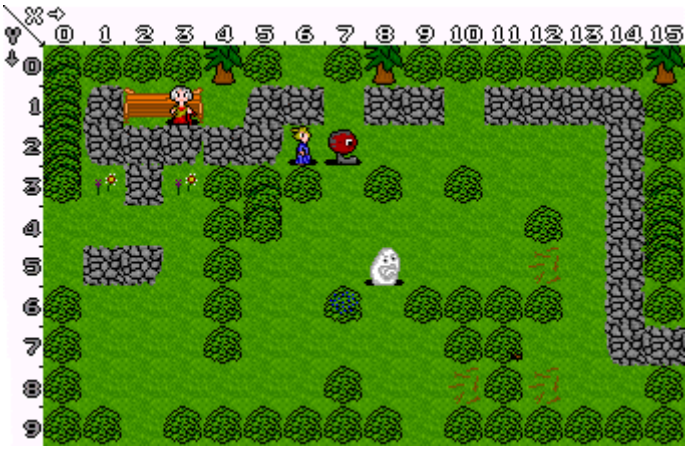
Schulmathematik hat einen Sinn! Hurraaaaaa!

Aber leider nur für Freaks. Hier anhand des OHRRPGCE einige Praxisbeispiele.

NPC = Nicht-Personengesteuerter Charakter

- Koordinatensystem:

Jedes Feld auf der schachbrettartigen Karte ist einer Koordinate (X|Y) zugeordnet. Gut ist zu wissen, dass die X-Achse horizontal verläuft und die Y-Achse vertikal. Oder hast du etwa in der Schule nicht aufgepasst? Übrigens darf der Koordinatenursprung auch mal oben links liegen, so wie die alten Computer eben anfangen, zu zählen.



Der Skriptbefehl...

```
delete NPC (NPC at spot (8,5))
```

...würde die weiße Figur, die sich gerade an Stelle (8|5) befindet, verschwinden lassen.

- Einheiten:

Hier ein Stück Programmcode, das dem Spiel mitteilt, dass ein Labor in 25 Minuten in die Luft fliegen soll.

```
boom := days of play * 1440 + hours of play * 60 + minutes of play + 25
```

und dazu an einer anderen Stelle

```
if (days of play * 1440 + hours of play * 60 + minutes of play > boom)
then ( ... )
```

1440? 60? Komische, willkürlich gewählte Zahlen? Natürlich sind es Unrechnungsfaktoren, um auf Minuten zu kommen.

- Größer-/Kleiner-Zeichen

Das Krokodilmaul zeigt immer auf das, was größer ist. 40 < 60. Hier ein Beispiel für eine der zahlreichen Gabeln, die sich im Programmskript auftun, je nach dem, ob eine Zahl größer oder kleiner als eine Vorgabe ist.

```
if (current wealth > 99)
then (show text box (2788))
else (NPC does not face player (1,me))
```

Wenn der eigene Kontostand größer als 99 ist, wird ein Dialog eingeblendet. Ansonsten dreht sich ein NPC weg von der eigenen Spielfigur. Tja, dem ging es wohl ums Geld.

`if (current wealth >= 100)` würde dasselbe tun. Immer schön an die Grenzen denken!

- Betrag:

Kompliziert gesagt misst der Betrag den Abstand einer Zahl zur Zahl 0. Leicht gesagt macht der Betrag ein Vorzeichen-Minus vor einer Zahl einfach weg. Das wird zum Beispiel in der Flugszene über die Weltkarte benutzt, wo der nächstliegende Ort zu der eigenen Vogelfigur ausgewählt werden soll. „Nächster Ort“ wird im Programmskript mit „geringster Abstand“ übersetzt. Dort wird dann gelandet.

43 - 12 = **31**

95 - 12 = **83**

Beim Vergleich zwischen diesen zwei Punkten ist kein Betrag nötig, da der kleinere Abstand eindeutig 31 ist.

33 - 76 = **-43**

97 - 76 = **21**

Natürlich ist der Abstand 21 geringer als -43, nur ist -43 die kleinere Zahl... wie sag ich es meinem Computer?

Die Y-Koordinaten sind in beiden Beispielen vernachlässigbar, da sie fast identisch sind. Jetzt reicht es aber nicht, einfach nach X-Position des Zielortes minus X-Position der Spielfigur zu fragen. Dann würden wir im zweiten Beispiel bei Punkt (33|132) herauskommen, obwohl der eigentlich weiter weg von der Spielfigur liegt.

- Variablen:

„Als wir in der Schule dann mit Buchstaben rechnen sollten, bin ich ausgestiegen.“ Eine nur zu häufig vorkommende Erfahrung zu Variablen im Mathematikunterricht. Beim Programmieren sind sie nicht wegzudenken, diese nie müde werdenden Zahlenspeicher.

```
variable (a)
variable (b)
variable (c)
if ( (900 -- a) * (900 -- b) * (900 -- c) == 0 ) then (...)
```

Hier ist eine Formel mit drei Variablen, die im Spiel Startnummern darstellen. Ist eine der drei Zahlen 900, geht die Rechnung auf, da irgendein Produkt automatisch 0 ergibt, sobald sich eine 0 unter den zu multiplizierenden Zahlen befindet. Es gäbe auch einfachere Möglichkeiten, die obere Formel im Programmcode darzustellen, z.B.

```
if ( x == 900,or,y == 900,or,z == 900 ) then (...)
```

- Funktionen:

Richtig mächtig werden Variablen aber erst mit komplizierten Formeln. Dazu ein Beispiel mit einem schwankenden Aktienkonto.

```
global variable (74,x)
x := 89 * x / 100 + 9
```

Dieses Skript wird im Spiel alle fünf Minuten aufgerufen und ändert nach der Formel 89 * x / 100 + 9 die Kurse. Das ist ein linearer Vorgang, aber da er von sich selbst anhängt (also von seinem eigenen Kurs), werden Wachstumsraten geringer, je höher Variable „Kurs“ ist. Für x = 5 eingesetzt ergibt sich als Ergebnis gerundet 13. Im nächsten Schritt ergibt sich für x = 13 als Ergebnis 20, also ein Plus von 7 im Gegensatz zum vorherigen Kurs. Später jedoch z.B. bei x = 60 ergibt sich lediglich 62 als neues Ergebnis, sodass also das Wachstum sichtbar langsamer wird.

Eine Zufallsvariable verleiht der Situation im Spiel natürlich eine gewisse Würze, damit es nicht so langweilig gleich daherkommt.

- Zufallsexperimente:

Anstelle einen Würfel zu werfen, können wir einfach den Computer zufällige Zahlen bestimmen lassen.

```
if (random (1,6) == 6) then (...)
```

Irgendeine Zahl zwischen 1 und 6 wird bestimmt und ist sie zufällig 6, passiert irgend etwas.

```
if (random (1,100) <= 44) then (...)
```

So ließen sich auch Wahrscheinlichkeiten in Prozent angeben. Hier beträgt sie 44% dafür, dass das Skript weitermacht. Oh, sagte ich etwa „Prozent“?

- Prozentrechnung:

Ein etwas komplexeres Beispiel von einem Rennspiel: In den Variablen 113 bis 132 stehen die Rundenzeiten der 20 Fahrer. Zuerst findet das Skript heraus, welches die schnellste Zeit war. „Read global“ ist erneut der Befehl, um die Variablen auszulesen.

```
variable (x)
variable (best)
best := 113 # Beginn bei 113, sonst wäre ´best´ 0
for (x,113,132,1) do (
  if (read global (x) < read global (best))
  then (best := x))
```

Und jetzt kommt die Prozentrechnung ins Spiel! Jeder, der nämlich langsamer war als 155% der Bestzeit, der hat sich nicht für das Rennen qualifiziert.

```
hürde := read global (best) * 155 / 100

if (read global (best) > hürde) then ( ... )
```

Auch schön zu sehen ist eine andere Schreibweise für 155%, und zwar 155/100, anders ausgedrückt 1,55. In der Rechnung bedeutet das, dass bei einer Bestzeit von zum Beispiel 50s die 155%-Hürde bei 77,5s liegen würde.

- Terme:

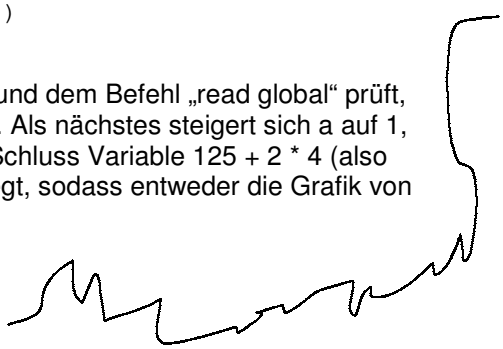
Ja, wie denkt man sich denn diese Formeln bzw. Terme eigentlich aus? Es nachmachen von Anderen.

Beispielsweise ergibt sich im Spiel folgende Situation, nach der das Programm handeln soll:
Wenn Variable 125 = 62 ist, dann ändere Bild von NPC 2
Wenn Variable 129 = 62 ist, dann ändere Bild von NPC 4
Wenn Variable 133 = 62 ist, dann ändere Bild von NPC 6
Und diese Liste könnte noch weit länger sein... ist sie hier aber glücklicherweise nicht.

Benötigt wird also eine Reihe für 125, 129, 133 und eine Reihe für 2, 4, 6. Gelöst wurde es wie folgt.

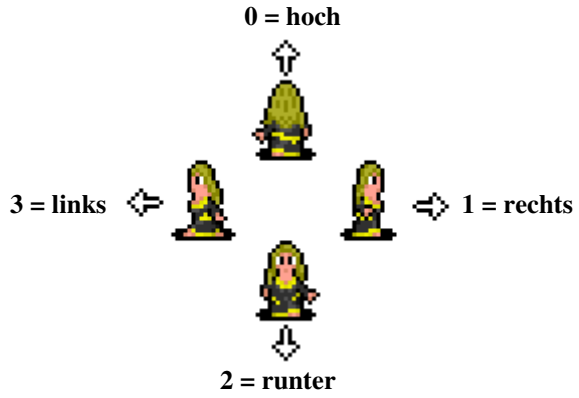
```
variable (a)
for (a,0,2,1) do (
  if (read global (125 + a * 4) == 62)
  then (alter NPC (2 + 2 * a,NPCstat:picture,194))
)
```

Eine Wiederholungsschleife, die drei Mal abläuft und erst mit a=0 und dem Befehl „read global“ prüft, ob Variable 125 + 0 * 4 (also 125) den gewünschten Wert (62) hat. Als nächstes steigert sich a auf 1, sodass nun Variable 125 + 1 * 4 (also 129) geprüft wird und zum Schluss Variable 125 + 2 * 4 (also 133). Die jeweilige Handlung wird mit der Formel 2 + 2 * a festgelegt, sodass entweder die Grafik von NPC 2, 4 oder 6 geändert wird.



- Restrechnen:

Oder Modulo-Rechnen genannt. Bei einer Division ist es also der Rest, mit dem als Ergebnis weitergerechnet werden kann. Z.B. ist 9:4 = 2 Rest 1. Somit ist 9 mod 4 = 1. Diese „mod"-Rechenoperation haben wir in einem Skript verwendet, um einen angesprochenen NPC irgendwo hin schauen zu lassen, aber hauptsache nicht in Richtung der eigenen Spielfigur. Dazu kurz eine OHRRPGCE-interne Sache zur Bezeichnung der Richtungen, die als Zahl dargestellt werden können:




```
script,NPC does not face hero,figur,held,begin
variable (x)
x := (hero direction (held) + random (1,3) + 2),mod,4
set NPC direction (figur,x)
end
```

Das Gegenteil muss also vermieden werden. Ist die Richtung der Spielfigur links (3), darf der NPC nicht nach rechts (1) schauen.

Vereinfacht gesagt geht es um diese Rechnung:

Richtung der Spielfigur + Zufall (1, 2 oder 3) + 2 mod 4

Richtung der Spielfigur	Zufallsvariable (1 bis 3)	+ 2 (kehrt die Richtung um)	Zwischen-ergebnis	mod 4 gerechnet, damit Richtung NPC
0 (hoch)	+1	+2	3	3 mod 4 = 3 (links)
0 (hoch)	+2	+2	4	4 mod 4 = 0 (hoch)
0 (hoch)	+3	+2	5	5 mod 4 = 1 (rechts)
1 (rechts)	+1	+2	4	4 mod 4 = 0 (hoch)
1 (rechts)	+2	+2	5	5 mod 4 = 1 (rechts)
1 (rechts)	+3	+2	6	6 mod 4 = 2 (runter)
2 (runter)	+1	+2	5	5 mod 4 = 1 (rechts)
2 (runter)	+2	+2	6	6 mod 4 = 2 (runter)
2 (runter)	+3	+2	7	7 mod 4 = 3 (links)
3 (links)	+1	+2	6	6 mod 4 = 2 (runter)
3 (links)	+2	+2	7	7 mod 4 = 3 (links)
3 (links)	+3	+2	8	8 mod 4 = 0 (hoch)

- Kommutativgesetz:

Ein Computer kann nicht mit unbegrenzt großen Zahlen rechnen. Also heutzutage schon, aber wenn man eine alte Programmiersprache nutzt, in der 32-Bit-Integer-Register lediglich 2.147.483.647 groß sein können, dann darf diese ach so große Zahl nicht überschritten werden, sonst stürzt das Programm ab. Um die Zahl nach jedem Rechenschritt also „möglichst klein“ zu halten, müssen Minusrechnungen zum Beispiel vor Plusrechnungen gesehen.

```
variable (a)
variable (b)
variable (c)
variable (d)
a := 90000000
b := 2000000000
c := 200000000
#d := b + c -- a   # würde zum Absturz führen, da Variable überläuft
d := -- a + b + c  # würde funktionieren, obwohl logisches Ergebnis ident.
```

Hier ein zweites Beispiel, wie eine quadratische Funktion aufgeschrieben werden muss, wenn die altmodische Programmierung keine Kommazahlen verstehen kann. Speicherplatz war eben knapp, da konnte man sich keine Kommazahlen mehr leisten.

```
variable (a)
a := 2 * b ^ 2 / 200 + 5 * b / 100    # Korrekte Berechnung
#a := 2 / 200 * b ^ 2 + 5 / 100 * b   # Führt zu Fehlern
```

Natürlich geben rein logisch beide Formeln dasselbe Ergebnis aus. Nicht aber, wenn die Rechnung (2 / 200) zu nichts führt. Und das ist der Fall, wenn in der Programmiersprache zu kleine Variablenräume benutzt werden, sodass das Ergebnis nicht als Kommazahl ausgegeben werden kann, sondern als „0“ gerundet wird. In der Programmiersprache BASIC sind die weiterführenden Stichworte zu diesem Thema SINGLE, DOUBLE, SHORT und LONG.

- Zahlensysteme:

Wir Menschen benutzen das 10er-System für unsere Zahlen. Zum Beispiel besteht die Zahl 532 aus 5 100-ern, 3 10-ern und 2 1-ern. Kurze Umschreibung: 1 ist 10^0, 10 ist 10^1, 100 ist 10^2, 1000 ist 10^3, usw. Sie symbolisieren zumindest unsere Ziffern im 10er-System.
Auf einem fernen Alienplaneten rechen sie möglicherweise mit einem 26er-System, sprich 26 möglichen Ziffern pro Stelle. Halunken! Hier bedenke man nun: 1 ist 26^0, 26 ist 26^1, 767 ist 26^2, usw. Statt mit Einer, Zehner und Hunderter rechnen die Aliens also mit Einer, Sechsundzwanziger, Sechshundertsechsundsiebziger...

```
variable (a)
variable (b)
variable (c)
variable (d)
zufall := random (0,17575)
a := val,mod,26                  # Berechnet die 1-er-Stelle (26^0)
b := ((val -- a),mod,676) / 26   # Berechnet die 26-er-Stelle (26^1)
c := (val-- a -- b * 26) / 676   # Berechnet die 676-er-Stelle (26^2)
```

Eine Zahl wird folgendermaßen zerlegt: 10000 = 14 * 676 + 20 * 26 + 16 * 1, woraus wir die Ziffern 14, 20 und 16 erhalten.

Wer tut sich so etwas an? Leute, die effiziente Speicherungen von Informationen in einer Zahl durchführen wollen. Schließlich musste nur eine große Zahl gespeichert werden anstelle von drei Kleinen.

So können auch „Bits“ zu einer leichter auslesbaren Maske zusammengefasst werden, aber das geschieht dann im Binärsystem (2-er-System). Die funktionieren dann mit zwei Ziffern, Null und Eins.