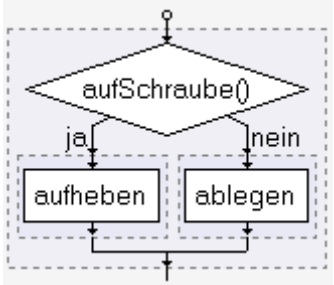
Einlasskontrollen: Bisher hatten Sie Glück. Die Mitarbeiter konnten die Wege recht genau beschreiben. Der Roboter wusste immer, was zu tun ist. So ist das aber nicht immer. Manche Türen gehen von allein auf. Andere müssen mit einem Schlüssel aufgeschlossen werden. Oder es ist gar eine Strombarriere im Weg, die durch einen Schalter deaktiviert werden muss. Gänge knicken unvorhergesehen nach links oder rechts ab. Da muss der Roboter selbstständig Entscheidungen treffen können.

Die Roboter treffen Entscheidungen ...

ZIEL: Alternativen in Handlungen erkennen, als FALLS-DANN-SONST-Entscheidungen formulieren und in Programmiersprache umsetzen können.

In vielen Situationen sind Anweisungen nur unter bestimmten Bedingungen auszuführen. Um z.B. ein Muster zu invertierten, muss ein Roboter dort, wo eine Schraube liegt, diese aufheben und dort, wo keine liegt, eine ablegen. Dazu muss er prüfen, ob eine Schraube da liegt oder nicht und jeweils passend handeln. Die Prüfung erfolgt in einer Prüfbedingung, die nur wahr oder falsch sein kann.

-- normierte Sprache	*-- Programmiersprache*	
FALLS (Schraube ist darunter)	if (istAufGegenstand(„Schraube“))	
DANN	{	
aufheben	aufnehmen();	
*ENDE DANN	}	
SONST	else {	
ablegen	ablegen("Schraube");	
*ENDE SONST	}	

Vorsicht: statt 'if (Prüfbedingung) then {... ' steht nur: 'if (Prüfbedingung) {... ' .

Man nennt solche Entscheidungen in der Programmierung auch **Verzweigungen**.

Aufgaben:

1. ***Invertierer:*** *Der Roboter links unten steht in einer Reihe mit vielen Schrauben. (Zur Sicherheit üben wir erst mal wieder mit Schrauben, wer weiß, was der Roboter anstellt...). Analysiere die beiden Methoden tausche() und tauscheUndVor(): Wie verhält sich der Roboter, falls er auf einer Schraube steht? Wie verhält er sich, falls er auf einem leeren Feld steht? Wozu ist in tauscheUndVor() die Entscheidung if(istVorneFrei())... verwendet worden? Wie unterscheidet sich die Form der Entscheidung in den beiden Methoden?*
2. *Wiederholung der while-Schleife: Jedes Mal, wenn du auf „Reset“ drückst, entsteht eine neue Schraubenreihe. Implementiere eine Methode tauscheBisWand(), die in einer while-Schleife so lange tauscheUndVor aufruft, wie die Wand noch nicht erreicht ist. Versichere dich, dass auch direkt vor der Wand getauscht wird.*
3. ***Fleckenfrei:*** *Ölflecken schaden den Robotern. Sie verlieren die Haftung auf dem Boden. Dadurch verlieren sie Energie. Der Roboter soll um die Ölflecken herum laufen. Mit istVorne("Oelfleck") kann er überprüfen, ob vor ihm ein Ölfleck ist.*
   Implementiere eine Methode umgeheOelfleck(), die den Roboter einen Schritt nach vorne gehen lässt, falls kein Ölfleck vor ihm ist. Ansonsten läuft er um den Ölfleck drumherum. Erweitere diese Methode so, dass der Roboter bis zur Wand läuft.
4. ***Schlüssel aufheben:*** *Schreibe eine Methode sammleSchluessel(), die einen Schlüssel aufnimmt, falls der Roboter auf einem steht. Falls nicht, soll er nichts machen. Teste diese Methode an den beiden Robotern oben links.*
5. ***Sesam öffne dich:*** *Implementiere eine Methode oeffneTuer(), die die beiden Roboter oben links aus ihren Gängen heraus führt. Der Roboter soll einen Schlüssel aufheben, falls er an*

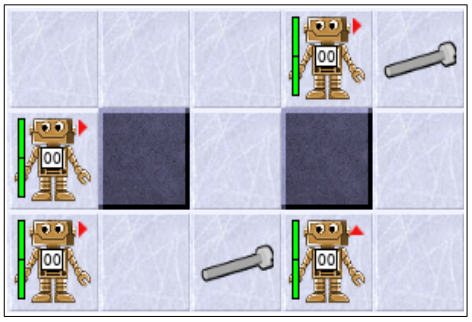
der Startposition liegt. Danach soll er drei Schritte vor gehen, und sich nach links drehen. Falls er vor einem Schloss steht (istVorne("Schloss")), soll er den Schlüssel benutzen (benutze("Schluessel")), andernfalls den Schalter benutzen (benutze("Schalter")). Dann sollte sich die Tür bzw. die Elektrobarriere öffnen. Anschließend soll der Roboter zwei Schritte vor gehen, um hinaus zu treten.

6. *Korrigiere die beiden Schreibfehler! Dieser Quelltext wird so nicht übersetzt, sondern mit einer Fehlermeldung zurückgewiesen. Der zweite Fehler führt zwar nicht zu einer Fehlermeldung, ist daher aber noch schwieriger zu finden.*

```
if (aufSchraube() {
    schraubeAufnehmen();
}
if (istVorneFrei()); {
  einsVor();
}
```

7. *Zeichne für jeden der vier AB4-Roboter ein, wie sie sich bewegen, wenn sie diese Anweisungen ausführen:*

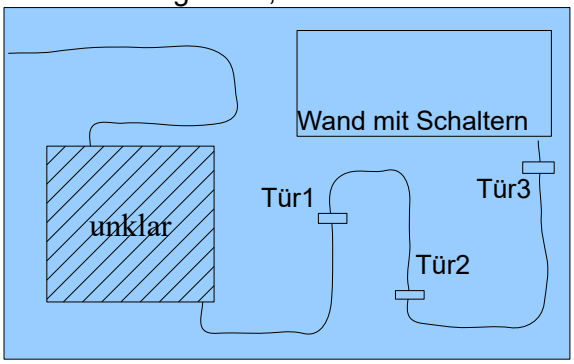
```
if (!istVorneFrei()) {
    dreheLinks();
    einsVor();
    dreheRechts();
}
else {
    einsVor();
}
dreheRechts();
```



8. *a) Nenne zwei Bewegungsbefehle, die ein Roboter ausführen kann und zwei Fragebefehle, die er beantworten kann.*
   b) Gib an, welche Ausdrücke von A bis *E nicht als Prüfbedingung in einer Entscheidungsanweisung **if (...)** benutzt werden kann. Dabei sind x, alter und anzBlaetter Variablen, die für Zahlen stehen.*
   A: (!istVorneFrei()) B: (x>5) C: (alter)
   D: (anzBlaetter<=7) E: (einsVor())
9. ***Labyrinth:** Der Roboter in der Mitte soll sich in einem Gang alleine zurecht finden. Der Gang hat keine Verzweigungen, knickt aber immer wieder nach links oder nach rechts ab. Implementiere eine Methode, die den Roboter bis zur Sackgasse laufen lässt:*
   - Stelle den Roboter zunächst vor eine Wand. Sorge dafür, dass er sich nach links dreht, falls links frei ist. Falls rechts frei ist, nach rechts. Sonst soll er einfach stehen bleiben.
   - Erweitere deine Methode, indem du den Roboter an einer beliebigen Stelle vor der Wand starten lässt und dann die Methode laufeBisWand() benutzt.
   - Lasse diese beiden Schritte solange wiederholen, wie die Sackgasse noch nicht erreicht ist. Um eine Sackgasse zu erkennen, benötigst du komplexere Bedingungen (siehe dazu auch Präsentation 2_Ablauf_von_Schleifen.odp und dort die letzte Folie). Du kannst mehrere Bedingungen mit „und“ (in Java &&) bzw. „oder“ (in Java ||) verbinden. So kannst du beispielsweise mit istWandVorne() && istWandLinks() && istWandRechts() überprüfen, ob dein Roboter in einer Sackgasse steht. Das Gegenteil – also „keine Sackgasse“ – erreicht man, indem man die ganze Bedingung einklammert und ein „nicht“ davor setzt.

Einsatz 4: Gelangen Sie in den Kontrollraum und schalten Sie das Kernkraftwerk ab.

Die Mannschaft hat das Kernkraftwerk fluchtartig verlassen und leider die elementaren Sicherheitsvorkehrungen vernachlässigt. Sie haben vergessen, das Kernkraftwerk herunterzufahren. Dazu müssen im Kontrollraum alle Schalter umgelegt werden. Der Kontrollraum befindet sich irgendwo versteckt am Ende eines langen Ganges, der durch drei Türen gesichert ist. Jede Tür ist durch einen Schalter oder ein Schloss direkt links neben der Tür zu bedienen. Hier haben Sie die nötigen Schlüssel. Im Kontrollraum gibt es an der Wand links neben dem Eingang zahlreiche Schaltern. Wie viele es sind und wo genau sie sich befinden, weiß ich nicht. Schalten Sie alle aus, dann haben Sie Ihren Auftrag erfüllt.



Tipps:

- nutze die Labyrinth-Methode, um zu der Tür 1 zu gelangen. Öffne diese.
- es sind genau drei Türen. Daher kannst du die Befehle einfach 3x kopieren.
- die letzte Tür ist immer genau vor dem Eingang in den Raum. Da reicht ein einsVor()-Befehl.
- für die Schalter brauchst du wieder eine Schleife, da nicht klar ist, wie breit die Wand ist.

Zusammenfassung: Du kannst Verzweigungen in Algorithmen nutzen, im Quelltext erkennen und formulieren. Du kennst die Schreibweise dieser Entscheidungs-Anweisung mit dem Schlüsselwort **if** und der Prüfbedingung im runden Klammerpaar dahinter.

```
if  ( Prüfbedingung ) {
  // DANN-Teil
  // Anweisungen, die ausgeführt werden
  // falls die Prüfbedingung stimmt.
}
else {
  // SONST-Teil
  // Anweisungen, die ausgeführt werden
  // falls die Prüfbedingung NICHT stimmt.
}
```

Manchmal ist im SONST-Fall nichts zu tun. Dann entfällt der Teil ab else. Du kannst die Antworten der Ja/Nein-Abfragen wie istVorneFrei() als Prüfbedingung in einer Entscheidung nutzen, auch in ihrer negierten Form wie bei: if (!istVorneFrei()) {...}. Dies wird gelesen als: „Falls NICHT vorne frei ist...“ oder „Falls vorne frei falsch ist...“ oder „Falls vorne nicht frei ist...“. Die Verneinung NICHT wird durch das Ausrufezeichen **!** geschrieben.

Die Begriffe Verzweigung, Entscheidungsanweisung und auch Alternative werden gleichwertig genutzt.