



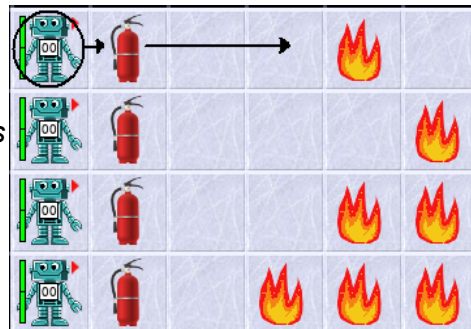
In einem anderen Kraftwerk ist ein Feuer ausgebrochen. Deine ersten Erfolge haben sich rumgesprochen. Daher wird deine RoboRescue-Firma angesprochen, ob sie auch in diesem Fall helfen kann. Natürlich bist du sofort bereit... aber wie löscht man ein Feuer?

Die Roboter machen vieles immer wieder ...

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

Aufgaben:

- Löschen:** Der Roboter **AB3** ganz oben links soll den vor ihm stehenden Feuerlöscher aufnehmen, damit auf das Feld vor dem Feuer (nicht auf das Feuer!!!) gehen und es löschen. Führe dies zunächst von Hand durch. Via Rechtsklick unter „geerbt von Roboter“ findest du auch die Methode "benutze", um den Feuerlöscher zu benutzen.



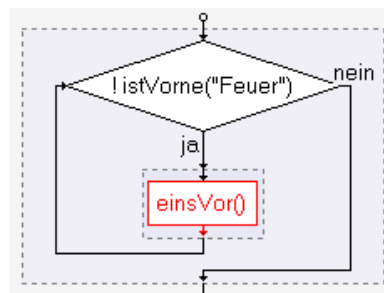
Hinweis: Es gibt Methoden, die eine Zusatzinformation (einen sogenannten Parameter) benötigen, um korrekt arbeiten zu können. Die Methode „benutze(String name)“ ist so eine. Es wird also der Name des Gegenstandes erwartet, der benutzt werden soll. Dieser ist ein String (=Text). Strings müssen in Anführungszeichen gesetzt werden. Um den Feuerlöscher zu benutzen, muss man also `benutze("Feuerloescher");` schreiben.

Vervollständige dann die Methode `loesche()` im Quelltext, die dies alles automatisch für den oberen Roboter machen soll.

- Die weiteren Roboter darunter sollen ihre ersten Feuer in der Reihe auch mit dem Befehl `loesche()` löschen können. Diese Feuer sind aber in einer anderen Entfernung platziert. Überlege dir, welche Zeilen deines Programms überarbeitet werden müssen. Worauf muss der Roboter reagieren können?
- VorBisFeuer:** Gib jedem Roboter den Auftrag: `vorBisFeuer()` und beobachte, was sie tun. Lies danach im Quelltext bei der Methode `vorBisFeuer()` die Anweisungen und versuche sie zu verstehen. Hinweis: das Ausrufezeichen bedeutet „nicht“. Setze diese Methode sinnvoll in der `loesche()`-Methode ein, indem du einige Zeilen deines Quelltextes durch den Befehl `vorBisFeuer()`; ersetzt. Teste nun die neue `loesche()`-Methode an allen vier Löschrobotern.
- Ergänze testweise die Methode `vorBisFeuer` so, dass die Roboter danach noch einen Schritt ins Feuer hinein machen. Was passiert dann? Was passiert, wenn du innerhalb der lila Farbhinterlegung des Quelltextes (also innerhalb der `{}`-Klammern) noch ein zweites `einsVor()`; einfügst? Entferne die Veränderungen wieder.

Der Auftrag `vorBisFeuer()` muss korrekt ausgeführt werden, egal wie weit das Feuer entfernt ist. Daher muss der Roboter prüfen, wie lange er vorwärts gehen muss. Auf die Anfrage: `istVorne("Feuer")` liefert er die Antwort `true` bzw. `false`. Damit können wir diese bedingte Wiederholung so formulieren:

<i>-- normierte Sprache</i>	<i>-- Programmiersprache</i>
SOLANGE (vorne kein Feuer ist)	<code>while (istVorne("Feuer"))</code>
WDH:	<code>{</code>
ein Schritt vor	<code>einsVor();</code>
ENDE WDH	<code>}</code>

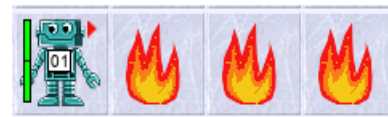


Genau so steht es auch im Quelltext. Im runden Klammerpaar hinter dem Schlüsselwort `while (...)` steht die Bedingung, welche die Wiederholungen steuert. Solange diese Ausführungsbedingung gilt, werden alle Anweisungen im Schleifenblock zwischen `{` und `}` wiederholt.



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.

5. **Feuersbrunst löschen:** Die Feuer des 3. und 4. Roboters bestehen aus mehreren Flammen hintereinander (ohne Lücke dazwischen). Um diese zu löschen, muss man immer wieder löschen, dann einen Schritt gehen, dann wieder löschen usw.. Das wiederholt man so lange, wie nach dem Schritt noch ein Feuer vor dem Roboter ist.



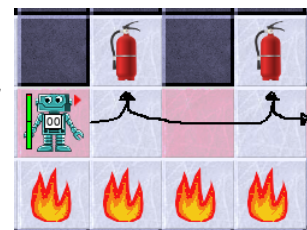
Gehe davon aus, dass der Roboter schon einen Feuerlöscher hat und vor dem ersten Feuer steht. Implementiere eine Methode `loescheReihe()`, die dann alle Feuer löscht. Verwende dazu eine While-Schleife. Teste die Methode, indem du von Hand den Feuerlöscher aufhebst und zum Feuer läufst. Rufe dann die Methode `loescheReihe()` auf.

Ersetze anschließend in deiner `loesche`-Methode das Benutzen des Feuerlöschers durch die neue Methode `loescheReihe()`. Teste die Veränderung.

6. **Reichweite testen:** Mit einem Feuerlöscher kommt man nicht weit. Die sind recht schnell leer. Schreibe eine Methode `testeReichweite()`, bei der der Roboter unten links, den ersten Feuerlöscher einsammelt, zur Feuerspur darunter geht und dort so viel Feuer wie möglich löscht. Verwende eine while-Schleife, um zu überprüfen, ob der Feuerlöscher noch nicht leer ist. Hinweis: Die Bedingung (`getAnzahl("Feuerloescher") >= 1`) testet, ob der Roboter noch mind. 1 Löscher hat. Dies testet automatisch, ob er leer ist, da er einfach aus dem Inventar des Roboters verschwindet, wenn er verbraucht ist.



7. **Feuerlöscher einsammeln:** Für ein großes Feuer braucht man mehrere Feuerlöscher. Vor dem Roboter unten links ist ein Gang mit vielen Feuerlöschern in den Nischen (drücke ggf. unten auf den Reset-Knopf). Er soll nach vorne bis zur Wand laufen und dabei alle Löscher einsammeln. Das soll funktionieren, egal auf welchem orangenen Feld er startet. Er muss aber nur die Löscher einsammeln, die sich nach seiner Startposition befinden.



Implementiere eine Methode `sammleLoescher()`, die zunächst den Roboter bis zur Wand laufen lässt, ohne die Löscher einzusammeln. (Die Methode `istVorneFrei()` testet, ob vor dem Roboter keine Wand ist). Ergänze die Methode dann so, dass er die Löscher einsammelt. Teste verschiedene Startpositionen (irgendwo auf einem roten Feld), indem du neue AB3-Roboter erzeugst und auf einem der roten Startfelder platzierst. (Tipp: über den Geschwindigkeitsregler unten rechts kannst du die Ausführungsgeschwindigkeit anpassen!)

8. **Homerun:** Lasse den Roboter rechts unten in die Sackgasse („Schneckenhaus“) laufen.
- a) Implementiere dazu zunächst die Methode `laufeBisWand()`, die den Roboter bis zur nächsten Wand bewegt.
Tipp: Verwende eine While-Schleife mit `istVorneFrei()` als Bedingung.
- b) Implementiere `laufeBisSackgasse()`, indem du zunächst `laufeBisWand()` aufrufst und dann eine while-Schleife verwendest, um zu testen, ob du noch mal bis zur Wand laufen musst oder schon in der Sackgasse angekommen bist. Dies erkennst du, indem du testet, ob links keine Wand ist (!`istWandLinks()` - das ! steht für „nicht“).

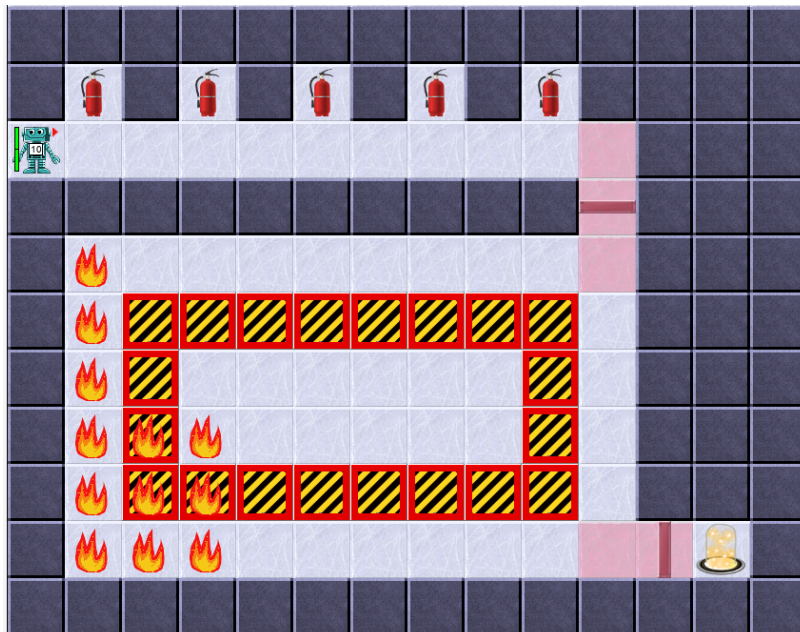
Bildquellen: Die verwendeten Bilder des Roboterszenarios sind alle ohne Bildnachweis verwendbar (selbst gezeichnet, Pixabay Lizenz oder Public Domain). Genaue Nachweise: siehe [bildquellen.html](#).



Einsatz 3: Feuerlöschen im Kernkraftwerk

„Oh mein Gott, in Kernkraftwerk Fessenheim ist ein Feuer ausgebrochen. Sie müssen unbedingt helfen. Wenn das Feuer nicht schnell gelöscht wird, gibt es eine Katastrophe. Unsere Arbeiter können wir nicht mehr rein schicken. Die Gefahr ist einfach zu groß.“

Wenn man reinkommt, gibt es gleich links eine Reihe von Nischen, in der die Feuerlöscher hängen. Sammeln Sie am besten alle ein. Wir wissen nicht, wie groß das Feuer schon ist. Gehen Sie dann in die große Halle. Löschen Sie dort das Feuer. Es ist in der linken unteren Ecke ausgebrochen. Wie weit es sich inzwischen ausgebreitet hat, müssen Sie selbst herausfinden (Hinweis: in jeder Zeile ist direkt vor der Wand mindestens eine Flamme und es gibt keine Lücken im Feuer).



Rechts unten befindet sich ein Schutzraum (mit Portal), in das sich der Roboter am Ende begeben muss. Schaffen Ihre Roboter das?

Ach so, der Weg ist übrigens weit – nutzen Sie die Akkus, die der Roboter für die Mission bei sich hat!“

Tipps:

- Versuche zunächst, nur die oberste Reihe Flammen zu löschen und danach wieder nach rechts zur Wand zu laufen. Drehe den Roboter wieder so, dass er nach unten blickt (Ausgangsstellung).
- Welche der benutzen Befehle müssen wiederholt werden, damit der Roboter die nächste Zeile löscht? Wiederhole diese mit einer While-Schleife, solange das Portal noch nicht erreicht ist (warum erreicht man es automatisch?).


Zusammenfassung: Du kannst Algorithmen formulieren und im Quelltext notieren, die eine oder mehrere Anweisungen solange 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.

Fragebefehle wie `istVorratLeer()` oder `istWandVorne()`, die alle Roboter entweder mit wahr oder mit falsch beantworten können, eignen sich als Ausführungsbedingung.

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.