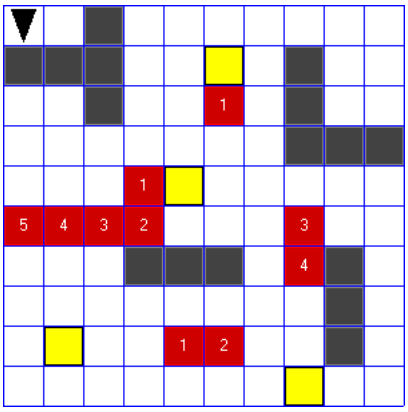
Algorithmische Grundstrukturen mit Robot Karol

Mathias Müller

15. September 2005

1 Karol in seiner Welt

1.1 Eine Welt für Robot Karol



Über das Icon  oder mit `Welt` ⇒ `Neue Welt` erstellt man eine neue, leere Welt. Die beim Erstellen festgelegten Maße lassen sich später nicht mehr beeinflussen.
Im 2D–Modus lassen sich Welten konfigurieren. Drücken Sie dazu `F6` oder wählen Sie `Welt` ⇒ `Welt direkt festlegen`.

Aufgabe: Erstellen Sie eine neue Welt mit der Höhe 5 und konfigurieren Sie sie wie abgebildet. Speichern Sie die Welt als `Handsteuerung.kdw`.

1.2 Robot Karol konfigurieren

Robot Karol selbst konfiguriert man über `Einstellungen` ⇒ `Karol`.

Aufgabe: Geben Sie Karol folgende Eigenschaften:

- Karol soll überprüfen können, wieviele Ziegel er bei sich trägt. Er soll maximal 10 Ziegel tragen können und aktuell 3 Ziegel tragen.
- Karol soll maximal einen Ziegel überspringen können.
- Bei einem Programmfehler bricht das Programm ab.

1.3 Karol von Hand steuern

Über die Leiste 

erreicht man die Steuerbefehle für Robot Karol. Die Sensoren sind im Handsteuermodus nicht zugänglich.

Aufgabe: Bewegen Sie Robot Karol durch seine Welt und lassen Sie ihn verschiedene Aktionen ausführen. Überprüfen Sie auch sein Verhalten bei folgenden Aktionen:

- 6 Ziegel übereiander stapeln,

- einen Ziegel auf eine Marke legen,
- auf verschiedene und von verschiedenen Stufen der „Treppe“ springen,
- von einem Ziegelstapel auf einen Quader laufen

2 Robot Karols erstes Programm

2.1 Hinweise zu Robot–Karol–Programmen

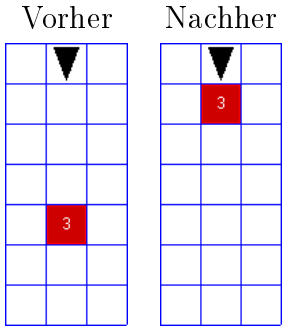
Programmdateien haben das Format `*.kdp`, Welten `*.kdw`. Die Endungen werden beim Speichern automatisch angehängt.

Lädt man ein Programm, dann wird ggf. die gleichnamige Welt mit geladen.

Die Konfiguration für Robot Karol wird nicht mit dem Programm gespeichert. Sie müssen die gewünschte Konfiguration also jedesmal von Hand einstellen.

2.2 Robot Karol holt Material

Problem: Karol will einen Stapel Ziegel bearbeiten. Dazu muss er ihn zuerst holen:



Aufgabe: Schreiben Sie ein einfaches Programm für dieses Problem.

Hinweis: Das Hauptprogramm sollte wegen der besseren Lesbarkeit in die Anweisungen `Programm ... *Programm` eingeschlossen werden.

3 Robot Karol lernt neue Anweisungen

Bestimmte Aktionen, wie das Wenden, muss Robot Karol mehrmals ausführen. Um nicht immer dieselben Anweisungsfolgen eintippen zu müssen, kann man Robot Karol neue Anweisungen beibringen. Diese schreibt man *vor* das Hauptprogramm in der Form:

```
Anweisung <Name>
  <Anweisungsfolge>
*Anweisung
```

Die neue Anweisung kann dann durch Aufrufen ihres Namens benutzt werden.

Hinweis: Der Befehl `Schnell` zu Beginn einer Anweisung entfernt die Verzögerung zwischen den einzelnen Anweisungsschritten. Die Anweisung wirkt dann „wie aus einem Guss“.

Aufgabe 1: Schreiben Sie die Anweisung `Wenden`. Verwenden Sie die neue Anweisung für alle Wendungen im Hauptprogramm.

Aufgabe 2: Schreiben Sie weitere sinnvolle Anweisungen.

Aufgabe 3: Robot Karol soll rückwärts zu seinem Platz zurück laufen. Wie könnte die Anweisung `SchrittRueckwaerts` aussehen?

Aufgabe 4: Robot Karol soll den gesamten Ziegelhaufen Schritt für Schritt rückwärts ziehen. Schreiben Sie eine geeignete Anweisung.

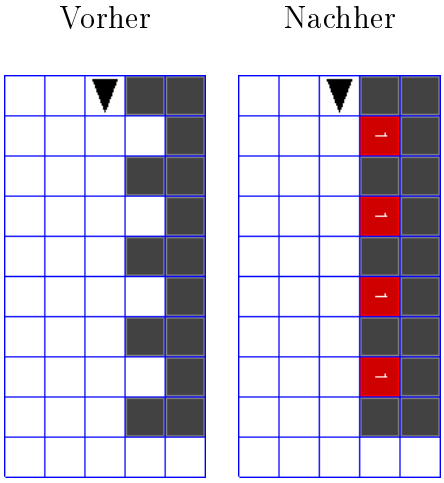
4 Ein Speicher für Robot Karols Befehle

Viele Anweisungen werden in verschiedenen Programmen benötigt. Um sie nicht immer wieder neu schreiben zu müssen, kann man sie in sogenannten Bibliotheken speichern. Man schreibt die gewünschten Anweisungen in eine separate Datei, die man wie eine normale Programmdatei speichert. Will man die Anweisungen in einem Programm verwenden, muss man zu Beginn des Programmtextes die Bibliotheksdatei importieren:

```
Einfügen
  <Pfad und Dateiname der Bibliothek>
*Einfügen
```

Aufgabe: Schreiben Sie Ihre selbst erstellten Anweisungen in eine Bibliotheksdatei. Verwenden Sie dann in Ihrem Programm nur noch die Anweisungen aus der Bibliothek.

5 Robot Karol wiederholt sich



Problem: Robot Karol arbeitet in einer Bücherei. Er soll in jedes Fach eines Regals genau ein Buch legen und sich dann in seine Ausgangsposition begeben.

Aufgabe: Schreiben Sie ein Programm für dieses Problem. Erstellen Sie eigene Anweisungen für mehrmals wiederholte Vorgänge.

Hinweis: Bestimmt können Sie auch einige Anweisungen aus Ihrer Bibliotheksdatei verwenden.

5.1 Robot Karol in der Schleife

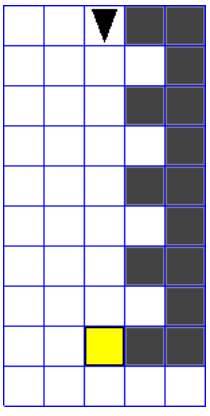
Im Büchereiprogramm mussten viermal dieselben Anweisungen aufgerufen werden. Das kann lästig werden, vor allem bei zehn, fünfzig oder mehr Wiederholungen. Man verwendet deshalb eine Programmschleife:

```
Wiederhole <n> mal
  <Anweisungen>
*Wiederhole
```

Für `<n>` setzt man die gewünschte Anzahl Wiederholungen ein.

Aufgabe: Lösen Sie das Büchereiproblem mit Hilfe einer Programmschleife.

5.2 Robot Karol wird flexibel



Problem: Bisher hat Robot Karol nur Regale mit vier Fächern bestückt. Für jede andere Regalgröße müsste man ein neues Programm schreiben. Besser wäre eine Anweisung: „Bestücke die Fächer solange, bis das Regal zuende ist!“

Zunächst muss Karol das Ende des Regals erkennen können. Dazu setzt man hinter das letzte Fach eine Marke.

Man benötigt nun eine Programmschleife, die läuft, solange bzw. bis eine Bedingung erfüllt ist. Mögliche Formen dieser Schleife sind:

```
Wiederhole
  <Anweisungen>
*Wiederhole bis <Bedingung>
```

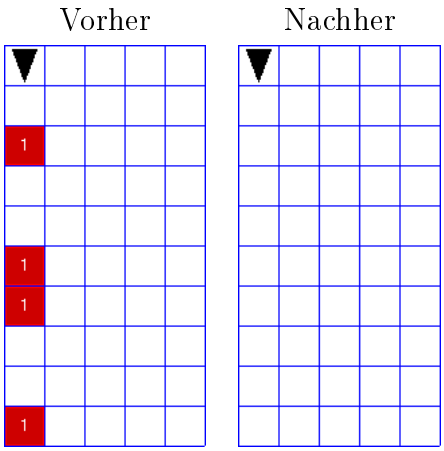
```
Wiederhole solange <Bedingung>
  <Anweisungen>
*Wiederhole
```

Aufgabe 1: Ändern Sie das Büchereiprogramm so, dass Robot Karol Regale beliebiger Größe bestücken kann. Verwenden Sie dazu beide Formen der Programmschleife. Testen Sie das Programm mit verschiedenen Regalgrößen.

Aufgabe 2: Beschreiben Sie die Unterschiede zwischen beiden Schleifenformen (wann wird geprüft, wie oft läuft die Schleife mindestens, wie muss die Bedingung formuliert werden ...).

6 Robot Karol muss Entscheidungen treffen

6.1 Robot Karol räumt auf



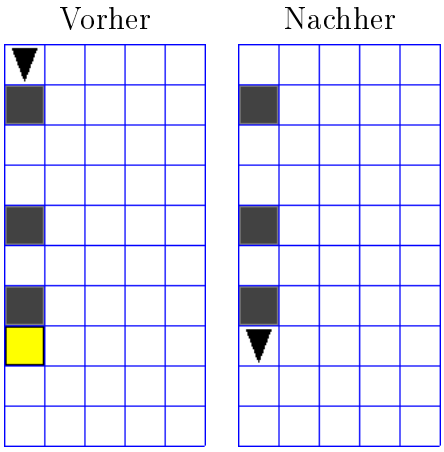
Problem: Es ist Feierabend — und die Baustelle ein einziges Chaos. Überall liegen verstreute Ziegel herum! Karol muss also die Reihe ablaufen und alle Ziegel einsammeln, bevor er an seinen Ruheplatz zurückkehren kann. Allerdings soll er sich nur bücken, wenn wirklich ein Ziegel vor ihm liegt.

Da hier Anweisungen nur unter bestimmten Bedingungen ausgeführt werden sollen, benötigt man eine bedingte Anweisung:

```
wenn <Bedingung> dann
  <Anweisungen>
*wenn
```

Aufgabe: Lassen Sie Robot Karol alle Ziegel in der Reihe einsammeln. Verwenden Sie, wo es sinnvoll erscheint, Programmschleifen. Testen Sie das Programm mit unterschiedlich langen Reihen und verschiedenen Ziegelpositionen.

6.2 Robot Karol sucht ein Handy



Problem: Der Meister hat in der Maschinenhalle sein Handy verloren. Robot Karol soll es suchen und einstecken. Auf der Suche stehen ihm jedoch immer wieder Maschinen im Weg.

Robot Karol muss also entweder eine Maschine umgehen oder einfach einen Schritt vorwärts gehen. Hier benötigt man eine bedingte Anweisung, die eine Entscheidung zwischen zwei Möglichkeiten zulässt:

```
wenn <Bedingung> dann
  <Anweisungen>
sonst
  <Anweisungen>
*wenn
```

Hinweis: Einen Quader kann man mit der Bedingung `istWand` aufspüren.

Aufgabe: Schreiben Sie ein Programm, das Robot Karol das Handy finden lässt. Verwenden Sie Programmschleifen und eigene Anweisungen (z.B. für das Umgehen eines Hindernisses). Variieren Sie das Aussehen der Maschinenhalle.

7 Robot Karol stellt Bedingungen

Die Programmierumgebung bringt bereits einige vordefinierte Bedingungen mit. Für bestimmte Problemstellungen ist es aber sinnvoll, zugeschnittene Bedingungen zu definieren:

```
Bedingung
  <Anweisungen>
*Bedingung
```

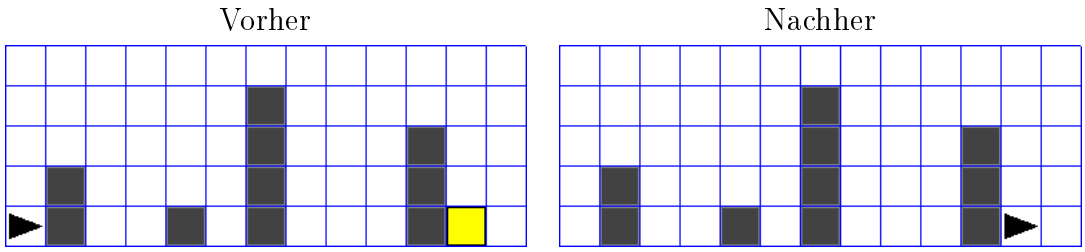
Beispiel: Karol soll feststellen, ob hinter ihm ein Ziegel liegt. Eine solche Bedingung definiert man wie folgt:

```
Bedingung istZiegelHinten
  Schnell
  Linksdrehen
  Linksdrehen
  wenn istZiegel dann
    Wahr              ← Rückgabewert 1
  sonst
    Falsch            ← Rückgabewert 2
  *wenn
  Linksdrehen
  Linksdrehen
*Bedingung
```

Hinweis: Stellen Sie beim Erstellen eigener Bedingungen sicher, dass Robot Karols Ausgangsposition nach dem Prüfen wieder hergestellt wird

7.1 Handysuche unter erschwerten Bedingungen

Problem: Wieder muss Robot Karol ein Handy suchen. Diesmal haben aber alle Maschinen eine unterschiedliche Größe:



Aufgabe 1: Formulieren Sie eine Bedingung, die prüft, ob rechts neben Robot Karol ein Quader liegt.

Aufgabe 2: Modifizieren Sie die Anweisung zum Umgehen einer Maschine so, dass Robot Karol beliebig große Maschinen umgehen kann.

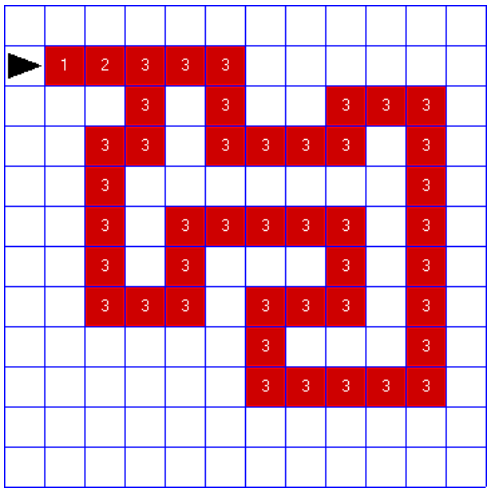
7.2 Karol bildet sich weiter

Ein gut strukturierter Algorithmus lässt sich leicht erweitern bzw. auf ähnliche Probleme anpassen, wie der Vergleich der Algorithmen aus den Abschnitten 6.2 und 7.1 zeigt. Nur die Anweisung für das Umgehen einer Maschine musste verändert werden.

8 Komplexe Übungen

8.1 Robot Karol auf Wacht

Problem: Robot Karol soll einen Burgwall ablaufen, ohne herunterzufallen. Dazu muss er bei jedem Schritt prüfen, in welche Richtung es weiter geht. Zur Sicherheit vor einem eventuellen Überfall soll er hinter sich die Treppe abbauen.



Aufgabe 1: Formulieren Sie die nötigen Bedingungen.

Aufgabe 2: Schreiben Sie eine Anweisung für einen „sicheren Schritt“.

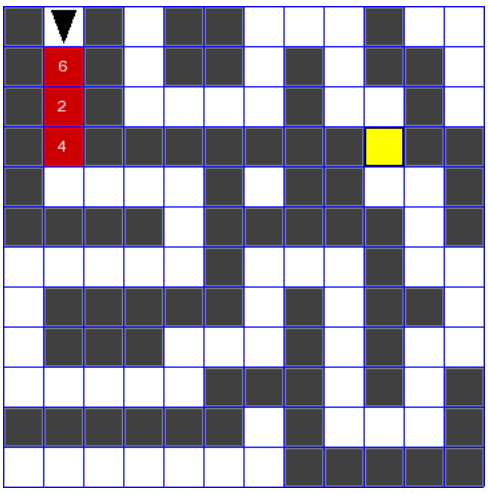
Aufgabe 3: Lassen Sie Robot Karol in einer Endlosschleife den Burgwall ablaufen. Testen Sie das Programm mit verschiedenen Burgwällen.

Hinweis: Endlosschleifen formuliert man einfach wie folgt:

```
Wiederhole immer
  <Anweisungen>
*Wiederhole
```

8.2 Robot Karol bei den Pharaonen

Problem: Robot Karol soll im Labyrinth einer Pyramide einen Goldschatz suchen. Dazu muss er vor jedem Schritt prüfen, wie es weiter geht. Leider muss Karol erst den zugemauerten Eingang der Pyramide öffnen. Er weiß nicht, wie dick und wie hoch die Mauer ist.



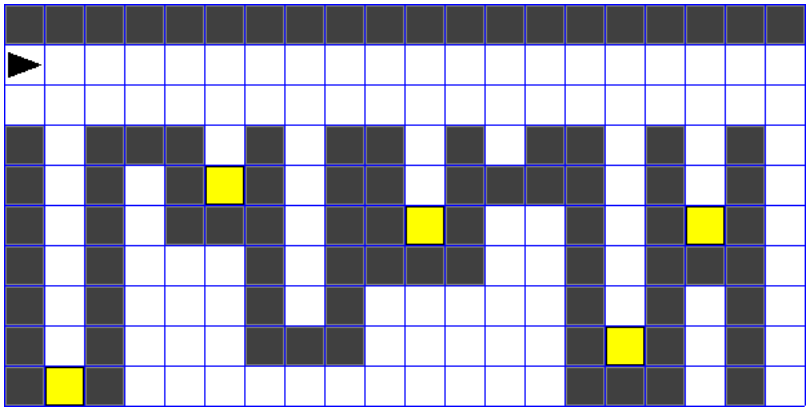
Aufgabe 1: Formulieren Sie die nötigen Bedingungen.

Aufgabe 2: Schreiben Sie eine Anweisung für einen „sicheren Schritt“.

Aufgabe 3: Lassen Sie Robot Karol das Labyrinth nach dem Schatz (Marke) absuchen und diesen aufnehmen. Testen Sie das Programm mit verschiedenen Labyrinthen.

8.3 Puuhh — Robot Karol in der Kanalisation

Problem: Robot Karol befindet sich in einem Abwasserkanal mit zahlreichen Nebenarmen. Einige davon haben sich im Laufe der Zeit mit Schmutz (Marke) zugesetzt. Karol hat die Aufgabe sie zu reinigen. Anschließend soll er sich wieder zu seinem Ausgangspunkt begeben.



Aufgabe 1: Formulieren Sie vorab eine Bedingung, mit der Robot Karol prüfen kann, ob zwei Plätze rechts neben ihm ein Kanal beginnt.

Aufgabe 2: Entwickeln Sie ein gut strukturiertes Programm. Lagern Sie Teilprobleme in Unterprogramme aus.

Aufbage 3: Testen Sie Ihr Programm in verschieden gestalteten und unterschiedlich stark verschmutzten Kanalisationen.

9 Einführung in die Rekursion

Die folgenden Abschnitte basieren auf dem Dokument:

> Rekursion oder: weniger vom Gleichen `http://ada.rg16.asn-wien.ac.at/~python/Bonus/bonus3-rekursion.pdf`

Erste Definition der Rekursion: Um zu erfahren, was Rekursion ist, lesen Sie die *Definition der Rekursion.*

Zweite Definition der Rekursion: Um zu verstehen, was Rekursion ist, müssen zuerst einmal verstehen, was Rekursion ist.

9.1 Salamitaktik iterativ

Stellen Sie sich vor, Sie haben ein Stück Salami, das in Scheiben geschnitten werden soll. Mit Hilfe einer Programmschleife können Sie bestimmt schnell einen Algorithmus für das Problem entwickeln:

```
Anweisung ZerschneideSalami
  Wiederhole solange NichtSalamiAlle
    SchneideEineScheibeAb
  *Wiederhole
*Anweisung
```

Dieses Verfahren nennt man *Iteration* (lat.: Wiederholung). Durch die Wiederholung eines Verfahrens (hier: Scheibe abschneiden) nähert man sich Schritt für Schritt dem angestrebten Ergebnis.

9.2 Salamitaktik rekursiv

Es geht aber auch anders. Stellen Sie sich dazu folgendes vor:

- Sie müssen eine *Salami zerschneiden.* Sie schneiden eine Scheibe ab. Was dann?
- Sie müssen eine (kürzere) *Salami zerschneiden.* Sie schneiden eine Scheibe ab. Was dann?
- Sie müssen eine (noch kürzere) *Salami zerschneiden.* Sie schneiden eine Scheibe ab ...

Nach jeder Scheibe stehen Sie — mit etwas kürzerer Salami — wieder vor demselben Problem. Der Algorithmus könnte also auch lauten:

```
Anweisung ZerschneideSalami
  SchneideEineScheibeAb
  ZerschneideSalami
*Anweisung
```

Haben Sie's bemerkt? Eine Anweisung, die sich selbst aufruft. Eben das ist *Rekursion* (lat.: „Zurücklaufen"). Ganz ohne Programmschleife wird hier eine ganze Salami zerschnitten. Probieren Sie es aus!

9.3 In Ewigkeit und Ewigkeit?

Einen Schönheitsfehler hat der Algorithmus schon noch — Sie haben es sicher bemerkt — er hört nie auf. Bis in alle Ewigkeit würde sich die Anweisung selbst aufrufen, selbst wenn die Salami längst zerschnitten wäre. Der Algorithmus braucht also einen „Rettungsanker" für den Fall, dass die Arbeit getan ist. Das könnte so aussehen:

```
Anweisung ZerschneideSalami
  Wenn SalamiAlle
    Dann              (Der Basisfall: Der Algorithmus stoppt.)
      LegeMesserAb
    Sonst             (Der rekursive Fall: Der Algorithmus läuft weiter.)
      SchneideEineScheibeAb
      ZerschneideSalami
  *Wenn
*Anweisung
```

Ein rekursiver Algorithmus benötigt also immer zwei Fälle: den *Basisfall* zum Anhalten des Algorithmus und den *rekursiven Fall*, in dem sich der Algorithmus wieder selbst aufruft.

Aufgabe 1: Erklären Sie den Begriff *Iteration*.

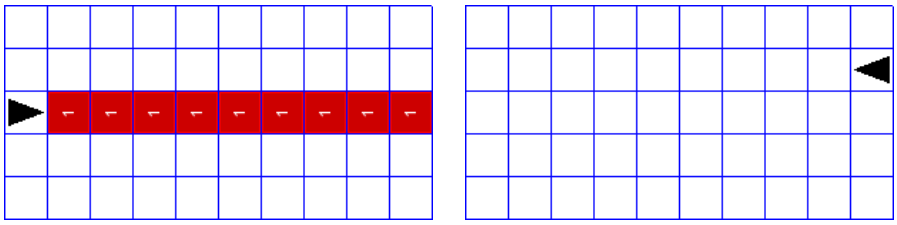
Aufgabe 2: Erklären Sie den Begriff *Rekursion*.

Aufgabe 3: Beschreiben Sie den grundsätzlichen Aufbau eines rekursiven Algorithmus.

10 Robot Karol kehrt zurück — Übungen zur Rekursion

10.1 Robot Karol erntet Kartoffeln

Problem: Robot Karol steht vor einer Ackerfurche mit Kartoffeln. Er soll alle Kartoffeln auflesen, dann in die nächste Furche einschwenken und stehen bleiben:



Vorher Nachher

Aufgabe 1: Schreiben Sie eine Anweisung, in der Robot Karol rekursiv Kartoffeln sammelt.

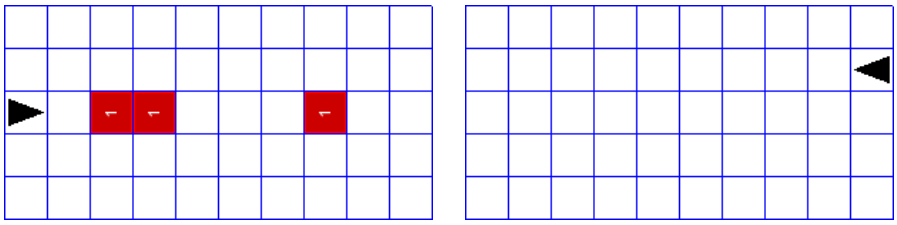
Aufgabe 2: Rufen Sie die Anweisung aus dem Hauptprogramm auf und verfolgen Sie im Editorfenster genau den Programmablauf.

Aufgabe 3: Entwickeln Sie ein eigenes Problem für Robot Karol, das er sowohl mit einem iterativen als auch mit einem rekursiven Algorithmus lösen kann.

Aufgabe 4: Demonstrieren Sie anhand der beiden Lösungen das Wesen der Iteration und das der Rekursion.

10.2 Robot Karol beim Stoppeln

Problem: Aufgrund einer gestörten Zieloptik hat der Ernteroboter einige Kartoffeln in der Furche übersehen. Robot Karol soll nun die Reste auflesen:



Vorher Nachher

Aufgabe: Passen Sie die Anweisung zum Kartoffelernten der geänderten Problematik an.

10.3 Verflixte Rekursion!

Aufgabe 1: Analysieren Sie das nebenstehende Programm und stellen Sie Vermutungen darüber an, wieviele Ziegel nach dem Programm neben Robot Karol liegen. Zeichnen Sie den vermuteten Endzustand.

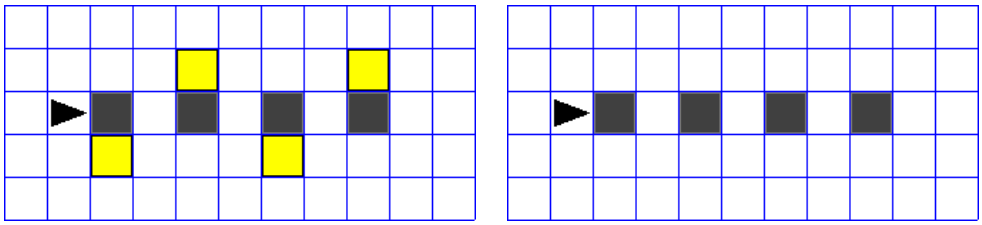
Aufgabe 2: Geben Sie das Programm ein und testen Sie es. Was stellen Sie fest?

Aufgabe 3: Informieren Sie sich über den Verlauf eines rekursiven Algorithmus. Überlegen Sie, wieviele rekursive Aufrufe im Beispiel erfolgen. Erklären Sie jetzt den beobachteten Endzustand des Beispielprogramms.

```
Anweisung Umsetzen
  Wenn IstZiegel
    Dann
      Aufheben
      Umsetzen
    Sonst
      LinksDrehen
      LinksDrehen
  *Wenn
  Hinlegen
*Anweisung
```

10.4 Robot Karol im Wintersport

Problem: Karol läuft bei einem Slalom–Wettbewerb. Um sicherzugehen, dass er kein Hindernis auslässt, muss er unterwegs alle Kontrollmarken entfernen. Nachdem er den Lauf beendet hat, soll er rechts an der Hindernisstrecke entlang an seinen Ausgangspunkt zurückkehren.



Vorher Nachher

Aufgabe 1: Formulieren Sie den Algorithmus zunächst iterativ.

Aufgabe 2: Übersetzen Sie Ihre iterative Lösung in eine rekursive.

Aufgabe 3: Schreiben Sie den Verlauf der rekursiven Aufrufe ihrer Lösung auf.

Aufgabe 4: Realisieren Sie Robot Karols Rückweg, indem Sie die Anzahl der rekursiven Aufrufe nutzen.