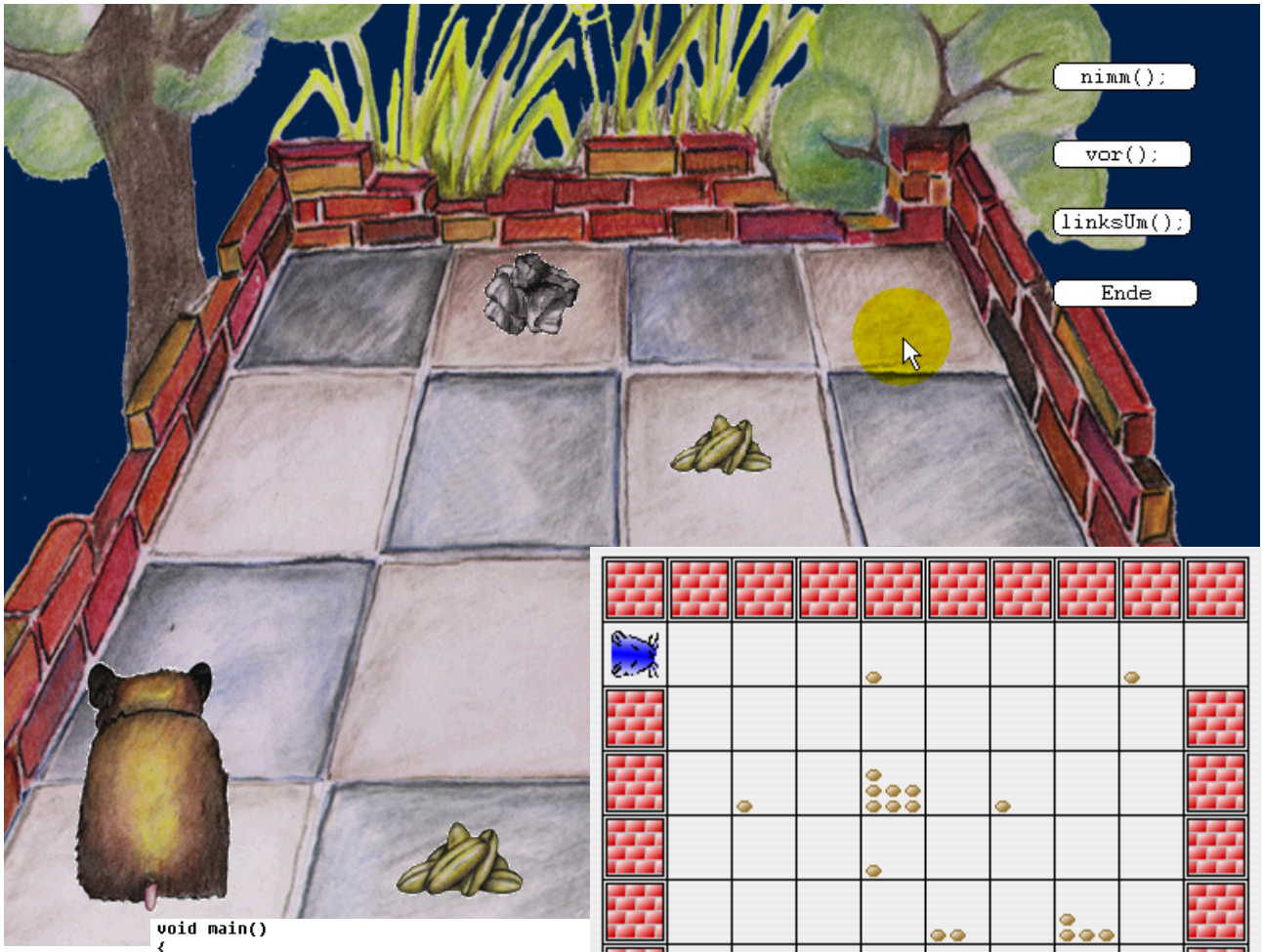
Einführung Java

mit dem

Hamstermodell




```
void main()
{
   for (int i=1; i<=12; i++) vorwaerts();
}

void vorwaerts()
{
   while (vornFrei()) vor();
   rechtsUm();
}

void rechtsUm()
{
   for (int i=1; i <= 3; i++) linksUm();
}
```

1. Wir lernen das Programm kennen

Der Hamstersimulator ist ein JAVA-Programm, das von Dietrich Boles entwickelt wurde, um Programmieranfängern die Grundlagen der Programmierung auf spielerische Weise nahezubringen.

Installation

Der Hamster-Simulator ist ein in Java geschriebenes Programm. Um es ausführen zu können, muss auf deinem Rechner eine Java-Laufzeitumgebung installiert werden.
Dieser Schritt wurde bereits gemacht: Von der Website `http://www.java-hamster-modell.de` den Simulator herunterladen. Du findest ihn im Klassenordner.

Start

Um den Hamstersimulator zu starten, führst du mit der Maus einen Doppelklick auf die Datei *`hamstersimulator.jar`* oder die Datei *`hamstersimulator.bat`* aus.

Das Territorium im Simulator

Nach dem Starten des Simulators siehst du zwei Fenster: den Editor und den Simulator. Im *Simulatorfenster* kann der Hamster herumlaufen, Körner aufheben und wieder weglegen, Hindernissen aus dem Weg gehen (oder auch nicht) etc. Im *Editorfenster* gibst du die Befehle ein, die den Hamster durch den Simulator steuern.

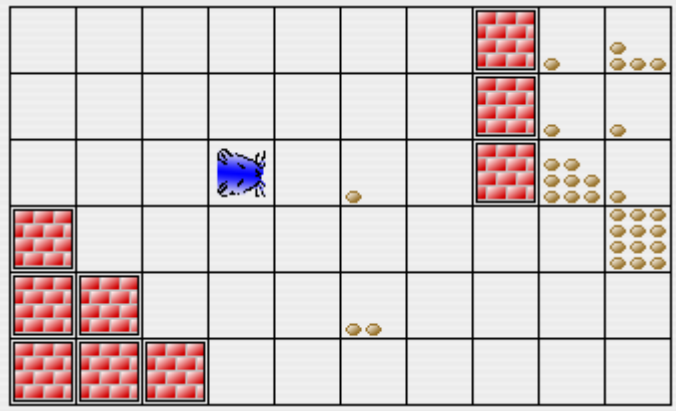
Wie erzeugt man ein *Hamster-Territorium*, in dem dann das eigentliche Programm abläuft?

Betrachte die Buttonleiste im oberen Bereich des Simulationsfensters:

- Ganz links findest du den Button zum **Erzeugen einer neuen Hamster-Welt**. Du werden jetzt aufgefordert, die Breite und Höhe des neuen Territoriums einzugeben. Im obigen Beispiel wurden die Werte 5 und 5 gewählt.
- Mit dem vierten Button von links können wir die **Startposition des Hamsters** ändern. Erst den Button anklicken, dann in das Feld klicken, in den du den Hamster hineinsetzen willst.
- Der fünfte Button von links dient zum **Drehen des Hamsters um 90°**. Somit kannst du die Startrichtung des Tieres ändern.
- Der sechste Button von links dient zum **"Füttern" des Hamsters**. Du kannst ihm Körner ins Maul legen, die er aber nicht sofort runterschluckt, sondern im Maul behält.
- Jetzt sind wir beim siebten Button. Das Ähren-Symbol deutet schon darauf hin: Mit diesem Button kann man **Körner auf die einzelnen Felder legen**. Probiere es einfach aus.
- Mit dem achten Button können wir dem Hamster das Leben ein wenig schwerer machen: Er dient zum **Setzen von Mauern**.
- Der neunte Button schliesslich dient zum **Löschen von Körnern oder Mauern**.

Beispiel 1

Erzeuge nun folgendes *Territorium* und speichere es mit dem dritten Button von links unter **Klasse/.../Hamster/Vorname/Beispiel1** ab:



Die Programmierung im Editor

Klicke nun das Editor-Fenster an und bereite dich sich seelisch darauf vor, dass du gleich programmieren musst!

Mit dem ersten Button von links in der Buttonleiste erzeugen wir ein neues Java-Programm. Wählen "imperatives Programm".

Es erscheint dann ein fast leeres Quelltextfenster, in dem nur ein kurzer Text steht:

```
void main() {

}
```

Übrigens: } erzeugst du mit „AltGr“

Du hast gerade ein neues leeres Programm erzeugt, das den Namen main trägt.
Ergänze das Programm, indem du die folgenden Zeilen eintippst.

```
void main() {
  vor();
  vor();
  nimm();
  linksUm();
  linksUm();
  linksUm();
  vor();
  vor();
  nimm();
}
```

Speichere das Programm jetzt im gleichen Ordner, in dem du vorher das Territorium gespeichert hast.

Kompilieren und Ausführen des Programms

Warum tut sich im Simulator noch nichts? Weil du das Programm noch nicht kompiliert hast!

Drücke dazu im Editorfenster den Button mit dem Pfeil 

Jetzt wird der Programmquelltext, den du gerade gespeichert hast, kompiliert. Das heisst: der Computer übersetzt den Quelltext in eine Sprache, die er versteht und ausführen kann.

Wenn du keinen Tippfehler gemacht hast, dann wird die Übersetzung anschliessend ausgeführt, der Hamster bewegt sich in seinem Territorium gemäss deinen Befehlen.

Fehlermeldungen

Hast du einen Tippfehler eingegeben, so gibt es *Fehlermeldungen* in englischer Sprache, die man nach Möglichkeit von oben nach unten abarbeiten sollte. Es kann nämlich sein, dass alle Fehlermeldungen, die man sieht, eine Folge des ersten Fehlers sind. Wenn du diesen beseitigst, verschwinden beim nächsten Kompilieren auch alle Folgefehler.

Debuggen

Auf das Debuggen oder "Entwanzen" von Programmen verzichten wir hier erst mal.

Funktionen

Stattdessen solltest du jetzt deine erste *Java-Funktion* schreiben (eigentlich ist auch main eine Funktion, aber die zählt nicht). Verändere dazu den Quelltext folgendermassen:

```
void main() {
  vor();
  vor();
  nimm();
  rechtsUm();
  vor();
  vor();
  nimm();
}

void rechtsUm() {
  linksUm();
  linksUm();
  linksUm();
}
```

Du hast die Funktion rechtsUm geschrieben. Eine *Funktion ist eine Art Befehlssammlung*. Ein Bündel von Befehlen also, das gemeinsam ausgeführt wird. Die Funktion rechtsUm führt dreimal den linksUm-Befehl aus, wenn die Funktion aufgerufen wird. Der Quelltext ist jetzt deutlich länger geworden. Wenn du in einem Programm den Hamster aber mehrmals rechts drehen willst, so ist diese Variante übersichtlicher als die vorherige.

Zurücksetzen des Hamsters

Wenn du das gleiche Programm mehrmals hintereinander ausführst, tauchen neue Fehlermeldungen auf, weil der Hamster vor eine Wand läuft oder Körner von einem leeren Feld aufnehmen soll. Du hast hier überhaupt nichts falsch eingegeben, sondern nur vergessen, den Hamster vor der Programmdurchführung **wieder auf seine Startposition zurückzusetzen** bzw. wieder Körner auf die Felder zu legen.

Für dieses Zurücksetzen gibt es einen weiteren Button im Simulationsfenster, es ist der vierte von rechts.

Aufgabe 1

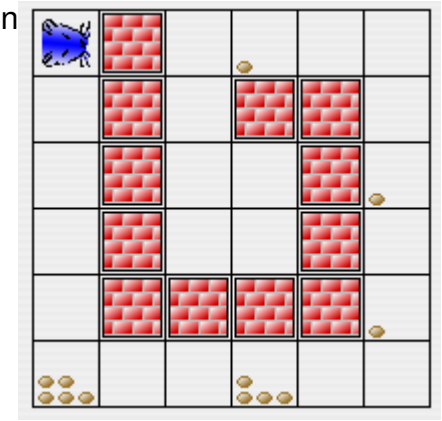
Vervollständige dein erstes Hamsterprogramm so dass der Hamster alle Körner aufnimmt! Du darfst gerne eigene Funktionen erfinden und in dein Programm einbauen. Schaffst du es mit höchstens **48** Zeilen Quelltext?

Speichere dein *Programm* unter **Klasse/.../Hamster/Vorname/Aufgabe1** ab.

- **Tipp01:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

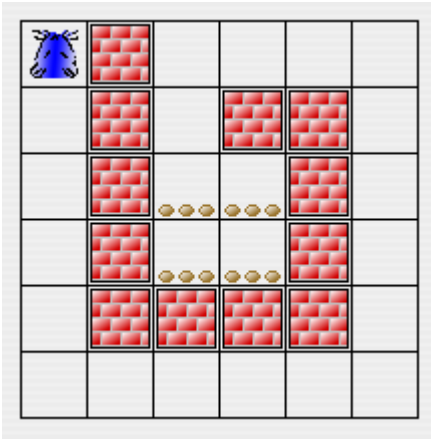
Aufgabe 2

Erstelle ein neues Territorium (Bild 1) und speichere es in deinem Ordner unter *Aufgabe2* ab:



Schreibe ein Programm, welches zur Situation in Bild 2 führt:

- Der Hamster sammelt also alle Körner, die er findet, ein und bringt sie in seinen Bau (4 Felder in der Mitte).
- Mit dem Befehl gib() kann er die Körner ablegen.
- Anschliessend verlässt er seinen Bau wieder und rennt zu seinem Ausgangspunkt, an der er Wache hält.



Hinweise zu den beiden Aufgaben:
Das Programm kann recht lang werden, durch den geschickten Einsatz von selbstgeschriebenen Funktionen kannst du die Zahl der Programmzeilen auf unter **75** reduzieren.

Programmierregeln:
1. Nur ein Befehl pro Zeile.
2. Funktionen werden durch Leerzeilen getrennt, die natürlich mitgezählt werden müssen.
3. Die Schlussklammer } einer Funktion muss sich in einer eigenen Zeile befinden.

- **Tipp02:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

2. Wir programmieren Schleifen

Allgemeines über Schleifen

Schleifen sind Programmierblöcke, die vom Programm immer wieder durchlaufen werden.
In den Programmiersprachen der Pascal-Familie gibt es grundsätzlich drei Schleifentypen:

FOR
WHILE
REPEAT

Unser Java-Hamster beherrscht zwei von diesen drei Schleifentypen: die WHILE-Schleife und die FOR-Schleife.

FOR-Schleifen

Die *FOR-Schleife* ist dann sinnvoll, wenn die Anzahl der Schleifendurchgänge bereits bei dem Erstellen des Programms bekannt ist. Man muss also genau wissen, wie oft die Schleife durchlaufen werden soll.

FOR-Beispiel 1

Fangen wir mit einem ganz einfachen Beispiel an: Unser Hamster kennt ja bekanntlich keinen Befehl zum Rechtsdrehen, daher haben wir eine Funktion rechtsUm() programmiert:

```
void rechtsUm() {
 linksUm();
 linksUm();
 linksUm();
}
```

Wir wollen diese Funktion nun mit Hilfe einer FOR-Schleife schreiben. Sie sieht dann so aus:

```
void rechtsUm() {
  for (int i = 1; i <= 3; i++) linksUm();
}
```

Schauen wir uns den Aufbau einer FOR-Schleife im Detail an:

Laufvariable deklarieren (int i = 1)

Zunächst wird eine so genannte *Laufvariable* deklariert. Wie sie heisst, ist egal, nur muss sie vom Typ int sein! Im ersten Beispiel nennen wir sie i (im zweiten heisst sie x). Wir haben hier ein Beispiel einer *Zählschleife*. Ein Befehl soll hier dreimal wiederholt werden. Daher ist es sinnvoll, die Laufvariable bei 1 zu starten. Wir könnten aber auch bei 12 starten:

```
for (int i = 12; i <= 14; i++) ....
```

Diese FOR-Schleife würde den folgenden Befehl ebenfalls dreimal ausführen.

Testen der Laufvariable (i <= 3)

Der zweite Ausdruck dient zur Überprüfung, ob die so genannte Schleifenbedingung noch erfüllt ist. Eine *Schleifenbedingung* ist ein beliebiger *boolscher Ausdruck*. Er kann TRUE (wahr) oder FALSE (falsch) sein. Nur wenn er TRUE ist, wird der anschliessende Befehl ausgeführt.

Im ersten Beispiel wird also vor jedem Schleifendurchlauf getestet, ob i kleiner oder gleich 3 ist. Vor dem ersten Schleifendurchlauf hat i den Wert 1, also wird die Schleife durchlaufen. Im zweiten Durchlauf ist der Wert von i auf 2 angewachsen. Die Bedingung gilt noch, die Schleife wird ein zweites Mal durchlaufen. Nach dem zweiten Durchlauf hat i den Wert 3. Der Test lautet 3 <= 3. Das ist TRUE, also wird die Schleife ein drittes Mal durchlaufen. Danach hat i den Wert 4, und der Test 4 <= 3 liefert den Wert FALSE, so dass die Schleife jetzt nicht mehr durchlaufen wird.

Veränderung der Laufvariable
Damit eine FOR-Schleife funktioniert, muss die Laufvariable bei jedem Schleifendurchgang um einen bestimmten Betrag hochgesetzt (oder erniedrigt) werden. Bei Zählschleifen erhöht man die Laufvariable normalerweise um 1. Dazu gibt es mehrere Möglichkeiten:

```
i = i + 1
i+=1
i++
```

Das Gegenteil dieses Befehls wäre:

```
x = x – 1
i-=1
x--
```

Das folgende Programm erfüllt keinen besonderen Zweck, es soll nur demonstrieren, wie man eine Laufvariable um beliebige Werte wachsen lassen kann, hier zum Beispiel um den Wert 3. Dazu setzt man den Operator += ein:

```
void main() {
for (int i = 1; i <= 12; i+=3) vor();
}
```

FOR-Beispiel 2
Mit FOR-Schleifen können auch mehrere Befehle nacheinander durchgeführt werden:

Der Hamster soll zum Beispiel einen Schritt machen, ein Korn fressen, wieder einen Schritt machen, wieder ein Korn fressen, usw. Im Hauptprogramm kommt also mehrmals die folgende Befehlssequenz vor:

```
vor(); nimm();
vor(); nimm();
vor(); nimm();
vor(); nimm();
```

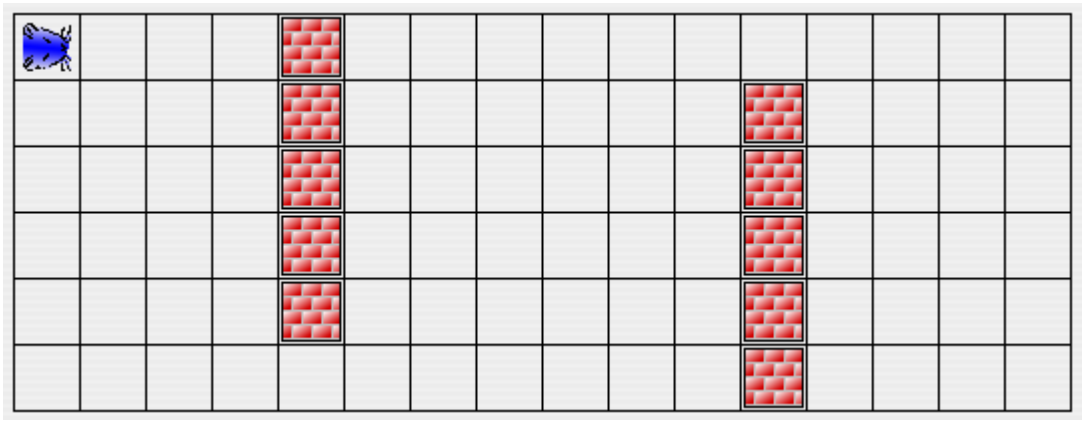
Mit Hilfe einer FOR-Schleife können wir stattdessen schreiben:

```
for (int x = 1; x <= 4; x++)
{
   vor();
   nimm();
}
```

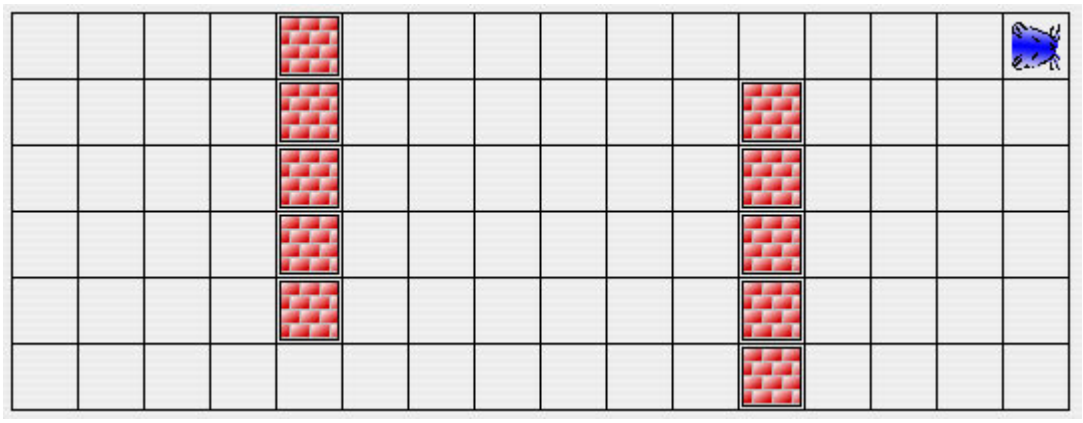
Die Befehle zwischen den geschweiften Klammern werden also 4x ausgeführt.

Aufgabe 3

Erzeuge folgendes Territorium:



Schreibe ein Programm, das mit Hilfe von FOR-Schleifen folgenden Zustand erzeugt:

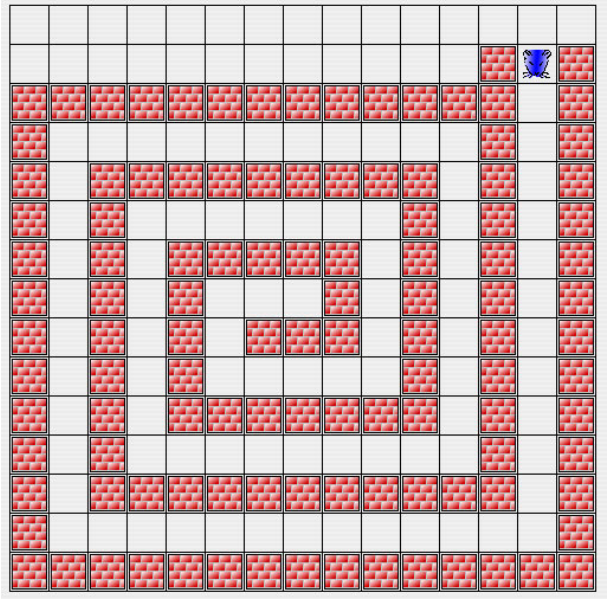


Klappt es mit höchstens **12** Programmzeilen?

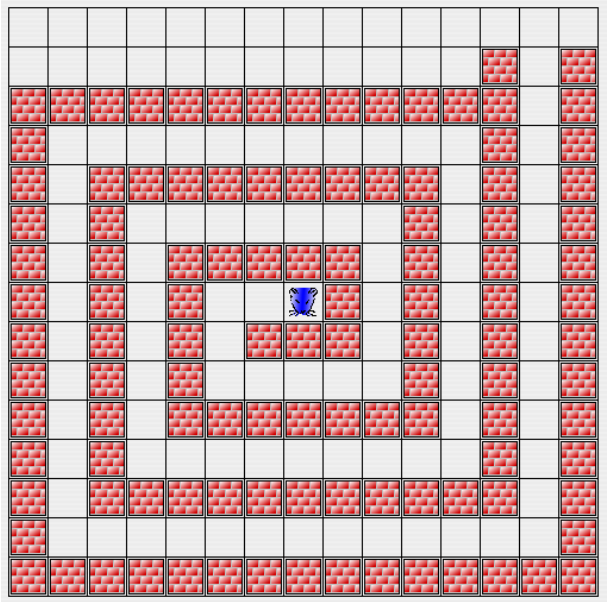
- **Tipp03:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

Aufgabe 4

Erzeuge folgendes Territorium (15 x 15 Felder):



Dein Programm bringt den Hamster schliesslich in folgenden Zustand:



Dein Programm sollte weniger als **30** Zeilen ** haben.

- **Tipp04:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

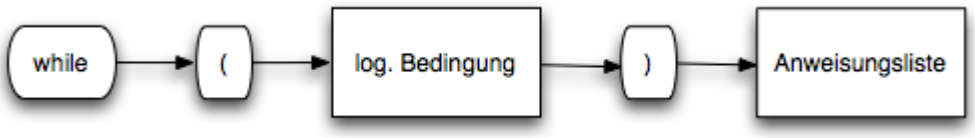
** Durch geschicktes Schachteln von FOR-Schleifen und Setzen von Variabeln schaffst du es auch mit höchstens **9** Zeilen.

WHILE-Schleifen

Bei der *WHILE-Schleife* wird vor dem ersten Schleifendurchgang geprüft, ob die Schleifenbedingung erfüllt ist. Wenn ja, werden die in der Schleife stehenden Befehle ausgeführt. Andernfalls eben nicht. Es kann also sein, dass die WHILE-Schleife übersprungen wird, wenn die Bedingung vor dem ersten Schleifendurchgang nicht erfüllt ist.

Aufbau der WHILE-Schleife

Erst kommt das Schlüsselwort "while", dann in runden Klammern die Bedingung, die für einen Schleifendurchgang erfüllt sein muss, dann eine oder mehrere Anweisungen:



Beispiel 1

Wir wollen den bekannten Befehl zum Rechtsdrehen

```
void rechtsUm() {
 linksUm();
 linksUm();
 linksUm();
}
```

jetzt mit Hilfe einer WHILE-Schleife programmieren:

```
void rechtsUm() {
   int i = 1;
   while (i <= 3)
   {
     linksUm();
     i++;
   }
}
```

Wir haben hier die WHILE-Schleife als *Zählschleife* benutzt. Normalerweise hätte man dazu eine FOR-Schleife verwendet. Aber hier sollte demonstriert werden, dass man auch eine WHILE-Schleife zum Zählen verwenden kann:

Vor dem ersten *Schleifendurchlauf* hat i den Wert 1, die *Schleifenbedingung* i <= 3 ist somit erfüllt und hat den Wert TRUE. Die Schleife wird also durchlaufen. Dabei dreht sich nicht nur der Hamster um 90° nach links, sondern der Wert der Variable i wird vergrössert. Vor dem zweiten Durchlauf der WHILE-Schleife hat i den Wert 2, somit ist die Bedingung wieder erfüllt, und die Schleife wird ein zweites Mal durchlaufen. Nach dem zweiten Durchlauf hat i den Wert 3. Immer noch ist die Bedingung erfüllt, und es findet ein dritter Schleifendurchlauf statt. Danach hat i den Wert 4, so dass die Bedingung 4 <= 3 nicht mehr erfüllt ist. Die Schleife wird beendet.

Beispiel 2
Wie kann man den Hamster in das Innere des Labyrinths bringen, ohne dass er gegen eine der Wände stösst?

In Aufgabe 4 hast du das mit Hilfe von FOR-Schleifen geschafft.

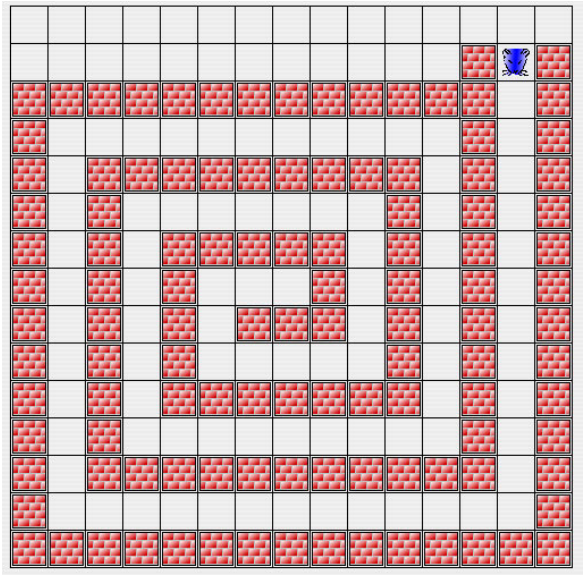
Mit WHILE-Schleifen kann man aber eine noch elegantere Lösung programmieren.

Zunächst setzen wir die Funktion vornFrei(); ein. Die Funktion liefert den Wert TRUE, falls der Hamster nicht direkt vor einer Wand steht. Sonst liefert sie den Wert FALSE.

Betrachte dazu den folgenden Quelltext:

```
void main()
{
   for (int i=1; i<=12; i++) vorwaerts();
}

void vorwaerts()
{
   while (vornFrei()) vor();
   rechtsUm();
}

void rechtsUm()
{
   for (int i=1; i <= 3; i++) linksUm();
}
```

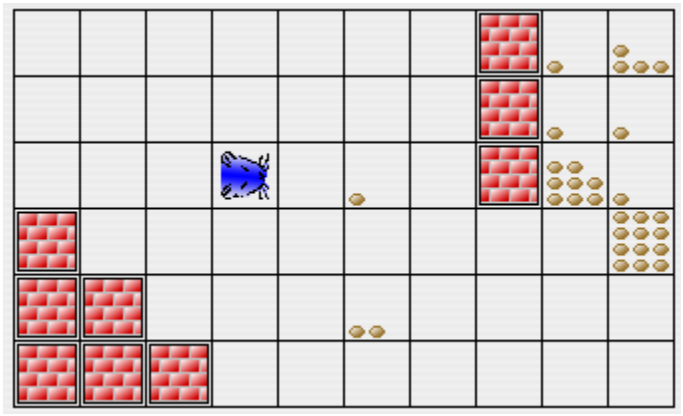
for (int i=1; i<=12; i++) vorwaerts();	Das Hauptprogramm führt die Funktion „vorwaerts()“ 12 mal aus. Die Funktion vorwaerts() enthält zwei Befehle: gehen und rechts drehen
while (vornFrei()) vor();	Der erste Befehl mit While-Schleife: Solange keine Mauer im Weg ist, macht der Hamster 1 Schritt.
rechtsUm();	Der zweite Befehl ist die Funktion rechtsUm()...
for (int i=1; i <= 3; i++) linksUm();	… die als FOR-Schleife aufgebaut ist.



An diesem Beispiel wird gezeigt, dass eine Funktion eine andere Funktion aufrufen darf.

Aufgabe 5

Lade dein Territorium aus der Aufgabe 1, das so aussehen sollte:



Benutze FOR-Schleifen, um den Hamster laufen und sich drehen zu lassen.
Mit WHILE-Schleifen und der Funktion kornDa() lässt du den Hamster alle Körner aufnehmen.
Dein Programm sollte höchstens **35** Zeilen umfassen.

- **Tipp05:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

Befehlsübersicht

Funktion	Beschreibung	Typ
vor()	Der Hamster geht genau 1 Feld weiter	void
linksUm()	Der Hamster dreht sich um 90° nach links	void
nimm()	Der Hamster nimmt ein Korn auf	void
gib()	Der Hamster legt ein Korn ab	void
vornFrei()	Liefert TRUE, falls der Hamster nicht vor einer Wand steht	boolean
kornDa()	Liefert TRUE, falls das Feld, auf dem der Hamster gerade steht, mindestens ein Korn enthält.	boolean
maulLeer()	Liefert TRUE, falls der Hamster kein Korn im Maul hat.	boolean

Typ "void" heisst, dass die Funktion kein Ergebnis zurück liefert. In Pascal oder Delphi würde man eine solche Funktion einfach als "Prozedur" bezeichnen.

Eine Funktion vom Typ "boolean" liefert einen von zwei Werten zurück: TRUE oder FALSE.

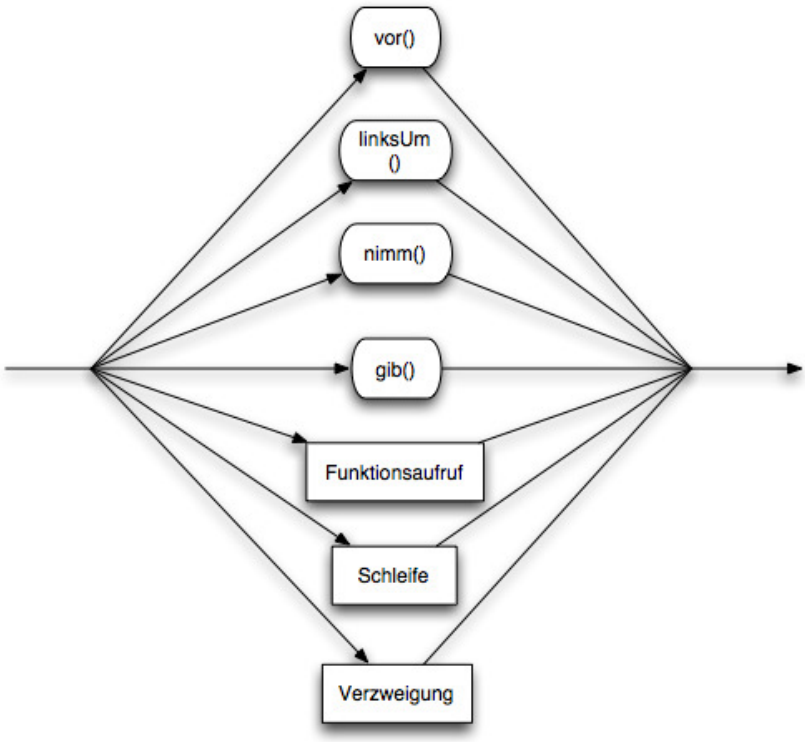
3. Noch mehr Schleifen

Geschachtelte Schleifen

Was ist eine Anweisung?

Dafür gibt es mehrere Möglichkeiten: Eine Anweisung kann eine einfache Hamster-anweisung wie zum Beispiel vor() sein, oder der Aufruf einer selbstgeschriebenen Funktion wie zum Beispiel rechtsUm(), oder eine WHILE-Schleife bzw. eine FOR-Schleife.

Die folgende Abbildung zeigt die verschiedenen Möglichkeiten:



Eine Schleife kann also auch eine „eingeschachtelte“ Anweisung aufrufen, z.B. dies:

```
while (vornFrei())
{
  while (kornDa())
  {
    nimm();
  }
  vor();
}
```

Hier haben wir eine so genannte "geschachtelte Schleife". Die erste WHILE-Schleife enthält eine zweite WHILE-Schleife und eine vor-Anweisung. Was diese doppelte Schleife bewirkt, kannst du durch kurzes Nachdenken selbst herausfinden.

Du hast jedenfalls jetzt ein mächtiges Werkzeug in der Hand, mit dem du bereits recht komplexe Probleme lösen kannst.

Übrigens: die obige doppelte WHILE-Schleife kann man noch etwas vereinfachen. Wenn nur eine einzige Anweisung in der Liste vorkommt, benötigen wir die geschweiften Klammern nicht. Also können wir einfach schreiben:

```
while (vornFrei())
{
  while (kornDa())
    nimm();
  vor();
}
```

Für die äussere WHILE-Schleife (bei geschachtelten Schleifen spricht man von einer "äusseren" und einer "inneren" Schleife) brauchen wir die geschweiften Klammern allerdings, denn diese Schleife führt ja zwei Anweisungen aus: die innere WHILE-Schleife und den vor-Befehl.

Den Quelltext dieser doppelten Schleife bekommen wir noch eine Zeile kürzer, wenn wir folgende Regel einführen:

Bei einer Schleife mit nur einer Anweisung darf diese in der gleichen Zeile stehen wie das Wort WHILE.

```
while (vornFrei())
{
  while (kornDa()) nimm();
  vor();
}
```

Es kann übersichtlicher sein, die geschweifte Klammer in einer neuen Zeile unterzubringen. Selbstverständlich wäre aber auch folgende Schreibweise erlaubt:

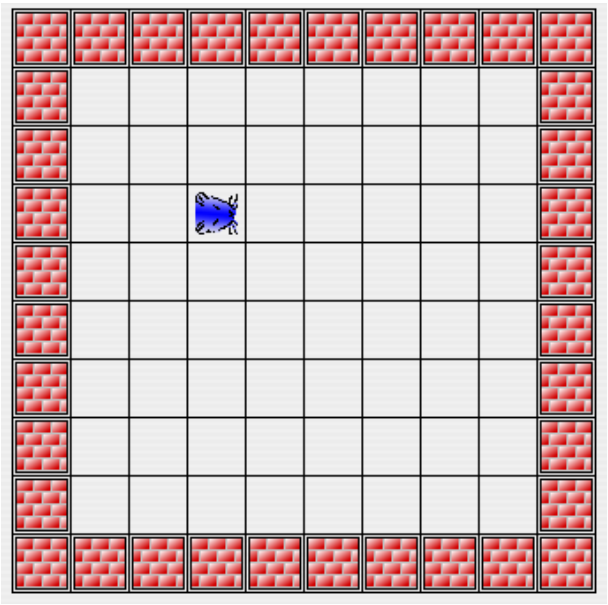
```
while (vornFrei()) {
  while (kornDa()) nimm();
  vor(); }
```

Jetzt beurteile selbst, was übersichtlicher und damit weniger fehleranfällig ist.

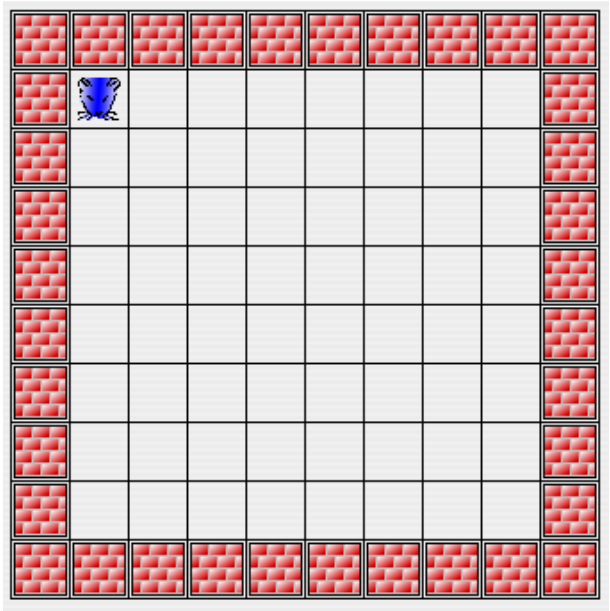
Aufgabe 6

Erstelle ein quadratisches oder rechteckiges Territorium, das von vier Wänden umgeben ist. Im Innern soll es völlig leer sein. Wo die Startposition des Hamsters sich befindet, ist völlig egal. Wichtig ist nur, dass der Hamster nach rechts schaut.

Dein Territorium könnte zum Beispiel so aussehen:



Schreibe ein Programm, das den Hamster immer in folgende Position bringt:



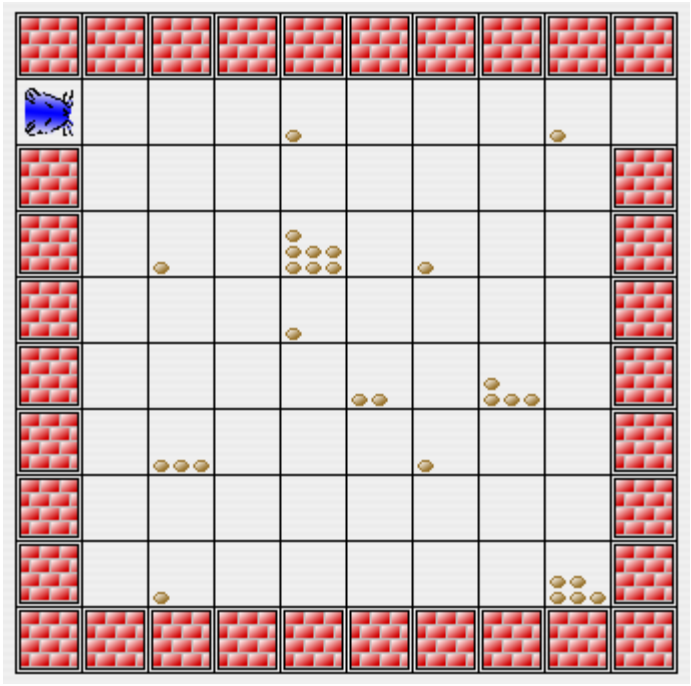
Am Ende des Programms soll sich der Hamster also in der linken oberen Ecke befinden und nach unten schauen. Und zwar unabhängig davon, wie gross das Territorium ist und wo sich der Hamster am Anfang befunden hat.

Versuche ein Programm zu schreiben, das höchstens **10** Zeilen lang ist.

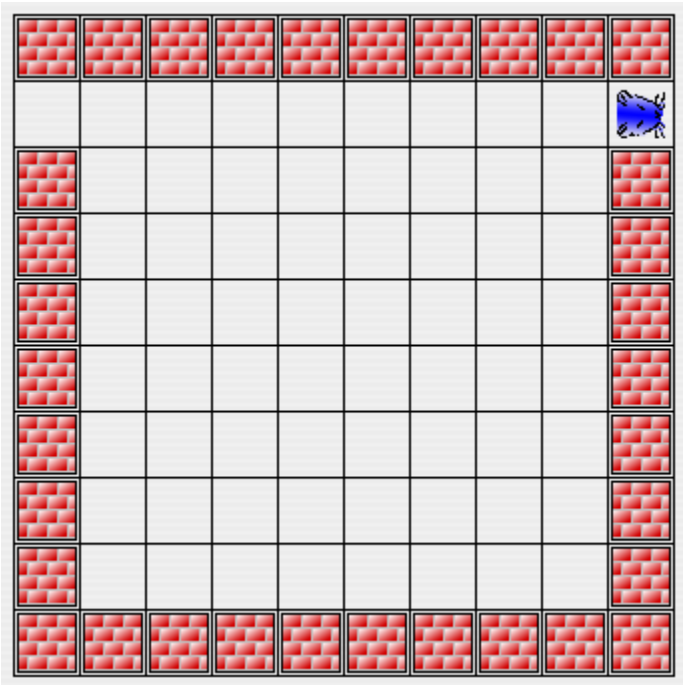
- **Tipp06:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

Aufgabe 7

Erstelle folgendes Territorium (10 x 10):



Wo du die Körner platzierst, ist dabei völlig egal, auch wie viele Körner du platzierst spielt keine Rolle. Der Hamster startet in der angegebenen Position und Richtung. Wenn das Programm fertig ist, soll er alle Körner aufgenommen haben und sich dann im Ausgang befinden und nach rechts schauen:



Teste das Programm auch für mindestens eine andere Körnerbelegung!

Tipp07: http://infopfaffnau.jimdo.com/zusatz/programmieren/

Aufgabe 8

Verändere dein Programm aus Aufgabe 7 so, dass beliebig grosse Territorien abgeerntet werden, die nicht unbedingt quadratisch sein müssen!

Schreibe also eine Funktion, in der der Hamster eine senkrechte Reihe aberntet und dann in derselben Reihe wieder zurückläuft.

- **Tipp08:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

Befehlsübersicht

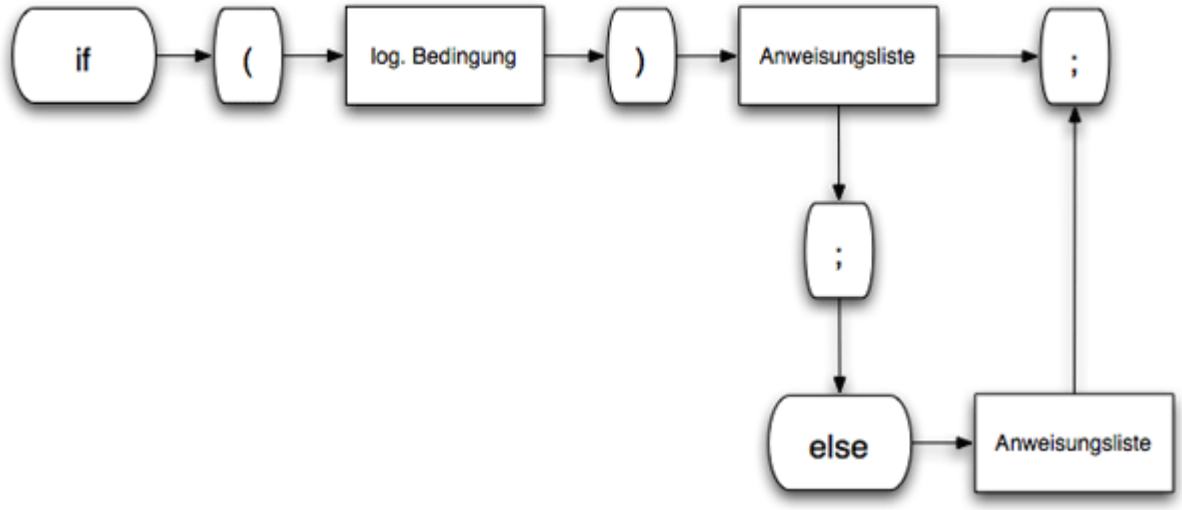
Funktion	Beschreibung	Typ
vor()	Der Hamster geht genau 1 Feld weiter	void
linksUm()	Der Hamster dreht sich um 90° nach links	void
nimm()	Der Hamster nimmt ein Korn auf	void
gib()	Der Hamster legt ein Korn ab	void
vornFrei()	Liefert TRUE, falls der Hamster nicht vor einer Wand steht	boolean
kornDa()	Liefert TRUE, falls das Feld, auf dem der Hamster gerade steht, mindestens ein Korn enthält.	boolean
maulLeer()	Liefert TRUE, falls der Hamster kein Korn im Maul hat.	boolean

4. Verzweigungen

Eine Verzweigungen ist eine *bedingte* Anweisung, die *nur dann* ausgeführt wird, *wenn* eine Bedingung erfüllt ist.

In Java werden solche Verzweigungen mit Hilfe von IF-ELSE-Anweisungen gelöst.

IF-ELSE-Anweisung (Wenn – Dann – Sonst)



Das obige Diagramm zeigt, wie eine IF-ELSE-Anweisung aufgebaut sein muss.

Unter einer Anweisungsliste verstehen wir immer:

1. entweder eine einzelne Anweisung
2. oder eine in geschweiften Klammern stehende Liste von Anweisungen, die durch je ein **;** getrennt sind (siehe Kapitel 2).

Beispiele

1) Ein sehr einfaches Beispiel. Der Hamster geht vorwärts, wenn vor ihm ein Feld frei ist:

```
if (vorneFrei()) vor();
```

2) Diesmal werden zwei Anweisungen ausgeführt, wenn die Bedingung erfüllt ist. Die beiden Anweisungen stehen deshalb in geschweiften Klammern:

```
if (vorneFrei()){
vor():
linksUm();
}
```

In beiden Beispielen wird zuerst überprüft, ob vorne frei ist.
- Falls ja, wird etwas ausgeführt.
- Falls nein, wird nichts gemacht (keine ELSE-Anweisung).

3) Hier wird - falls vorne nicht frei ist – eine Sonst-Anweisung (ELSE) programmiert:

```
if (vornFrei())
                    vor();
else
                    linksUm();
```

Wenn vorne frei ist, läuft die Maus; *sonst* dreht sie sich nach links.

4) Im IF-Zweig steht jetzt nur eine Anweisung, daher sind keine geschweiften Klammern nötig. Im ELSE-Zweig stehen zwei Anweisungen, darum sind Klammern erforderlich.

```
if (vorneFrei()) vor();
else {
  linksUm();
   if (vorneFrei()) vor();
}
```

Und noch etwas ist neu:

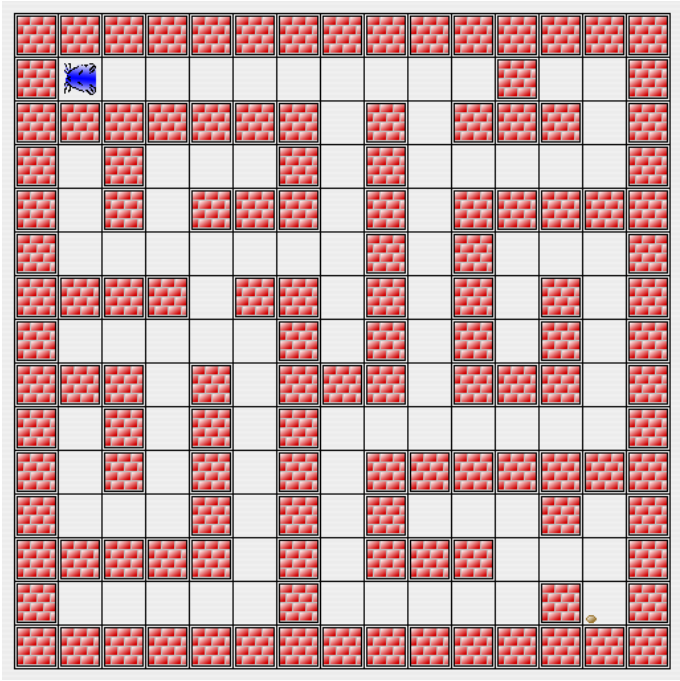
Im ELSE-Zweig ist die zweite Anweisung wiederum eine IF-Anweisung.

Wir dürfen IF- bzw. IF-ELSE-Anweisungen also schachteln. Etwas Ähnliches hatten wir ja bereits bei FOR-Schleifen kennen gelernt.

Mit diesem neuen Baustein solltest du jetzt in der Lage sein, auch die kompliziertesten Aufgaben zu lösen. Fangen wir aber erstmal (scheinbar) harmlos an.

Aufgabe 9

Erstelle ein Territorium mit einem Labyrinth wie zum Beispiel das folgende:



Wichtig: die Wege müssen genau 1 Feld breit sein. Am besten füllst du zunächst das ganze Territorium mit Wänden und erzeugst dann mit dem Löschen-Button einen Weg.

Irgendwo in dem Labyrinth befindet sich ein einziges Korn. Das soll der Hamster finden und auf dem entsprechenden Feld soll er stehen bleiben.

Dein Programm soll jetzt für alle denkbaren Labyrinthe funktionieren! Und es soll möglichst kurz sein. Ideal wären 20 Zeilen Quelltext oder weniger. Das ist aber nur sehr schwer zu schaffen.

Überlege mal, wie man aus einem Labyrinth garantiert herausfindet…

Befehlsübersicht

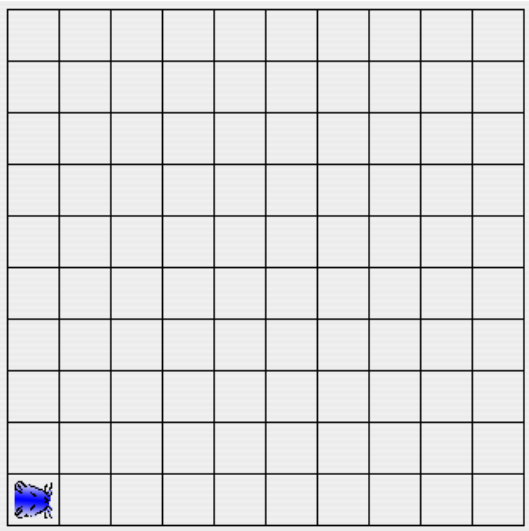
Funktion	**Beschreibung**	**Typ**
vor()	Der Hamster geht genau 1 Feld weiter	void
linksUm()	Der Hamster dreht sich um 90° nach links	void
nimm()	Der Hamster nimmt ein Korn auf	void
gib()	Der Hamster legt ein Korn ab	void
vornFrei()	Liefert TRUE, falls der Hamster nicht vor einer Wand steht	boolean
kornDa()	Liefert TRUE, falls das Feld, auf dem der Hamster gerade steht, mindestestens ein Korn enthält.	boolean
maulLeer()	Liefert TRUE, falls der Hamster kein Korn im Maul hat.	boolean

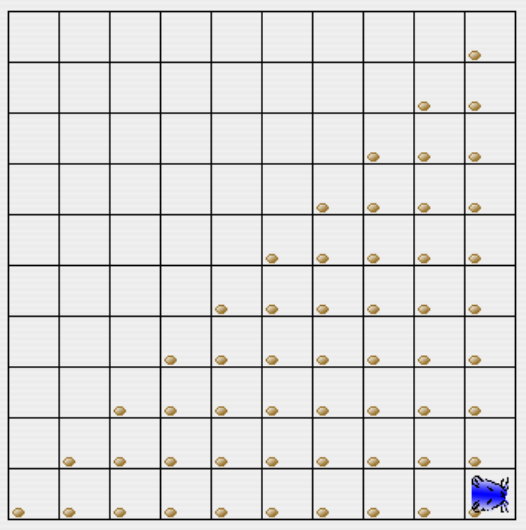
- **Tipp01:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

5. Unser Hamster lernt rechnen

Unser Hamster soll jetzt in die Schule gehen und lernen zu zählen.

Hier ist die Ausgangssituation. Der Hamster sitzt unten links in einem 10x10-Territorium. Er soll nun zeigen, dass er von 1 bis 10 zählen kann, indem er folgendes Muster legt:





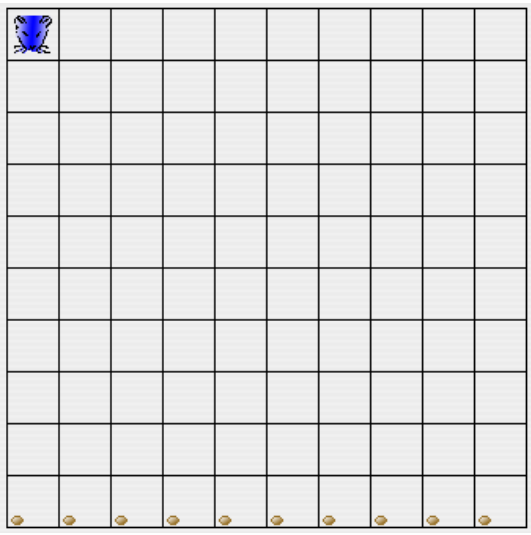
Hier nun der zugehörige Quelltext:

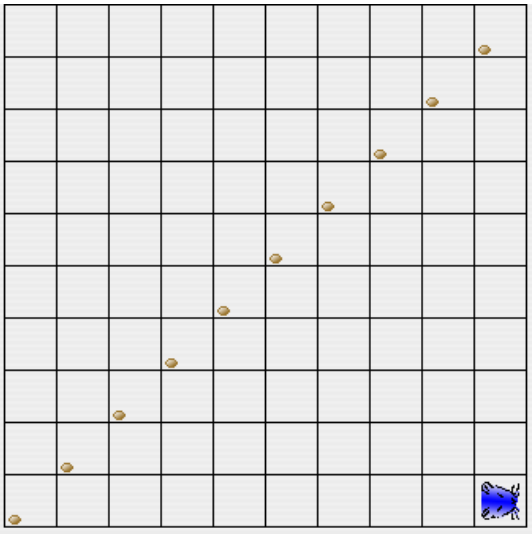
```
void main()
{
  for (int i=1; i<=10; i++)
  {
    legeReihe(i);
    if (i<10) vor();
  }
}

void legeReihe(int n)
{
  linksUm();
  for (int i=1; i<=n; i++)
  {
    gib();
    if (i<n) vor();
  }
  linksUm();
  linksUm();
  for (int i=1; i<n; i++) vor();
  linksUm();
}
```

Betrachten wir zunächst die Hauptfunktion main(). Der zentrale Gedanke: In einer FOR-Schleife bearbeitet der Hamster jede der 10 Spalten des quadratischen Territoriums. Die Anweisungen sind schliesslich in der Funktion legeReihe() mit der neu definierten Variablen n vorgegeben: Wenn er zum Beispiel fünf Körner legen soll, geht er an das untere Ende der Spalte, dreht sich nach oben, legt ein Korn ab, geht ein Feld vor, legt wieder ein Korn ab und so weiter, bis er seine fünf Körner los geworden ist. Dann dreht sich der Hamster um 180° und läuft die fünf Felder zurück. Anschliessend dreht er sich noch einmal, damit er wieder die gleiche Position hat wie zuvor.

Aufgabe 10

Schreibe das Programm um, so dass der Hamster die Körner aus der Ausgangslage wie folgt verteilt:





Der Hamster soll auch auf beliebigen anderen quadratischen Territorien eine solche Diagonale erzeugen!

Wichtige Teilaufgabe:
Schreibe die Funktion legeReihe(int n) so um, dass der Hamster immer auf dem untersten Feld der Reihe startet, dann n-1 Schritte vorwärts geht, ein Korn legt und danach auf das unterste Feld zurückkehrt.
Wenn du die Funktion derart umgeschrieben hast, so kannst du beliebige mathematische Funktionen mit dem Hamster zeichnen lassen.

- **Tipp10:** http://infopfaffnau.jimdo.com/zusatz/programmieren/

Aufgabe 11

Verändere Programm aus Aufgabe 10 so, dass der Hamster die Funktion $y = x^2$ zeichnet. Verändere dazu das Territorium so, dass es für x = 5 gross genug ist (5 breit und 25 hoch).

Befehlsübersicht

Funktion	**Beschreibung**	**Typ**
vor()	Der Hamster geht genau 1 Feld weiter	void
linksUm()	Der Hamster dreht sich um 90° nach links	void
nimm()	Der Hamster nimmt ein Korn auf	void
gib()	Der Hamster legt ein Korn ab	void
vornFrei()	Liefert TRUE, falls der Hamster nicht vor einer Wand steht	boolean
kornDa()	Liefert TRUE, falls das Feld, auf dem der Hamster gerade steht, mindestestens ein Korn enthält.	boolean
maulLeer()	Liefert TRUE, falls der Hamster kein Korn im Maul hat.	boolean

- **Tipp11:** http://infopfaffnau.jimdo.com/zusatz/programmieren/