Karol – Beschreibung von Abläufen durch Algorithmen

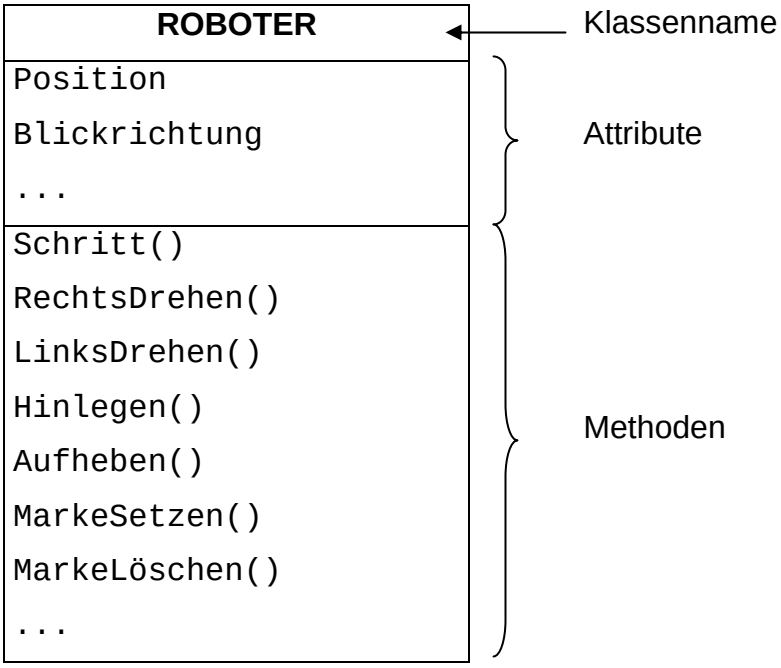
Du hast bereits gesehen, dass bei Bastelanleitungen, Gebrauchsanweisungen oder Kochrezepten folgende Kriterien wichtig sind:

- Knappe, präzise Beschreibung
- Einfache Formulierung
- Schrittweise Abarbeitung der einzelnen Punkte (manchmal müssen einige Schritte auch mehrmals ausgeführt werden)
- Die Abarbeitung muss in der vorgegebenen Reihenfolge erfolgen

⇗ **INFO:** Diese Kriterien gelten auch für die Programmierung von Software, egal, ob es sich um die sehr komplexe Programmierung eines Betriebssystems wie Unix, Windows oder Vista handelt oder um die Programmierung eines Roboters.
Einen echten oder durch Software simulierten Roboter kann man durch **Algorithmen** (=Befehle, die bestimmte Vorschriften befolgen) steuern.

Unser Roboter Karol ist ein **Objekt** der **Klasse** ROBOTER. Diese Klasse hat **Attribute** wie Position, Blickrichtung etc. Diese Attribute können verändert werden, wenn sich Karol z.B. um einen Schritt bewegt: das Attribut „Position“ hat nun einen anderen Wert.
Damit Karol sich in seiner gitterartigen Welt in vier Richtungen bewegen kann oder Ziegelsteine hinlegen / aufheben kann etc., muss er über gewisse **Methoden** wie „Schritt“ oder „Hinlegen“ verfügen. Das Klassendiagramm sieht also etwa folgendermaßen aus:

Beispiel 1:



Um Karol nun in der Welt zu steuern, muss man ihm **Anweisungen** (Befehle) geben. Dies geschieht im Programmierfenster auf der linken Seite. Dabei handelt es sich im Prinzip um einen Texteditor, in dem man die Befehle tippt. Ein sog. „**Compiler**“ übersetzt die für uns verständlichen Befehle Schritt o.ä. dann in eine für den Computer verständliche Sprache.

I. Iterative Programmierung

(=schrittweise Abarbeitung von Befehlen von der ersten bis zur letzten Zeile, ohne Wiederholungsanweisungen oder Sprunganweisungen)

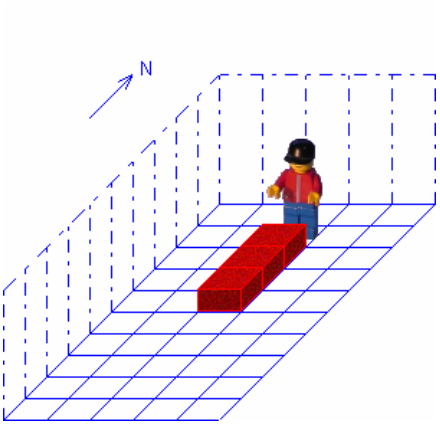
I.1 Vordefinierte Anweisungen

Befehl	Erklärung
`Schritt`	einen Schritt in Blickrichtung gehen
`Schritt(n)`	n Schritte in Blickrichtung gehen
`RechtsDrehen`	Drehung um 90° im Uhrzeigersinn
`LinksDrehen`	Drehung um 90° gegen den Uhrzeigersinn
`Hinlegen`	einen Ziegel auf das Feld vor Karol hinlegen
`Aufheben`	Einen Ziegel vom Feld vor Karol aufheben
`MarkeSetzen`	Markiert das Feld, auf dem Karol steht
`MarkeLöschen`	Löscht die Markierung des Feldes, auf dem Karol steht

Aufgabe 1: Reihe legen
Lege, ausgehend von einem bestimmten Punkt in Karols Welt, eine gerade Reihe mit drei Ziegelsteinen. Karol soll am Ende nicht auf einem Stein stehen.

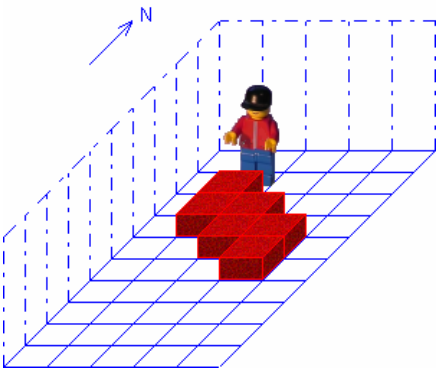
Das dazugehörige Programm lautet:

```
Hinlegen
Schritt
Hinlegen
Schritt
Hinlegen
Schritt
Schritt
```



Aufgabe 2: Muster
Nun soll Karol ein kleines Muster auf den Boden legen, da er einen kleinen Sitzplatz in seinem Garten pflastern möchte. Dazu muss er sich immer wieder drehen:

```
Hinlegen
Schritt
Hinlegen
Schritt
LinksDrehen
Hinlegen
Schritt
RechtsDrehen
Hinlegen
Schritt
LinksDrehen
Hinlegen
Schritt
RechtsDrehen
Hinlegen
Schritt
Schritt
```



Du hast vielleicht bereits jetzt festgestellt, dass es sehr schnell umständlich werden kann, auf diese Weise zu programmieren: wenn man z.B. möchte, dass Karol eine Welt im Format 20*20 komplett mit Ziegelsteinen auslegt, muss man 400 mal die Befehle `Hinlegen` und `Schritt` tippen, zudem noch eine größere Anzahl an Anweisungen, um Karol zu drehen.
Der Programmcode, also die getippten Befehle, wird also sehr umfangreich – die zeitliche Dauer und der Speicherplatzbedarf des Codes stehen in keinem Verhältnis zum Nutzen. Deshalb schauen wir uns einmal an, wie man einen Code verkürzen (=optimieren) kann:

I.2 Selbstdefinierte Methoden

Um Zeit zu sparen und um den Speicherplatz und den eigentlichen Anweisungsteil, das „Hauptprogramm“ möglichst knapp und übersichtlich zu halten, bietet es sich an, selbstdefinierte Methoden zu verfassen – auf diese kann man dann beim Programmieren praktischerweise immer wieder zurückgreifen.

⇗ **INFO:** Dieses Verfahren ist nicht nur beim Programmieren von Karol von Nutzen, sondern gilt auch und vor allem für komplexe Anwendungen: ein Hersteller von 3D-Computerspielen verwendet bestimmte (oft selbstentwickelte) 3D-Routinen in verschiedenen Spielen, dennoch kann es sich um sehr unterschiedliche Spiele handeln. Oder aber bei Supercomputern zur Simulation von Wetterdaten wird auf vieles modulartig zurückgegriffen; die Entwicklung solcher Datenbanken oder Grafikroutinen dauert oft Jahre, und es wäre ineffektiv, ein Nachfolgeprogramm wieder völlig von vorne zu entwickeln. Es gibt ganze Bibliotheken solcher Methoden und Funktionen, die bei Bedarf dann vom Programmierer einfach in das jeweilige Programm eingebunden werden.

Aufgabe 3: Reihe als Methode
Schreibe eine Methode „`DreierReihe`“, bei deren Aufruf Karol drei Ziegel hintereinander legt (vgl. Bild bei Aufgabe1):

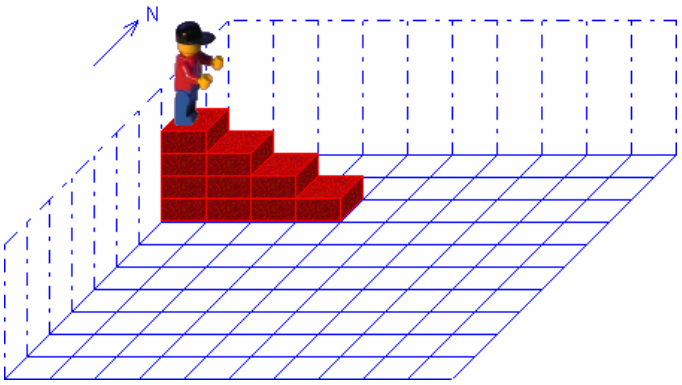
```
Anweisung DreierReihe
      Hinlegen
      Schritt
      Hinlegen
      Schritt
      Hinlegen
      Schritt
*Anweisung

DreierReihe
```

wie man an „DreierReihe“ sieht, beginnen selbstdefinierte Methoden mit **`Anweisung <Name>`**, danach folgt der Block von Einzelbefehlen. Beendet werden muss die Methode mit **`*Anweisung`** (kein Leerzeichen nach dem Stern!).

Die Methode zu definieren ist nur ein erster Schritt – wenn man das Programm nun startet, passiert nichts: erst wenn man die Methode mit dem Befehl `DreierReihe` aufruft, führt Karol sie aus.
Aus Gründen der Übersichtlichkeit solltest du dir angewöhnen, mit Einrückungen im Text zu arbeiten – das erhöht die Übersicht für dich, aber auch für jemand anderen, der deinen **Quellcode**, also dein Programm, liest.

Aufgabe 4: Aussichtstreppe
Karol soll nun aus den bereits bekannten Befehlen und mithilfe von selbstdefinierten Methoden eine Aussichtstreppe bauen, die 4 Steine lang ist und am höchsten Punkt 4 Steine hoch ist. Danach soll sich Karol zur Treppe wenden.

Ein mögliches Programm dazu lautet:

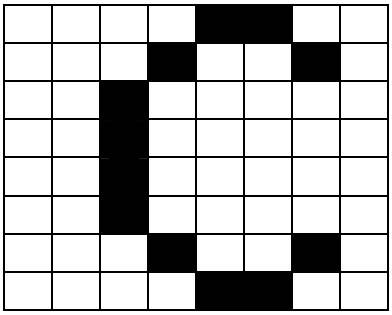


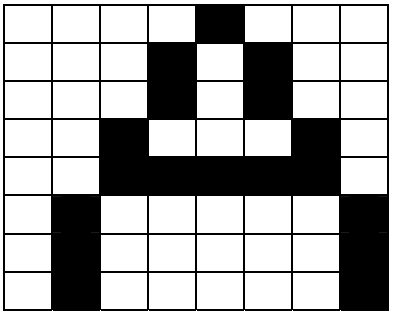

```
Anweisung ZweierReihe
  Hinlegen
  Schritt
  Hinlegen
  Schritt
*Anweisung

Anweisung DreierReihe
  Hinlegen
  Schritt
  Hinlegen
  Schritt
  Hinlegen
  Schritt
*Anweisung

Anweisung ViererReihe // besteht aus 2*ZweierReihe
ZweierReihe
ZweierReihe
*Anweisung

Anweisung Startpunkt
LinksDrehen
Schritt(4)
RechtsDrehen
Schritt(2)
RechtsDrehen
*Anweisung

Anweisung Umdrehen
LinksDrehen
LinksDrehen
*Anweisung

Startpunkt                  // gehe zum Startpunkt
ViererReihe                 // lege die unterste Reihe (=4 Steine)
  Umdrehen
  Schritt(3)                // gehe nur 3 Schritte zurück
  Umdrehen
    DreierReihe             // lege die zweite Reihe (=3 Steine)
      Umdrehen
      Schritt(2)            // gehe nur 2 Schritte zurück
      Umdrehen
        ZweierReihe         // lege dritte Reihe (=2 Steine)
          Umdrehen
          Schritt
          Umdrehen
            Hinlegen        // ganz oben nur einen Stein
            Schritt
Umdrehen                    // hurra! geschafft!
```

Wie aus dem Beispielprogramm ersichtlich, scheint es erst mal mühsam und umständlich, sich eigene Methoden zu definieren, die dann als Befehle verwendet werden können. Da man aber nach der Definitionsphase immer wieder darauf zurückgreifen kann, ist der modulartige Aufbau dann aber doch bequem: falls Karol z.B. zwei Treppen bauen will, vergrößert sich der Aufwand nur geringfügig. Hätte man keine selbstdefinierten Methoden vereinbart, würde sich der Programmcode verdoppeln – was sehr zeitaufwändig, speicherplatzverschwendend und unübersichtlich wäre.

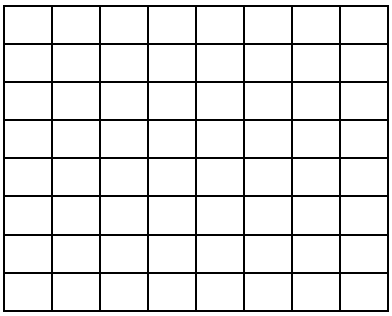
Aufgabe 5: Initialen
Karol möchte dich beeindrucken und in seinem Garten den Anfangsbuchstaben deines Vornamens pflastern. Programmiere Karol so, dass er einen Buchstaben aus Ziegelsteinen auf den Boden legt. Du kannst dir dabei vorstellen, dass der Buchstabe aus einzelnen Punkten aufgebaut ist – er besteht also aus Punkten in einem „Gitter“. Wähle die Größe der Welt 8*8 Felder.

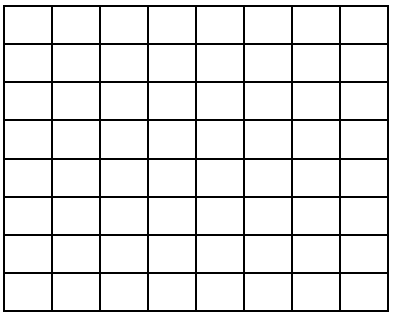
Beispiel 2:





Hier sind zwei Raster, in die du einen Buchstaben schreiben kannst:



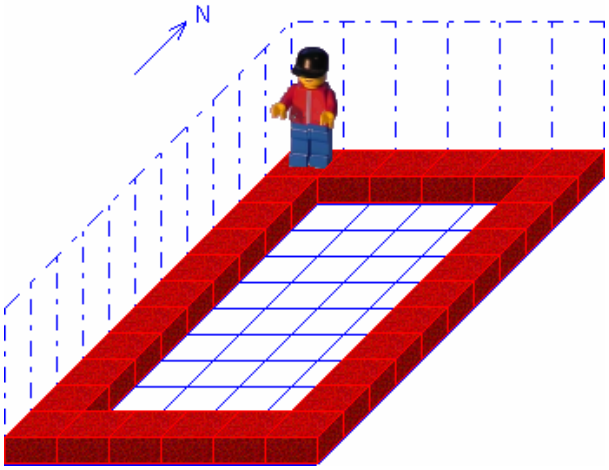


⇗ **INFO:** Bis zur Einführung von ***true type*** basierten alle Buchstaben auf einem Raster von 8*8 **Pixeln** (=*picture elements* =Bildschirmpunkte). *True type* bedeutet, dass alle Zeichen einer Schriftart ihre Größe ändern können – in einer Textverarbeitung kann man dies durch die Schriftgröße einstellen, wobei die Schriftgröße der Anzahl von „echten“ (=*true*), also „abzählbaren“ Pixeln entspricht: bei Schriftgröße 24 wird das Raster automatisch auf 24 Pixel umgerechnet. Man hat die gleiche Schriftart, nur eben größer ⇨ „grobkörniger“. Deutlich sichtbar wird dies bei einem Buchstaben, wenn du eine riesige Schriftgröße einstellst (z.B. 500). Am Rand liegt der sog. „Treppeneffekt“ vor, der durch „auspixeln“ der Buchstaben entsteht.
Auch beim digitalen Fernsehen sieht man manchmal bei nicht optimaler Empfangsqualität einzelne kleine Quadrate, aus denen sich das Fernsehbild zusammensetzt.

Aufgabe 6: Gartenmauer
Karol möchte eine Mauer um sein Grundstück bauen, d.h. am Rand der Welt. Wähle dabei die Größe der Welt 6*11 Felder. Verwende eigene Methoden, um das Hauptprogramm möglichst kurz zu halten.

Ein mögliches Programm lautet:

```
//eigene Methoden definieren
Anweisung FünferReihe
  Hinlegen
  Schritt
  Hinlegen
  Schritt
  Hinlegen
  Schritt
  Hinlegen
  Schritt
  Hinlegen
  Schritt
*Anweisung

Anweisung ZehnerReihe
  FünferReihe
  FünferReihe
*Anweisung

//Hauptprogramm
LinksDrehen           // obere Mauer
FünferReihe
  RechtsDrehen        // rechte Mauer
  ZehnerReihe
  RechtsDrehen        // untere Mauer
    FünferReihe
      RechtsDrehen    // linke Mauer
      ZehnerReihe
      RechtsDrehen
      RechtsDrehen
```



Obwohl Karol nur eine Mauer von 5*10 legt, muss seine Welt aus 6*11 Feldern bestehen, da er die Steine nicht auf das Feld legt, worauf er gerade steht, sondern auf das Feld daneben (vgl. Grafik).

I.3 Struktogramme

⇗ **INFO:** Die Befehlsabfolge eines Programms kann mithilfe eines sog. „**Struktogramms**“ dargestellt werden. Dabei werden Befehle und **Sequenzen** (=Folgen nacheinander auszuführender Anweisungen) nach bestimmten einheitlichen Vorschriften strukturiert dargestellt. Struktogramme dienten in der Vergangenheit oft zur Dokumentation von kleineren Programmen. Dabei war das Struktogramm immer der erste Schritt – anhand der einzelnen Strukturblöcke war ersichtlich, was das Programm leisten sollte. Vorteil von Struktogrammen ist, das man ein Programm ohne Computer bzw. Compiler auf dem Papier programmieren kann. Ein Struktogramm lässt sich in alle existierenden Programmiersprachen umsetzen, da es sehr allgemein ist. Große Nachteile sind aber, dass sich nur einfache Programme darstellen lassen und Struktogramme schnell sehr unübersichtlich werden können. Deshalb sind sie für die heutige Programmierung i.d.R. nicht mehr bedeutend.

Einzelbefehle werden bei Struktogrammen einfach in untereinanderliegende Kästchen geschrieben:

Struktogramm 1:

Hauptprogramm

LinksDrehen
FÜNFERREIHE
RechtsDrehen
ZEHNERREIHE
RechtsDrehen
FÜNFERREIHE
RechtsDrehen
ZEHNERREIHE
RechtsDrehen
RechtsDrehen

Anw.: FÜNFERREIHE

Hinlegen
Schritt
Hinlegen
Schritt
Hinlegen
Schritt
Hinlegen
Schritt
Hinlegen
Schritt

Anw.: ZEHNERREIHE

FÜNFERREIHE
FÜNFERREIHE

Spätestens jetzt weißt du, warum die bisher bekannten Anweisungen unter „**iterativ**“ (=schrittweise) zusammengefasst werden können: deine Programme 1-6 werden schrittweise abgearbeitet, deshalb sind die Struktogramme jeweils aufgebaut wie Struktogramm 1.

I.4 Ablaufgeschwindigkeit

Die Geschwindigkeit der Programmausführung kann bei Karol beeinflusst werden. Karol kann Befehle auf zwei Arten ausführen: langsam oder schnell. Bei langsamer Ausführung legt Karol nach jedem Befehl eine „kleine Verschnaufpause“ ein, damit man den Ablauf des Befehls am Bildschirm besser verfolgen kann. Bei einer Programmausführung mittels Schnelldurchlauf tritt diese Verzögerung nicht auf. Mittels der Schlüsselwörter `schnell` und `langsam` lässt sich die Ausführungsgeschwindigkeit steuern.

Aufgabe 7a: Tanzkurs für Anfänger
Karol möchte einen Tanzkurs machen – und Du sollst dabei der Tanzlehrer sein! Dazu muss die Geschwindigkeit der Programmausführung beeinflusst werden. Damit sich Karol in alle vier Richtungen bewegen kann, musst Du 3 eigene Anweisungen schreiben (der Befehl `Schritt` existiert ja bereits): `RechtsSchritt`, `LinksSchritt` und `SchrittZurück`. Auf diese Weise soll Karol die Grundschritte des Tanzens lernen.

Wie immer bietet es sich also an, möglichst viele Anweisungen selbst zu definieren, um danach bequem darauf zugreifen zu können. Beachte dabei, dass die eigenen Anweisungsblöcke zunächst definiert werden müssen, bevor Du auf sie zugreifen kannst.
Achte auch weiterhin darauf, dass die Programmierung möglichst übersichtlich ist, also z.B. eine Leerzeile zwischen zwei Blöcken, Einrücken innerhalb einer Anweisung, sinnvolle Namen der Anweisungen etc.

Eine mögliche Lösung lautet:

```
// Eigene Anweisungen werden definiert
Anweisung Umdrehen
     Schnell
     LinksDrehen
     LinksDrehen
*Anweisung

Anweisung SchrittZurück
     Schnell
     Umdrehen
     Schritt
     Umdrehen
*Anweisung

Anweisung LinksSchritt
     Schnell
     LinksDrehen
     Schritt
     RechtsDrehen
*Anweisung

Anweisung RechtsSchritt
     Schnell
     RechtsDrehen
     Schritt
     LinksDrehen
*Anweisung
```

Aufgabe 7b: Tanzkurs für Fortgeschrittene
Mithilfe der vier Richtungsanweisungen kann Karol sich nun nach Belieben bewegen und nicht nur verschiedene Tänze, sondern sogar verschiedene Choreographien ausführen! Erweitere nun Dein Tanzprogramm um zwei eigens definierten Anweisungen „Squaredance“ und „Rap“, in denen Du die Choreographien festlegst, nach denen sich Karol über das Tanzparkett bewegen soll.
„Squaredance“ bedeutet dabei, dass sich Karol, in eine Richtung schauend, in einem Quadrat bewegt.
Beim „Rap“ kannst Du Dir eine coole Freestyle-Variante überlegen – hier gibt es keine Regeln.
Beachte bei beiden Tanzrichtungen, dass sich Karol in einer nicht zu kleinen Welt etwa in der Mitte befindet, um nicht an den Seiten anzustoßen.

Eine mögliche Lösung lautet:

```
Anweisung Squaredance
     LinksSchritt
     LinksSchritt
     Schritt (2)
     RechtsSchritt
     RechtsSchritt
     SchrittZurück
     SchrittZurück
*Anweisung

Anweisung Rap
     LinksSchritt
     LinksSchritt
     RechtsSchritt
     Schritt
     RechtsSchritt
     RechtsSchritt
     SchrittZurück
*Anweisung
```

Nun sind zwar beide Choreographien definiert, aber noch passiert nichts: Wie immer musst Du als letzten Befehl einen der Tänze aufrufen:

```
// Hauptprogramm
Squaredance
```

bzw.

```
// Hauptprogramm
Rap
```

Wenn du möchtest, kannst du Karol noch weitere Tänze beibringen. Frag doch deine Eltern nach ihrem Lieblingstanz, lass dir die Schrittfolge erklären und programmiere diese dann mit Karol. Vergiss nicht, dass du die Geschwindigkeit am Ende eines Anweisungsblockes mit `langsam` wieder verringern kannst...

II. Wiederholungsanweisungen

II.1 Wiederholung mit fester Anzahl

Bisher musste Karol beim Legen einer Reihe von 5 Ziegelsteinen immer wieder das Gleiche tun: Er wiederholte fünfmal die Sequenz „`Hinlegen Schritt`“. Für eine solche Wiederholung mit fester Anzahl kennt Karol eine eigene Anweisung:

Beispiel 3:

```
Wiederhole 5 mal
     Hinlegen
     Schritt
*Wiederhole
```

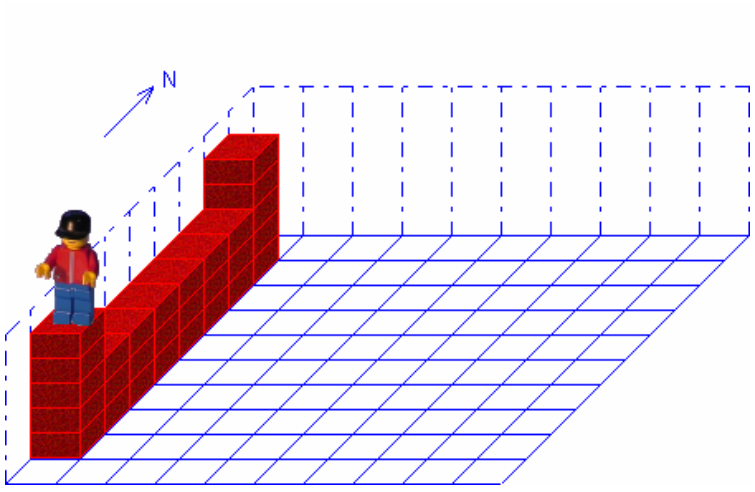
Struktogramm zu Beispiel 3:

Wiederhole 5 mal
Hinlegen
Schritt

Aufgabe 8a:
Karols Gartenzaun aus Holz ist teilweise morsch geworden, er möchte deswegen das kaputte Element durch ein Element aus Ziegelsteinen ersetzen. In einem Gartenprospekt hat er eine passende Vorlage gefunden: Das Mauerelement besteht aus 18 Steinen mit der Länge 6 und der Höhe 3. Eingerahmt wird das Mauerelement von zwei Pfeilern, die jeweils 5 Steine hoch sind.
Definiere zwei eigene Anweisungen „`Pfeiler`“ und „`Element`“ und lasse Karol die neue Teilmauer bauen!

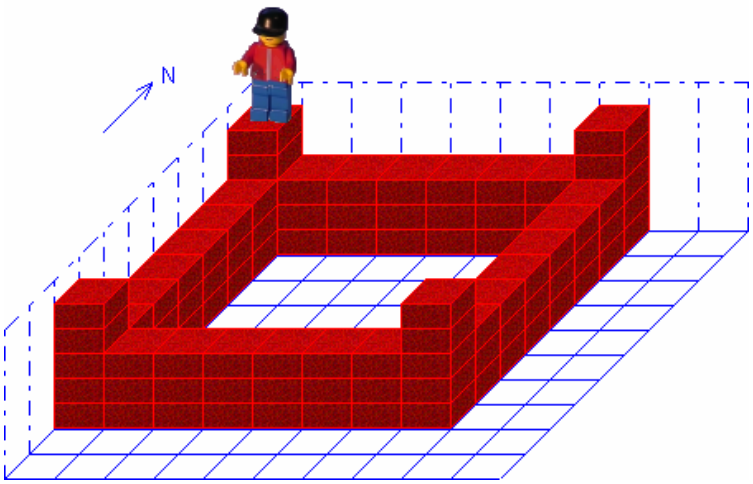
Eine mögliche Variante ist:

```
// Anweisungen
Anweisung Pfeiler
     Wiederhole 5 mal
          Hinlegen
     *Wiederhole
     Schritt
*Anweisung

Anweisung Element
     Wiederhole 3 mal
          Hinlegen
     *Wiederhole
     Schritt
*Anweisung

Anweisung Teilmauer
     Pfeiler                  // setze den ersten Pfeiler
     Wiederhole 6 mal         // lege 6 Mal 3 Steine aufeinander
          Element
     *Wiederhole
     Pfeiler                  // setze den zweiten Pfeiler
*Anweisung

// Hauptprogramm
Teilmauer
```



Aufgabe 8b:
Karols alter Holzzaun ist nun völlig kaputt, er möchte deshalb um sein ganzes Grundstück eine Mauer bauen. Verändere Dein bestehendes Programm so, dass Karol eine Mauer baut, die aus 4 solcher Teilmauern besteht. Dabei sollen insgesamt nur 4 Pfeiler entstehen, also in jeder Ecke einer.
Verwende dazu eine weitere Wiederholungsanweisung, so dass das Mauerbauen „geschachtelt" wird.
Tipp: Damit Du nur 4 Pfeiler baust und nicht 8, beginne jedes Teilstück mit den niederen Elementen, an die Du nur am Schluss einen Pfeiler anbaust.

```
// Anweisungen
Anweisung Pfeiler
      Wiederhole 5 mal
            Hinlegen
      *Wiederhole
      Schritt
*Anweisung

Anweisung Element
      Wiederhole 3 mal
            Hinlegen
      *Wiederhole
      Schritt
*Anweisung

Anweisung Mauer
    Wiederhole 4 mal          // wiederhole 4 mal, (für alle Seiten):
        Wiederhole 6 mal      // lege 6 Mal 3 Steine aufeinander
            Element
        *Wiederhole
    Pfeiler                   // und setze den Endpfeiler
    LinksDrehen               // dann dreh dich und wiederhole alles
    *Wiederhole
*Anweisung

// Hauptprogramm
Mauer
```

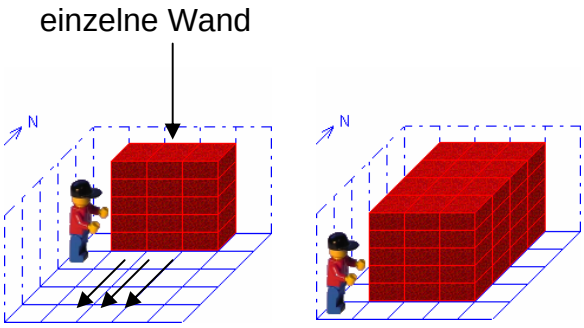


Durch sinnvolles Einrücken in der Anweisung „Mauer" erkennst Du, was „zusammengehört": alle Befehle mit gleicher Einrückung bedeuten eine Einheit. So kannst Du die innere Wiederholungsanweisung erkennen, die nur den Befehl „Element" wiederholt und die Äußere, die dafür sorgt, dass genau viermal die niederen Elemente aneinandergesetzt und mit einem Pfeiler abgeschlossen werden, bevor Karol sich nach Links dreht und die restlichen Mauerteile anbaut.

Aufgabe 9: Tanzfläche
Karol ist sehr stolz auf seine neue Gartenmauer. Er hatte ja vorher bereits seinen restlichen Garten umgestaltet (Sitzplatz, vgl. Aufgabe 2, Aussichtstreppe, vgl. Aufgabe 4, Initialen, Aufgabe 5) und möchte nun für seine Freunde eine Gartenparty veranstalten. Da er auch den Tanzkurs erfolgreich abgeschlossen hat, möchte er eine Tanzfläche bauen, um so richtig abrocken zu können. Die Tanzfläche soll ein Quader mit den Maßen 4*3*5 (Länge * Breite * Höhe haben) .

Tipp: Um den Quader zu bauen, stelle dir einfach einzelne „Wände“ vor, die aneinander gebaut werden. Somit wird der Quader nach vorne gezogen (vgl. Beispielprogramm, evtl. im Einzelschrittmodus mit F7).

Ein mögliches Programm lautet:




```
// Programm Tanzfläche in Quaderform
Anweisung Wand
  Wiederhole 3 mal
    Hinlegen(5)
    Schritt
  *Wiederhole
*Anweisung

Anweisung Tanzfläche
LinksDrehen
  Wiederhole 4 mal // baue 4 Mal eine Wand nach vorne in den Raum
    Wand
    RechtsDrehen
    Schritt
    RechtsDrehen   // und gehe dann zum nächsten Anfangspunkt
    Schritt(3)     // einer Wand
    RechtsDrehen
    RechtsDrehen
  *Wiederhole
*Anweisung

Tanzfläche
```

II.2 Bedingungen

Bisher konnte Karol nur dann eine Mauer ganz außen um seine Welt bauen, wenn die Größe der Welt bekannt war. Geht das auch anders? Wünschenswert wäre auch, dass Karol von jedem beliebigen Punkt in seiner Welt genau bis zur Wand läuft, ohne daran anzustoßen. Ist dies mit einer Wiederholung mit fester Anzahl möglich?
Karol muss überprüfen können, ob er die Wand erreicht hat. Dazu verwendet er die Anweisung „`IstWand`“. Steht er vor einer Wand, liefert diese die Antwort (den **Wahrheitswert**) „wahr“ (was soviel wie „ja“ bedeutet). Befindet sich vor ihm keine Wand, liefert die Anweisung „falsch“ (was wiederum soviel wie „nein“ bedeutet). Derartige Anweisungen gehören zu einer speziellen Klasse von Anweisungen, die **Bedingungen** genannt werden.
Weitere vordefinierte Bedingungen sind:

Befehl	Erklärung
`IstWand`	WAHR, wenn Karol vor der Wand steht und in diese Richtung schaut; ansonsten FALSCH
`NichtIstWand`	WAHR, wenn Karol nicht vor einer Wand steht; ansonsten FALSCH
`IstZiegel`	WAHR, wenn Karol vor einem Ziegel(stapel) steht und zu diesem schaut; ansonsten FALSCH
`NichtIstZiegel`	WAHR, wenn Karol nicht vor einem Ziegel(stapel) steht; ansonsten FALSCH

Es gibt noch weitere vordefinierte Bedingungen, die hier aber aus zeitlichen Gründen nicht behandelt werden. Falls du sich dafür interessierst, kannst du im unteren linken Kontrollfenster unter dem Menüpunkt „vordefinierte Bedingungen“ nachschauen.

II.3 Bedingte Wiederholung

Um eine Aufgabe wie unter II.2 beschrieben lösen zu können, brauchen wir noch eine Kontrollstruktur, die eine Sequenz von Anweisungen solange ausführt, bis eine Bedingung ihren Wahrheitswert von „wahr“ auf „falsch“ ändert.

Aufgabe 10a: Blinde Kuh
Karol möchte auf seiner Gartenparty ein Spiel spielen: er bekommt die Augen verbunden und muss solange wie möglich in eine Richtung gehen, ohne an die Wand anzustoßen. Dabei legt er bei jedem Schritt einen Ziegel auf den Boden. Um zu gewinnen, muss er direkt vor der Wand stehen bleiben.

Variante 1:

```
Solange NichtIstWand tue
      Hinlegen
      Schritt
*Solange
```

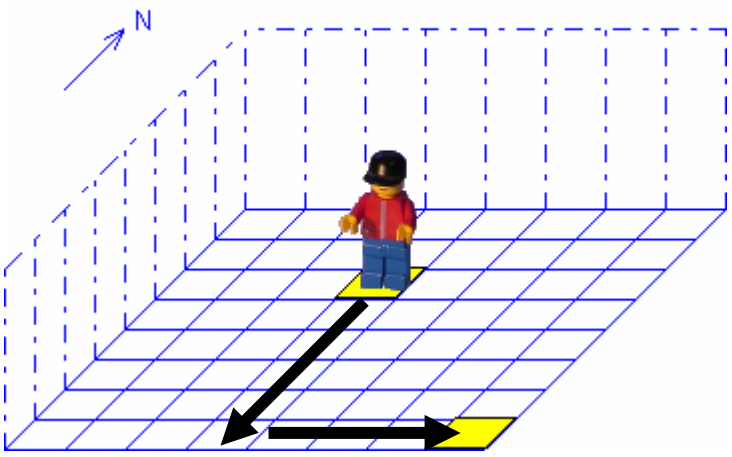
```
Solange NichtIstWand
      Hinlegen
      Schritt
```

Variante 2:

```
Wiederhole Solange NichtIstWand
      Hinlegen
      Schritt
*Wiederhole
```

```
Wiederhole Solange NichtIstWand
      Hinlegen
      Schritt
```

Aufgabe 10b: Eckestehen
Karol trug in der Schule eine Baseballkappe und war zudem mehrfach unaufmerksam, deshalb muss er den Rest der Stunde mit dem Gesicht zur Wand in einer Ecke verbringen.
Egal, wo sich Karol im Klassenzimmer befindet, soll er ohne Umweg in die von seiner Blickrichtung aus gesehen linke vordere Ecke gehen (vgl. Markierung in der Grafik) und über seine Unaufmerksamkeit nachdenken. Dabei darf Karol nicht an die Wand stoßen, sonst bekommt er schon vor dem Nachdenken Kopfweh.
Schreibe zwei Anweisungen „`BisZurWand`“ und „`EckeStehen`“ – Karol soll durch den Befehl „`EckeStehen`“ dann loslaufen.



```
// bis zur Wand geradeaus und nicht weiter
Anweisung BisZurWand
     Solange NichtIstWand tue
          Schritt
     *Solange
*Anweisung

Anweisung EckeStehen
     BisZurWand          // bis zur 1.Wand
     LinksDrehen         // drehen
     BisZurWand          // bis zur 2.Wand = in die Ecke
*Anweisung

EckeStehen
```

Wie bereits in Aufgabe 6 gesehen hast du Karol eine Mauer um seinen Garten bauen lassen. Dabei war es wichtig, die Größe des Grundstücks zu kennen. Nun sollen die Aufgaben 10a und 10b als Grundstein für eine weitere Variante dienen:

Aufgabe 10c: Gartenmauer, Variante 2
Ohne die Länge und Breite des Gartens zu kennen, d.h. egal, wie lange oder breit das Grundstück ist, soll Karol ganz außen eine Mauer bauen. Verwende dabei die bereits bestehenden Anweisungen aus Aufgabe 10b und erweitere dein Programm um die eigene Methode „MauerBauen“.

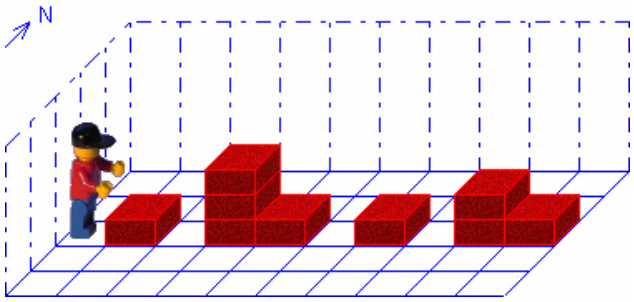
Ein mögliches Programm lautet:

```
// bis zur Wand geradeaus und nicht weiter
Anweisung BisZurWand
     Solange NichtIstWand tue
          Schritt
     *Solange
*Anweisung

Anweisung EckeStehen
  BisZurWand          // bis zur 1.Wand
  LinksDrehen         // drehen
  BisZurWand          // bis zur 2.Wand = in die Ecke
*Anweisung

Anweisung MauerBauen
  EckeStehen          // erst mal zur Ecke, diese dient als Startpunkt
  LinksDrehen         // drehen und los geht's!
  Wiederhole 4 mal    // für alle 4 Seiten soll Folgendes geschehen:
    Solange NichtIstWand tue
      Hinlegen
      Schritt
    *Solange
    LinksDrehen
  *Wiederhole
*Anweisung

MauerBauen
```

Aufgabe 10b

Aufgabe 11: Garten aufräumen

Karols coole Gartenparty ist vorbei – leider ist einiges an Müll übriggeblieben, und Karol muss alles alleine wegräumen. Der Gartenweg ist von Müll übersät, und Karol kommt nicht mehr in sein Haus!
Lege eine Welt mit 10*5 an und lege einige Ziegelsteine (=Müll) auf beliebigen zufällig gewählten Feldern ab. Die Ziegelsteine (=Müll) dürfen sich dabei beliebig hoch auftürmen, sollen sich aber alle in einer Reihe befinden.
Positioniere Karol nun vor einer Wand und lasse ihn bis zur anderen Wand gehen. Dabei soll er alle Ziegelsteine (=Müll) restlos aufsammeln.



Ein mögliches Programm lautet:

```
Anweisung Aufräumen
     Solange NichtIstWand tue      // gehe bis zur Wand
          Solange IstZiegel tue    // falls Müll vorhanden: aufheben!
          Aufheben
          *Solange                 // ansonsten: ein Feld weiter
          Schritt
     *Solange
*Anweisung

Aufräumen
```

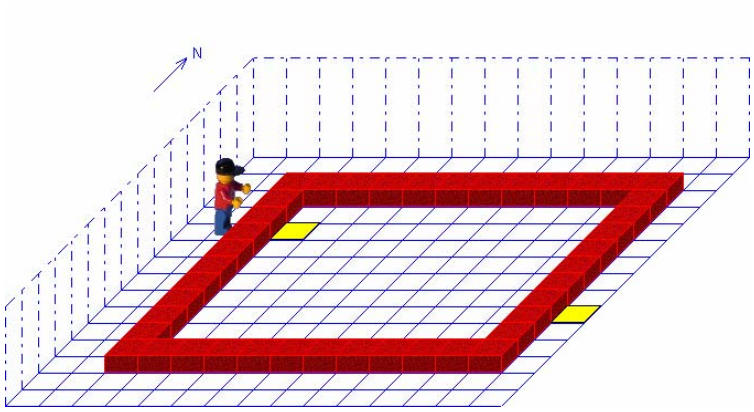
Aufgabe 12 (*anspruchsvoll*): Ali Karol und die Schatzhöhle

Kaum ist Karols Gartenparty vorbei, denkt er auch schon wieder über neue Partyhighlights nach – er hat sich ein Spiel überlegt, bei dem seine Gäste auf unterhaltsame Weise eine Überraschung erleben können. Das Grundprinzip ist wie bei Ali Babas Schatzhöhle: Die Tür öffnet sich nur unter bestimmten Umständen. Karol hat eine Trittplatte mit elektronischem Sensor verlegt; steigt man darauf, so öffnet sich das Bauwerk und man kann eintreten. Da man nicht weiß, wo genau sich die Platte befindet, muss man am Rand um die Höhle herumspazieren. Im Inneren verbirgt sich der gleiche Mechanismus – man will die Höhle ja auch wieder verlassen.

1. Lege eine Höhle an (die Begrenzung bestehe aus Ziegelsteinen) und lass Karol außen an der Mauer entlang laufen (Endlosschleife).
2. Baue nun eine Trittplatte in Form einer Marke vor die Mauer und lass Karol wieder der Mauer entlang starten und bis auf die Platte laufen.
3. Lass bei Betreten der Platte die Mauer aufgehen, Karol hindurchtreten und schließe die Mauer wieder.
4. Wiederhole nun die Schritte 2 und 3 für das Höhleninnere, so dass Karol wieder nach draußen gelangt.

Wenn du alles richtig gemacht hast, durchläuft Karol eine Endlosschleife von außen nach innen, von innen nach außen etc.

⇗ **INFO:** Nach diesem Prinzip werden übrigens auch Roboter programmiert, um den Weg aus einem beliebigen Labyrinth zu finden: sie gehen einfach so lange immer rechtsrum oder immer linksrum, bis sie automatisch an ihr Ziel gelangen...




```
// Ali Karol und die 40 Räuber – Welt z.B. nach dieser Vorlage anlegen
Anweisung Umdrehen
  LinksDrehen
  LinksDrehen
*Anweisung

Anweisung SesamÖffneDich
Aufheben
Schritt(2)
Umdrehen
Hinlegen
*Anweisung

Anweisung GeheimtürAussen
Solange NichtIstMarke tue
  Wenn IstZiegel dann
    RechtsDrehen
      Wenn NichtIstZiegel dann
        Schritt
        LinksDrehen
      sonst
        Schritt
        LinksDrehen
      *Wenn
      sonst
      Schritt
      LinksDrehen
  *Wenn
*Solange
SesamÖffneDich
*Anweisung

Anweisung GeheimtürInnen
Solange NichtIstMarke tue
  Wenn IstZiegel dann
    RechtsDrehen
      Wenn IstZiegel dann
        RechtsDrehen
        Schritt
        LinksDrehen
      sonst
        Schritt
        LinksDrehen
      *Wenn
  *Wenn
*Solange
SesamÖffneDich
*Anweisung

//Hauptprogramm
GeheimtürAussen
GeheimtürInnen
```