Karel

**Eine Übersicht über verschiedene Entwicklungen,
die auf der Idee von „Karel, the Robot“ basieren.**

Arbeitsstand: Juni 2002

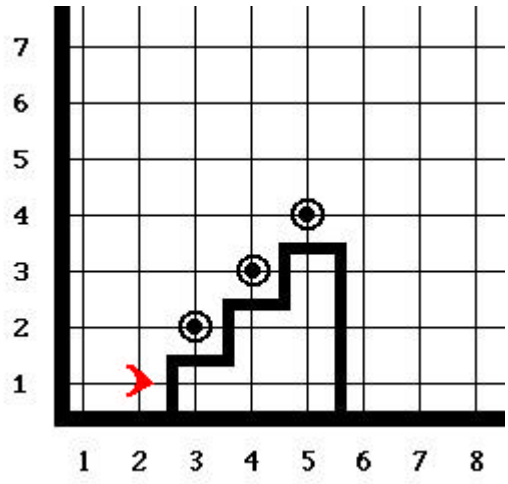
Ulli Freiberger
Luitpold-Gymnasium
München
freiberger@schule.bayern.de

Karel, the Robot

Die Grundidee für alle Programmierumgebungen und Veröffentlichungen zu diesem Thema entstammt dem Buch:

Titel: Karel the Robot, A Gentle Introduction to the Art of Programming
Authoren: Richard E. Pattis (2nd Edition), revised by Jim Roberts and Mark Stehlik
Herausgeber: John Wiley & Sons; 1981, 1995
ISBN: 0-471-59725-2
Abstract: Karel the Robot teaches you the fundamental concepts and skills of programming -- quickly and easily! By emphasizing logic and structure over calculation, it provides a non-threatening introduction to the central ideas in programming -- the same ideas that apply to all programming languages. This widely-praised guide begins by introducing Karel, a literal-minded robot whose built-in capabilities allow him to explore his world and manipulate simple objects in it. It then introduces Karel's programming language, which emphasizes logical deduction and spatial reasoning rather than calculation and algebraic reasoning.

Originalzitat: „The name Karel is used in recognition of the Czechoslovakian dramatist Karel Èapek, who popularized the word *robot* in his play *R.U.R.* (Rossum's Universal Robots). The word *robot* is derived from the Czech word *robota*, meaning "forced labor."



Die Idee ist, einen Roboter zu programmieren, der in einer „Bildschirmwelt" lebt. Wenn die Programme laufen, sehen die Benutzer 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.

Die Welt bei „Karel, the Robot" ist zweidimensional und besteht aus quadratisch angeordneten Straßen (vertikal Avenues und horizontal Streets), die nummeriert sind und sich kreuzen. In diese Welt können zwischen den Straßen Wände (dicke Linien) eingezogen werden, die Karel nicht überwinden kann und ihn daran hindern direkt von der einen zu der anderen Straße zu gehen. Karel, als roter Pfeil dargestellt, kann sich entlang der Straßen bewegen und

dabei „Beeper“ (doppelte Kreise) aufheben und hinlegen. Diese „Beeper“ kann er in einer Tasche mit sich tragen.

Karel wird mit einer Programmiersprache, die nur einen kleinen, überschaubaren Satz an Befehlen kennt gesteuert. Die zugrundeliegende Programmiersprache ist für den Einstieg in die Algorithmik bewusst einfach gehalten.
Die Programme benutzen eine einfache, klare Syntax und beinhalten:

- Vordefinierte Anweisungen: TurnOn, Move, TurnLeft, PickBeeper, PutBeeper, TurnOff
- Vordefinierte Bedingungen: frontIsClear, frontIsBlocked, leftIsClear, leftIsBlocked, facingNorth, notFacingNorth, facingSouth, notFacingSouth, facingEast, notFacingEast, facingWest, notFacingWest, nextToABeeper, notNextToABeeper, anyBeepersInBeeperBag, notAnyBeepersInBeeperBag
- Kontrollstrukturen: if then, if then else, iterate times, while do
- Benutzerdefinierte Anweisungen sind möglich

Pattis selbst hat in seinem Buch „Bergin, Joseph and Mark Stehlik and Jim Roberts and Richard Pattis; Karel++: A Gentle Introduction to the Art of Object-Oriented Programming; New York: John Wiley and Sons, Inc., 1997“ die ursprüngliche Konzeption auf die Objektorientierte Sicht ausgeweitet. Siehe <www.csis.pace.edu/~bergin/karel.html>.

Diese Idee zur Einführung in die Programmierung wurde in vielen Arbeiten aufgegriffen. Die Folgeprodukte lassen sich in vier Kategorien unterteilen:

- zweidimensionale Welten, die sich streng an der Idee von Pattis orientiert; manchmal ist Karel ein Pfeil, andere Autoren verwenden roboterartige Figuren (in Draufsicht); meist gleicher Befehlsumfang wie bei Pattis;
- abgeänderte zweidimensionale Welten mit deutlich anderen Karel-Figuren wie Hamster, Marienkäfer oder ähnliches; erweiterte Objekte und Befehle;
- dreidimensionale Welten mit zusätzlichen Steuermöglichkeiten;
- Konzepte, die nicht die Algorithmik zur Grundlage haben, sondern andere Modellierungstechniken der Informatik wie z.B. Zustandsorientierte Modellierung.

2D-Karel-Welten

R.Untch

Dr. Roland H. Untch, Middle Tennessee State University, beschreibt ausführlich die Idee und Konzeption des ursprünglichen „Karel, the Robot“ und bringt dabei viele Beispiele. Die Karel-Welt entspricht dem Originalaufbau wie bei Pattis mit quadratisch angeordneten Starßen. Untch bietet eine Realisierung der Karel-Befehle unter Verwendung der Sprache C. Die Karel-Befehle sind in einem Headerfile definiert, das am Programmanfang eingebunden werden muss.

Als Anweisungen stehen zur Verfügung: TurnOn(), Move(), TurnLeft(), PickBeeper(), PutBeeper(), TurnOff()

TurnOn muss die erste Anweisung im Programm sein und TurnOff die letzte.

Vordefinierte Bedingung sind: frontIsClear, frontIsBlocked, leftIsClear, leftIsBlocked, rightIsClear, rightIsBlocked, facingNorth, notFacingNorth, facingSouth, notFacingSouth, facingEast, notFacingEast, facingWest, notFacingWest, nextToABeeper, notNextToABeeper, anyBeepersInBeeperBag, noBeepersInBeeperBag

Der Programmcode ist in C-Syntax zu erstellen, selbstdefinierte Anweisungen werden als C-Funktionen definiert, die Kontrollstrukturen werden direkt von C verwendet und die Compilierung muss mit einem C-Compiler erfolgen. Einzig die Wiederholung mit fester Anzahl [iterate(n)] wurde zur üblichen C-Syntax ergänzt.

Näheres unter <www.mtsu.edu/~untch/karel/>. Zitat: „By initially limiting the student's language repertoire to easily grasped imperative commands whose actions are visually displayed, the Karel approach quickly introduces students to such concepts as procedures and the major control structures. Although originally based on Pascal, the Karel approach has been used successfully with several different computer programming languages. These pages describe a version of Karel that uses the C / C++ language.“

```
#include "karel.h"
int main()
{
    Move();
    PickBeeber();
    TurnLeft()
    PutBeeper();
    Move;
    }
}
```

```
void PickBeepersToWall()
{
    if (nextToABeeper)
        do
        {
            PickBeeper();
        } while (nextToABeeper)
    while (frontIsClear)
    {
        Move();
        if (nextToABeeper)
            do
            {
                PickBeeper();
            } while (nextToABeeper)
    }
}
```

B.Wachsmuth

Weitere, umfangreiche Ausführungen beschreibt Bert G. Wachsmuth, Seton Hall University, auf den Seiten <pirate.shu.edu/~wachsmut/Teaching/CSAS1111/ Notes-Karel/ > und </karel1.html> bis </karel3.html>. An der Seton Hall University wurde „Karel the Robot“ 1995 im Anfangssemester zu „Introduction to Computer Science“ eingesetzt. Wachsmuth hält sich sehr eng an das Buch von Pattis. Er verwendet die Software von Pattis, die in USA dem Buch in der 2.Auflage beiliegt (August 1994). Dieses Programm ist unter MSDOS im Textmodus lauffähig. Die Programmierumgebung ist sehr umständlich zu bedienen, voll Textkommando orientiert und bietet nur beschränkte Ausgabemöglichkeiten. Die Welt wird im Textmodus mit Sonderzeichen wie . - ! + usw. dargestellt und die „Beeper“ mit Ziffern bzw. O angezeigt.
Die hier </manual.html> beschriebene Originalsprache von Pattis ist Pascal angenähert und hat folgenden Umfang:

- Anweisungen: move, turnleft, pickbeeper, putbeeper, turnoff
- Bedingungen: front-is-blocked, front-is-clear, left-is-blocked, left-is-clear, right-is-blocked, right-is-clear, next-to-a-beeper, not-next-to-a-beeper, any-beepers-in-beeper-bag, no-beepers-in-beeper-bag, facing-north, not-facing-north, facing-south, not-facing-south, facing-west, not-facing-west, facing-east, not-facing-east
- Kontrollstrukturen: if then, if then else, iterate .. times, while .. do; Blockbildung mit begin end
- selbstdefinierte Anweisungen sind möglich mit: define-new-instruction .. as
- das Programm muss mit beginning-of-program starten und mit end-of-program enden; der eigentliche Hauptteil muss von beginning-of-execution und end-of-execution eingefasst werden.

Auf der Seite <.../CSAS1111/Notes-Pascal/pascal1.html> beschreibt Wachsmuth den Vergleich von Karel und Pascal und führt dann weiter in Pascal als Programmiersprache ein.

R. Hasker

Dr. Robert W. Hasker bietet an der University of Wisconsin, Platteville eine Vorlesung zur Einführung in die Programmierung an (CS113). Er verwendet dabei in den ersten 9 Wochen eine Karel-Implementation auf der Basis von C++ (ähnlich zu Untch). Hasker führt schon sehr früh bei den Bedingungen boolsche Operatoren ein und benutzt am Ende der Einführung auch Variable (Deklaration, Zuweisung, Terme). Der Kurs ist detailiert beschrieben unter <http:// baobab.cs.uwplatt.edu/classes/cs113/ >. Nach der Karel-Einführung wird der Kurs mit C++ fortgeführt.

M.Moore

Eine Umsetzung für Unix-Systeme wurde 1994 von Michael D. Moore, Ohio State University, durchgeführt. Näheres siehe <www.geocities.com/SiliconValley/Platform/3173/ karel.html>.
„This is the first official internet release of Karel the Robot, a graphical UNIX-based IDE (Integrated Development Environment) for the programming language of the same name (originally deisgned by Richard E. Pattis). This release contains binary executables for the three major platforms we have here: SunOS4.1.3, Solaris 2.3, and HP/UX 9.x. The package also requires a common library directory where the scripts and configuration files are kept. A source distribution is also available for those bold adventurers who want to try to compile this beast themselves.“

T.Mitchell

Eine neue Auflage (Juli 2000) dieser Version, die auf der Basis von Java unter UNIX lauffähig ist liefert als Opensource-Code Tom Mitchell bei <karel.sourceforge.net/>.
Der Sprachumfang ist:

- Anweisungen: move, turnleft, pickbeeper, putbeeper, turnoff
- Bedingungen: front-is-blocked, front-is-clear, left-is-blocked, left-is-clear, right-is-blocked, right-is-clear, next-to-a-beeper, not-next-to-a-beeper, any-beepers-in-beeper-bag, no-beepers-in-beeper-bag, facing-north, not-facing-north, facing-south, not-facing-south, facing-west, not-facing-west, facing-east, not-facing-east
- Kontrollstrukturen: if then, if then else, iterate .. times, while .. do; Blockbildung mit begin end
- selbstdefinierte Anweisungen sind möglich mit: define-new-instruction

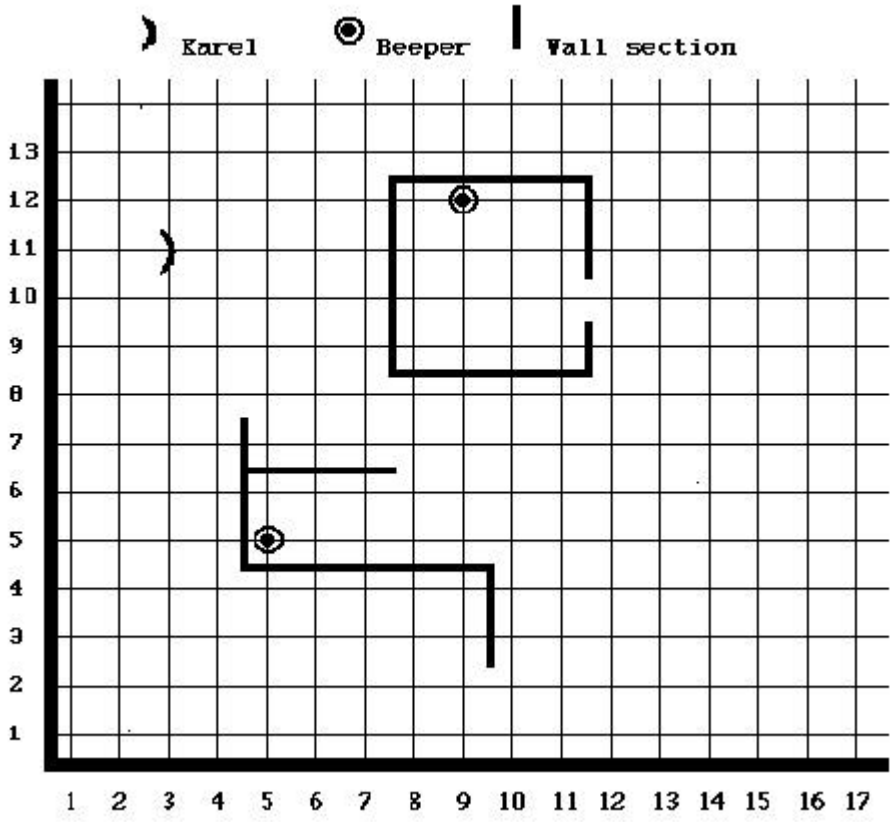
Einen passenden Einführungskurs bietet <www.a2w.net/introcs/karel-unit/>.
Eine schöne Bedienoberfläche zu der Mitchell-Version entwickelte die TREW-Gruppe <www.calpoly.edu/~sholzer/index.html>.

J. Wilson

Eine Standardversion des Pattis-Konzepts für Unix mit Graphik ähnlich zu T.Mitchell.
Umgesetzt 1994 von Joseph N. Wilson, University of Florida,
<www.cise.ufl.edu/~jnw/Karel/>

Karel, the Robot in Pascal - J. Hunter

Hier sind die spezifischen Befehle von Karel als zusätzliche Prozeduren zum normalen Turbo-Pascal realisiert. Der Autor bietet eine Unit (csKarel), die in ein Pascal-Programm eingebunden werden muss. Diese Version von Karel findet man bei Jim Hunter, University of Aberdeen, unter <www.abdn.ac.uk/~csc111/cs_pas/notes/lectures/ch1.htm> (Oktober 1999).
Der Programmtext muss, wie ein normales Pascal-Programm in einem Editor mit genauer Pascal-Syntax eingegeben werden. Die ursprüngliche Idee von Pattis, die auf einem beschränkten Befehlssatz beruhte und keine Variablen verwendete, geht dabei verloren, da eine ganz normale vollständige Turbo-Pascal-Version zum Einsatz kommt. Der Autor weicht dabei von den Notationen von Pattis ab, gibt aber hierfür detaillierte Umsetzungen unter <www.abdn.ac.uk/~csc111/cs_pas/notes/misc/pattis.htm> an. Hunter führt zahlreiche Beispiele für Karel zusammen mit Lösungen aus, geht aber dann sehr bald vollständig zu Pascal über.
Die Welt ist, genau wie bei Pattis, zweidimensional mit Straßenkreuzungen. Zusätzlich enthält die Welt gesetzte Trennwände und abgelegte Beeper. Die grafische Darstellung der Welt ist sehr einfach gehalten.

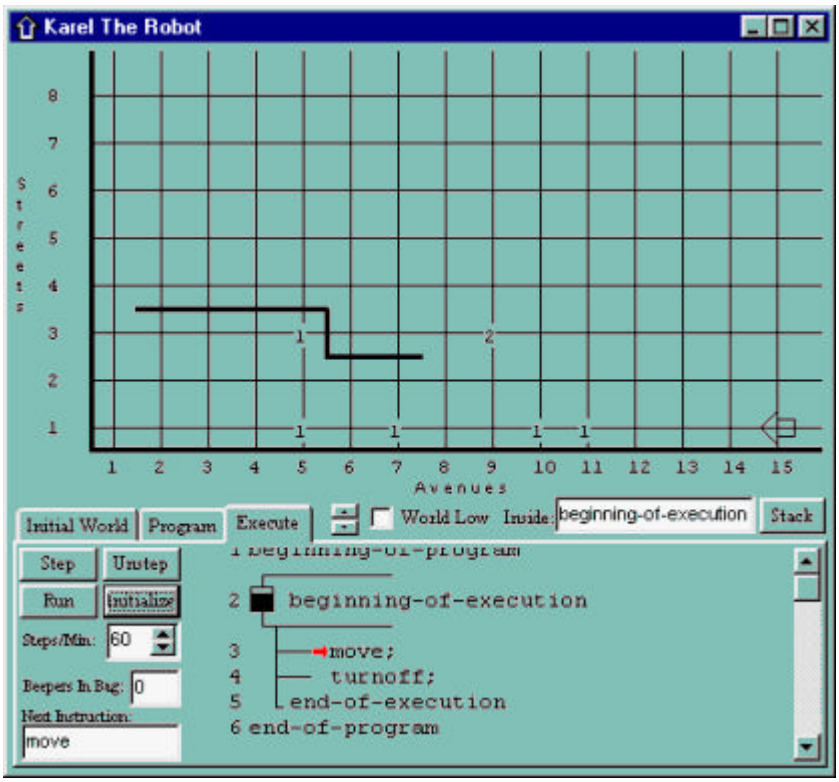


Duane Buck

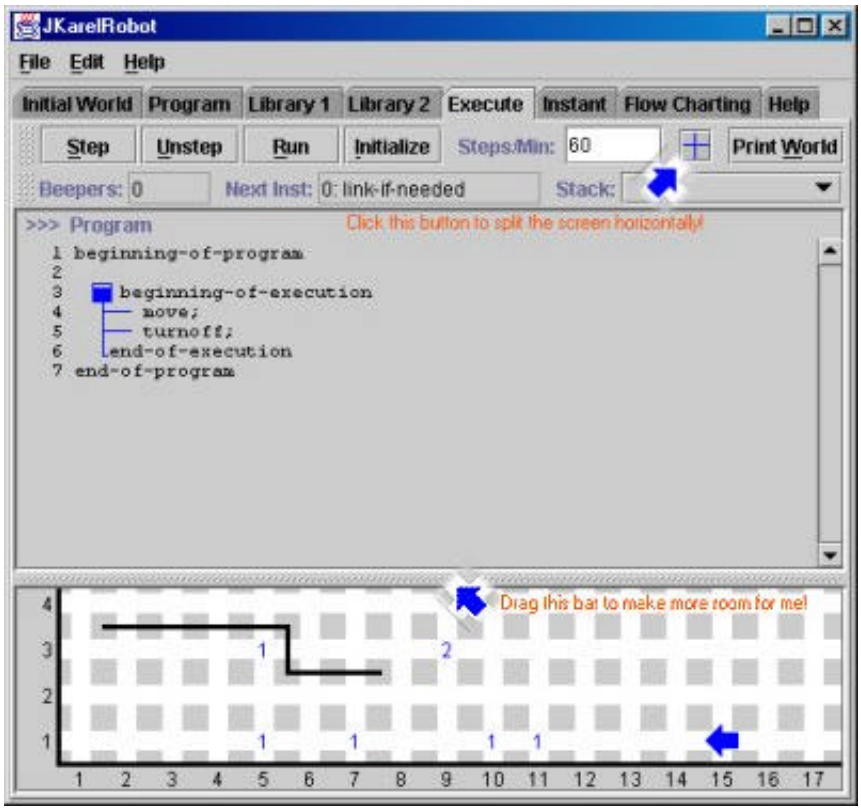
Diese Windows-Version wurde 1999 von Duane Buck, Departement of Mathematics and Computer Science, Otterbein College, Westerville Ohio entwickelt. Zu finden unter <math.otterbein.edu/Class/Csc100/Karel/web/Karel/Karel.htm>. Buck bietet eine vollständige Programmierumgebung mit einfach zu bedienendem Weltgenerator, einem Programmeditor und einer umfangreichen Steuerung des Programmablaufs. Nach dem Compilieren kann zusätzlich eine Programmdarstellung in strukturierter Form („Control Structure Diagram“) angezeigt werden. Karel wird in der Welt als Pfeil und die Beeper, je nach Anzahl, als Zahl dargestellt.
Der Sprachumfang ist:

- Anweisungen (primitive commands): move, turnleft, pickbeeper, putbeeper, turnoff
- Bedingungen (conditiones): front-is-blocked, front-is-clear, left-is-blocked, left-is-clear, right-is-blocked, right-is-clear, next-to-a-beeper, not-next-to-a-beeper, any-beepers-in-beeper-bag, no-beepers-in-beeper-bag, facing-north, not-facing-north, facing-south, not-facing-south, facing-west, not-facing-west, facing-east, not-facing-east
- Kontrollstrukturen (control statements): if .. then, if .. then .. else .., iterate .. times, while .. do; Blockbildung mit begin end
- selbstdefinierte Anweisungen sind möglich mit define-new-instruction

Ablauffehler (z.B. Aufheben ohne, dass ein Beeper am Boden liegt) führen zum Programmstopp.

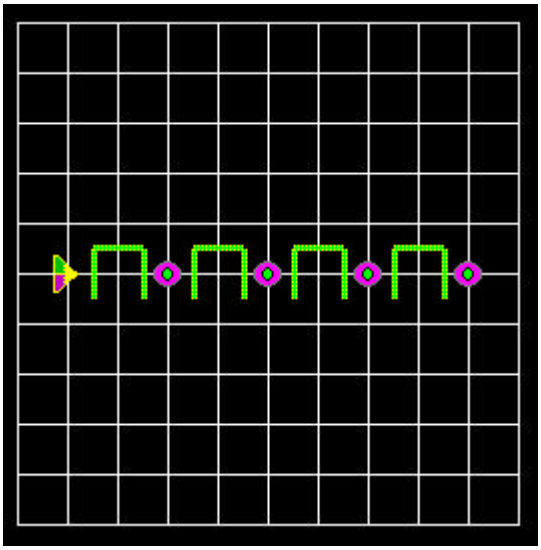


Inzwischen gibt es eine neue Programmierumgebung die mit Java entwickelt wurde <math.otterbein.edu/JKarelRobot/ParelTutorial/Karel.htm>. Die realisierte Karel-Sprache und die Karel-Welt sind jedoch identisch zur Vorgängerversion.



D.Barnett

Eine Version, die das Konzept von Pattis umsetzt, ist dieses DOS-Grafikprogramm. Es ist unter Windows 3.1, NT und Windows 95 lauffähig und wird von David.D.Barnett auf der Seite <home.att.net/~David.D.Barnett/karel-home.html> vorgestellt. (Shareware $35)



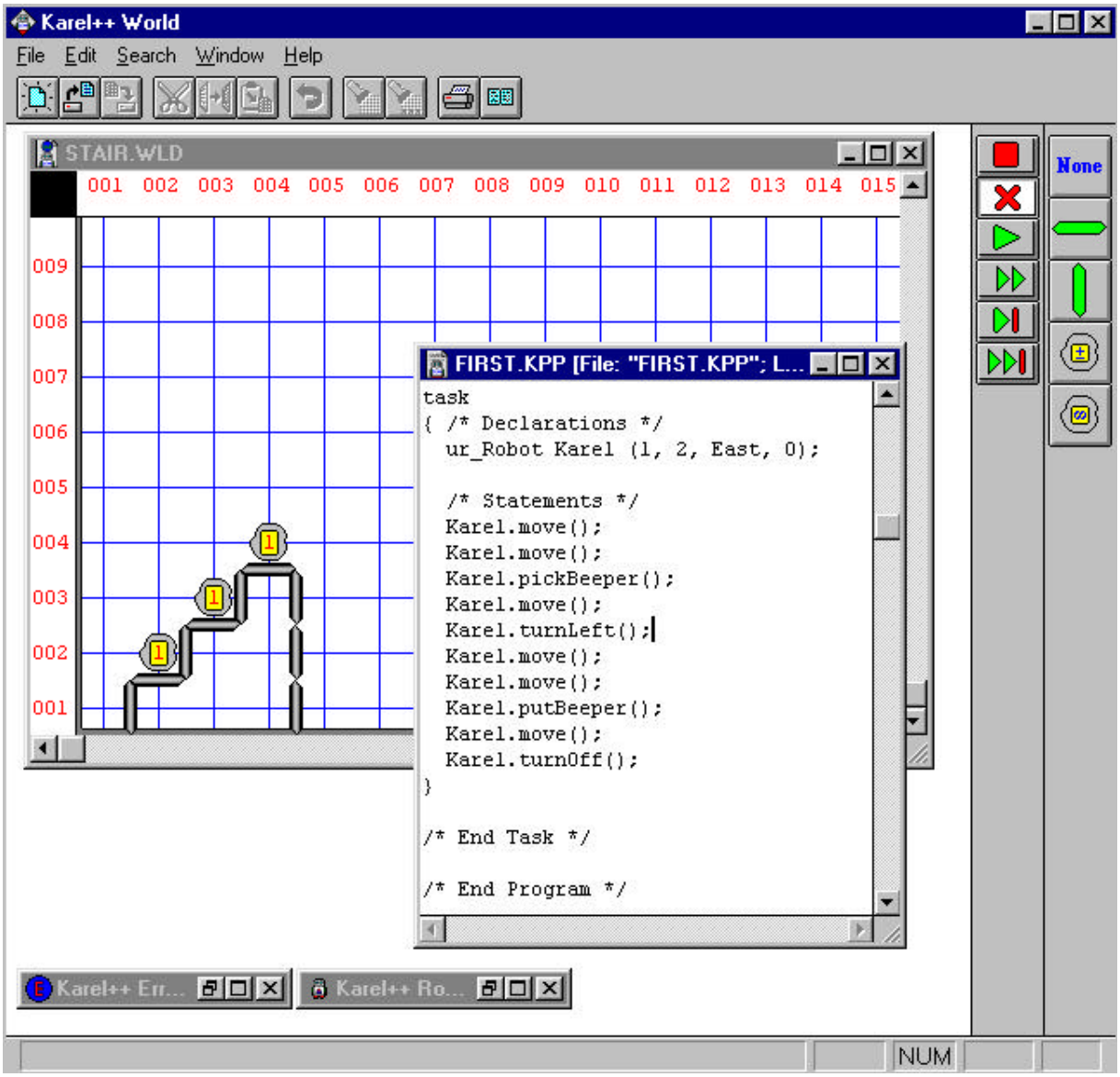
Auch hier besteht die Welt aus Straßenkreuzungen und verstreuten „Beepern“. Karel ist als Dreieck dargestellt. In der Welt können Wände aufgestellt werden. Eine direkte Steuerung ist möglich.
Der Sprachumfang ist gegenüber Pattis etwas erweitert:

- Anweisungen: TurnLeft, Move, PickBeeper, PutBeeper, TurnOff
- Bedingungen: front-is-blocked, front-is-clear, left-is-blocked, left-is-clear, right-is-blocked, right-is-clear, next-to-a-beeper, not-next-to-a-beeper, any-beepers-in-beeper-bag, no-beepers-in-beeper-bag, facing-north, not-facing-north, facing-south, not-facing-south, facing-west, not-facing-west, facing-east, not-facing-east
- Kontrollstrukturen: if then, if then else, iterate .. times, while .. do; Blockbildung mit begin end
- selbstdefinierte Anweisungen sind möglich: Define-New_Instruction *name* As ...

Die Karel-Programme können auch schrittweise ausgeführt werden, wobei die jeweilige Programmzeile im Listing markiert wird.

Karel++ World

Dieses neue Konzept bietet eine einfache, klare und übersichtliche Einführung in die Programmierung aus Objektorientierter Sicht. Es richtet sich stark nach dem Buch „Bergin, Joseph and Mark Stehlik and Jim Roberts and Richard Pattis; Karel++: A Gentle Introduction to the Art of Object-Oriented Programming; New York: John Wiley and Sons, Inc., 1997“.
Ein Windows-Programm hierzu wurde von Adrian Iley und Josh de Cesare entwickelt. Die Version 1.0d7 vom Dezember 1998 ist unter <www.csis.pace.edu/~bergin/temp/findkarel.html> zu finden. Ebenso gibt es eine Macintosh-Version.
Die Welt ist wie bei Pattis „Karel, the Robot“ aufgebaut, besteht aus Straßenkreuzungen an denen „Beeper“ liegen und die durch unüberwindbare Wände voneinander getrennt werden können.



Eine Anleitung zu dem Programm gibt es unter <www.csis.pace.edu/~bergin/KWorld/karelwin.html>. Die Programmierung verfolgt den objektorientierten Ansatz und ist stark an C++ angelehnt. Man definiert Objekte der Klasse ur_Robot, die schon über gewisse Fähigkeiten verfügen. Die Befehle sind als Methodenaufrufe des Objekts definiert. Es können neue Klassen definiert werden, die von der Klasse ur_Robot abstammen und zusätzlich neue Methoden bekommen. Diese neuen Methoden müssen in der C++-Syntax im Programmeditor festgelegt werden.
Der ur_Robot kennt die Methoden

- Befehle: move, turnOff, turnLeft, pickBeeper, putBeeper
- Abfragen: frontIsClear, nextToABeeper, nextToARobot, facingNorth, facingSouth, facingEast, facingWest, anyBeepersInBeeperBag
  zusätzlich existiert der Not-Operator !

Es können die Kontrollstrukturen loop(n), while, if else, verwendet werden.
Bei Karel++ sind auch mehrere Roboter gleichzeitig in der selben Welt möglich.

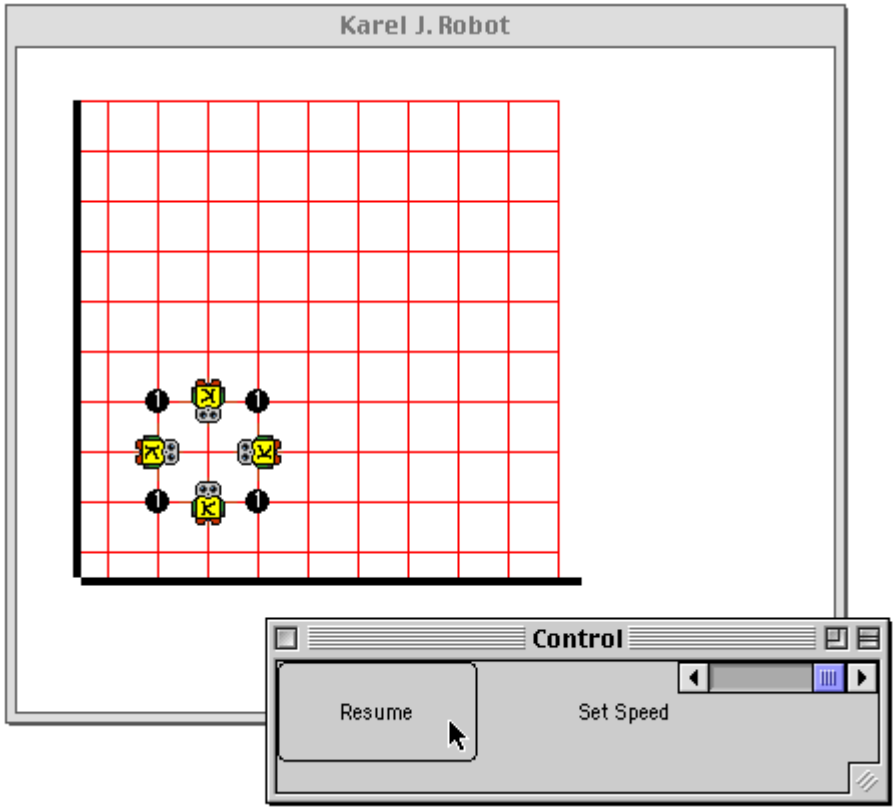
Ein gute und ausführliche Anleitung (in Englisch) hierzu stammt von Shane Torbert: „C++ Karel, Steps to Object Orientation, Karel the Robot, July 2001, Edited by Shane Torbert, Under the direction of Jerry Berry, Thomas Jefferson High School for Science and Technology Fairfax County Public Schools, Fairfax, Virginia“. Diese Arbeit und vieles mehr

ist im Web unter <www.tjhsst.edu/compsci/karel/> zu finden. Ebenso liegen an dieser Stelle viele passende Karel-Welten (<www.tjhsst.edu/compsci/karel/files/karel++worlds.zip>).

Gerry Bilodeau bietet in <fc.bchigh.edu/~gebilodeau/Karel/> eine kommentierte Einführung in Karel++ und entsprechende Folien.

Karel J. Robot

Ist die analoge Entwicklung wie Karel++ World jedoch jetzt in Java eingebunden. Zum Start der Programmieroberfläche und zum Ausführen der Karel-Programme benötigt man den Java-Developer-Kit JDK in der Version 1.2 (SDK und JRE). Diese Entwicklung wurde von Joseph Bergin vorangetrieben, der hierzu eine Java-Bibliothek KarelLib.jar zur Verfügung stellt. Ein Manuskript des nicht veröffentlichten Buches „Karel J. Robot, A Gentle Introduction to the Art of Object-Oriented Programming in Java, Joseph Bergin and Mark Stehlik and Jim Roberts and Richard Pattis“ ist bei <www.csis.pace.edu/~bergin/KarelJava/Karel++JavaEdition.html> abgelegt. Hier findet sich auch die zugehörige Java-Bibliothek <www.csis.pace.edu/~bergin/KarelJava/sim/>.
Weiteres hat Bergin in einem Didaktikvortrag zur Einführung in die objektorientierte Programmierung ausgearbeitet <csis.pace.edu/~bergin/karel/ecoop2000JBKarel.html>.



Die Programmierung erfolgt in Java mit einem Texteditor. Es steht eine vordefinierte Roboter-Klasse ur_robot zur Verfügung. Vor der Ausführung muss das Karel-Programm zusammen mit den Bibliotheken mit javac compiliert werden.
Die Klasse ur_Robot kennt die Methoden („All robots produced by Karel-Werke have at least the capabilities just described.“):

- Anweisungen, Methoden („actions“): move(), turnOff(), turnLeft(), pickBeeper(), putBeeper()

- Abfragen: frontIsClear, nextToABeeper, nextToARobot, facingNorth, facingSouth, facingEast, facingWest, anyBeepersInBeeperBag
  zusätzlich existiert der NOT-Operator !
- Es gibt die Kontrollstrukturen loop(n), while, if else

Im Programm wird ein Objekt der Klasse ur_Robot erzeugt, mit Anfangswerten versehen und dann dessen Methoden aufgerufen. Zum Beispiel das Objekt karel :

```
task
{
        ur_Robot Karel = new ur_Robot(1, 1, East, 0);
        Karel.move();
        Karel.move();
        Karel.pickBeeper();
        Karel.turnOff();
}
```

Es sind mehrere Objekte in einem Programm möglich, ebenfalls können neue Klassen definiert werden, die von der Klasse ur_Robot abstammen.

Auch hierzu gibt es die bewährten Ausarbeitungen von Shane Torbert: Java, Steps to Object Orientation Karel J. Robot, July 2001, Edited by Shane Torbert, Ankur Shah, Greg W Price, Under the direction of Jerry Berry, Thomas Jefferson High School for Science and Technology, Fairfax County Public Schools, Fairfax, Virginia . Diese Arbeit und vieles mehr ist im Web unter <www.tjhsst.edu/compsci/karel/> zu finden. Passende Karel-Welten und Karel-Programme sind ebenso unter dieser Adresse vorhanden <www.tjhsst.edu/compsci/karel/files/KarelJWorlds.zip> bzw. <KarelJFiles.zip>.

Auf der Arbeit von Bergin setzt der Simulator von Christoph Bockisch, Technische Universität Darmstadt, November 2001 auf, der unter <www.st.informatik.tu-darmstadt.de/lehre/ws01/inf1/karelj/welcome.html> vorgestellt wird.

W.Becker

Eine ähnliche Arbeit wie Karel J. Robot. Sie bietet eine einfache Einführung in die Art der objektorientierten Programmierung mit Java. Erstellt von Byron Weber Becker, <www.math.uwaterloo.ca/~bwbecker/karel/> (August 2001) zusammen mit einem Einführungslehrgang. Die Applikation karel.jar benötigt JRE 1.2 .

Michael Winikoff

Eine Standard-Version als Ergänzung zu der Sprache PocketC für den Palm Pilot. Von Michael Winikoff, michael@winikoff.net zu finden unter <www.winikoff.net/palm/Karel> bzw. <goanna.cs.rmit.edu.au/~winikoff/palm/Karel/>.
Zitat: „This implementation takes the form of a collection of library procedures for PocketC. You write PocketC programs using these procedures and the library provides the display (as well as a world editor). Since programs are compiled with PocketC you'll need the full version of PocketC, not the runtime only version.“

Karel Python

Eine Python-Implementation von Karel the Robot, entwickelt von Steve Howell (November 2001) <www.showell.westhost.com/karel/>

Andere 2D-Karel-Welten

Niki Roboter

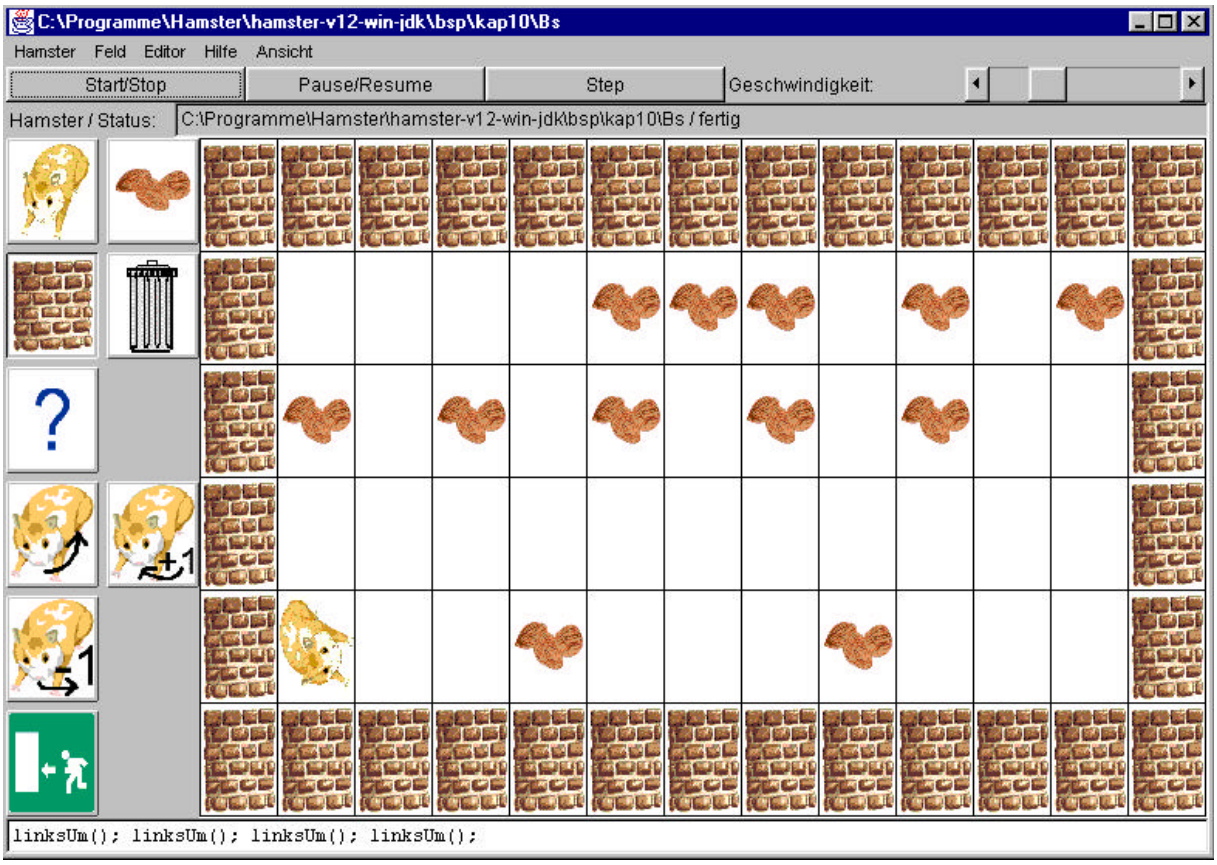
In ihrem Schulbuch Informatik Eins, Erst Klett Verlag, Stuttgart 1988 stellen Alfred Hermes und Dieter Stobbe ihre Programmierumgebung „Niki Roboter" vor, mit der variablenfrei in die Algorithmik eingeführt wird. Sie arbeitet mit dem textorientierten Bildschirm von MS-DOS.
Das Arbeitsfeld von Niki besteht aus quadratisch angeordneten Punkten in einem zweidimensionalen Koordinatensystem. Niki wird als Pfeil dargestellt. Er kann Scheiben ablegen und aufnehmen und es können an einem Punkt der Welt bis zu 9 Scheiben liegen. Zusätzlich können in der Welt Hindernisse eingebracht werden, die für Niki unüberwindbar sind. Die Anzahl an Scheiben die Niki transportiert wird beim Programmablauf kontrolliert. Nicht ausführbare Anweisungen (z.B. stößt gegen die Wand) führen zum Abbruch des Niki-Programms. Die Notation der Sprache ist an Pascal angelehnt und benutzt für die Kontrollstrukturen englische Bezeichner, für die Anweisungen („Befehle“) und Bedingungen („Sensorfunktionen“) von Niki jedoch deutsche. Die Sprache umfasst:

- vordefinierte Anweisungen: drehe_links, nimm_auf, gib_ab, vor
- vordefinierte Bedingungen: vorne_frei, links_frei, rechts_frei, Platz-Belegt, hat_Vorrat
- Kontrollstrukturen: if then, if then else, while do, repeat until und begin, end zur Blockbildung
- selbstdefinierte Anweisungen sind möglich

Das Programm muss, ohne Hilfen, in einem Wordstar-ähnlichen Texteditor eingetippt werden.

Hamster

Die deutsche Programmierumgebung wurde an der Universität Oldenburg unter der Federführung von Dietrich Boles (E-Mail: boles@informatik.uni-oldenburg.de) entwickelt. Sie ist in der Version 1.2.2 (Stand Februar 2000) unter <www-is.informatik.uni-oldenburg.de/~dibo/hamster> erhältlich und benötigt zum Starten und Arbeiten den Java-Developer-Kit JDK in der Version 1.1 oder 1.2. Zusätzlich stehen im Internet eine ausführliche Anleitung und Beispiele bereit.
Zitat aus der Anleitung: „Das Java-Hamster-Modell ist ein einfaches aber mächtiges Modell, mit dessen Hilfe Programmieranfänger die Grundkonzepte der Programmierung auf spielerische Art und Weise erlernen können. Sie als Programmierer steuern einen virtuellen Hamster durch eine virtuelle Landschaft und lassen ihn bestimmte Aufgaben lösen. Das Modell orientiert sich dabei an den Konzepten der Programmiersprache Java. Das Hamster-Modell wird ausführlich in meinem Buch "Programmieren spielend gelernt - mit dem Java-Hamster-Modell" beschrieben, das im Teubner-Verlag erschienen ist.“



Der „Roboter“ ist ein kleiner Hamster. Seine Welt („Hamster-Territorium“) ist ein zweidimensionales, quadratisches Gitter in dem sich der Hamster von Feld zu Feld („Kachel“) bewegen kann (entweder im Direktmodus oder durch Programm). Als platzierbare Objekte stehen Körner und Wände zur Verfügung. Die Wände kann der Hamster nicht überwinden. Die Körner kann er einsammeln und über die Anzahl der Körner „in den Backen“ wird Buch geführt. Durch Direktsteuerung kann eine Hamster-Welt definiert werden, ebenso der Hamster bewegt bzw. gefüttert werden.
Das Hamster-Programm wird in einer Java-Notation in einem gesonderten Texteditor erfasst. Die Kontrollstrukturen werden in Java-Syntax notiert und die selbstdefinierten Anweisungen werden als Java-Funktionen definiert. Auch die vordefinierten Anweisungen sind als Java-Funktionen realisiert. Der Befehlsumfang für den Hamster umfasst die üblichen Möglichkeiten:

- vordefinierte Anweisungen: vor(), linksum(), gib(), nimm()
- vordefinierte Bedingungen: vornfrei(), kornDa(), maulLeer()

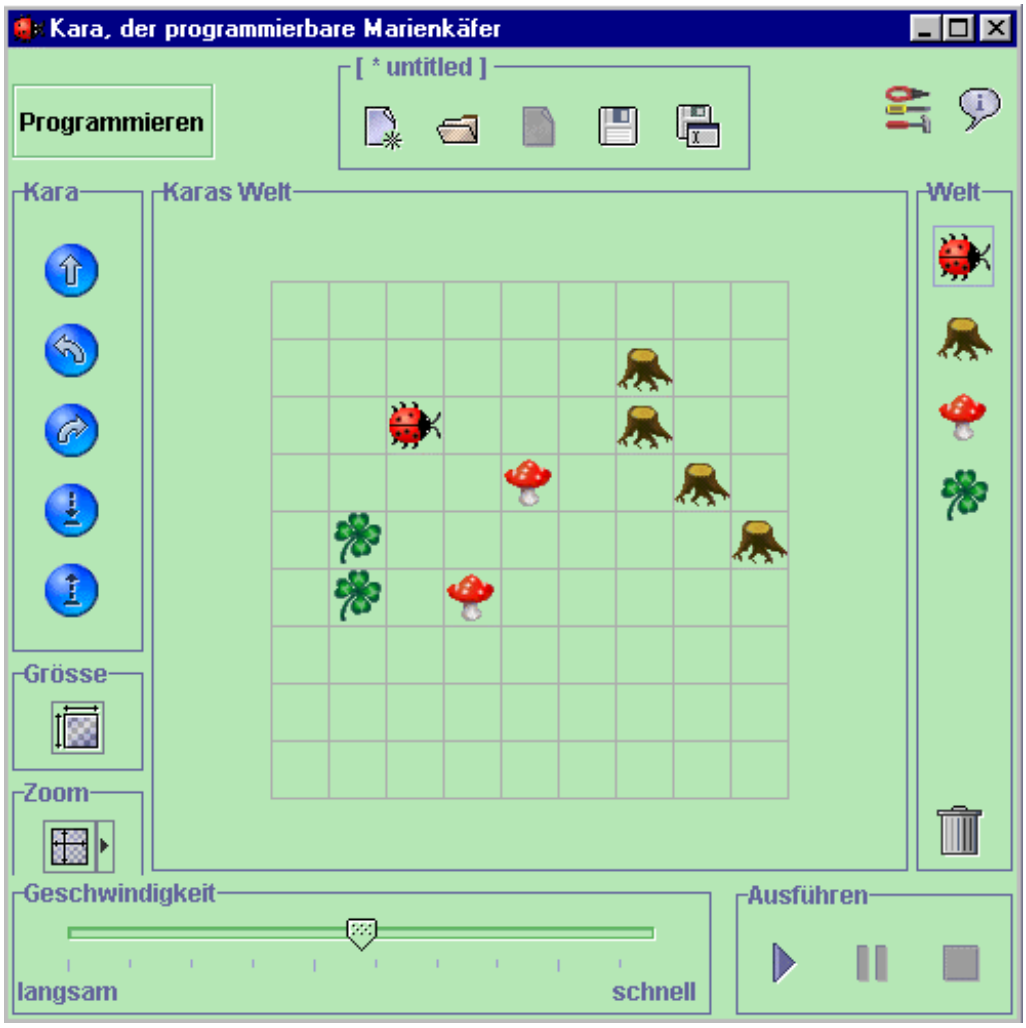
Vor dem Ablauf muss das Hamster-Programm mit dem Java-Compiler Javac compiliert werden. Compiler-Fehlermeldungen dabei erfolgen in Englisch. Falsche, nicht ausführbare Anweisungen an den Hamster „führen zum Tod des Hamsters“. Beim schrittweisen Programmablauf wird nicht die jeweilige Programmzeile angezeigt (auch nicht bei Einzelschritt), sondern nur der jeweils ausgeführte Befehl.

JavaKara

Bei JavaKara wird die Figur Kara, ein Marienkäfer, durch Programme gesteuert, die in Java erstellt werden. Entwickelt von Raimond Reichert, Markus Brändle, Reto Lamprecht; ETH Zürich. Die aktuelle Version (Juni 2002) der Programmieroberfläche javakara.jar ist unter <www.educeth.ch/informatik/ karatojava/javakara/> zu finden. Dort liegen auch umfangreiche Beispiele und ein gut dokumentierter Lehrgang zur Einführung in Java mit

JavaKara vor. Zum Schreiben eigener JavaKaraProgramme und zum Benutzen der JavaKara-Oberfläche benötigt man den Java-Developer-Kit JDK (J2SE, SDK und JRE) in der Version 1.2 (ist sowohl unter Windows/MacOS und Unix lauffähig).
Zitat: „Mit JavaKara bieten Sie Ihren Schüler/innen einen einfachen Einstieg in die Programmierung mit Java: Sie können Kara, den Marienkäfer, in Java programmieren! Damit haben Sie einen perfekten spielerischen Einstieg in die Java-Programmierung. Der Vorteil beim Programmieren mit JavaKara: die Schüler/innen sehen sofort, was ihre Programme tun.
Folgendes Programmfragment sammelt zum Beispiel alle Kleeblätter bis zum nächsten Baum ein:

```
while (!kara.treeFront()) {
  if (kara.onLeaf())
  kara.removeLeaf();
  kara.move();
  }
```



Kara lebt in einer rechteckigen Welt, die aus einzelnen quadratischen Feldern besteht. Er kann sich nur von Feld zu Feld bewegen. Auf den Feldern in dieser Welt gibt es

- unbewegliche Baumstümpfe. Wenn Kara in einen Baumstumpf läuft, beschwert er sich. Auf einem Feld mit einem Baumstumpf kann kein anderer Gegenstand liegen.
- verschiebbare Pilze. Läuft Kara in einen Pilz hinein, so verschiebt er ihn geradeaus ins nächste Feld. Allerdings ist er zu schwach, um zwei Pilze miteinander zu verschieben! Kara kann nicht auf einem Feld stehen, das von einem Pilz belegt ist.

- Kleeblätter. Kara kann beliebig viele Kleeblätter aufnehmen. Auch hat er einen unerschöpflichen Vorrat an Blättern zum Ablegen. Kara kann auf einem Kleeblatt stehen, und ein Pilz kann über einem Kleeblatt sein.

Kara-Welten werden durch Ziehen mit der Maus im Direktmodus generiert, man hat aber auch aus Kara-Programmen heraus Zugriff auf die Welt. Eine Direktsteuerung von Kara ist durch die Symbole am linken Rand der Entwicklungsumgebung möglich.

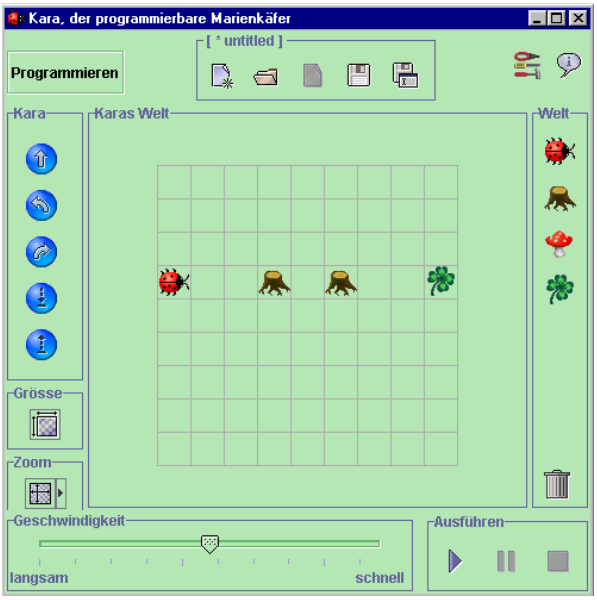
Zum Erstellen von JavaKaraProgrammen steht ein, in die Entwicklungsumgebung integrierter Texteditor zur Verfügung. Bei dessen Aufruf wird ein Programmierrahmen für ein typisches JavaKaraProgramm vorgegeben.
Zitat: „Ihre JavaKara-Programme müssen Sie von der Klasse JavaKaraProgram ableiten. Dadurch werden Ihre Programme in die Laufzeitumgebung von JavaKara eingebettet. In Ihrer ProgrammKlasse müssen Sie die Methode protected void myProgram() überschreiben. Diese Methode wird als "Hauptprogramm" ausgeführt. Innerhalb Ihrer ProgrammKlasse haben Sie Zugriff auf die drei Objekte kara, world und tools, die Sie von JavaKaraProgram erben. Sie sollten diese Objekte nie ändern, indem Sie zum Beispiel versuchen, mit kara=new JavaKara(); einen neuen Kara zu erzeugen! Der von Ihnen erzeugte Kara wäre nicht in die Umgebung von JavaKara eingebunden und somit unsichtbar.“
Die vordefinierte Klasse `kara` verfügt über entsprechende Methoden:

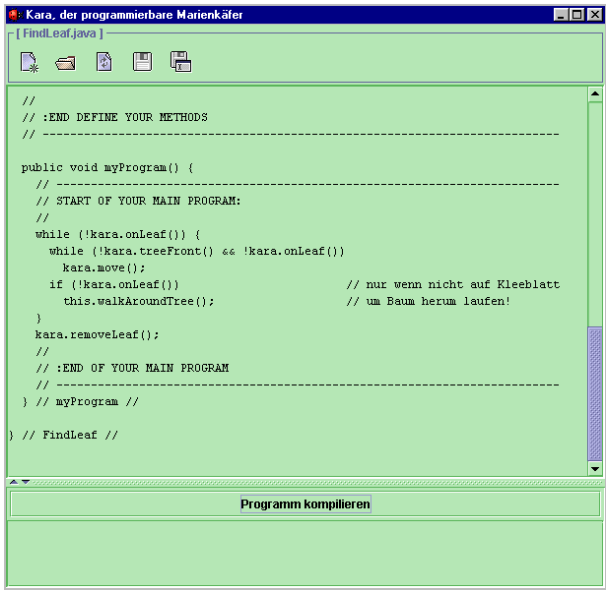
- einfache Methoden (Anweisungen): move(), turnLeft(), turnRight(), putLeaf(), removeLeaf()
- boolsche Methoden (Bedingungen): treeFront(), treeLeft(), treeRight(), onLeaf(), mushroomFront(); Negation mit ! wie in Java üblich;
- eigene Methoden können entsprechend Java-Syntax definiert werden

Es stehen die üblichen Java-Kontrollstrukturen zur Verfügung: while, do while, for, if else, usw. In den ausgelieferten Beispielen von JavaKaraProgrammen werden oft auch Variable und Parameter verwendet.
Typisches JavaKara-Programm:

```
import JavaKaraProgram;
public class CollectLeaves extends JavaKaraProgram {
    // Das Hauptprogramm.
    public void myProgram() {
        if (kara.onLeaf()) {     // falls Kara zu Beginn
            kara.removeLeaf();   // auf Kleeblatt steht: aufnehmen!
        }
        while (!kara.treeFront()) {
            kara.move();
            if (kara.onLeaf()) {
                kara.removeLeaf();
            }
        }
    }
}
```

Das JavaKara-Programm muss unter dem Namen der Programm-Klasse abgespeichert werden (hier CollectLeaves.java) und mit javac compiliert werden, bevor es in der Kara-Oberfläche verwendet werden kann. Der Javacompiler kann aus der Entwicklungsumgebung direkt aufgerufen werden. Eventuelle Fehlermeldungen werden im integrierten Texteditor ausgegeben. Diese Fehlermeldungen fallen jedoch sehr spartanisch aus, da sie von den Standardmeldungen des JavaCompilers übernommen werden.

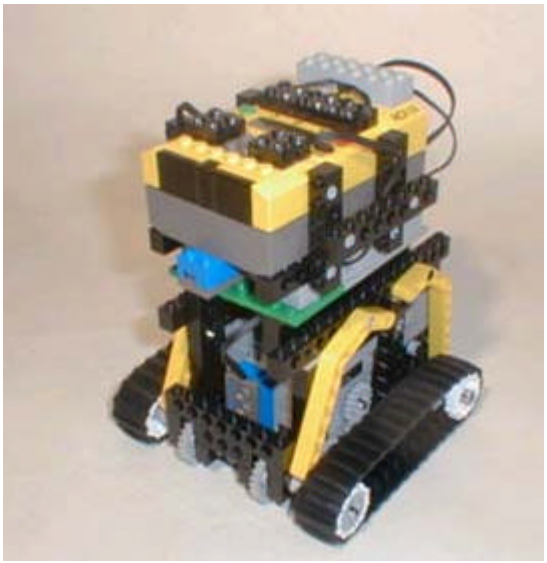




Die kompilierten JavaKaraProgramme können aus der Entwicklungsumgebung heraus gestartet werden. Der Ablauf der Kara-Programme kann verzögert durchgeführt werden. Eine schrittweise Verfolgung im Kara-Programmlisting ist jedoch nicht möglich.

LegoKara

LegoKara ist eine Weiterentwicklung von JavaKara. Autoren sind Remo Meier, Samuel Zürcher von der ETH-Zürich. Es benötigt ebenso als Basis den Java-Developer-Kit JDK (J2SE, SDK und JRE) in der Version 1.2. Das Material, die Programmieroberfläche legokara.jar, eine Bauanleitung und die Dokumentation, ist unter <www.educeth.ch/informatik/karatojava/legokara/> zu finden (Juli 2000).
Zitat: „LegoKara ist ein Roboter, der auf dem Lego Mindstorms Set basiert. LegoKara wird in der Umgebung von JavaKara programmiert. LegoKara ermöglicht es Schülerinnen und Schülern, ihre Kara-Programm an einem physischen Roboter zu testen. Aus der virtuellen Bewegung von Kara am Bildschirm wird eine Bewegung des LegoKara durch das Schulzimmer. So ist das Programmieren weniger abstrakt - und es macht noch mehr Spass!



Die Kara-Programme werden in NQC-Programme zur Steuerung des Lego-Mindstorm-Roboters übertragen.
Zitat: „Um LegoKara-Programme vom Computer auf den LegoRoboter zu übertragen, benötigen Sie das "Lego Invention System". Diese Software liegt dem "Lego Mindstorms Set" bei. Zudem benötigen Sie NQC. Diese Software ermöglicht die Compilation des von LegoKara erzeugten C-Programms.“

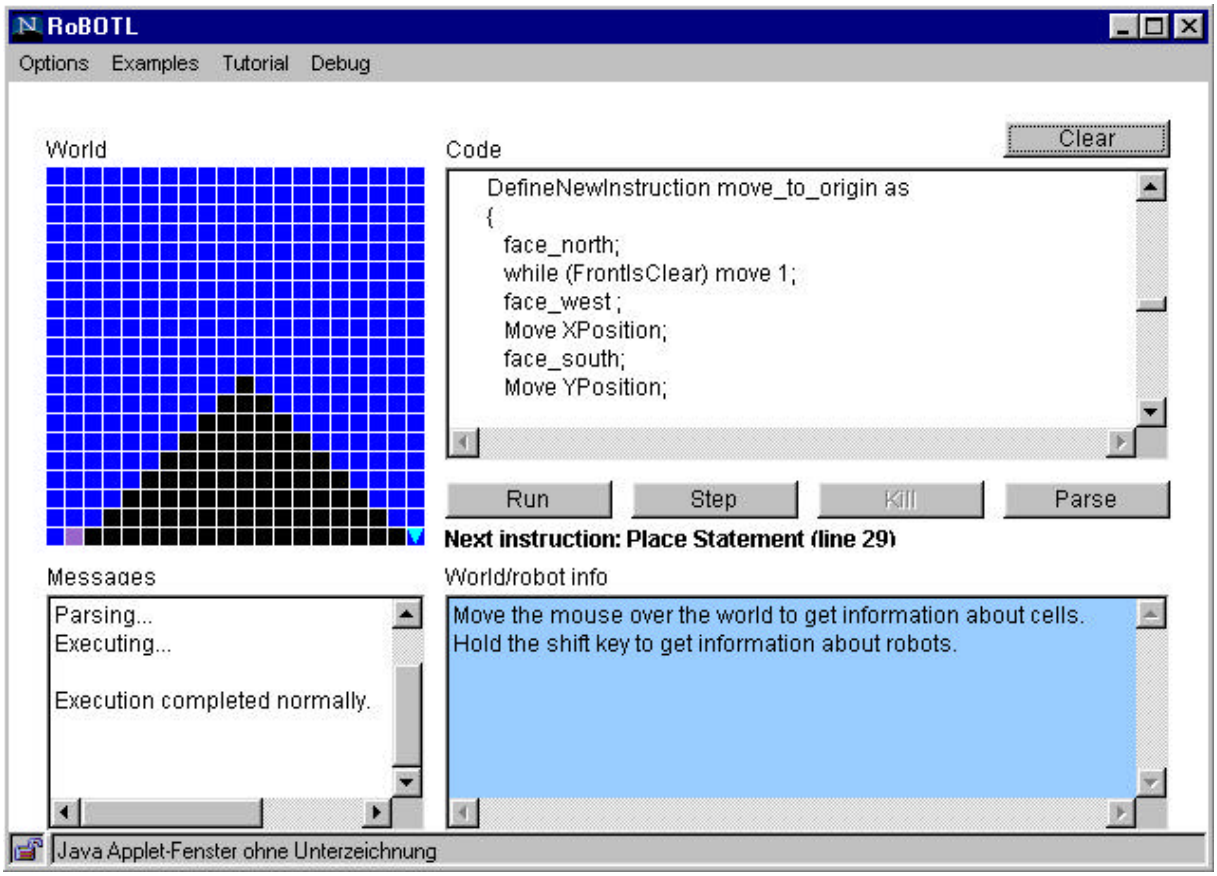
RoBOTL

RoBOTL ist eine einfache, aber mächtige Programmiersprache, die die Grundkonzepte der Programmierung wie Anweisungen und Bedingungen unterstützt, ebenso objektorientierte Konzepte wie Objekte und Vererbung.
Ein DOS-Programm wurde 1995 von Professor Karen Lemone, WPI's Computer Science Department, Worcester, MA und Walter S. Ching, Senior S/W Engineer, Raytheon Electronic Systems, Marlboro, MA entwickelt. Näheres siehe <www.mmra.org/~n1hbr/robotl/robotl.html>.
RoBOTL hat eine C-ähnliche Syntax mit vordefinierten Klassen. Die Welt wird nicht gesondert sondern im Programm definiert. Ein RoBOTL-Programm besteht aus 2 Teile dem Definitionsbereich (unter Anderem auch für die Welt) und dem Ausführungsbereich.
Die Welt ist eine Fläche mit quadratischen Feldern. Der Roboter bewegt sich von Feld zu Feld und wird als Dreieck dargestellt. Die Welt kann Beeper und Wände enthalten.
Die Grundklasse basic_bot verfügt über folgende Fähigkeiten (werden mit „tell robname“ aufgerufen):

- einfache Methoden: Move, Move n, TurnLeft, PickBeeper, PutBeeper, TurnOff, TurnOn
- Methoden mit boolschem Rückgabewert: FrontIsClear, LeftIsClear, RightIsClear, FacingNorth, FacingSouth, FacingWest, FacingEast
- Methoden mit ganzzahligem Rückgabewert: BeepersInBag, BeepersOnFloor, XPosition, YPosition
- selbstdefinierte Anweisungen werden über neue Roboterklassen, die vom Grundroboter abstammen, realisiert

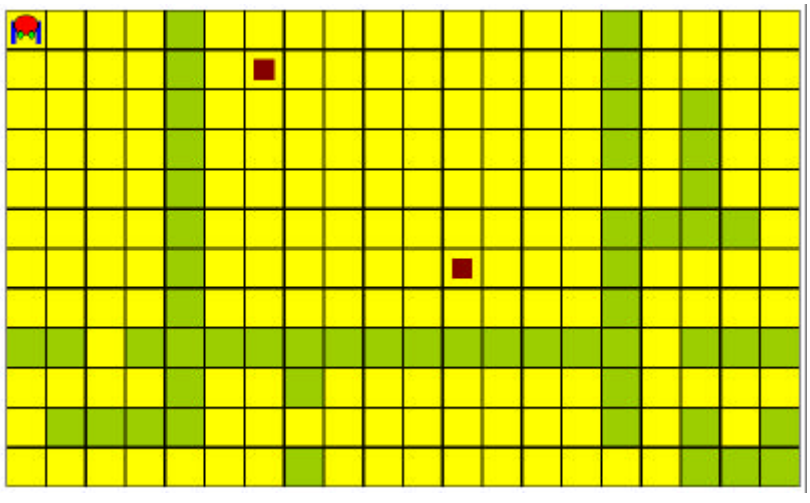
Als Kontrollstrukturen sind iterate times, do while, while, if , if else möglich und die Blockbildung erfolgt mit { }. Die Sprache RoBOTL bietet boolsche Operatoren (and, or, Negation) und Ganzzahl-Operatoren (+, -, *, /).
Bei RoBOTL sind ebenfalls mehrere Roboter gleichzeitig in der Welt möglich, sie werden mit New angelegt und mit Remove entfernt. Es können neue Roboter definiert werden, die von anderen abstammen (Vererbung; „NewRobotType IsLikeA“) und über zusätzliche Methoden verfügen können („DefineNewInstruction“) und zusätzliche Attribute besitzen.

Marvin, eine Realisierung von RoBOTL als Java-Applet ist bei <www.kluge.net/mqp/applet.html> vorhanden (Dezember 1997). Dort befindet sich auch eine umfassende Beschreibung der Sprache RoBOTL.



Der Roboter Robi

Entwickelt von J.-O. Wehner an der Sophie-Scholl-Oberschule, Berlin-Schöneberg auf der Basis von Javascript. Wird als einzelne HTML-Dateien ausgeliefert. Software und umfangreiche Lernsequenzen im WWW unter <scholl.be.schule.de/schule/faecher/inform/material/robi/>.
Zitat: „"Robi" stellt einen virtuellen Roboter dar, der mit Hilfe einiger weniger Befehle in einer vorgegebenen "Robiwelt" Tätigkeiten ausführt. Die Oberfläche zur Erzeugung von Robiwelten, Teachin- und Robiprogrammen wurde mit Hilfe von JavaScript erstellt und funktioniert in einem javascriptfähigen Internetbrowser (außer Netscape 6), um auch online arbeiten zu können.“



Die Welt („Robiwelt“) besteht aus einem festen (13x21) Gitter von Quadraten. In einem Quadrat kann ein Hinderniss (grün) oder ein Gegenstand plaziert werden. Es sind auch mehrere Gegenstände auf einem Quadrat möglich, jedoch ist Anzahl aufeinanderliegender Gegenstände in der Welt nicht direkt erkennbar. Robi kann die Gegenstände aufnehmen, in seiner Tasche transportieren und ablegen. Robi wird als Roboter in Draufsicht dargestellt.
Die Programmierumgebung zu Robi besteht aus einem Steuerprogramm und drei Module (in Form von HTML-Dateien) zum Erstellen einer Welt, zum Aufbau von Robi-Programmen im Teachin-Modus und zum Ablaufen lassen selbsterstellter Robi-Programme.
Ein Robi-Programm muss mit Javascript-Notation in einem gesonderten Texteditor oder HTML-Editor erfasst und als HTML-Datei gespeichert werden, wobei eine ausgelieferte Skriptdatei robi_pro.js einzubinden ist. Die Kontrollstrukturen müssen in Javascript-Syntax geschrieben werden und die selbstdefinierten Anweisungen werden als Javascript-Funktionen definiert. Auch die vordefinierten Anweisungen und Bedingungen sind als Javascript-Funktionen (in der Datei robi.js) realisiert.
Der Befehlsumfang für Robi umfasst nur wenige Möglichkeiten:

- vordefinierte Anweisungen: vor(), rechts(), nimm(), gib()
- vordefinierte Bedingungen mit boolschem Rückgabewert: vorne_frei(), feld_leer(), tasche_leer()
- selbstdefinierte Anweisungen und Bedingungen sind als normale Javascript-Funktionen möglich

Als Kontrollstrukturen werden verwendet: if (bedingung) anweisung; if (bedingung) anweisung1; else anweisung2; while (bedingung) anweisung; do anweisung while (bedingung); . Blockbildung erfolgt mit den Klammern { }. Bei den Bedingungen werden die boolsche Operatoren &&, || und die Negation ! verwendet.
Die Ablaufgeschwindigkeit des Robi-Programms kann eingestllt werden. Während des Programmablaufs werden die Robi-Eigenschaften (Position Zeile, Spalte; Anzahl Schritte seit Programmbeginn; Taschеninhalt als Anzahl aufgesammelter Gegenstände; Anzahl der Gegenstände auf dem Feld von Robi) angezeigt. Einzelschritt der Befehle ist möglich. Das Robi-Programm ist während des Programmablaufs nicht direkt sichtbar (muss mit einem Texteditor zusätzlich geöffnet werden) und der Programmfortschritt wird nirgends angezeigt (auch nicht beim Einzelschritt).

Robi, der Roboter

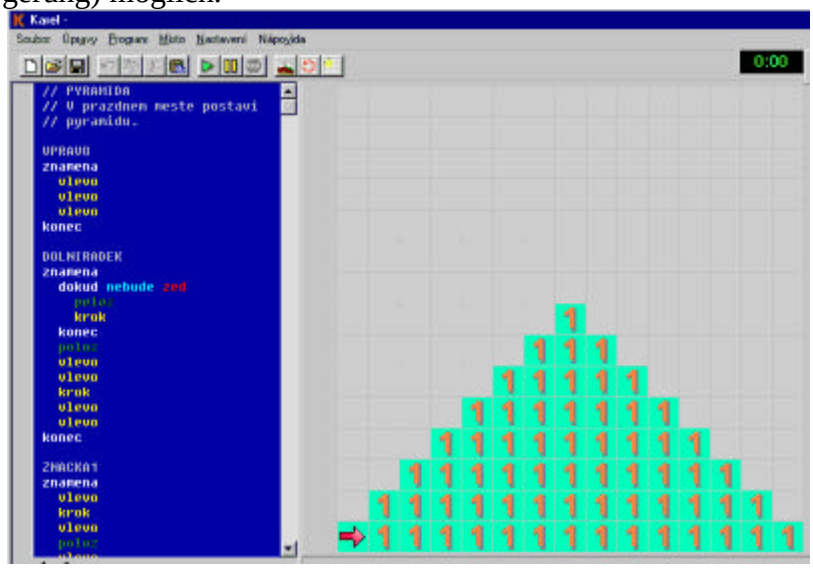
Eine Version, die auf Modula aufsetzt und unter SCO Unix bzw. LINUX mit mcs-Modula-2 lauffähig ist. Es wird in Modula programmiert und zur Steuerung von Robi werden einige Zusatz-Module zur Verfügung gestellt. Entwickelt von Dr. Bernd Kokavecz, Humboldt-Gymnasium Berlin-Tegel, Landesbildstelle Berlin (BICS), August 1994.
Zitat: „In der Simulation "arbeitet" Robi in einer schachbrettartig eingeteilten "Werkhalle", durch die er sich mit Hilfe des Steuerbefehls "vor" und der 90 - Grad - Drehung "links" bewegen kann. Robi hat einen Sensor, der ihm meldet, ob vor ihm ein Hindernis liegt (vorne_frei), und einen Sensor, der meldet, ob auf dem Feld unter ROBI Gegenstände liegen (Feld_leer). Robi kann Gegenstände aufnehmen (nimm) oder Gegenstände aus seinem Vorrat ablegen (gib). Ein weiterer Sensor meldet, ob noch Gegenstände geladen sind (Tasche_leer). In der Bildschirmdarstellung (im Textmodus) werden Robi durch einen Pfeil, Gegenstände (z.B. Werkzeuge) mit Hilfe der "0" dargestellt. Mit Hilfe einer Textdatei bzw. eines "Welteditors" lassen sich in ROBIs Welt auch Hindernisse einbauen. Das Programm ROBI-Teach-In erlaubt es, mit Hilfe von Steuertasten Robi manuell zu führen und beispielsweise "Aufräumarbeiten" in der Halle ausführen zu lassen. Das Programm erzeugt dabei automatisch eine Textdatei, die ein ROBI-Quellcode-Modula-2-Programm darstellt, das anschließend mit dem mcs-Compiler

übersetzt werden kann und dann automatisch abläuft. Die Forderung, nicht nur eine Einzellösung, sondern Problemlösungsklassen zu programmieren, bedingt die Einführung von wertelieferndenProzeduren, Ablaufstrukturen und den Einsatz der "Sensoren". Nun werden die Programme mit Hilfe eines Editors geschrieben, wobei vorübersetzte Programmteile eingebunden werden (MODULE Robi).“

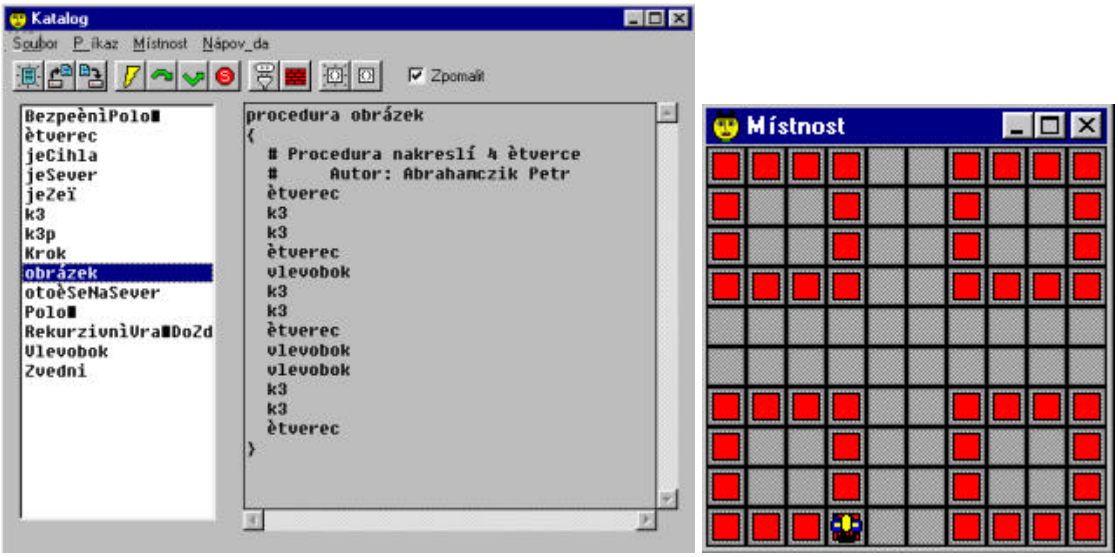
Semesterarbeiten in tschechisch oder slowkisch

An Hochschulen und Fachhochschulen der Tschechei und Slowakei werden des öfteren Semesterarbeiten zum Thema „Karel, the Robot“ vergeben. Die Aufgabe besteht in der Entwicklung eines Interpreters für die Sprache Karel und die Erstellung einer zugehörigen Programmierumgebung mit einer Karel-Welt. Diese Arbeiten liegen, zusammen mit der Anleitung, meist nur in tschechischer oder slowakischer Sprache vor. Im folgenden einige Beispiele.

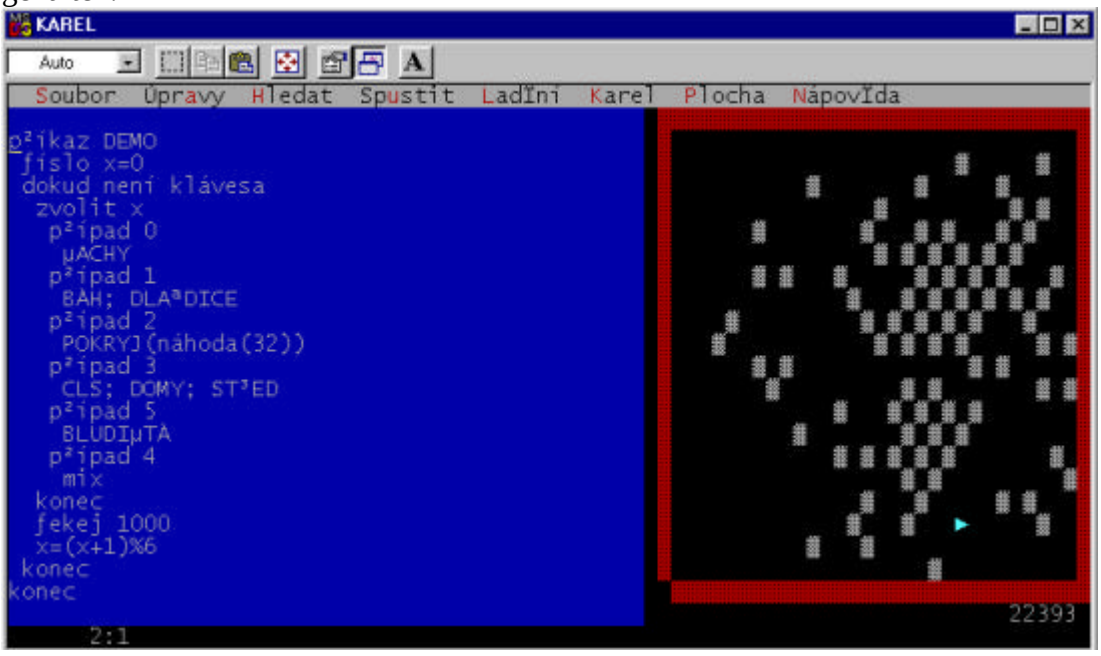
Der Karel von Petr Plavjanik wird als Pfeil dargestellt und lebt in einer Welt aus Quadraten in der Mauern angebracht und Marker plaziert werden können. An einer Stelle sind bis zu 9 Marker möglich, was durch Zahl auf dem Marker angezeigt wird. Die Größe der Welt ist wählbar und die Marker und Mauern können in einem Welt-Editmodus gesetzt werden. Die zugehörige Sprache hat den üblichen Befehlsumfang, wobei eigene Anweisungen und Bedingungen definiert werden können. Es ist sowohl Einzelschritt als auch normaler Ablauf (ggf. mit Verzögerung) möglich.



Eine andere Arbeit ist von Rádim Dostal (Radim.Dostal.fei@vsb.cz) und Petr Abrahamczik. Die Karel-Welt wird in einem eigenen Fenster dargestellt, wobei Karel ein kleiner Roboter in Draufsicht ist. Die Welt besteht aus Quadraten auf denen ein Ziegel plaziert werden kann. Die Größe der Welt ist einstellbar, jedoch wird die Lage der Ziegel in der Welt vom Programm automatisch generiert und kann nicht beeinflusst werden. Der Sprachumfang ist der Übliche mit der Möglichkeit eigene Anweisungen und Bedingungen zu verfassen.
Weiteres unter <linux456.vsb.cz/~dos083/Karel>



Das DOS-Programm von Petr Lastovicka verfügt über einen großen, pascalartigen Sprachumfang mit integer und boolschen Operatoren. Die Karel-Welt ist sehr schlicht gehalten.

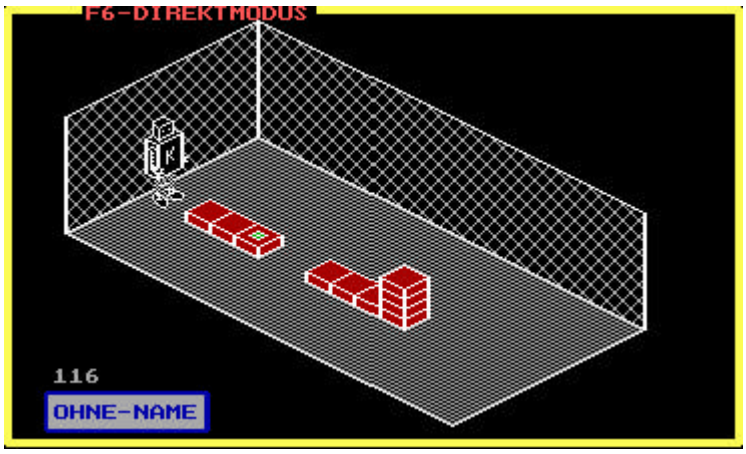


3D-Karel-Welten

Die meisten Realisierungen verwenden die zweidimensionale Welt, wie sie von Pattis vorgesehen wurde. Wesentlich mehr Möglichkeiten bieten dreidimensionale Welten und trotzdem bleiben sie einfach genug um zur Einführung in die Algorithmik eingesetzt werden zu können. Die dreidimensionale Welt hat einen Boden mit quadratischem Grundpflaster und einer maximalen Höhe (die Größen Breite, Länge, Höhe sind wählbar). In dieser Welt kann sich Karel von Quadrat zu Quadrat bewegen, Ziegelstapel aufbauen und abbauen, auf die Ziegelstapel klettern, Markierungen setzen und entfernen.

Robot Karel

Ein Karel-Programm dieser 3D-Art wurde in dem MSDOS-Grafikprogramm „Robot Karel“ von Andrej Blaho (blaho@fmph.uniba.sk), Chlebikova und Marian Vitek (vittek@fmph.uniba.sk) (1986-1992) in slowakischer Sprache realisiert und später auch ins Deutsche und Englische übertragen. Literatur hierzu auch: „Jozef Hvorecky (hvorecky@vseba.sk), Karel the Robot for PC, 1992, Proc.\ of the East-West Conference on Emerging Computer Technologies in Education, P. Brusilovsky and V. Stefanuk, Moscow, Russia.“



Der Bildschirm ist in vier Bereiche aufgeteilt:

- die 3D-Welt von Karel mit den Ziegeln und Markierungen; Karel wird als kleiner Roboter dargestellt; jeder Befehl an Karel wird sofort in eine visuelle Reaktion umgesetzt
- der Editierbereich; hier wird der Programmtext eingegeben
- eine Auswahlliste aller Anweisungen (Befehle) (vordefinierte und selbstdefinierte)
- eine Auswahlliste aller Bedingungen (vordefinierte und selbstdefinierte).

Karol kann in seiner Welt direkt oder durch Programmcode gesteuert werden. Der Aufbau einer Welt erfolgt durch Direktsteuerung von Karel und hinlegen von Ziegel bzw. Setzen von Markierungen. Die Bedienung der Programmieroberfläche erfolgt mit Funktionstasten, mit diesen wechselt man zwischen den einzelnen Bildschirmbereichen. Die möglichen Befehle (Anweisungen) und Bedingungen können - ohne Eintippen - aus Auswahllisten gewählt und zu einem Programm zusammengestellt werden.
Es wird folgender Sprachumfang angeboten:

- Vordefinierte Anweisungen (Befehle): Schritt, Linksum, Rechtsum, Hinlegen, Aufheben, Markieren, Löschen, Schnell, Langsam, Piep

- Vordefinierte Bedingungen: Ist Ziegel, Ist Wand, Ist Marke, Ist frei, Nicht Ist Ziegel, Nicht Ist Wand, Nicht Ist Marke, Nicht Ist Frei
- Kontrollstrukturen: wiederhole ... mal, solange ... tue, wenn ... dann ... sonst ..., wenn ... dann
- Benutzerdefinierte Anweisungen und Bedingungen sind möglich

Eine schrittweise Ausführung der Karel-Programme mit Anzeige der aktuellen Zeile wird unterstützt. Ebenso kann wahlweise bei falschen Anweisungen an Karel eine Programmstop erfolgen.
Die deutsche Version kann von <www-schulen.informatik.tu-muenchen.de/didaktik/vorlesung-werkzeuge.html> geladen werden.

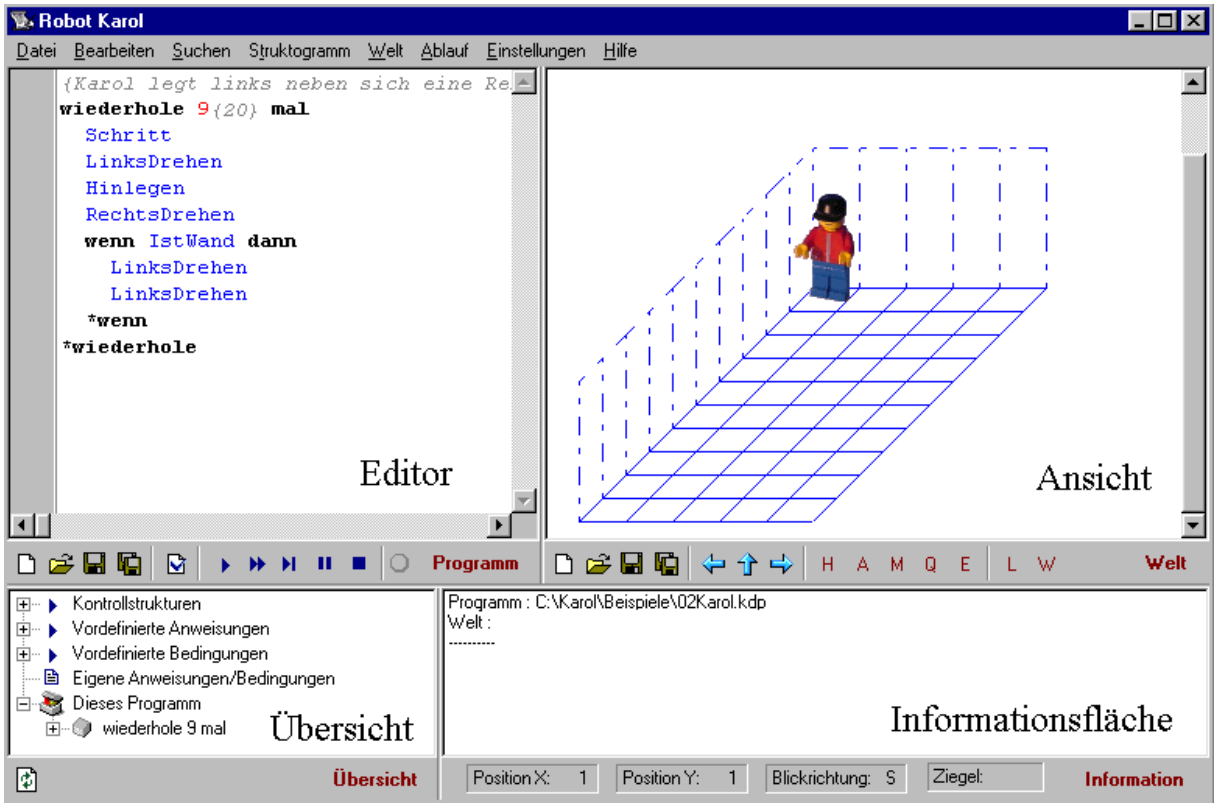
Karel 3D pre Windows

Dieses Produkt ist nur in Slowakisch vorhanden. Geschrieben von Lubomir Kosut, Nabr. Stefanika 2/13, 034 00 RUZOMBEROK, S L O V A K I A, Lubomir.Kosut@dcs.fmph.uniba.sk. Arbeitsstand Version 1.30 etwa 1997. Es stellt eine Implementation von Robot Karel unter der grafischen Bedienoberfläche Windows dar. Die Welt bei Karel 3D ist, wie auch der Roboter Karel, ganz ähnlich aufgebaut und kann Ziegel und Wände enthalten. Der Umfang der Programmiersprache ist ebenfalls gleich (natürlich in slowakisch). Das Programm wird in einem zeilenorientierten Editor mit Mausunterstützung erfasst. Der Aufruf und das Editieren von Programmen ist in der Programmieroberfläche etwas unübersichtlich realisiert und schwer zu verstehen. Man muss erst einige Dialoge öffnen bevor man zum Editieren gelangt.

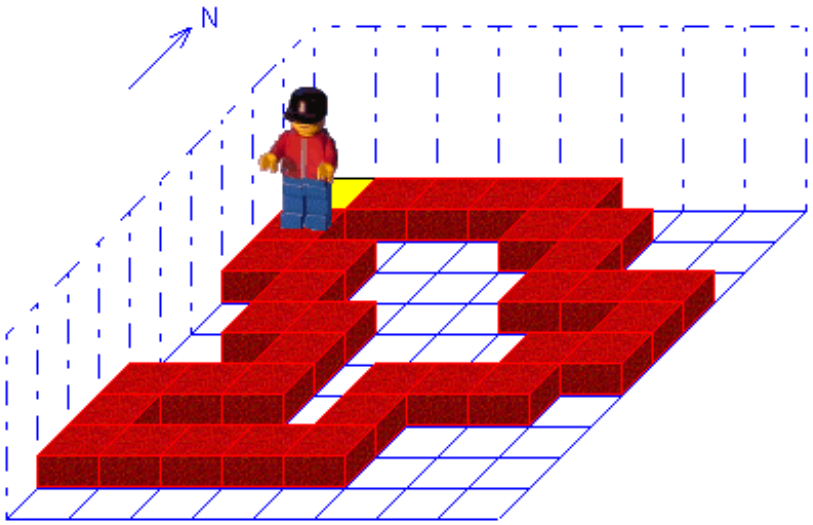
Robot Karol

Das Programm „Robot Karol“ greift die Idee von „Robot Karel“ auf, setzt das Konzept in eine zeitgemäßere Form unter der Bedienoberfläche Windows um und erweitert die Einsatzmöglichkeiten wesentlich. Es wurde in Zusammenarbeit von Ondrej Krško, Slowakei, krsko@gjh.sk und Ulli Freiberger, Luitpold-Gymnasium München, freiberger@schule.bayern.de entwickelt und ist über <www.schule.bayern.de/karol> erhältlich.
Zitat: „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-Ergänzungen, 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, und vieles mehr.“



Die Karol-Welt ist eine übliche 3D-Karel-Welt. In dieser können von Karol Ziegel hingelegt, gestapelt und aufgehoben; Marken gesetzt und gelöscht; Quader aufgestellt und entfernt werden. Eine Welt mit ihren Objekten wird durch direkte Steuerung von Karol, Ziegel hinlegen und Marken setzen aufgebaut und ggf. gespeichert. Die Karol-Welt kann wahlweise auch in einer 2D Grundrißansicht betrachtet werden.



Karol-Programme werden im Editor-Bereich erstellt. Es gibt zahlreiche Hilfen bei der Erstellung wie Aufklappmenü mit den wichtigsten Anweisungen/Bedingungen, automatische Code-Vervollständigung und kontextsensitive Code-Nachschlagemöglichkeit. Zur Übersicht wird die Syntax farblich hervorgehoben.

Robot Karol verfügt über die Sprachelemente:

- vordefinierte Anweisungen: Schritt, LinksDrehen, Rechtsdrehen, Hinlegen, Aufheben, MarkeSetzen, MarkeLöschen, Warten, Ton
- vordefinierte Bedingungen: IstWand, NichtIstWand, IstZiegel, NichtIstziegel, IstMarke, NichtIstMarke, IstSüden, IstNorden, IstWesten, IstOsten, IstVoll, NichtIstVoll, IstLeer, NichtIstLeer
- Kontrollstrukturen: wiederhole immer, wiederhole mal, solange tue, wiederhole solange, wiederhole bis, wenn dann, wenn dann sonst
- selbstdefinierte Anweisungen und Bedingungen sind möglich, auch lokale

In einer Übersicht kann das Karol-Programm als hierarchische Struktur dargestellt werden bzw. im Ansichtbereich als Struktogramm.
Sytnaxfehler werden im Informationsfenster angezeigt und im Editorbereich markiert.

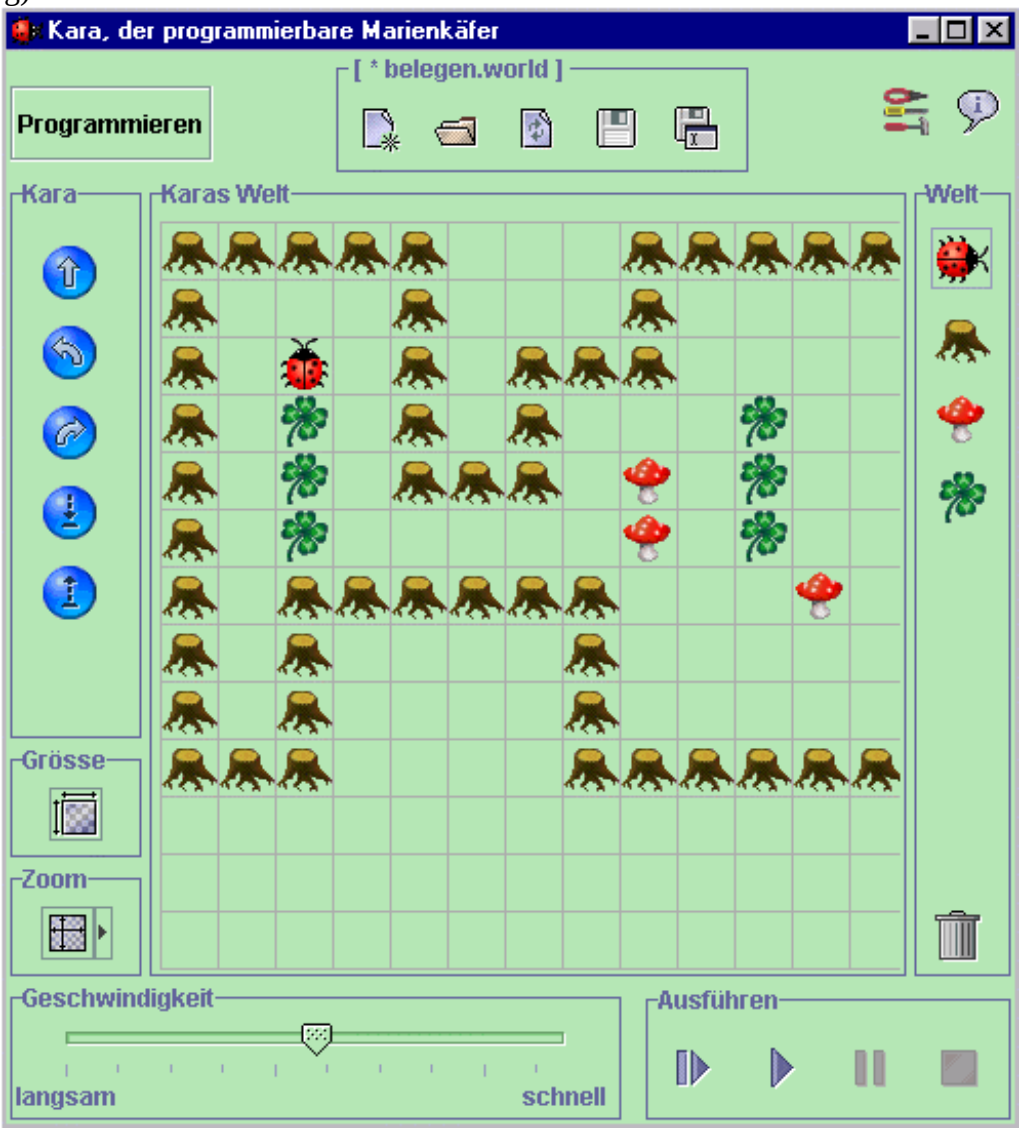
```
{ Karol läuft auf einem geschlossenen
  Mauerzug entlang, bis er die Marke
  erreicht }
Anweisung Umdrehen
  LinksDrehen
  LinksDrehen
*Anweisung

{ Hauptprogramm }
Programm
  solange NichtIstMarke tue
    wenn IstZiegel dann
      Schritt
    sonst
      LinksDrehen
      wenn NichtIstZiegel dann
        Umdrehen
      *wenn
      Schritt
    *wenn
  *solange
*Programm
```

Die Karol-Programme können im Einzelschrittmodus, im Normalmodus mit einer festlegbaren Verzögerungsdauer und im Schnellmodus ohne Verzögerung ausgeführt werden. Der Programmablauf wird im Editorbereich Zeile für Zeile durch Markierungen angezeigt. Ein Fehlverhalten von Karol (z.B. läuft gegen die Wand) kann ignoriert werden, nur zu einer Hinweisausgabe führen oder einen Programmstopp bewirken.

Andere Programmierkonzepte

Kara

Die Welt von Kara ist analog zu JavaKara aufgebaut (siehe dort), jedoch wird Kara als endlicher Automat rein grafisch programmiert. Entwickelt von Raimond Reichert, Markus Brändle, Reto Lamprecht; ETH Zürich. Die aktuelle Version (Juni 2002) der Programmierumgebung kara.jar, Beispiele, Hinweise für den Unterrichtseinsatz und eine ausführliche Dokumentation sind unter <www.educeth.ch/informatik/karatojava/kara/> zu finden. Die Programmierumgebung kann entweder als Java-Applet im Browser mit dem Java-Plugin Version 1.2 oder als Applikation unter Verwendung der Java Runtime Environment (JRE) in der Version 1.2 oder höher ausgeführt werden (ist sowohl unter Windows/MacOS und Unix lauffähig).



Zitat: „Kara ist ein Marienkäfer, der in einer einfachen grafischen Welt auf dem Bildschirm lebt. Er kann programmiert werden, um in seiner Welt Aufgaben zu erledigen, zum Beispiel Kleeblätter zu sammeln. Kara wird in einer grafischen "Entwicklungsumgebung" programmiert.“
Kara lebt in einer rechteckigen Welt, die aus einzelnen quadratischen Feldern besteht. Er kann sich nur von Feld zu Feld bewegen. Auf den Feldern in dieser Welt gibt es

- unbewegliche Baumstümpfe. Wenn Kara in einen Baumstumpf läuft, beschwert er sich. Auf einem Feld mit einem Baumstumpf kann kein anderer Gegenstand liegen.
- verschiebbare Pilze. Läuft Kara in einen Pilz hinein, so verschiebt er ihn geradeaus ins nächste Feld. Allerdings ist er zu schwach, um zwei Pilze miteinander zu verschieben! Kara kann nicht auf einem Feld stehen, das von einem Pilz belegt ist.
- Kleeblätter. Kara kann beliebig viele Kleeblätter aufnehmen. Auch hat er einen unerschöpflichen Vorrat an Blättern zum Ablegen. Kara kann auf einem Kleeblatt stehen, und ein Pilz kann über einem Kleeblatt sein.

Kara-Welten werden durch Ziehen mit der Maus im Direktmodus generiert. Eine Direktsteuerung von Kara ist durch die Symbole am linken Rand der Entwicklungsumgebung möglich.

Zitat: „Kara’s Programme werden rein grafisch mit der Maus als Automaten "zusammengeklickt". Keine Syntax der verwendeten Sprache muss erlernt werden, Syntaxfehler gibt es nicht! Jedes erstellte Programm kann sofort gestartet werden.“
Kara wird programmiert, indem ein Automat erstellt wird. Es werden die einzelnen Zustände festgelegt und für jeden Zustand wird angegeben bei welchen Sensorwerten welche Kara-Reaktionen ausgeführt und welcher neue Zustand angenommen wird. Die Erfassung der Zustände, Sensorwerte und der Reaktionen erfolgt übersichtlich in einer einfach zu bedienenden Tabelle.
Zusätzlich werden in einer grafischen Übersicht alle Zustände und ihre Übergänge mit Symbolen eines Zustandsdiagramms dargestellt.

Zitat: „Damit Kara programmiert werden kann, verfügt er über Sensoren. Diese erlauben es ihm, seine unmittelbare Umgebung wahrzunehmen:

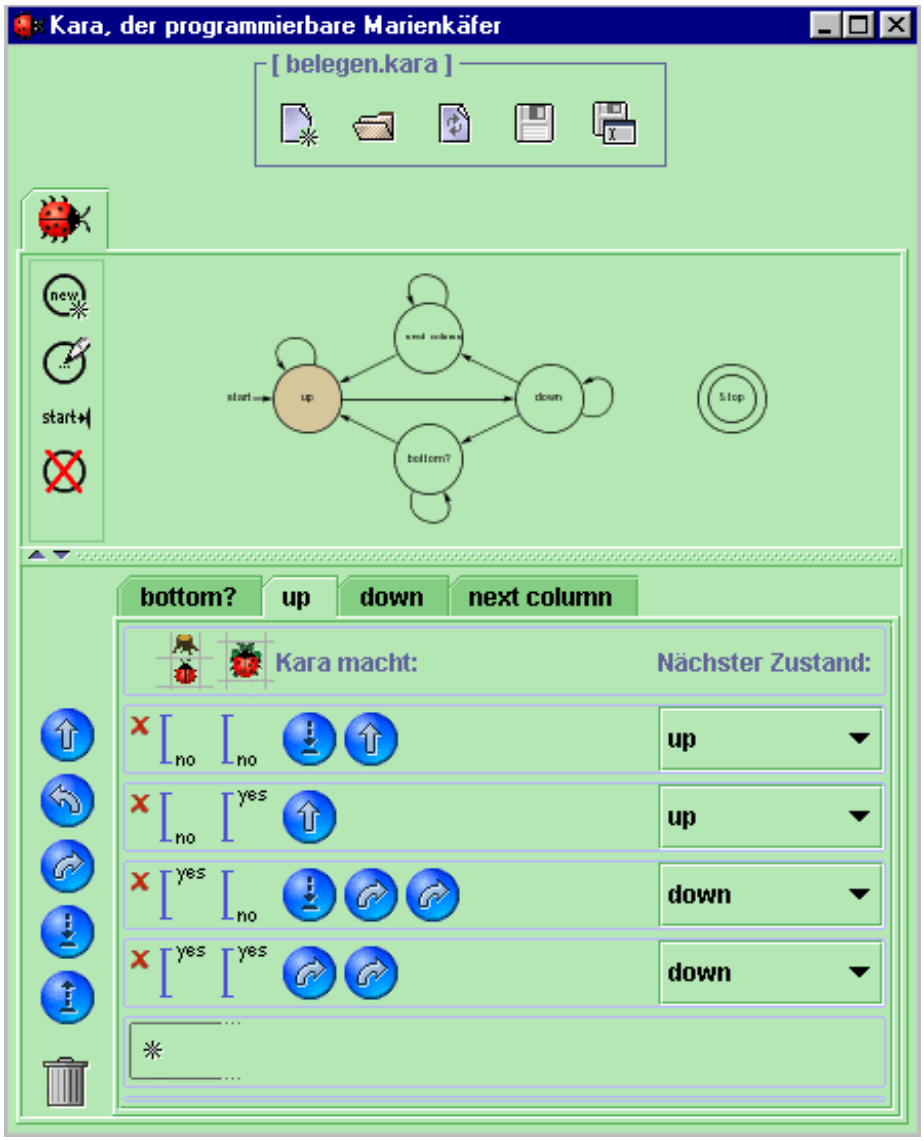
Stehe ich vor einem Baumstumpf? - Ist links neben mir ein Baumstumpf? - Ist rechts neben mir ein Baumstumpf? - Stehe ich vor einem Pilz? - Stehe ich auf einem Kleeblatt?

Das ist alles, was Kara über die Welt weiß! Er weiß nicht, auf welchem Feld er steht oder in welche Himmelsrichtung er schaut.

Kara kennt nur wenige Befehle:

Mache einen Schritt vorwärts - Drehe um 90° nach links - Drehe um 90° nach rechts - Lege ein Kleeblatt hin - Nimm ein Kleeblatt auf

Diese Befehle müssen für die Programmierung genügen! Kara kann in Abhängigkeit von dem, was ihm seine Sensoren melden, eine beliebige Anzahl von Befehlen ausführen.“



Ein Kara-Programm kann schrittweise und automatisch ablaufen, wobei die Ablaufgeschwindigkeit einstellbar ist.

Stand 20.Juni 2002 U.Freiberger