AB03 - Schleifen: Die Schafe machen etwas immer wieder...

Nicht jede Wiese ist gleich groß, der Stall ist nicht immer gleich weit weg. Trotzdem sollen die Schafe richtig reagieren. Daher müssen sie ihre Umwelt wahrnehmen und darauf reagieren.

> **Ziel:** Wiederholungen in Handlungen erkennen, als SOLANGE-Schleife formulieren und in Programmiersprache umsetzen können. Methoden mit Parametern benutzen können.

(A1) Wiese erkunden

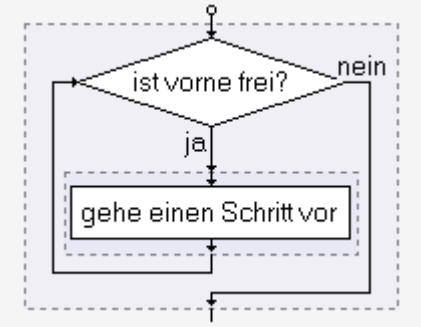
Die Schafe sollen eine Runde auf ihrer Wiese drehen, egal wie groß diese Wiese ist. Dazu gibt es eine Methode `vorBisZaun()`. Teste diese Methode und lies dann im Quelltext vom `AB3_Schaf` nach, wie sie implementiert wurde.

Implementiere dann den Code für die Methode `wieseErkunden()` und setze dabei die Methode `vorBisZaun()` ein, um die Schafe eine Runde entlang des Zauns ihrer Wiese drehen zu lassen. Teste diese Methode an verschiedenen Schafen.

Der Auftrag `vorBisZaun()` muss korrekt ausgeführt werden, egal wie weit der Zaun entfernt ist. Daher muss das Schaf prüfen, wie lange es vorwärts gehen muss. Auf die Anfrage: `istVorneFrei()` liefert es die Antwort `true` bzw. `false`. Damit können wir diese **bedingte Wiederholung** so formulieren:

-- normierte Sprache	-- Programmiersprache

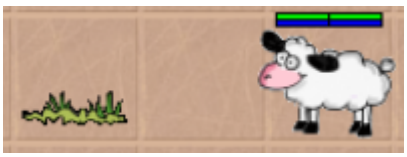
SOLANGE (vorne frei ist)
WDH:
  ein Schritt vor
ENDE WDH
	`while (istVorneFrei())`
	`{`
	`  einsVor();`
	`}`



Genau so steht es auch im Quelltext. Im runden Klammerpaar hinter dem Schlüsselwort while(...) steht die Bedingung, welche die Wiederholungen steuert. Solange diese Ausführungsbedingung gilt, werden alle Anweisungen im Schleifenblock zwischen { und } wiederholt.

Oder anders ausgedrückt: Die Ausführungsbedingung wird überprüft. Ist sie wahr, so wird jede Anweisung in der Blockklammer zwischen { und } ausgeführt. Danach wird wieder überprüft, ob die Ausführungsbedingung immer noch wahr ist. Die Anweisungen innerhalb der Blockklammer werden wieder ausgeführt. Und so weiter und so fort. Erst wenn die Ausführungsbedingung falsch ist, wird die Schleife beendet und die Befehle hinter der schließenden Klammer } ausgeführt. Innerhalb des wiederholten Vorgangs muss sich die Ausführungsbedingung verändern, damit die Wiederholungen schließlich aufhören und nicht endlos laufen.

(A2) Zum Gras

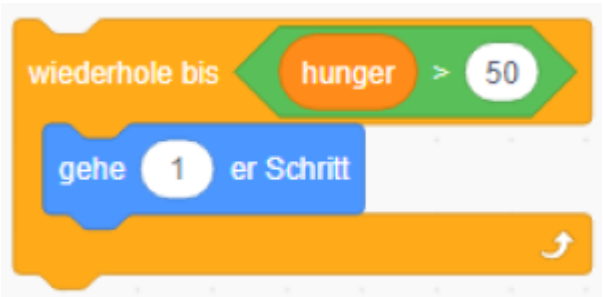


Jedes Schaf muss fressen und dafür muss es einen schönen Grasbüschel finden. Es gibt eine Methode `istAufGegenstand()`, die testet, ob eine Figur auf einem Gegenstand (also z.B. auch Gras) steht. Nutze diese Methode, um die Schafe bis zum nächsten Grasbüschel laufen zu lassen und danach das Gras zu fressen. Implementiere dies in der Methode `vorBisGrasUndFriss()`.
Hinweis: Es muss also so lange laufen, wie es (noch) nicht auf einem Grasbüschel steht. In Java hat `while(!Bedingung)` die Bedeutung "solange die Bedingung nicht erfüllt ist, mache...". Das Ausrufezeichen heißt also "nicht".

Teste die Methode an verschiedenen Schafen. Beschreibe, was passiert, wenn das Schaf schon auf einem Grasbüschel steht oder nirgends vor dem Schaf Gras zu finden ist.

(A3) Scratch

In Scratch gibt es auch eine vergleichbare Schleife. Analysiere, welchen entscheidenden Unterschied es zwischen dem Scratch-Programm und dem Java-Programm gibt.




```
while(getHunger() > 50)
{
   einsVor();
}
```

Wir haben bisher Methoden benutzt, die keine zusätzliche Information benötigen, um ausgeführt werden zu können. Viele Methoden brauchen aber zusätzliche Informationen. Diese werden als **Parameter** bezeichnet. Die Methode `istVorne(String name)` testet beispielsweise, ob die Figur vor einem Feld mit `"name"` steht. Dabei können `name` verschiedene Dinge (z.B. `"Wasser"`, `"Gras"`, `"Zaun"`) sein. Also z.B. `while(istVorne("Zaun"))` ...

(A4) Methode mit Parameter

Teste diese Methode direkt an einem Schaf. Der Kommentar beim Aufruf gibt an, welche Werte der Parameter haben kann. Untersuche, woran man bei einer Methode erkennen kann, ob sie einen Parameter benötigt.

(A5) Zum Wasser

Jedes Schaf muss mal trinken. Aber der Teich oder der Brunnen sind nicht immer gleich weit weg. Es muss also so lange laufen, wie kein Wasser vorne ist. Implementiere die Methode `vorBisWasserUndTrinke()` im Quellcode und teste sie an allen drei Schafen oben. Die Methode funktioniert analog zur Methode `vorBisGrasUndFriss()`. Wichtig: Das Schaf muss sich vollständig **satt trinken**!

Bisher haben wir Methoden für die Bedingungen der Schleifen verwendet, die nur `true` oder `false` zurückgegeben haben. Andere Methoden geben aber Zahlen oder Texte zurück. Möchte man diese für eine Bedingung verwenden, muss man das Ergebnis der Methode mit einem Wert vergleichen. Für die Vergleiche gibt es folgende Zeichen:

gleich	ungleich	größer	größer oder gleich	kleiner	kleiner oder gleich
==	!=	>	>=	<	⇐

z.B. `while(getHunger() < 70)` oder `while(getDurst() != 0)`

Ganz wichtig: Wenn du prüfen willst, ob zwei Werte identisch sind, so musst du mit einem **doppelten Gleichheitszeichen** (==) prüfen!

(A6) Zum Wasser und trinken

Ergänze `vorBisWasserUndTrinke()` aus Aufgabe 5 so, dass das Schaf so lange trinkt, wie es noch Durst hat (der Wert 100 noch nicht erreicht ist).

(A7) Satt fressen

Das Schaf links unten steht auf einer saftigen Wiese. Implementiere eine Methode, die das Schaf die Wiese so lange abfressen lässt, bis eine Sättigung von mindestens 80 erreicht ist (die vier Grasfelder reichen dazu auf jeden Fall). Teste die Methode mehrmals. Drücke dazwischen den "Reset-Knopf". Prüfe, ob der gewünschte Wert auch wirklich jedes Mal erreicht wird.

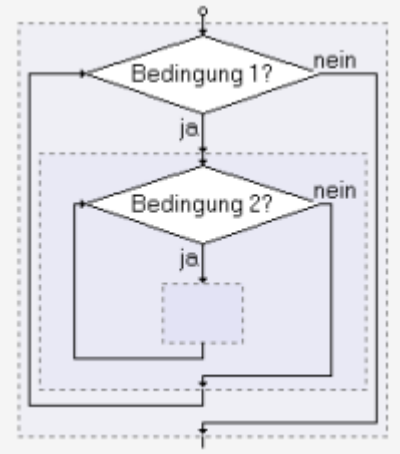
(A8) Faules Schaf

Fred ist faul. Statt über die Wiese zu laufen und nach schönen Grasbüscheln zu suchen, wartet er einfach, bis das Gras genügend gewachsen ist. Implementiere eine Methode, die das Schaf so lange warten lässt, bis das Gras, auf dem es steht, vollständig gewachsen ist und es dann frisst. Nutze zum Warten die Methode `warte()`. Teste deine Methode an den drei Schafen in den Gattern.

Hinweis: Mit der Methode `pruefe(String name)` kannst du das Gras testen. Wenn der Wert 15 erreicht ist, ist es vollständig gewachsen. Für den Parameter `"name"` musst du natürlich wieder den richtigen Wert einsetzen.

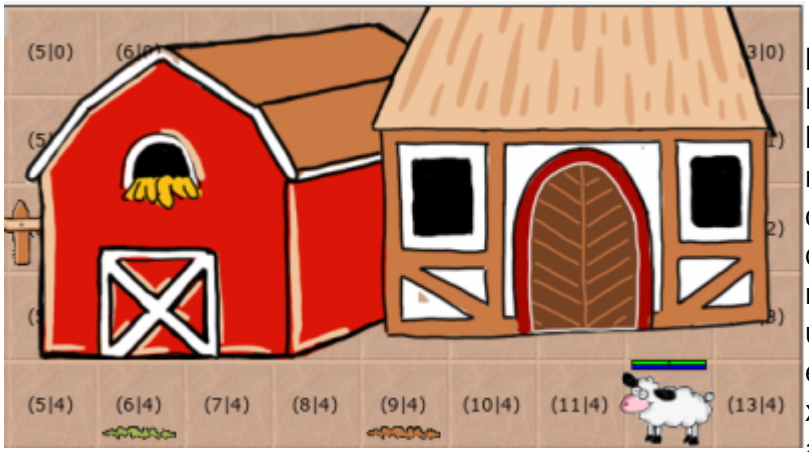
(A9) Faules Schaf 2

Von einem Grasbüschel wird Fred natürlich nicht satt. Verbessere die Methode so, dass die Befehle von Aufgabe 8 solange immer wieder ausgeführt werden, wie ein Sättigungsgrad von 80 noch nicht erreicht wurde.



Hinweis: Man kann Schleifen ineinander schachteln (siehe rechtes Bild).

(A10) Zurück zum Stall



Die Methoden `getX()` und `getY()` liefern die Koordinaten der aktuellen Position des Schafs. `getRotation()` liefert die Richtung, in die es schaut (0 = rechts, 90 = unten, 180 = links, 270 = oben). Implementiere eine Methode, die das Schaf zunächst so lange dreht, bis es nach links schaut, dann nach links laufen und am Ende nach oben laufen lässt, bis es im Stall ist. Der Stall hat die Position x=6 und y=3. Teste deine Methode an allen drei Schafen oben.

Leveltest 3: Karges Land

Überprüfe wie immer deine Implementierungen am Ende des Levels mit der entsprechenden checkup-Methode. Die komplexere Zusatzaufgabe (Leveltest) ist auch dieses Mal eine Bonusaufgabe. Führe diesen Schritt ab sofort am Ende jedes Levels durch.

Aufgabe

„Mäh, mäh, mäh, wo seid ihr alle?" Das arme kleine Schaf hat den Anschluss verloren und steht als einziges noch auf dem Hof des Bauerhofes. Es ist schon ganz ausgehungert, aber auf dem Hof findet man fast nichts zu fressen. Sorge trotzdem dafür, dass es sicher in den Stall kommt.
Irgendwo vor dem Schaf ist ein karges Grasfleckchen, weiter vorne ein Brunnen. Die exakte Position ist jedes Mal etwas anders.
Versuche mit möglichst wenigen Befehlen das Schaf in seinen Stall zu bekommen. Nutze wenn möglich die zuvor programmierten Methoden - es kann aber auch sinnvoll sein, deren Inhalt zu übernehmen und ein wenig anzupassen!

Tipps:

- Es genügt, wenn das Schaf einmal frisst und anschließend einmal trinkt. Beim Sattessen bzw -trinken würde es andernfalls jeweils verdursten oder verhungern…
- Zusammengesetzte Bedingungen kann man mit && (= und) bzw. || (= oder) formulieren.

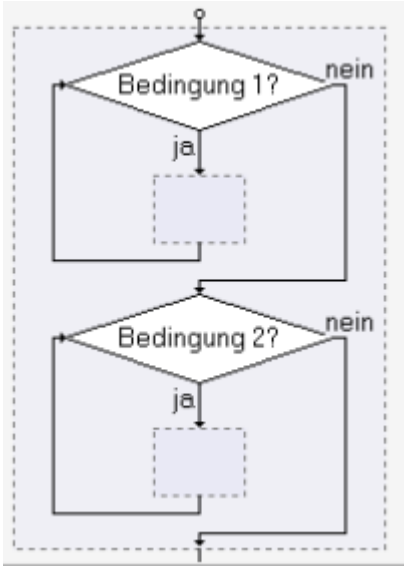
Zusammenfassung: Du kannst Algorithmen formulieren und im Quelltext notieren, die eine oder mehrere Anweisungen so lange wiederholen, wie eine Ausführungsbedingung gilt. Als Bedingung eignet sich alles, was wahr oder falsch sein kann. Wir verwenden oft einen Fragebefehl wie `istVorneFrei()`, dessen Ja/Nein-Antwort die Wiederholung steuert. Aber auch Vergleiche wie „größer“ können verwendet werden.

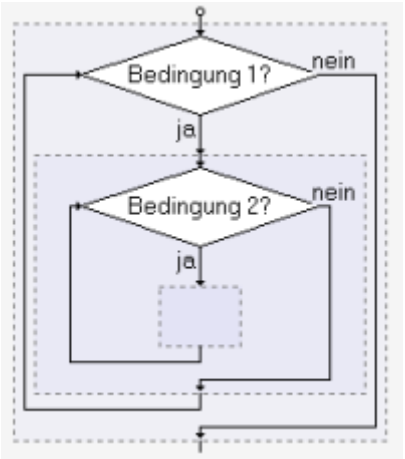
Im Quelltext schreibt man Wiederholungen mit dem Schlüsselwort `while` im **Schleifenkopf** und dahinter eine Ausführungsbedingung in runder Klammer `(…)` sowie einen **Schleifenrumpf** innerhalb des Blockklammerpaares `{…}`.!

```
while ( Ausführungsbedingung ) {
    // zu wiederholende Anweisung;
    // zu wiederholende Anweisung;
    ...
}
```

Die Ausführungsbedingung muss im Laufe der Wiederholungen einmal falsch werden, damit es keine Endlosschleife gibt. Falls es dennoch mal eine Endlosschleife gibt, kann man diese mit dem Knopf  unten rechts unterbrechen.

Mehrere Schleifen können nacheinander ausgeführt werden oder ineinander verschachtelt werden:





<<< Zurück zu Level 2 **AB03** Weiter zu Level 4 >>>