

Das Karol-Programm kann auch als Struktogramm dargestellt werden. Nach dem Menüpunkt „Struktogramm - Anzeigen“ wird die Karol-Welt ausgeblendet und dafür das Struktogramm des aktuellen Karol-Programms im Ansichtbereich angezeigt.



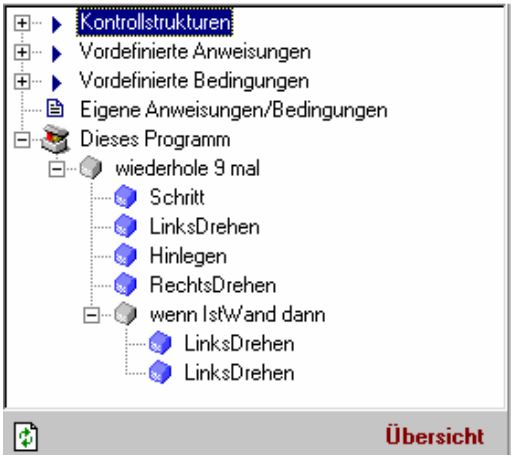
Vor der Darstellung des Struktogramms wird das Programm einer Syntaxprüfung unterzogen. Bei Syntaxfehlern erfolgt keine Struktogrammdarstellung.

Das Struktogramm kann als Grafik im Windows-Metafile-Format (*.emf) abgespeichert werden. Ein Ausdruck und das Kopieren in die Zwischenablage wird ebenfalls unterstützt. Diese Kopie kann dann über Einfügen in Dokumente einer anderen Anwendung (z. B. Textverarbeitung) eingebunden werden.

Während der Struktogrammanzeige ist kein Programmablauf möglich. Eine Direktsteuerung von Karol kann ebenfalls nicht ausgeführt werden. Hierzu muss die Struktogrammdarstellung erst durch Klick auf „Struktogramm ausblenden“ bzw. Menüpunkt „Struktogramm - Ausblenden“ geschlossen werden.

Übersicht

Die Übersicht stellt in hierarchischen Listen sowohl vordefinierte als auch eigene Anweisungen und Bedingungen dar. Auf Wunsch kann das ganze Programm in Form einer Baumstruktur angezeigt werden (hierzu ist vorher ein Klick auf die Schaltfläche „Übersicht aktualisieren“ nötig). Dabei sind die einzelnen Kontrollstrukturen und die eingebunden Blöcke hierarchisch dargestellt. Auch selbstdefinierte Anweisungen/Bedingungen aus Bibliotheken, die in das Programm eingefügt wurden werden dargestellt.



Informationsfläche

In der Informationsfläche wird ständig die aktuelle Position und Blickrichtung von Karol eingeblendet. Auf Wunsch kann die Anzahl Ziegel angezeigt werden, die Karol momentan trägt (siehe „Einstellungen - Karol“).
Im Textfeld werden die Namen und Pfade für die Dateien der aktuellen Welt und des Programms aufgeführt. Bei der Syntaxprüfung erfolgt die Ausgabe der Fehlermeldung in diesem Bereich. Während des Programmlaufs werden hier Laufzeitfehler gemeldet. Diese treten auf wenn Karol einen Bewegungsfehler (z.B. läuft gegen die Wand) macht und im Dialog „Einstellungen - Karol“ die Anzeige dieser Fehler gewählt wurde.

Einstellungen Karol

Über diesen Menüpunkt lassen sich einige Programmeinstellungen die Karol betreffen festlegen.

- Die Ablaufverzögerung legt fest wieviele Sekunden Karol beim normalen Programmablauf nach jedem Programmschritt verzögern soll, damit die Bewegung am Bildschirm besser mitverfolgt werden kann. (Standardeinstellung 0,2 sec; Bereich 0 sec ... 2 sec)
- Wurde die Welt abgespeichert, so kann man einstellen, dass vor jedem Programmstart diese gespeicherte Welt erneut geladen wird. Das hat den Vorteil, dass der Programmablauf immer auf dem selben Zustand der Welt aufsetzt.

- Karol kann nicht beliebig hoch/tief springen. Man kann festlegen um wieviele Ziegel Karol maximal hoch-/tiefspringen kann. (Standardeinstellung 1 Ziegel; Bereich 1 .. 10 Ziegel)
- Die Aufgaben, die Karol bearbeiten kann werden schwieriger, wenn man berücksichtigt wieviele Ziegel Karol in seinem „Rucksack“ tragen kann. Ist die Kontrolle eingeschalten, so kann man bestimmen wieviele Ziegel Karol beim Programmstart mit sich trägt und wieviele Ziegel er maximal tragen kann.
- Bei Ablauffehler - z.B. Karol ist gegen die Wand gestoßen - kann die Programmieroberfläche unterschiedlich reagieren:
  a) der Ablauffehler wird ignoriert
  b) es wird ein entsprechender Hinweis im Informationsbereich ausgegeben, das Programm läuft aber weiter
  c) das Programm wird abgebrochen

Einstellungen Editor

Mit diesem Menüpunkt lassen sich einige Programmeinstellungen die den Editor betreffen festlegen.

- Tabulatorabstand legt fest um wieviele Leerstellen durch die Taste <TAB> eingerückt wird. (Standardeinstellung 2)
- Die Schriftgöße für den Programmtext kann an dieser Stelle in Punktgröße gewählt werden. (8Pkt ... 24Pkt)
- Je nach Lernfortschritt kommen die erweiterten Kontrollstrukturen zum Einsatz. Über „erweiterte Kontrollstrukturen anbieten“ kann man festlegen, ob beim Klick mit der rechten Maustaste im Editorfenster nur die Grundstrukturen oder zusätzlich die erweiterten Strukturen angezeigt werden.
- Es können die Farben für die Hervorhebungen gewählt werden. Man unterscheidet zwischen Kontrollstrukturen, vordefinierte Anweisungen/Bedingungen, selbstdefinierte Anweisungen/Bedingungen, Zahlen und Kommentare.
- Klick auf „Standardeinstellungen“ stellt empfohlene Farbfestlegungen ein.
- Beim Formatieren des Programmtextes (Menü „Bearbeiten - Formatieren“) wird der Text entsprechend den Kontrollstrukturen eingerückt. Zusätzlich kann man die vordefinierten Bezeichner automatisch auf eine standardisierte Schreibweise ändern lassen.
- In der Editorspalte können Zeilennummern eingeblendet werden. (Standardeinstellung aus)
- Beim Ausdruck Zeilennummern ausgeben (Standardeinstellung aus)
- Farbig Ausdrucken (Standardeinstellung aus)

Roboterfiguren

Zur 3D-Darstellungen des Roboter Karol werden vier Bilder - je nach Blickrichtung - verwendet. Sie liegen im Verzeichnis imgs und heißen robot0.bmp, robot1.bmp, robot2.bmp und robot3.bmp. Mit dem Programm werden zusätzlich noch weitere Sätze von Figurenbildern für Roboter Karol ausgeliefert.
Über das Menü „Einstellungen - Figur wechseln“ kann das Bild von Roboter Karol temporär geändert werden, diese Änderung wird beim nächsten Programmstart zurückgenommen. Zum

dauerhaften Auswechseln muss ein Satz von vier Bilddateien aus den entsprechenden Unterverzeichnissen in das Verzeichnis imgs kopiert und die dortigen Bilder ersetzt werden.
Weitere Roboterfiguren :

- \Karel
  Dieser Karol erinnert an die ursprüngliche Karel-Figur.
- \KarelFarbe
  Wie Karel jedoch mit Farben, damit man die Blickrichtung besser unterscheiden kann.
- \Figur
  Karol als kleine Spielfigur. Entspricht der Grundauslieferung.
- \FigurGelb
  Karol als kleine Spielfigur. Wie Figur jedoch mit gelber Hose.
- \Kind1
  Karol wird durch ein Kind mit anliegenden Armen dargestellt.
- \Kind2
  Karol wird durch ein Kind mit abgewinkelten Armen dargestellt.
- \Mann
  Karol wird durch einen Mann mit abgewinkelten Armen dargestellt.

Man kann auch eigene Karol-Bilder verwenden. Diese müssen Bitmap-Dateien sein mit den Maßen 40 Pixel breit und 65-95 Pixel hoch. Für jede Blickrichtung muss ein entsprechendes Bild erstellt werden.

Beispiele

In den folgenden Beispielen wird das Prinzip der Karol-Programme gezeigt.

Programm1

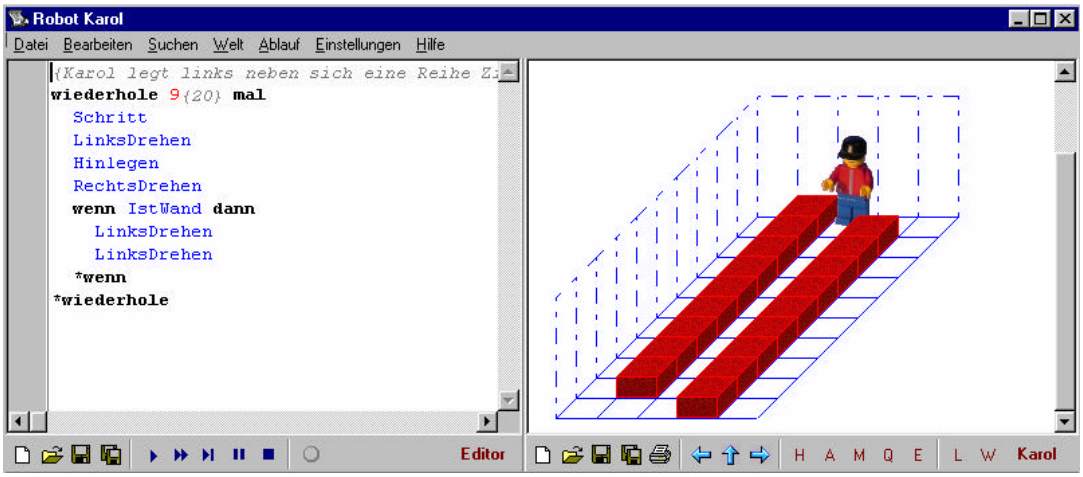
Das erste Programm zeigt die Benutzung der bedingten Wiederholung mit Anfangsbedingung und der Bedingung „Nicht IstWand“. Karol durchquert den Raum und bleibt an der Wand stehen.

```
solange NichtIstWand tue
  Schritt
*solange
```

Aufgabe: Versuche andere Wiederholungen, Anweisungen und Bedingungen. Schreibe ein Programm, so dass sich Karol an der Wand entlang bewegt.

Programm2

Dieses Programm bringt Karol bei, die Welt zu durchqueren und links von sich Ziegel zu legen. Wenn Karol an der Wand angekommen ist, dreht er um und legt weiter Ziegel.

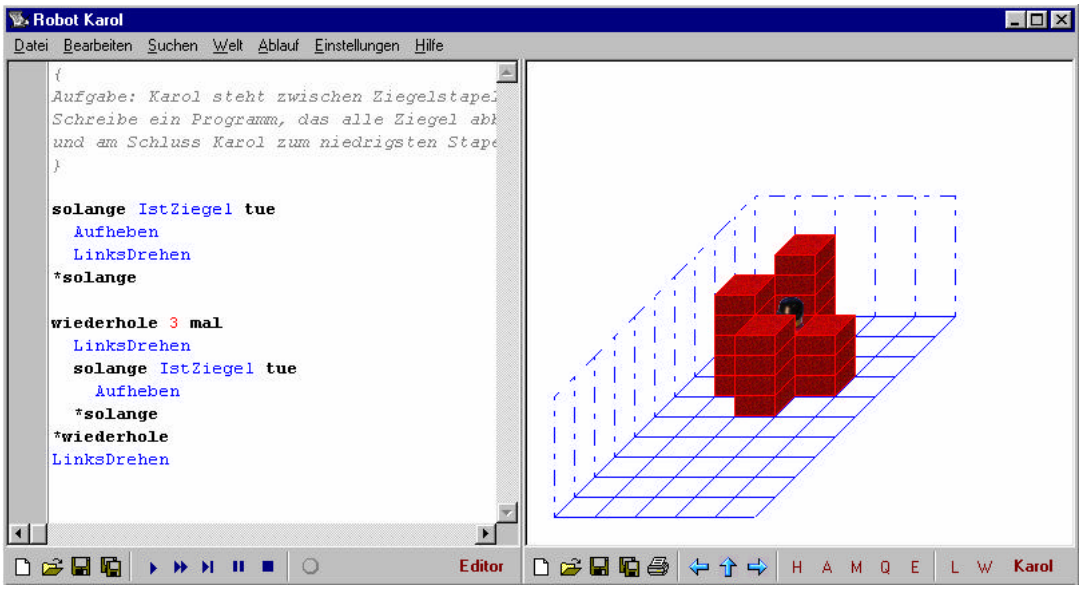


```
{Karol legt links neben sich eine Reihe Ziegel}
wiederhole 20 mal
  Schritt
  LinksDrehen
  Hinlegen
  RechtsDrehen
  wenn IstWand dann
    LinksDrehen
    LinksDrehen
  *wenn
*wiederhole
```

Aufgabe: Schreibe ein Programm, damit Karol an der Wand entlang Ziegel legt. Verwende nur eine Wiederholung.

Ziegelstapel vergleichen

Karol steht zwischen vier Stapeln aus Ziegeln. Man soll ein Programm schreiben bei dem Karol alle Ziegel abbaut. Am Schluss soll er so stehen bleiben, dass er in Richtung des (ehemals) kleinsten Stapel schaut.
Das Programm funktioniert folgendermaßen: Karol nimmt nacheinander von jedem Stapel einen Ziegel, bis ein Stapel ganz abgebaut ist. Dann leert er alle 3 restlichen Stapel jeweils auf einmal.



```
{ Karol steht zwischen Ziegelstapel.
Schreibe ein Programm, das alle Ziegel abbaut
und am Schluss Karol zum niedrigsten Stapel
dreht. }
solange IstZiegel tue
  Aufheben
  LinksDrehen
*solange

wiederhole 3 mal
  LinksDrehen
  solange IstZiegel tue
    Aufheben
  *solange
*wiederhole
LinksDrehen
```

Anweisung Umdrehen

Diesmal schreiben wir eine eigene Anweisung, die Karol um 180 Grad dreht.

```
Anweisung Umdrehen
  LinksDrehen
  LinksDrehen
*Anweisung
```

Aufgabe: Schreibe weitere Anweisungen, die du später gebrauchen kannst. Zum Beispiel LinksHinlegen, RechtsHinlegen, ZweiHinlegen, usw.

Anweisung SchrittZurück

Wir schreiben eine weitere eigene Anweisung, in der wir Karol beibringen einen Schritt zurückzugehen. Diese Anweisung kommt in späteren Beispielen zur Anwendung. Das Beispiel zeigt die Verwendung lokaler Anweisungen, es kann natürlich auch mit einer globalen Anweisung Umdrehen gelöst werden.

```
Anweisung SchrittZurück

  Anweisung Umdrehen
    LinksDrehen
    LinksDrehen
  *Anweisung

  Umdrehen
  Schritt
  Umdrehen
*Anweisung
```

Bedingung ZweiZiegel

Dem Roboter Karol kann man auch eigene Bedingungen beibringen. Diesmal erkennt er ob genau zwei Ziegel vor ihm liegen.

```
Bedingung ZweiZiegel
  falsch
  wenn IstZiegel dann
     Aufheben
     wenn IstZiegel dann
        Aufheben
        wahr
        wenn IstZiegel dann falsch *wenn
        Hinlegen
     *wenn
     Hinlegen
  *wenn
*Bedingung
```

Diese Programm könnte man einfacher lösen, wenn man Bedingungen mit Parameter verwendet, aber das zeigen wir später (z.B. IstZiegel(2)).

Aufgabe: Schreibe die Bedingung „ZweiZiegelHinten“, welche die Frage beantwortet ob zwei Ziegel hinter Karol liegen.

Ein Schwimmbad bauen

In diesem Beispielprogramm kommen die Anweisungen schnell und langsam zur Anwendung und es wird das Programm in viele einzelne eigene Anweisungen zerlegt.

```
{ Programm: Ein Schwimmbecken bauen
Karol soll in der linken, hinteren Ecke stehen
mit Blick nach Vorne = Süden }

// Anweisung Becken bauen
Anweisung BaueBecken
   schnell
   wiederhole 12 mal
     solange NichtIstWand tue
       Hinlegen
       Schritt
     *solange
     LinksDrehen
   *wiederhole
   langsam
*Anweisung
```

```
// Anweisung Becken abreissen
Anweisung AbbauenBecken
   schnell
   wiederhole 12 mal
     solange NichtIstWand tue
       Aufheben
       Schritt
     *solange
     RechtsDrehen
   *wiederhole
   langsam
*Anweisung

// Anweisung Becken durchschwimmen
Anweisung Schwimmen

    // lokale Anweisung
    Anweisung Umdrehen
       LinksDrehen
       LinksDrehen
    *Anweisung

  // Schwimmkörper bauen
   wiederhole 3 mal Hinlegen *wiederhole
   Schritt
   solange NichtIstZiegel tue
     wiederhole 3 mal Hinlegen *wiederhole
     Schritt
     Umdrehen
     wiederhole 3 mal Aufheben *wiederhole
     Umdrehen
   *solange
   Schritt
   Umdrehen
   wiederhole 3 mal Aufheben *wiederhole
   Umdrehen
*Anweisung


// Definition des Hauptteils
Anweisung Hauptteil
   BaueBecken
   { zur Mitte bewegen }
   LinksDrehen
   wiederhole 2 mal Schritt *wiederhole
   RechtsDrehen
    { jetzt hinüber schwimmen }
   Schwimmen
   { zur Ecke bewegen }
   RechtsDrehen
   wiederhole 2 mal Schritt *wiederhole
   RechtsDrehen
```

```
   AbbauenBecken
   { zurück zur Ausgangsposition }
   solange NichtIstWand tue Schritt *solange
   LinksDrehen LinksDrehen
*Anweisung

{ ************************************* }
{ ********** Programmanfang  ************ }
{ ************************************* }
Programm
   wiederhole 4 mal
      Hauptteil
   *wiederhole
*Programm
{ *********** Programmende ********** }
```

Rekursion - Stapel verlegen

Dieses Programm zeigt die Verwendung der Rekursion. Diese Aufgabe wäre sicher auch ohne sie zu lösen (sogar einfacher). Es gibt aber Fälle, bei denen die Rekursion unvermeidlich ist.

```
{ Aufgabe: Karol steht vor einem Stapel Ziegel.
Schreibe ein Programm, welches Karol beibringt
den ganzen Stapel von vorne nach hinten
umzusetzen }

Anweisung PfeilerSetzen
  wenn IstZiegel dann
     Aufheben
     PfeilerSetzen
  sonst
     LinksDrehen
     LinksDrehen
  *wenn
  Hinlegen
*Anweisung

Programm
  PfeilerSetzen
  Aufheben
*Programm
```

Stapel verlegen ohne Rekursion

Wir schreiben das vorherige Programm ohne Verwendung der Rekursion. Beobachte den Unterschied vor allem darin, auf welche Art Karol den Stapel überträgt.

```
solange IstZiegel tue
  Aufheben
  LinksDrehen
  LinksDrehen
  Hinlegen
  LinksDrehen
  LinksDrehen
*solange
```

Schachbrett

Das Beispielprogramm legt im ganzen Raum (Dimension 6*10) Marken nach einem Schachbrettmuster aus. Felder werden mit dem Befehl MarkeSetzen markiert und mit dem Befehl MarkeLöschen wird eine Markierung entfernt.

```
{ Karol zeichnet ein Schachbrettmuster.
Er soll links hinten starten mit Blick
nach vorne }

Anweisung MarkiereZeile
  solange NichtIstWand tue
    MarkeSetzen
    Schritt
    Schritt
  *solange
*Anweisung

wiederhole 3 mal
  MarkiereZeile
  LinksDrehen Schritt LinksDrehen
  MarkiereZeile
  RechtsDrehen Schritt Rechtsdrehen
*wiederhole
```

Aufgabe: Schreibe ein Programm, welches Karol beibringt das Schachbrettmuster in einem Raum beliebiger Größe auszulegen.

Parameter bei Anweisungen

Manche Anweisungen und Bedingungen kann man mit Parameter erweitern. Zum Beispiel ergibt die Bedingung „IstZiegel(3)“ wahr, wenn vor Karol genau 3 Ziegel aufeinander stehen. Bei der Anweisung Schritt(2) macht Karol 2 Schritte.

```
solange IstZiegel(3) tue
  Schritt(2)
*solange
```

Aufgabe: Schreibe ein Programm, bei dem Karol im ganzen Raum umhergeht und alle Stapel markiert, die aus genau zwei Ziegel bestehen.

Wir addieren Zahlen

Dieses Beispielprogramm zeigt die Anwendung vieler Befehle der Sprache Karol. Das Programm überträgt Ziegel für Ziegel von einem Stapel auf einen anderen (die Anzahl Ziegel ist dann die Summe der beiden). Wenn sich an dem Stapel, auf den die Ziegel gelegt werden, mehr als 10 ergeben tritt ein Übertrag ein. Dabei setzt Karol eine Marke auf das Feld unter sich, wodurch er sich merkt, dass er beim nächsten Umsetzen einen Ziegel zusätzlich obendrauf legen muss. Es empfiehlt sich beim Programmablauf zwischen 3D- und 2D-Darstellung hin und her zu schalten.

```
Anweisung Umdrehen;
  schnell
  LinksDrehen
  LinksDrehen
  langsam
*Anweisung

{ Eine Marke ist 10 Ziegel wert }
Anweisung PrüfeÜbertrag
     wenn IstZiegel(10) dann
       wiederhole 10 mal Aufheben *wiederhole;
       MarkeSetzen
     *wenn
*Anweisung

{ Einen Stapel auf den zweiten versetzen. Wenn
sich ein "Übertrag" ergibt, wird eine Marke
gesetzt }
Anweisung Versetzen
   solange IstZiegel tue
     Aufheben
     Umdrehen
     Hinlegen

     PrüfeÜbertrag
```

```
    Umdrehen
   *solange
*Anweisung

{ ergibt wahr, wenn rechterhand von Karol eine
Wand ist }
Bedingung IstRechtsWand
  schnell
  RechtsDrehen
  wenn IstWand dann
    wahr
  sonst
    falsch
  *wenn
  LinksDrehen
  langsam
*Bedingung

Anweisung Rechnen
  //wiederhole 9 mal
  solange nicht IstRechtsWand tue
    Versetzen
    RechtsDrehen

    wenn IstMarke dann
      MarkeLöschen
      Schritt
      RechtsDrehen
      Hinlegen

      PrüfeÜbertrag

      Umdrehen
    sonst
      Schritt
      LinksDrehen
    *wenn
  *solange
*Anweisung

// Hauptprogramm
Programm
  { Karol zur richtigen Seite drehen }
  solange nicht IstOsten tue
    LinksDrehen
  *solange
  { Summe berechnen }
  Rechnen
*Programm
```

Aufgabe: Schreibe das Programm so um, dass es die Summe im 8-er System berechnet (es sind nur einige Änderungen nötig).

Wir gehen durch ein Labyrinth

Das ist das Schlußbeispiel. Es zeigt eine Vielzahl der Eigenschaften der Sprache Karol. Viele wurden schon in den vorherigen Beispielen besprochen, neu sind die Anweisungen Ton, Warten und Beenden.
Bei dem Durchgang durch das Labyrinth verwenden wir die Rekursion - damit Karol weiß, wohin er zurückkehren soll (In diesem Labyrinth wird das Mauerwerk durch zwei aufeinanderliegende Ziegel dargestellt, das Ziel durch einen einzelnen Ziegel).

```
{ Karol dreht sich um 180 Grad }
Anweisung Umdrehen
  LinksDrehen
  LinksDrehen
*Anweisung

{ Karol geht einen Schritt Rückwärts }
Anweisung SchrittRückwärts
  Umdrehen
  Schritt
  Umdrehen
*Anweisung

{*****************************************
** Bedingung gibt wahr zurück, wenn vor Karol
** keine Wand, keine Marke, keine Mauer
** (=2 Ziegel) und nicht das Ziel ist.
*****************************************}
Bedingung IstGehenErlaubt
  { gibt wahr, wenn vor Karol eine Marke ist }
  Bedingung IstMarkeVorne
    Schritt
    wenn IstMarke dann
      wahr
    sonst
      falsch
    *wenn
    SchrittRückwärts
  *Bedingung

  // Hauptteil der Bedingung
  schnell
  falsch
  wenn NichtIstWand dann
    wenn NichtIstziegel dann
      wenn nicht IstMarkeVorne dann
        wahr
      *wenn
    *wenn
  *wenn
  langsam
*Bedingung
```

```
{*****************************************
**   Gibt wahr, wenn Karol das Ziel sieht
**   = 1 Ziegel
*****************************************}
Bedingung IstZiel
  schnell
  falsch
  wiederhole 4 mal
    wenn IstZiegel(1) dann
      wahr
    *wenn
    LinksDrehen
  *wiederhole
  langsam
*Bedingung

{*****************************************
** Anweisung, welche sich rekursiv aufruft und
** damit alle Gänge durchläuft
*****************************************}
Anweisung ZweigGehen
  //  alle möglichen Wege durchsuchen ...
  MarkeSetzen
  wiederhole 4 mal
    wenn IstGehenErlaubt dann
      Schritt
      schnell
      ZweigGehen
      { feststellen, ob Karol am Ziel ist,
          sonst zurückgehen  }
      wenn IstZiel dann
        { jetzt wartet Karol 2,5 Sekunden,
            gibt einen Ton, dreht sich zum
            Ziel,
            und beendet dann das Programm }
        Warten(2500);
        Ton;
        solange NichtIstZiegel(1) tue
          LinksDrehen
        *solange
        Schritt
        langsam
        Beenden
      sonst
        // zurückkehren
        schnell
        SchrittRückwärts
      *wenn
    *wenn
    LinksDrehen
    schnell
```

```
  *wiederhole
*Anweisung


{****************************************
************* Hauptprogramm **************
****************************************}
Programm
  schnell
  ZweigGehen
  { hierher kommt das Programm nur, wenn es
    Karol nicht gelungen ist das Ziel zu
    erreichen }
  wiederhole 2 mal
    Ton
    Warten(2000)
  *wiederhole
*Programm
```

Stand: 3.Mai 2003; U.Freiberger