Einführung in die Algorithmik mit Kara

Mathias Müller

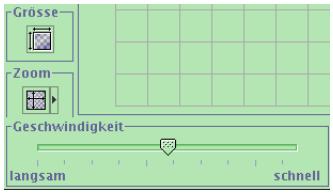
17. November 2005

1 Kara in seiner Welt

Karas Startbildschirm bietet verschiedene Kara-Varianten an. Ein Klick auf eines der Fenster öffnet Karas Welt. Wählen Sie hier „Kara — programmieren mit Automaten“.

Kara lebt in einer rechteckigen Welt mit Kleeblättern, Pilzen und Baumstümpfen. Um Karas Welt zu gestalten, zieht man einfach mit der Maus die Gegenstände von der Welt–Leiste in das Spielfeld. Nicht mehr benötigte Gegenstände entsorgt „wirft“ man per Maus in den Mülleimer. Durch Mausklicks auf die blauen Icons der Kara–Leiste kann man Kara durch seine Welt steuern.

Über das Icon „Größe“ werden Höhe und Breite von Karas Welt verändert. Die Größe der Darstellung stellt man mit „Zoom“ ein. Der Schiebeschalter „Geschwindigkeit“ legt fest, wie schnell Kara-Programme ausgeführt werden. Im Dateimenü lassen sich eigene Welten speichern und vorgefertigte Welten laden.



Aufgabe 1: Gestalten Sie die gezeigte Welt und speichern Sie sie als *Welt1*.

Aufgabe 2: Lassen Sie Kara per Handsteuerung seine neue Welt erkunden.

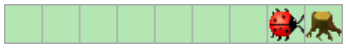
Aufgabe 3: Welche Aktionen kann Kara ausführen? Was geschieht, wenn Kara an den Rand seiner Welt gelangt?

Aufgabe 4: Ermitteln Sie die Eigenschaften der einzelnen Gegenstände:

- Welche kann Kara überwinden?
- Welche kann er verschieben? Wieviele auf einmal?
- Welche lassen sich aufheben und ablegen?
- Können mehrere Gegenstände übereinander liegen? Welche und wieviele?

Aufgabe 5: Gestalten und speichern Sie die Welt *Baumsuche*:

Aufgabe 6: Steuern Sie Kara so durch seine Welt, dass folgender Endzustand entsteht. Protokollieren Sie die einzelnen Schritte des „Programms“:



Aufgabe 7: Formulieren Sie den Programmablauf umgangssprachlich so, dass er auch für ähnliche Welten gültig ist, z.B.:

2 Kara mit Automaten programmieren

2.1 Automaten

Ein Automat ist, einfach gesprochen, ein Modell zur Steuerung von Maschinen, z.B. Ampeln, Handies, Geldwechselautomaten, Waschmaschinen und natürlich — Kara.

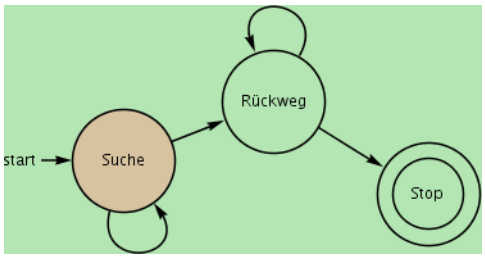
Ein Beispiel: Kara will sich zum Abendbrot ein leckeres Kleeblatt holen. Dazu muss er nacheinander verschiedene Teilaufgaben lösen:

1. das Kleeblatt suchen und aufnehmen
2. das Kleeblatt zu seinem Schlafplatz am Baumstumpf tragen

Anfang

Ende

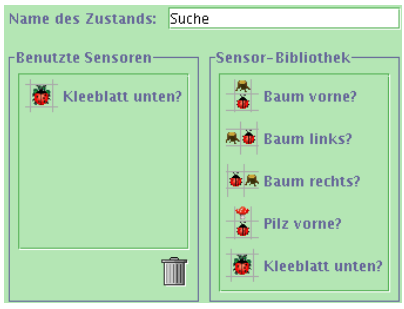
Das dazugehörige Programm bzw. den Automaten *Kleeblatt_holen* stellt man in einem *Zustandsdiagramm* dar. Dieses Diagramm zeigt alle *Zustände*, in denen sich Kara befinden kann, und die möglichen *Übergänge* zwischen diesen Zuständen. Die Gesamtheit aller Zustände und Übergänge nennt man auch den *Zustandsraum*.



2.2 Ein Programm für Kara

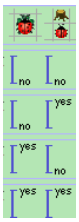
Ein Klick auf *Programmieren* öffnet Karas Programmeditor. Über die verschiedenen Menüpunkte in der linken Leiste kann man Zustände erzeugen, bearbeiten und löschen sowie den Startzustand eines Programms festlegen.

Über den Menüpunkt *new* öffnet man das Menü zur Erzeugung eines neuen Zustandes. Es bietet eine Auswahl von *Sensoren* an, mit denen Kara die Bedingungen seiner Umwelt prüfen kann. Je nach Aufgabenstellung zieht man die benötigten Sensoren in die linke Seite des Menüs.



Jeder Zustand sollte einen möglichst aussagekräftigen Namen erhalten. Das verbessert die Lesbarkeit der Programme.

Programmiert wird ein Zustand über die *Zustandsübergangstabelle*. Abhängig von den Ergebnissen der Sensoren legt sie fest, welche Aktionen Kara durchführen muss und in welchem Zustand er danach weiter arbeitet. Ein Sensor kann die Ergebnisse *yes* (wahr) oder *no* (falsch) liefern.



Beim Ausfüllen der Zustandsübergangstabelle sollte man der besseren Übersicht wegen eine bestimmte Reihenfolge einhalten. Bei nur einem Sensor beginnt man mit *no*. Die nebenstehende Abbildung zeigt die Reihenfolge für zwei Sensoren.

Aufgabe 1: Testen Sie das Programm *Kleeblatt_holen*. Erläutern Sie den Programmablauf anhand des Zustandsdiagramms.

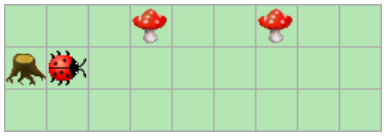
Aufgabe 2: Erläutern Sie die gemeinsamen Merkmale aller Welten, in denen Ihr Programm funktioniert (die sogenannte *Invariante* der Welten).

Aufgabe 3: Das Programm soll auch funktionieren, wenn sich in Karas Welt kein Kleeblatt befindet. Überlegen Sie, wie Kara feststellt, dass seine Suche vergeblich war.

Aufgabe 4: Passen Sie das Programm der erweiterten Aufgabenstellung an. Speichern Sie es als *Kleeblatt_holen_2*.

Inzwischen hat es geregnet und auf Karas Weg sind Pilze gewachsen. Wenn Kara auf einen Pilz trifft, muss er ihn zur Seite schieben.

Anfang



Ende

Aufgabe 5: Erstellen Sie das Programm *Kleeblatt_holen_mit_Pilzen*. Gehen Sie der Einfachheit halber davon aus, dass auf jeden Fall ein Kleeblatt vorhanden aus.

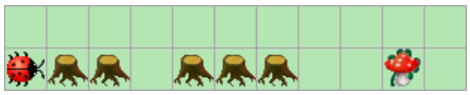
Aufgabe 6: Definieren Sie für das Verschieben der Pilze einen eigenen Zustand und ändern Sie Ihr Programm zu *Kleeblatt_holen_mit_Pilzen_2*.

Aufgabe 7: Überlegen Sie, welche Vorteile es hat Teilaufgaben in eigene Zustände auszulagern.

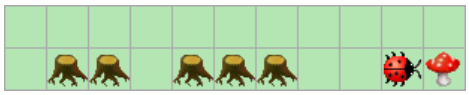
3 Vom Problem zum Programm

Die bisherigen Probleme waren noch so einfach, dass sie sich mehr oder weniger „zu Fuß“ lösen ließen. Komplexere Probleme erfordern eine sorgfältige Planung, um allgemeingültige und fehlerfreie Programme zu erstellen.

Beispiel: Kara sucht wieder ein Kleeblatt. Dabei muss er eventuell unterschiedlich lange Baumreihen umgehen. Außerdem kommt er nur dann an das begehrte Blatt heran, wenn er den darüber stehenden Pilz wegschiebt:



Anfang



Ende

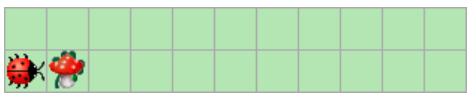
3.1 Problemanalyse

3.1.1 Beschreiben der Welt

Zuerst überlegt man sich die *Invariante* der Welt, für die das Programm erstellt wird. Das ist die Gesamtheit aller unveränderlichen Merkmale dieser Welt:

- Kara steht in Richtung eines Pilzes, der auf dem Kleeblatt steht.
- Bäume können, müssen aber nicht auf Karas Weg stehen.
- Die Baumreihen sind genau einen Baum breit, aber beliebig lang.
- Eine Baumreihe darf nicht direkt vor dem Pilz enden, da Kara den Pilz sonst nicht erkennt.
- Direkt hinter dem Pilz darf kein Baum stehen, da sich der Pilz sonst nicht schieben lässt.

Jetzt zeichnet man verschiedene Varianten der Welt. Besonders interessant sind dabei Spezialfälle, z.B.:



3.1.2 Ermitteln der Teilprobleme

Anschließend listet man die Teilprobleme auf, die Kara für die Erfüllung seiner Aufgabe zu lösen hat.

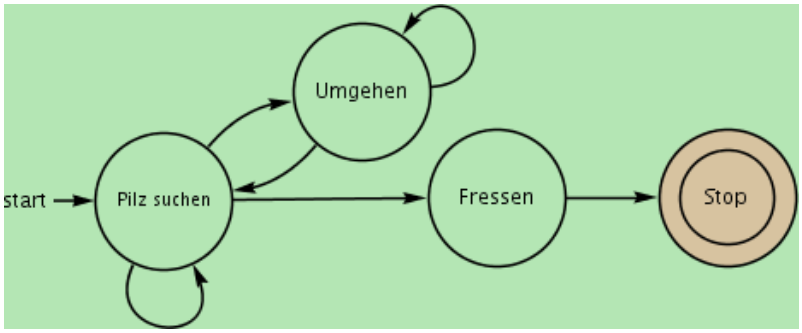
- nach dem Pilz suchen, dabei auf Bäume achten
- Umgehen einer Baumreihe
- Verschieben des Pilzes
- Fressen des Kleeblattes

3.2 Lösungsentwurf

3.2.1 Erstellen des Zustandsdiagramms

Die notwendigen Zustände ergeben sich oftmals direkt aus den aufgelisteten Teilproblemen. Eventuell lassen sich auch mehrere Teilprobleme in einem Zustand zusammenfassen.

Man überlegt sich alle möglichen Übergänge zwischen den Zuständen und zeichnet das Zustandsdiagramm. Eine mögliche Lösung:



3.2.2 Spezifizieren der Zustände

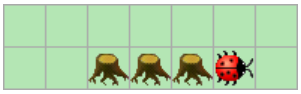
Für jeden Zustand wird jetzt festgelegt:

- Wo und wie steht Kara, wenn er in den Zustand wechselt?
- Wo und wie steht er, wenn er den Zustand verlässt?
- Welche Sensoren benötigt er für jeden Zustand?

Ein Beispiel: Im Zustand *Umgehen* muss Kara bei jedem Schritt prüfen, ob noch ein Baum rechts neben ihm steht. Daraus folgt:

- Er benötigt den Sensor *Baum rechts?*
- Beim Übergang *Pilz suchen*→*Umgehen* steht er links neben dem ersten Baum.
- Beim Übergang *Umgehen*→*Pilz suchen* steht er hinter dem letzten Baum in Richtung Pilz:

Anfang *Umgehen*



Ende *Umgehen*

3.3 Implementierung

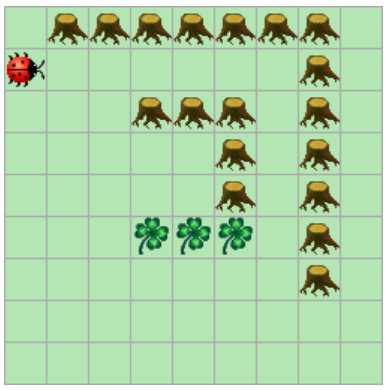
Nachdem die beschriebenen Vorbereitungen abgeschlossen sind, erstellt man in Karas Programmeditor die Zustände und programmiert die Zustandsübergangstabellen.

3.4 Test

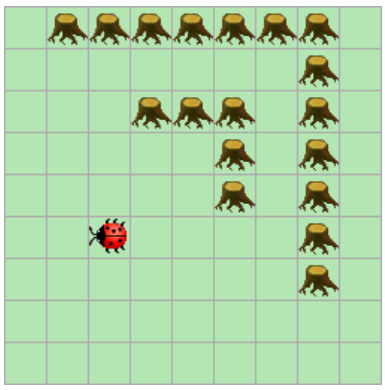
Das erstellte Programm wird nun mit den verschiedenen Varianten der Welt getestet. Besonders wichtig ist es Spezialfälle in die Tests einzubeziehen.

Aufgabe 1: Notieren Sie die Phasen der Programmentwicklung anhand eines eigenen Beispiels.

Aufgabe 2: Entwerfen, implementieren und testen Sie das Programm *Tunnel_mit_Abzweig*. Kara steht vor einem Tunnel, dessen Eingang er zunächst finden muss. An einer beliebigen Stelle biegt der Tunnel nach rechts ab. Rechts neben dem Ausgang des Tunnels liegt eine Kleeblattspur, die Kara fressen will:



Anfang



Ende

Aufgabe 3: Erweitern Sie das Programm für folgende Welt: Kara weiß nicht, ob der Tunnel nach rechts oder nach links abbiegt.

Aufgabe 4: Zusätzlich gilt für das Ende des Tunnels: Endet die rechte Wand zuerst, dann liegt die Kleeblattspur rechts. Endet die linke Wand zuerst, liegt sie links. Enden beide Wände gleichzeitig, dann verläuft sie in Richtung des Tunnels.

4 Von Klassen und Objekten

Die Informatik betrachtet alle Lebewesen und Gegenstände als *Objekte*. Jedes Objekt besitzt bestimmte Eigenschaften (*Attribute*) und Fähigkeiten (*Methoden*), z.B.:

Objekt: mein Auto

Attribute
- Farbe = taubenblau metallic
- Höchstgeschwindigkeit = 100 $\frac{km}{h}$
- Geschwindigkeit = 0

...

Methoden
+ beschleunigen
+ bremsen
+ blinken
+ hupen

...

Da alle Autos gemeinsame Attribute und Methoden besitzen, fasst man die Beschreibung für Autos in der *Klasse Auto* zusammen:

Auto
- farbe - hoechstGeschwindigkeit - aktuelleGeschwindigkeit ...
+ beschleunigen + bremsen + blinken + hupen ...

Die einzelnen Auto–Objekte unterscheiden sich nur in ihren *Attributwerten*, also in ihrer speziellen Farbe, ihrer Höchstgeschwindigkeit usw.

Aufgabe 1: Beschreiben Sie die Attribute und Methoden des Objektes Kara.

Aufgabe 2: Das Bild zeigt außer Kara noch drei andere Marienkäfer. Wie könnte eine Klasse *Marienkäfer* aussehen, zu der alle vier Käfer gehören?

Aufgabe 3: Erläutern Sie anhand des Beispiels Kara die Bedeutung der Begriffe Objekt, Klasse, Attribut, Attributwert und Methode.

Aufgabe 4: Starten Sie Kara und wählen Sie *JavaKara – mit Java programmieren*. Öffnen Sie mit einem Klick auf *Programmieren* den Programmeditor. Informieren Sie sich im Programmtext des Beispielprogramms über die Java–Schreibweise von Karas Methoden.

5 Einführung in die Arbeit mit JavaKara

5.1 Der Aufbau eines JavaKara–Programms

Auch ein Javaprogramm ist ein Objekt, genau wie ein Auto oder ein Marienkäfer. Deshalb (be-)schreibt man es in Form einer *Klasse*:

```
public class MeinProgramm extends JavaKaraProgram {
  //Hier können eigene Methoden stehen.
  public void myProgram() {
    // Hier steht das Hauptprogramm.
  }
}
```

Ein JavaKara–Programm enthält immer das Hauptprogramm `myProgram`. Außerdem kann es selbst geschriebene Methoden enthalten.

In Java wird jedes Programm, jede Methode und jeder Programmblock in geschwungene Klammern `{ }` eingeschlossen. Um bei längeren Programmen den Überblick zu behalten, rückt man den Programmtext zwischen den Klammern um zwei Stellen ein.

Kommentare erhöhen die Verständlichkeit des Programmtextes. Sie werden speziell gekennzeichnet:

```
// Alles bis zum Zeilenende ist Kommentar.
/* Das ist ein Kommentar
   über mehrere Zeilen. */
```

Java–Programme werden in speziellen Dateien gespeichert. Eine Java–Programmdatei hat denselben Namen wie die Klasse und die Endung `*.java`. Die Datei zu `MeinProgramm` heißt also `MeinProgramm.java`.

Java unterscheidet zwischen Groß- und Kleinschreibung. `Programm` ist z.B. nicht dasselbe wie `programm`.

Bevor sie ein Java–Programm starten können, müssen Sie den Programmtext in ein ausführbares Programm übersetzen lassen. Drücken Sie dazu auf *Programm kompilieren*. Wenn die Übersetzung erfolgreich war, erscheint nur die Zeile `executed: javac ...` Ansonsten werden zusätzliche Fehlermeldungen angezeigt.

Hinweis: Karas Programmeditor bietet jedesmal ein Beispielprogramm an. Löschen Sie zuerst alle überflüssigen Zeilen, bevor Sie mit dem Programmieren beginnen.

5.2 Ein altbekanntes Problem — neu betrachtet

Erinnern Sie sich noch an Ihr erstes Automaten–Programm? Kara sollte ein Kleeblatt zu seinem Schlafplatz holen:

Anfang

Ende

Will man mit dem Objekt Kara „javanisch“ sprechen, muss man es zuerst beim Namen nennen und ihm dann einen Befehl geben, z.B. „Kara, geh einen Schritt vorwärts“:

```
kara.move();
```

Aufgabe 1: Notieren Sie den Aufbau und die Besonderheiten von Javakara–Programmen.

Aufgabe 2: Schreiben Sie das Programm `KleeblattHolen` für die gezeigte Welt. Alle Befehle stehen innerhalb des Hauptprogramms `myProgram`. Übersetzen und testen Sie Ihr Programm.

6 Kara erweitert seine Fähigkeiten — eigene Methoden

Im Programm `KleeblattHolen` muss Kara Aktionen wie das Wenden mehrmals ausführen. Um nicht jedesmal zwei Rechts-oder Linksdrehungen programmieren zu müssen, bringt man Kara eine neue Fähigkeit bei — man schreibt die Methode `wenden`:

```
public void wenden() {
  kara.turnLeft();
  kara.turnLeft();
}
```

Die neue Methode kann man im Hauptprogramm und in anderen Methoden benutzen:

```
public void myProgram() {
  ...
  this.wenden();
  ...
}
```

Hinweis: Das aktuelle Programm(-objekt) wird über den Namen `this` („dieses hier“) angesprochen. Auf die Methoden des aktuellen Programms greift man deshalb mit `this.`*<methodenname>* zu.

Aufgabe 1: Schreiben Sie die Methode `vierSchritte`. Ersetzen Sie die entsprechenden Abschnitte im Hauptprogramm durch den neuen Befehl.

Aufgabe 2: Ändern Sie das Hauptprogramm so, dass es nur noch aus zwei Befehlen besteht und schreiben Sie die dazu nötigen Methoden (bereits existierende Methoden können Sie nutzen):

```
public void myProgram() {
  this.kleeblattSuchen();
  this.rueckWeg();
}
```

7 Kara wird flexibel — Programmschleifen

7.1 Kara wiederholt sich

Bisher funktioniert Ihr JavaKara–Programm nur in einer einzigen Welt. Liegt z.B. das Kleeblatt nicht vier, sondern fünf Schritte entfernt, muss ein neues Programm geschrieben werden. Besser wäre es, man könnte Kara den Befehl geben: „Laufe, *solange* du noch kein Kleeblatt gefunden hast!“ Das erreicht man mit einer *Programmschleife*.

Ein Beispiel: Kara soll links an einer Baumreihe vorbei gehen und dann stehen bleiben. Die Baumreihe kann beliebig lang sein.

Anfang

Ende

Kara erhält den Befehl:

solange rechts neben Kara ein Baum steht
  Kara, geh einen Schritt

oder auf „javanisch“:

```
while(kara.treeRight()) {
  kara.move();
}
```

Die Befehle in der Programmschleife werden solange wiederholt, wie die Sensormethode `treeRight()` den Wert `true` (wahr, yes) liefert.

Ein Ausrufezeichen kehrt den Wert der Sensormethode um. Dann läuft die Schleife, solange der Sensor `false` (falsch, no) liefert. Der Befehl „Gehe, solange kein Pilz vor dir steht“ würde z.B. lauten:

```
while(!kara.mushroomFront()) {
  kara.move();
}
```

7.2 Vorher oder nachher — das ist hier die Frage

Die bisher besprochene Form der Programmschleife nennt man die *vorprüfende Schleife*. Eine andere Form ist die *nachprüfende Schleife*, z.B.:

führe aus
  Kara, geh einen Schritt
solange rechts neben Kara ein Baum steht

oder auf „javanisch“:

```
do {
  kara.move();
}
while(kara.treeRight());
```

Aufgabe 1: Vergleichen Sie die vorprüfende und die nachprüfende Schleife:

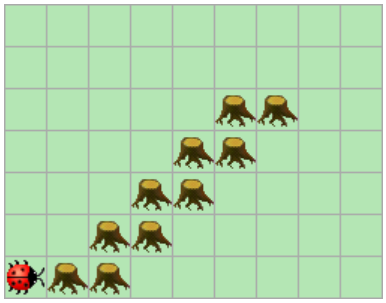
- Wann wird die Bedingung geprüft, wann die Befehle ausgeführt?
- Wie oft läuft die Schleife mindestens?
- In welchem Fall setzt man günstiger welche Schleife ein?

Aufgabe 2: Formulieren Sie folgende Programmschleifenköpfe:

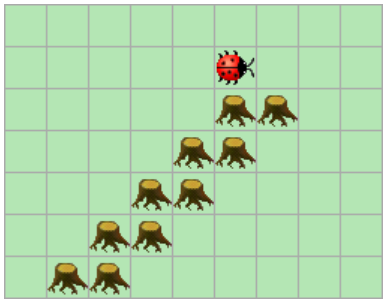
- solange vor Kara ein Baum steht ...
- solange rechts neben Kara kein Baum steht ...
- solange unter Kara ein Kleeblatt liegt ...
- solange vor Kara kein Pilz steht

Aufgabe 3: Ändern Sie Ihr Programm `KleeblattHolen` so, dass Kara beliebig weit entfernte Kleeblätter findet.

Aufgabe 4: Schreiben Sie das Programm `TreppenSteigen` für beliebig hohe Treppen. Überlegen Sie dazu, wie Kara erkennt, ob er noch eine Stufe hochsteigen muss. Schreiben Sie dazu zuerst die Methode `stufeSteigen`, die Sie dann innerhalb der Programmschleife aufrufen können:



Anfang



Ende

Aufgabe 5: Probieren Sie sowohl die vorprüfende als auch die nachprüfende Schleife aus. Überlegen Sie, für welche Variante der Welt die nachprüfende Schleife nicht geeignet ist.

8 Kara trifft Entscheidungen — bedingte Anweisungen

Im Alltag entscheidet oft die konkrete Situation über unser Tun. Die Entscheidung darüber, ob wir eine Handlung ausführen oder nicht, hängt von bestimmten Bedingungen ab, z.B.:

- ***Wenn*** es regnet, ***dann*** nehme ich den Regenschirm mit.
- ***Wenn*** ich kein Brot mehr habe, ***dann*** kaufe ich ein neues.
- ***Wenn*** die Mauer nicht höher als zwei Meter ist, ***dann*** springe ich.
- ***Wenn*** die Ampel rot zeigt, ***dann*** bleibe ich stehen, ***sonst*** gehe ich weiter.
- ***Wenn*** Schnee liegt, ***dann*** fahre ich mit der Bahn, ***sonst*** nehme ich das Fahrrad.

Auch Kara muss häufig Entscheidungen treffen. Das ermöglichen ihm *bedingte Anweisungen*.

8.1 Tu ich's oder tu ich's nicht — die einseitige Abfrage

Kara will einen Schritt gehen. Damit er sich keine Beule holt, muss er vorher prüfen, ob ein Baum vor ihm steht. Der Befehl an Kara lautet also:

wenn vor Kara kein Baum steht
  Kara, geh einen Schritt

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

Kara geht nur dann einen Schritt, wenn kein Baum vor ihm steht. Ansonsten wird der Befehl einfach übersprungen.

Aufgabe 1: Formulieren Sie folgende bedingte Anweisungen mit einseitiger Abfrage:

- Wenn rechts neben Kara ein Baum steht, dreht er sich nach links.
- Wenn unter Kara ein Kleeblatt liegt, nimmt er es auf und dreht sich um 180°.
- Wenn Kara vor einem Pilz steht, umgeht er ihn.

Aufgabe 2: Kara arbeitet in seinem Kleeblattbeet. Einige Kleeblätter sind wegen der herrschenden Trockenheit eingegangen und Kara muss sie ersetzen. Schreiben Sie dazu das Programm `KleeblaetterErsetzen`. Es enthält die Methode `kleeblattErsetzen`, die das Ersetzen eines Kleeblattes ermöglicht:

Anfang

Ende

Aufgabe 3: Kara sorgt für den Winter vor und bepflanzt die leeren Fächer in seinem Kräuterkasten. Unter jedes Kleeblatt schiebt er einen Düngepilz, damit es besser gedeiht:

Anfang

Ende

Hinweis: Schreiben Sie zuerst die Methode `fachBepflanzen`. Überlegen Sie dann, wie lange Kara diese Methode ausführen muss.

8.2 Die Qual der Wahl — die zweiseitige Abfrage

Wieder will Kara einen Schritt gehen. Diesmal soll er nicht einfach stehen bleiben, wenn er vor einem Baum steht, sondern den Baum umgehen. Er steht also vor der Entscheidung:

wenn vor Kara kein Baum steht
  Kara, geh einen Schritt
sonst
  umgehe den Baum

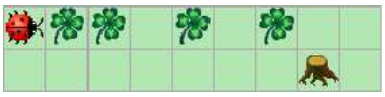
```
if(!kara.treeFront()) {
  kara.move();
}
else {
  this.umgehen();
}
```

Kara muss also entweder die eine oder die andere Handlung ausführen. Ein Überspringen der gesamten Anweisung ist nicht möglich.

Aufgabe 4: Formulieren Sie folgende bedingte Anweisungen mit zweiseitiger Abfrage:

- Wenn Kara vor einem Pilz steht, wendet er, sonst legt er ein Kleeblatt ab.
- Wenn rechts neben Kara ein Baum steht, dreht er sich nach links, sonst dreht er sich nach rechts.
- Wenn Kara auf einem Kleeblatt steht, nimmt er es auf und geht einen Schritt, sonst legt er eines ab und wendet.

Aufgabe 5: Kara organisiert sein Kleeblattbeet um: Alte Kleeblätter werden geerntet. Überall dort, wo bisher kein Kleeblatt stand, wird ein neues gepflanzt. Schreiben Sie das Programm `KleeblaetterTauschen`. Lagern Sie das Tauschen eines einzigen Kleeblattes in die Methode `kleeblattTauschen` aus:



Anfang

Ende

Aufgabe 6: Kara will die Seitennischen eines Ganges mit Kleeblättern bepflanzen. Schreiben Sie dazu das Programm `KleeblaetterVerteilen`. Schreiben Sie für das Ablegen eines Kleeblattes die Methode `pruefenUndAblegen`:

Anfang

Ende

Hinweis: Eine Nische befindet sich bei jedem Schritt entweder links oder rechts von Kara. Die Fälle *zwei Nischen* sowie *keine Nische* treten nicht auf:

Nische links

Nische rechts

zwei Nischen

keine Nische

9 Hmmm ... die Kunst des Verfeinerns

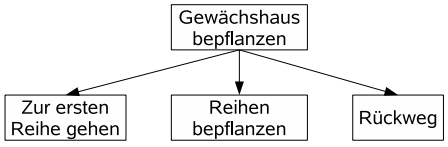
Kara hat sich ein Gewächshaus für Pilze und Kleeblätter angelegt. Die Pilze wachsen bereits, jetzt müssen die Kleeblätter gesteckt werden:

Anfang

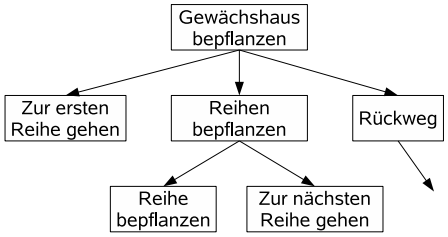
Ende

Bei diesem Problem hat man es mit einer Vielzahl größerer und kleinerer Teilprobleme zu tun, die z.T. mehrmals auftreten. Um hier zügig zu einer fehlerfreien und gut strukturierten Lösung zu kommen, beginnt man die Problemanalyse mit einer groben Einteilung des Problems. Die Teilprobleme werden dann nach und nach immer weiter zerlegt. Diese Methode wird als *schrittweises Verfeinern* bezeichnet:

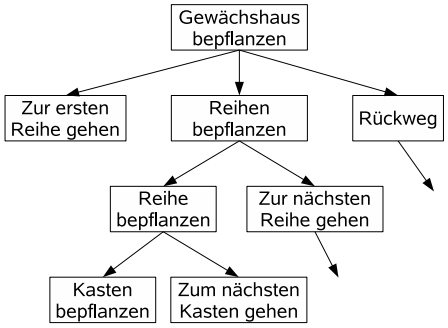
Grobeinteilung Das Problem kann grob in folgende Teilprobleme eingeteilt werden:

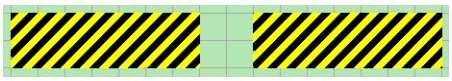


Erste Verfeinerung des Teilproblems *Reihen bepflanzen*: Das Gewächshaus besteht aus drei Reihen mit Blumenkästen. Kara hat also folgende Teilprobleme zu lösen:

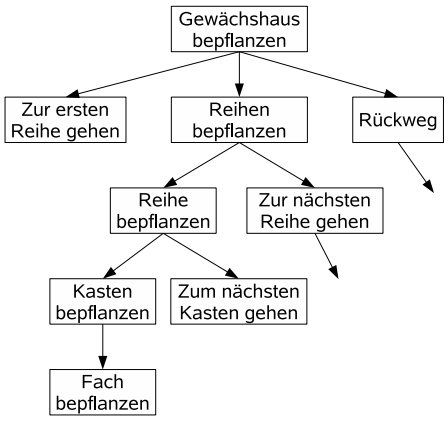


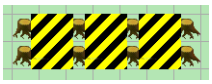
Zweite Verfeinerung Jede Reihe besteht aus zwei Blumenkästen:





Dritte Verfeinerung Jeder Kasten besteht aus drei Fächern:





Weitere Verfeinerungen Die Problemanalyse wird solange weiter verfeinert, bis die Teilprobleme so klein sind, dass sie sich problemlos lösen lassen..

Programmierung Die Programmierung erfolgt in umgekehrter Richtung. Zuerst schreibt man Methoden für die einfachsten Teilprobleme, z.B. `fachBepflanzen`. Jede Methoden wird sorgfältig getestet. Erst wenn man sicher ist, dass sie fehlerfrei funktioniert, kann man sie in Methoden wie `kastenBepflanzen` verwenden. So entsteht nach und nach das Programm aus immer größer werdenden Bausteinen.

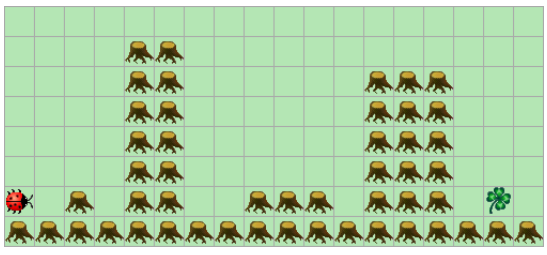
Aufgabe 1: Vervollständigen Sie die Problemanalyse für das gezeigte Problem.

Aufgabe 2: Programmieren Sie die Lösung im Programm `Gewaechshaus`.

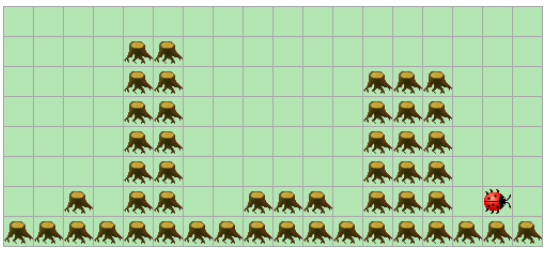
Aufgabe 3: Erweitern Sie Ihr Programm für Reihen mit beliebig vielen Kästen.

Aufgabe 4: Ändern Sie das Programm für ein Gewächshaus mit beliebig vielen Reihen. Überlegen Sie zuerst anhand der Problemanalyse die nötigen Änderungen. Denken Sie auch darüber nach, wie Kara seine Ausgangsposition so kennzeichnen kann, dass er sie wiederfindet. Ändern Sie dann gezielt Ihr Programm.

Aufgabe 5: Kara sucht ein Kleeblatt. Dabei muss er beliebig hohe und breite Waldstücke umgehen. Lösen Sie das Problem mit der Methode der schrittweisen Verfeinerung:



Anfang



Ende

10 Und ... oder nicht, oder wie — Kara denkt logisch

10.1 Noch einmal: Das Tunnelproblem

Bereits In der Aufgabe auf Seite 7 musste Kara einen Tunnel durchqueren. Den Ausgang des Tunnels erkannte er daran, dass nicht mehr rechts *und* links neben ihm ein Baum stand. Sehen Sie sich noch einmal die Zustandsübergangstabelle des Zustandes *Ausgang finden* an: Wenn die Sensoren  (*Baum links*) und  (*Baum rechts*) den Wert *yes* liefern, muss Kara weiter suchen.

10.2 Programmieren mit logischen Funktionen

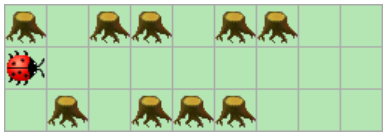
Jede Sensorfunktion, die Sie in Schleifen und bedingten Anweisungen verwenden, liefert einen der Werte `true` (wahr, yes, 1) oder `false` (falsch, no, 0). Um das Tunnelproblem auch für JavaKara lösen zu können, müssen muss man die Ergebnisse zweier Sensormethoden miteinander verknüpfen. Dazu erstellt man zuerst eine Wertetabelle. Im Beispiel muss Kara weiter suchen, wenn beide Sensormethoden den Wert `true` liefern:

`kara.treeRight()`	`kara.treeLeft()`	`kara.move()`
`false`	`false`	`false`
`false`	`true`	`false`
`true`	`false`	`false`
`true`	`true`	`true`

Diese Wertetabelle entspricht der UND–Funktion (siehe Anhang A). Die Suchschleife wird also programmiert als:

```
while(kara.treeRight() && kara.treeLeft()) {
  kara.move();
}
```

Aufgabe 1: Kara läuft eine Allee entlang. Das Ende der Allee erkennt er daran, dass keine Bäume mehr neben ihm stehen:



Anfang

Ende

Aufgabe 2: Kara läuft durch einen Tunnel. Zuerst muss er den Eingang, dann den Ausgang finden. Vervollständigen Sie die Lösung:

Anfang

Ende

Aufgabe 3: Kara läuft, bis er auf einen Pilz trifft. Jedesmal, wenn er auf einem Blatt oder neben einem Baum steht, dreht er sich vor Freude einmal um die eigene Achse:

Anfang

Ende

Aufgabe 4: Kara frisst eine Kleeblattspur. Welche Kleeblätter er jeweils liegen lässt, zeigen die verschiedenen Endzustände:

Anfang

Ende 1

Ende 2

Ende 3

A Logische Funktionen im Überblick

A.1 Grundfunktionen

Funktion	Wertetabelle	Programmierung in Java
NICHT (NOT) $Y = \overline{A}$	A | Y 0 | 1 1 | 0	`!kara.treeFront()`
UND (AND) $Y = A \wedge B$	A | B | Y 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1	`kara.onLeaf() && kara.treeFront()`
ODER (OR) $Y = A \vee B$	A | B | Y 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1	`kara.onLeaf() || kara.treeFront()`

A.2 Zusammengesetzte Funktionen

Durch Kombinationen logischer Grundfunktionen lassen sich eine Reihe weiterer Funktionen bilden. Dabei muss der Vorrang der Operationen beachtet werden (NOT vor AND vor OR). bei Bedarf werden Teilfunktionen eingeklammert:

Funktion	Wertetabelle	Programmierung in Java
NICHT UND (NAND) $Y = \overline{A \wedge B}$	A | B | Y 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0	`!(kara.onLeaf() && kara.treeFront())`
NICHT ODER (NOR) $Y = \overline{A \vee B}$	A | B | Y 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0	`!(kara.onLeaf() || kara.treeFront())`
EXCLUSIV–ODER (XOR) $Y = A \oplus B$ bzw. $Y = \overline{A} \wedge B \vee A \wedge \overline{B}$	A | B | Y 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0	`!kara.onLeaf() && kara.treeFront() || kara.onLeaf() && !kara.treeFront()`