T.A.G.-Tutorium

Martin Oehm

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einleitung

1.1 Über dieses Tutorium

Dieses Tutorium ist als Ergänzung zum Benutzerhandbuch von T.A.G. gedacht für diejenigen, die sich nicht gerne lange mit Papierkram aufhalten, sondern direkt loslegen möchten. Es enthält viele Beispiele und Tips zum Schreiben von Text-Adventures mit T.A.G.

III Ab und zu gebe ich auch ein paar Kommentare über die Gestaltung von Adventures, in denen ich meine Ansichten vertrete. Diese subjektiven Bemerkungen sind mit drei Strichen gekennzeichnet.

× Ein Kreuz bedeutet, dass es hier Kniffe für Fortgeschrittene gibt oder Passagen, die vielleicht für manche interessant sind, aber nicht direkt weiterführen.

Aufgabe

Ab und zu werden Aufgaben gestellt, die am Ende des Tutoriums beantwortet werden. Die Anzahl der Sterne gibt den Schwierigkeitsgrad an, kein Stern ist eine einfache Übungsaufgabe, zwei Sterne sind eine Kopfnuss.

1.2 Die ersten Schritte

Die ersten Schritte sind für Leute gedacht, die mit MS-DOS und Windows nicht ganz vertraut sind. Alte Hasen dürfen diesen Absatz lächelnd überspringen und weiter hinten weiterlesen.

Ihr habt das Paket entpackt, es liegt zum Beispiel im Verzeichnis **`c:\tag`**. Das Paket enthält eine Reihe von Dateien, darunter drei ausführbare. Schauen wir uns zunächst einmal **`tam.exe`** an. (Je nachdem, wie Windows eingestellt ist, wird die Endung **`.exe`** nicht angezeigt. Die ausführbaren Dateien **`*.exe`** werden immer mit einem Fenster-Icon dargestellt: ein weißes Fenster mit blauem Rand.)

`tam.exe` ist die Text-Adventure-Maschine, mit diesem Programm kann man Adventures spielen. Doppelklickt einmal darauf. Was passiert? Ein DOS-Fenster kommt hoch und sagt Euch, dass etwas fehlt. Außerdem erscheint eine Liste mit Optionen.

Funktioniert die T.A.M. nicht? Doch, sie funktioniert. Es fehlt nur eine wichtige Angabe: Welches Adventure Ihr spielen wollt. Anders als bei den meisten Programmen, wo man eine Datei öffnet, nachdem das Programmfenster erscheint, muss man bei T.A.M. diesen Dateinamen beim Aufruf mit angeben.

Im Paket ist ein fertiges Adventure enthalten, nämlich **`karn.tag`**. Fertige Adventures haben also die Endung **`.tag`**. Der Befehl um es zu spielen heißt:

```
> tag karn
```

Dieser Befehl kann auf verschiedene Arten eingegeben werden:

Auf herkömmliche Art

Dazu öffnet man ein DOS-Fenster unter *Start* → *Programme* → *MS-DOS-Eingabeaufforderung*. Es erscheint ein schwarzes Textfenster, in dem man hinter dem Prompt, dem „>“-Zeichen, Befehle eingeben kann. Zunächst wechseln wir das Verzeichnis mit

```
> cd /tag
```

(wenn das Paket nach **`c:/tag`** entpackt wurde). Dann starten wir die Text-Adventure-Maschine mit

```
> tam karn
```

und können dann das Test-Adventure „Die Höhlen von Karn“ spielen. Mit *F1* oder „Ende“ geht es wieder heraus, und es erscheint wieder der DOS-Prompt.

Auf etwas elegantere Art

Man kann in so gennanten *Batch-Dateien* DOS-Befehle speichern, die dann beim Öffnen dieser Datei ausgeführt werden. So erstellen wir in **`\tag`** mit *Neu* → *Textdatei* eine neue Textdatei und benennen sie um in **`karn.bat`**. (Die Warnung von Windows ignorieren wir und drücken beherzt „Ja“). Dann bearbeiten wir sie, indem wir aus dem Pull-Down-Menü, das beim Klick mit der rechten Maustaste auf die Datei erscheint, *Bearbeiten* wählen. Es öffnet sich ein Texteditor mit **`karn.bat`**. Diese Datei ist noch leer, und wir schreiben einfach unseren Befehl

```
tam karn
```

hinein und speichern die Datei ab. Beim Doppelklick auf **`karn.bat`** wird dann dieser Befehl ausgeführt.

Absolut elegant

Unter Windows kann man mit bestimmten Dateiendungen Programme verknüpfen, wie z.B. Word mit **`*.doc`**, Notepad mit **`*.txt`** usw. Die Endung **`*.tag`** ist vermutlich noch frei. Also verknüpfen wir sie mit der Text-Adventure-Maschine T.A.M.

Im Windows-Explorer wählen wir dazu unter *Ansicht* → *Optionen* die Karteikarte *Dateitypen* und erstellen mit *Neuer Typ…* eine Verknüpfung mit dem Typ **`tag`**. Beschreibung „Text-Adventure“ und Erweiterung „tag“ eingeben, mit *Neu* eine neue Aktion „open“ generieren und diese mit **`tam.exe`** verknüpfen. Nun sollte sich bei jedem Doppelklick auf eine **`tag`**-Datei die T.A.M. mit dem entsprechenden Adventure öffnen.

Beschränken wir uns erst einmal auf die herkömmliche Art, die explizite Eingabe im DOS-Fenster. Nachdem Ihr ein wenig mit Karn herumgespielt habt, können wir uns unserem

eigentlichen Ziel zuwenden, dem Erstellen von Adventures. Dem Paket liegt der Quelltext von **karn.tag** bei. Er heißt **karn.adv**. Der Quelltext ist eine reine Textdatei, in der mehr oder weniger im Klartext die einzelnen Bestandteile des Adventures beschrieben werden.

Der Text-Adventure-Generator erzeugt aus diesem Quelltext eine **tag**-Datei, die dann mit der T.A.M. gespielt werden kann.

× Warum spielt man nicht direkt die **adv**-Datei? Dadurch, dass die **adv**-Datei reiner Text ist, ist sie größer als die **tag**-Datei, in der die Daten des Adventures in kompakter (binärer) Form abgelegt sind. Außerdem werden die Texte in der **tag**-Datei verschlüsselt, so dass neugierige oder ungeduldige Spieler keine Hinweise aus der **tag**-Datei entnehmen können, während ihnen in der **adv**-Datei alles mehr oder weniger klar dargelegt würde.

Das probieren wir direkt einmal aus. Dazu erstellen wir zunächst eine Sicherheitskopie des **tag**-Adventures

```
> copy karn.tag karn_bak.tag
```

da das Original überschrieben wird. Nun erstellen wir **karn.tag** neu mit dem Kommando

```
> tag karn
```

und überschreiben damit karn.tag. Vergleichen wir doch einmal die beiden Adventures mit

```
> dir *.tag
```

Die Größe der beiden dateien **karn.tag** und **karn_bak.tag** sollte gleich sein. Auch sollte man keine Unterschiede beim Spielen von **karn.tag** und **karn_bak.tag** feststellen.

Sollte etwas nicht stimmen, so ist vermutlich **karn.adv** verändert worden.

1.3 Der Quelltext

Nun schauen wir uns einmal den Quelltext **karn.adv** an. Dazu öffnen wir einen Texteditor und laden **karn.adv**.

Beim Schreiben von Adventures wird der Texteditor Euer Haputwerkzeug sein. Es kann jeder beliebige Editor benutzt werden, der reinen ASCII-Text herausschreibt. Um gut und effektiv arbeiten zu können, sind einige Funktionen unerlässlich oder zumindest äußerst praktisch:

Suchen und Ersetzen von Wörtern, lokal im markierten Text und global. Dabei sollte man Optionen wie „Groß-/Kleinschreibung berücksichtigen“ oder „Nur ganze Wörter suchen“ ein- und abschalten können.

Angabe der aktuellen Schreibposition, d.h. Zeile und Spalte

Einfaches Markieren, Kopieren, Ausschneiden und Einfügen von Text

Unkomplizierte Handhabung von mehreren offenen Dateien

Sprungfunktion zu einer bestimmten Zeile

Ausführen von Systemkommandos, damit man zum Testen nicht immer den Editor verlassen muss

Windows bringt schon einige Editoren mit, diese sind aber nicht unbdeingt optimal:

Notepad kann nur Dateien bis zu 64 Kilobytes öffnen und hat keine Ersetzen-Funktion. Ansonsten schnell und zuverlässig.

Wordpad kann große Dateien öffnen. *Wordpad* unterstützt Text-Formate wie Fettdruck, Kursive usw., von denen man für T.A.G. *keinen* Gebrauch machen sollte. Beim Abspeichern immer die Option „Nur Text“ wählen. Mich stört, dass die voreingestellte Schrift Times ist, für reine ASCII-Editoren bevorzuge ich Schriften mit fester Breite.

Word, *StarWrite* und ähnliches ist zum Erstellen von Quelltext absolut unbrauchbar. Diese Textverarbeitungsprogramme erstellen formatierten Text, und sind viel zu unhandlich für unsere Bedürfnisse.

III Ich benutze für meine Programme und Adventures den Texteditor Proton von Ulli Meybohm, der über umfangreiche Funktionen verfügt und nahezu unbegrenzten Textspeicher hat. Ihr findet ihn unter

```
http://www.meybohm.de/proton.html
```

Im Netz gibt es aber bestimmt noch andere gute Editoren, so dass jeder seinen „Leib-und-Magen“-Schreibwerkzeug finden kann.

OK, wir haben jetzt den Quelltext vor unseren Augen. Wenn wir einmal ein bisschen herumschmökern, fallen uns folgende Grundelemente auf:

Schlüsselwörter wie **`Obj`**, **`Raum`**, **`Name`** oder **`Besch`**, die sich oft wiederholen.

Ausgabetexte, die in einfache Anführungszeichen gesetzt sind und die über mehere Zeilen gehen können. Diese Texte erscheinen irgendwann einmal auf dem Bildschirm.

Vokabeln sind ebenfalls in einfache Anführungsstriche gesetzt. Sie enthalten aber in der Regel keine Leerzeichen, sind eher kurz und klein geschrieben. Diese Wörter werden von der Text-Adventure-Maschine verstanden.

Anweisungen die in Blöcken zwischen **`Ausf`** und **`AusfEnde`** zusammengefasst sind.

Steuerkommandos, die aus einem Lattenzaun (#) und drei Buchstaben bestehen.

Diese Elemente werden wir uns im nächsten Kapitel näher anschauen, wenn wir unser erstes eigenes Adventure schreiben.

Zusammenfassung:

- T.A.G. und T.A.M. benötigen mindestens ein Argument, um gestartet werden zu können: den Namen des Spiels. Die einfachste Methode, T.A.G./T.A.M. zu starten, ist aus der MS-DOS-Eingabeaufforderung heraus.

- Zum Erstellen des Adventures kann ein beliebiger ASCII-Editor verwendet werden.

Verweis:

- Handbuch, Kapitel 2: Grundkonzept eines Adventures

2 Eine virtuelle Welt

2.1 Es geht endlich los

Nun wollen wir unser erstes Adventure schreiben. Kreativ, wie wir sind, nennen wir es **`test.adv`** und öffnen es im Editor.

Im Editor schreiben wir:

```
Raum    Vor_dem_Haus
Name    'Vor dem Haus'
Besch   'Du befindest Dich vor einem kleinen,
        weißen Haus, das im Osten liegt. Vor dem
        Haus wachsen wilde Blumen und ein kleiner
        Trampelpfad führt nach Norden in einen
        Wald.'
Ende
```

Abspeichern und mit **`tag test`** generieren. Es sollte keine Fehler geben, ansonsten noch einmal die Syntax überprüfen.

Damit haben wir das minimal mögliche Adventure mit nur einem Raum erstellt. T.A.G. verlangt nämlich, dass mindestens ein Raum da ist, damit der Spieler nicht im Nirgendwo ist. Das Wort **`Ende`** beendet den Quelltext, alles was danach geschrieben wird, wird nicht mehr untersucht.

Der Name des einzigen Raumes innerhalb von T.A.G. ist **`Vor_dem_Haus`**. Jeder Raum muss solch einen Namen besitzen, mit dem man sich im weiteren Quelltext auf ihn beziehen kann. Solche Namen, oder oft auch *IDs* genannt, können bis zu 36 Zeichen lang sein und dürfen nur Buchstaben, Ziffern und den Unterstrich enthalten, der oft dazu verwendet wird, Leerzeichen darzustellen. Das erste Zeichen in der ID darf keine Ziffer sein.

Groß- und Kleinschreibung ist bei IDs allerdings egal, genauso sind die Umlaute und ihre Umschreibung mit „ae“, „oe“ und „ue“ sowie „ss“ und „ß“ identisch. Das heißt, dass **`suedstrasse`**, **`Südstraße`** und **`SÜDSTRASSE`** denselben Ort meinen, aber verschieden von **`Süd_Straße`** sind. Wie diese ID verwendet wird, wird gleich gezeigt.

Dieses Adventure ist allerdings noch sehr langweilig. Ein Aufruf von **`tam test`** bringt dies an den Tag:

> Du befindest Dich vor einem kleinen, weißen Haus, das im Osten liegt. Vor dem Haus wachsen wilde Blumen und ein kleiner Trampelpfad führt nach Norden in einen Wald.
>
> **`> I`**
> Du hast nichts bei dir.

> N
Ich kenne „N“ nicht.

> U HAUS
Ich kenne „Haus“ nicht.

> GEHE NACH OSTEN
Ich kenne „Osten“ nicht.

> SCHAU DICH UM
Du befindest Dich vor einem kleinen, weißen Haus, das im Osten liegt. Vor dem Haus wachsen wilde Blumen und ein kleiner Trampelpfad führt nach Norden in einen Wald.

> ENDE
Spiel wirklich beenden? [J/N] **JA**

Da wir nur einen Raum definiert haben, kennt T.A.M. auch außer den von vorneherein in T.A.G. definierten Verben wie „gehe“, „schaue“ usw. nichts, noch nicht einmal die Richtungen.

2.2 Eine kleine „Welt“

Also erweitern wir unsere Definition:

```
#DAT    'normal.adv'

Raum    Vor_dem_Haus
Name    'Vor dem Haus'
O       Im_Haus
N       Lichtung
Besch   'Du befindest Dich vor einem kleinen,
        weißen Haus, das im Osten liegt. Vor dem
        Haus wachsen wilde Blumen und ein kleiner
        Trampelpfad führt nach Norden in einen
        Wald.'

Raum    Im_Haus
W       Vor_dem_Haus
Name    'In dem kleinen Haus'
Besch   'Das kleine Haus ist irgendwie leer: Es
        gibt keine Möbel und die Wände sind kahl.
        Im Westen geht es wieder hinaus.'

Raum    Lichtung
S       Vor_dem_Haus
Name    'Auf einer Lichtung'
Besch   'Dies ist eine kleine Lichtung in einem
        Laubwald, der hauptsächlich aus Eichen und
        einigen Birken besteht. Dichtes Unterholz
```

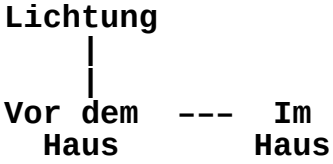
```
            macht den Wald undurchdringbar, nur ein
            Trampelpfad führt nach Süden.'

    Ende
```

Mit dieser Definition kann man nun in drei Räumen herumlaufen. Die Anweisung

```
    #DAT    'normal.adv'
```

liest die Datei **normal.adv** ein, als ob sie Bestandteil der Hauptdatei **test.adv** wäre. Mit diesem Befehl kann man sein Adventure unterteilen. Denkbar wäre z.B. eine Datei für die Befehle, eine für die Objekte oder eine Datei für jeden Abschnitt im Adventure. Die Datei **normal.adv** ist im Paket enthalten und definiert einige zusätzliche Befehle sowie zwölf Grundrichtungen, nämlich die acht Himmelsrichtungen, hoch, runter, herein und heraus.

Die dort festgelegten Richtungen kann man jetzt in der Raum-Definition angeben. Die Zeile **O Im_Haus** bedeutet „Von hier aus geht es nach Osten zu dem Ort mit der ID **Im_Haus**“. Mit diesen Definitionen erhalten wir eine „Landschaft“, die ungefähr so aussieht:



Die Reihenfolge der Angaben zu jedem Raum ist übrigens egal. Alles, was nach der Zeile, die mit **Raum** beginnt, definiert wird, gehört zum selben Raum, egal, ob es Leerzeilen dazwischen gibt oder nicht. Man nennt das einen Definitionsblock. Dieser Block hört automatisch dann auf, wenn ein anderer beginnt, was in unserem Fall die nächste Raum-Definition ist.

Aufgabe 1

Erweitere die Raum-Definitionen so, dass man vor dem Haus mit „rein“ ins Haus gelangt und von dort mit „raus“ wieder hinaus.

2.3 Einige Verfeinerungen

Wer mit **tag test** und **tam test** unser neues Opus schon ausprobiert hat, dem ist beim Umherwandern in unserer virtuellen Welt bestimmt einiges aufgefallen:

> Beim ersten Betreten des Raums wird der lange Text (**Besch**) angegeben, danach nur noch der kurze (**Name**). Dieser Modus heißt „knapp“. Mit „ausführlich“ kann man sich bei jedem Betreten eines Raums die ganze Beschreibung anzeigen lassen. In jedem Fall bringt „schau dich um“ oder „l“ die komplette Beschreibung.
>
> Der Text erscheint nicht so, wie er eingegeben wurde, die Umbrüche sind

woanders: T.A.M. behandelt die Umbrüche aus der **adv**-Datei als Leerzeichen und bricht automatisch bei 80 Zeichen um.

Wenn man versucht, in eine Richtung zu gehen, in der keine Ausgänge definiert werden, sagt T.A.M. „Du kannst nicht in diese Richtung gehen.“

Genau diese eintönige Aussage wollen wir ändern, denn sie erinnert doch sehr stark an die schlechte alte Zeit, als die Antwort auf fast alles „Das geht leider nicht“ war.

Dazu definieren wir eine so genannte *Antwort*:

```
Antwort Unterholz
Besch   'In dieser Richtung ist das Unterholz zu
        dicht. Der eizige Weg ist ein Trampelpfad
        nach Süden.'
```

Diese Antwort ist wieder ein Definitionsblock für sich, darf also nicht innnerhalb einem der anderen Blöcke stehen. Die einzige Angabe ist die Beschreibung. Nun ändern wir die Definition der Lichtung ab auf:

```
Raum    Lichtung
N       Unterholz
NO      Unterholz
O       Unterholz
SO      Unterholz
SW      Unterholz
W       Unterholz
NW      Unterholz
S       Vor_dem_Haus
Name    'Auf einer Lichtung'
Besch   'Dies ist eine kleine Lichtung in einem
        Laubwald, der hauptsächlich aus Eichen und
        einigen Birken besteht. Dichtes Unterholz
        macht den Wald undurchdringbar, nur ein
        Trampelpfad führt nach Süden.'
```

Dann erscheint jedesmal, wenn man von der Lichtung aus in eine der Himmelsrichtungen außer Süden gehen will, der Satz, dass dort das Unterholz zu dicht ist und dass der Ausgang im Süden ist. Nun weiß der Spieler, warum er nicht in die gewünschte Richtung gehen kann und was er für Alternativen hat, und das ist doch besser als der Standard-Satz, oder?

Was noch stört, sind die vielen **Unterholz**-Angaben. Da es oft mehr „Das geht nicht“-Richtungen als Ausgänge gibt, kann man diese Definition abkürzen auf:

```
Raum    Lichtung
Std     Unterholz
S       Vor_dem_Haus
Name    'Auf einer Lichtung'
Besch   'Dies ist eine kleine Lichtung in einem
        Laubwald, der hauptsächlich aus Eichen und
        einigen Birken besteht. Dichtes Unterholz
        macht den Wald undurchdringbar, nur ein
        Trampelpfad führt nach Süden.'
```

Std heißt *Standard* und belegt alle Standard-Richtungen mit der Antwort **Unterholz**. In **normal.adv** wurden nur die Himmelsrichtungen als Standard-Richtung definiert (mit dem Sternchen), und das macht auch Sinn, denn wenn der Spieler auf der Lichtung nach oben gehen will, ist der Grund, warum das nicht klappt, wohl nicht das dichte Gestrüpp.

Die **Std**-Definition muss vor den anderen Richtungsdefinitionen stehen, da zunächst alle Himmelsrichtungen vorbelegt werden, und dann der Ausgang im Süden mit **Vor_dem_Haus** überschrieben wird. In der **Std**-Definition können auch andere Räume angegeben werden, obwohl dies meist keinen Sinn macht, denn wo kommt man denn immer in denselben Raum, egal, ob man nach Süden, Norden oder Osten geht? Etwas surreal, aber technisch möglich.

Aufgabe 2

Schreibe passende Antworten für die beiden anderen Räume.

2.4 Im Dunkeln …

Wir haben die Haupt-Eigenschaften eines Raums kennengelernt:

> Die *ID*, die innerhalb des Quelltext diesen Raum repräsentiert
>
> Der *Name*, der in der Statuszeile erscheint und der als Kurzbeschreibung hergenommen wird, wenn der Raum bereits bekannt ist
>
> Die *Beschreibung*, die dem Spieler ein Bild von seinem Aufenthaltsort vermittelt
>
> Die *Ausgänge*, die entweder in einen anderen Raum führen oder einem nur sagen, dass es in dieser Richtung nicht weiter geht

Eine wichtige Sache fehlt noch: Räume können Attribute besitzen. Ein Attribut für Räume ist z.B. **besucht**. Dieses Attribut bezeichnet, ob ein Raum dem Spieler bereits bekannt ist oder nicht und wird unter anderem für die langen und kurzen Raumbeschreibungen herangezogen. Dieses Attribut wird allerdings automatisch von T.A.M. gesetzt, wenn ein Raum betreten wird.

Das zweite wichtige Attribut ist **dunkel**. Besitzt ein Raum dieses Attribut, so kann der Spieler in diesem Raum nichts sehen, es sei denn, er hat eine Lampe oder Ähnliches bei sich. Von einem dunklen Raum aus kann der Spieler nur in einen hellen, bekannten Raum gehen. So kommt er zumindest immer wieder zurück, wenn er will.

Als Beispiel definieren wir einen Keller für das Haus:

```
Raum    Im_Haus
W       Vor_dem_Haus
R       Keller
```

```
Name    'In dem kleinen Haus'
Besch   'Das kleine Haus ist irgendwie leer: Es
        gibt keine Möbel und die Wände sind kahl.
        Im Westen geht es wieder hinaus.

        Durch eine offenen Falltür im Boden siehst
        du eine Treppe, die nach unten in die
        Dunkelheit führt.'

Raum    Keller
Name    'Keller'
H       im_Haus
Besch   'Dieser Keller riecht modrig, die feuchten
        Wände sind mit Moos bewachsen. Das, was
        von den verfaulten Regalen übriggeblieben
        ist, ist leer.

        Eine intakte Holztreppe führt zurück nach
        oben.'
Attr    dunkel
```

(Die Leerzeile in der Raumbeschreibung bewirkt einen Absatz bei der Ausgabe.)

Der Keller ist dunkel, das bewirkt die Zeile **`Attr dunkel`**. Wenn wir das neue Adventure erstellen und spielen wie gehabt, so erhalten wir, wenn wir vom Haus aus nach unten gehen (**`r`** = „runter“, **`h`** = „hoch“), die Nachricht „Es ist zu dunkel hier, um etwas sehen zu können“, anstelle der Raumbeschreibung, und in der Statuszeile steht „Im Dunkeln …“. Wir befinden uns zwar im Keller, aber wir können nichts machen, außer wieder zurück nach oben zu gehen. Wenn wir versuchen, in eine andere Richtung zu gehen, stolpern wir nur umher. Aber das ändert sich im nächsten Kapitel, in dem wir anfangen, Objekte zu definieren.

Zusammenfassung:

- *Räume* bilden das Grundgerüst eines Adventures, die Landschaft.

- Die Räume können in abgeschlossenen Blöcken definiert werden. Sie bekommen dort einen *Namen* und eine *Beschreibung*.

- Um die Räume miteinander zu verbinden, muss man *Richtungen* definieren. Zu jedem Raum kann ein Ausgang in jede Richtung angegeben werden. Dieser Ausgang kann ein anderer Raum oder eine *Antwort* sein.

- Räume können *Attribute* besitzen.

Verweis:

- Handbuch, Kapitel 3.1: Räume und ihre Verbindungen

3 Objekte

3.1 Unser erstes Objekt: ein Zettel

Was ist ein Adventure ohne Schwert, Lampe, und Proviant? Sehr langweilig, wie wir aus dem vorigen Kapitel wissen. Deshalb wollen wir nun ein paar Objekte einfügen. Als Begrüßungsbotschaft legen wir dem Spieler einen Zettel vors Haus:

```
Obj     Zettel
Name    'Zettel' m
Subst   'zettel' m
Ort     vor_dem_Haus
```

Das ist im Grunde genommen alles. Das Objekt hat eine ID, die von allen anderen IDs eindeutig zu unterscheiden sein muss, auch von allen Räumen und Antworten.

Es hat einen Namen, der in der Ausgabe der T.A.M. verwendet wird. Dieser Name muss eine Angabe zum grammatikalischen Geschlecht enthalten, die dem Namen als **`m`**, **`f`**, **`n`** oder **`p`** nachgestellt wird. Es heißt „*der* Zettel“, also ist das Geschlecht **`m`** wie männlich oder maskulin.

Das **`Subst`**(antiv) gehört zum Vokabular des Objekts, mit dem es der Spieler in seinen Anweisungen ansprechen kann. Die einzige Angabe ist hier **`'zettel'`**, wieder gefolgt von einer Genus-Kennzeichnung. Wir können den Zettel also nur „Zettel“ nennen. Das Substantiv in Anführungsstrichen darf keine Leerzeichen haben. Alle Großbuchstaben werden in Kleinbuchstaben verwandelt, alle „ß“ in „ss“ und alle „ae“, „oe“ und „ue“ in „ä“, „ö“ und „ü“. Zum Zeichen, dass es sich um Vokabular handelt, benutze ich für die Substantive gerne Kleinbuchstaben.

Und schließlich hat der Zettel einen Ort, nämlich vor dem Haus. Adventure generieren und laufen lassen. Es erscheint:

> Du befindest Dich vor einem kleinen, weißen Haus, das im Osten liegt. Vor dem Haus wachsen wilde Blumen und ein kleiner Trampelpfad führt nach Norden in einen Wald.
>
> Hier ist ein Zettel.
>
> **`> NIMM DEN ZETTEL`**
> Du hast nun den Zettel.
>
> **`> LIES IHN`**
> Auf dem Zettel steht nichts.
>
> **`> UNTERSUCHE DEN ZETTEL`**

Du kannst an dem Zettel nichts Außergewöhnliches finden.

> I
Du hast folgendes bei dir:
- einen Zettel

> LEGE DEN ZETTEL HIN
Du hast den Zettel hingelegt.

> O
Das kleine Haus ist irgendwie leer: Es gibt keine Möbel und die Wände sind kahl. Im Westen geht es wieder hinaus.

Durch eine offene Falltür im Boden siehst du eine Treppe, die nach unten in die Dunkelheit führt.

> RAUS
Vor dem Haus

Hier ist ein Zettel.

> ENDE
Spiel wirklich beenden? [J/N] **JA**

Man kann den Zettel aufheben und irgendwo anders wieder ablegen, aber viel mehr gibt der Zettel im Moment nicht her.

3.2 Der Zettel bekommt Leben

Also erweitern wir die Definition des Zettels:

```
Obj     Zettel
Name    'Zettel' m
Subst   'zettel' m
Ort     vor_dem_Haus
Besch   'Der Zettel ist nur ein schmutziger Wisch.
        Es steht allerdings etwas darauf.'
Text    'Auf dem Zettel steht in krakeliger
        Schrift:

        "Willkommen in der Welt von T.A.G.!"'
Erst    'Etwa fünf Meter von der Hauswand entfernt
        liegt ein Stück Papier im Gras.'
```

Damit erhalten wir folgendes:

Du befindest Dich vor einem kleinen, weißen Haus, das im Osten liegt. Vor dem Haus wachsen wilde Blumen und ein kleiner Trampelpfad führt nach

Norden in einen Wald.

Etwa fünf Meter von der Hauswand entfernt liegt ein Stück Papier im Gras.

> NIMM DEN ZETTEL
Du hast nun den Zettel.

> UNTERSUCHE DEN ZETTEL
Der Zettel ist nur ein schmutziger Wisch. Es steht allerdings etwas darauf.

> LIES IHN
Auf dem Zettel steht in krakeliger Schrift:

„Willkommen in der Welt von T.A.G.!“

> LEGE DEN ZETTEL HIN
Du hast den Zettel hingelegt.

> L
Du befindest Dich vor einem kleinen, weißen Haus, das im Osten liegt. Vor dem Haus wachsen wilde Blumen und ein kleiner Trampelpfad führt nach Norden in einen Wald.

Hier ist ein Zettel.

Die Erst-Beschreibung ist also der Text, der anstelle von „Hier ist …“ ausgegeben wird, wenn das Objekt am Aufenthaltsort ist. Ist das Objekt aber schon einmal aufgehoben worden, gibt es wieder die „normale“ Beschreibung.

Die **`Besch`**(reibung) ist der Text, der beim Untersuchen ausgegeben wird, und **`Text`** beschreibt, was geschieht, wenn man das Objekt liest. Fehlen diese Angaben, so kommen die Antworten, die aus dem ersten Beispiel mit dem Zettel bekannt sind. Ist **`Text`** definiert, Besch aber nicht, so tritt ein Sonderfall in Kraft: Es wird beim Untersuchen der Lese-Text ausgegeben.

3.3 Das Vokabular

Es gibt jetzt natürlich ein Problem: der Zettel wird – aus stilistischen Gründen – nicht immer Zettel genannt, sondern auch „dreckiger Wisch“ und „Stück Papier“. Wenn der Spieler aber diese Wörter benutzen will, so bekommt er zu hören „Ich kenne ‚Wisch‘ nicht.“

Deshalb erweitern wir das Vokabular des Objekts Zettel:

```
Subst   'zettel' m   'wisch' m   'stück' n
Subst   'papier' n   'fetzen' m
Adj     'schmutzig'
```

```
Vor     'papier'
```

Es können **`Adj`**(ektive), **`Subst`**(antive) und **`Vor`**(wörter) definiert werden. Die Substantive benötigen alle eine Genus-Kennung, die Adjektive und Vorwörter nicht. Es können beliebig viele Vokabular-Definitionen je Objekt gemacht werden.

Mit diesen Angeben versucht die T.A.M. dann, aus der Eingabe des Spielers ein Objekt zu bestimmen. Dabei geht sie so vor:

(*a*) Suche einen Artikel.

(*b*) Suche ein oder zwei Adjektive mit passender Beugung. Werden zwei Adjektive angegeben, so kann das erste ungebeugt sein, also als Adverb benutzt werden.

(*c*) Suche eine mögliche Kombination aus Vorwörtern und Substantiven zu jedem Objekt, wobei das Vorwort nicht unbedingt vorkommen muss. Vor- und Hauptwort können auch durch einen Bindestrich getrennt sein.

(*d*) Ist zu (*c*) etwas gefunden worden, so suche weitere Kombinationen aus Vor- und Hauptwörtern.

Dabei muss nur (*c*) ein Ergebnis bringen. Die anderen Angaben dienen nur dazu, die Suche einzuschränken.

In unserem Beispiel könnten wir also eingeben

```
Zettel
schmutziger Wisch
Stück Papier
schmutziger Papierfetzen
```

und es würde als Objekt „Zettel“ verstanden. Allerdings könnte man auch sagen

```
schmutzig schmutziges Papier
Zettel Wisch Fetzen
Papierpapier
```

was sich etwas eigenartig liest, aber für die T.A.M. in Ordnung ist und ebenfalls bedeutet: Objekt „Zettel“. Um dem Spieler möglichst viele Eingaben zu erlauben, nimmt man die Erkennung von Unsinnigem in Kauf.

Es geht bei der Definition des Vokabulars natürlich nicht darum, akribisch alle Möglichkeiten aus dem Thesaurus abzuschreiben, aber ein gutes Objekt sollte mit seinem Vokabular schon die gängigsten Eingabemöglichkeiten abfangen.

3.4 Zustände und Attribute

Ein Zettel ist natürlich kein besonders ausgiebiges Beispiel für ein Text-Adventure-Tutorium. Außer einem Text zum Lesen, der sich noch dazu nicht ändert, hat er nicht viel zu bieten.

Objekte können nämlich, genau wie Räume, Attribute besitzen. Außerdem können sie – im Gegensatz zu Räumen – einen bestimmten Zustand haben. Erinnern wir uns also an den dunklen Keller und geben unserem Abenteurer eine Lampe an die Hand:

```
Obj     Lampe
Name    'Taschenlampe' f
Adj     'batteriebetrieben'
Vor     'taschen' 'batterie'
Subst   'lampe' f  'leuchte' f
Ort     beimir
Zust    aus
Attr    Licht
```

Diese Lampe ist zu Beginn **`beimir`**, also beim Spieler. Sie hat den Zustand **`aus`**. Außerdem besitzt sie das Attribut **`Licht`**, was bedeutet, dass sie eine Lichtquelle ist, wenn sie nicht aus ist. Mit dieser Lampe können wir uns nun in den Keller wagen. Der Keller ist zwar immer noch modrig, aber nicht mehr stockfinster. Besser so, oder?

Was ist der Unterschied zwischen Attribut und Zustand? Ein Objekt hat immer nur einen Zustand, der aus meheren gewählt werden kann. Bereits vordefinierte Zustände sind **`ein`**, **`aus`**, **`kaputt`**, **`offen`**, **`geschlossen`** und **`abgeschlossen`**.

Ein Attribut ist eine Eigenschaft, die ein Objekt hat oder nicht hat. Ein Objekt kann beliebig viele Attribute besitzen.

Der definierte Zustand bewirkt eine weitere Sache: Zustände können neben IDs auch Bezeichnungen haben. Die Lampe hat keine Beschreibung. Trotzdem erscheint nicht die Pauschalantwort „Du siehst nicht Außergewöhnliches“, sondern die Beschreibung des Zustands: „Die Lampe ist aus“.

Aufgabe 3

Erzeuge ein Taschenmesser, das der Spieler bei sich hat und das man auf- und zumachen kann.

Zusammenfassung:

- Jeder Gegenstand im Spiel wird durch ein Objekt dargestellt.

- Ähnlich wie Räume werden Objekte in Blöcken definiert. neben einem Namen und einer Beschreibung besitzen sie weitere Eigenschaften: Vokabular, Attribute, Zustände und einen Ort.

- Als Vokabular können Adjektive, Vorwörter (von zusammengesetzten Hauptwörtern) und Hauptwörter (Substantive) angegeben werden. Zu Substantiven muss zusätzlich das grammatikalische Geschlecht definiert werden. Jedes Objekt kann beliebig viel Vokabular

haben.

- Attribute sind Eigenschaften, die ein Objekt hat oder nicht.

- Objekte haben zusätzlich einen Zustand, eine Eigenschaft, die einen Wert aus einer Anzahl von definierten Zuständen (ein, aus, offen, kaputt, usw.) annehmen kann.

Verweis:
- Handbuch, Kapitel 3.2: Objekte

4 Einige besondere Objekte

4.1 Behälter

Bisher haben wir Objekte kennengelernt, die in Räumen liegen und die beim Spieler sind. Manchmal befinden sich aber auch Objekte in anderen Objekten. Objekte, die andere aufnehmen können, heißen *Behälter*.

`Behälter` ist ein Attribut, das einem Objekt gegeben werden kann. Für unser Test-Adventure geben wir dem Spieler einen Rucksack mit:

```
Obj     Rucksack
Name    'Wanderrucksack' m
Adj     'oliv' 'grau' 'olivgrau'
Vor     'wander' 'seiten'
Subst   'sack' m   'rucksack' m   'taschen' p
Ort     angezogen
Zust    geschlossen
Attr    Behälter Kleidung
Besch   'Es ist ein altmodischer Rucksack aus
        olivgrauem Segeltuch mit vielen
        Seitentaschen und Lederriemchen.'
```

Dieser Rucksack kann dank seines **`Behälter`**-Attributs andere Objekte aufnehmen. Außerdem erhält er das Attribut **`Kleidung`**, damit man ihn aufziehen kann. Zu Beginn des Spiels hat der Spieler den Rucksack auf dem Buckel, was durch **`Ort angezogen`** angegeben wird. **`Angezogen`** ist ein Speziallfall von Beimir und wird von T.A.G. anders gehandhabt. (Probiert es einmal aus, und gebt als ersten Befehl „Lege Rucksack hin“ ein.)

Probiert einmal, mit den bereits vorhandenen Objekten herumzuspielen. Legt die Lampe und den Zettel in den Rucksack. Macht den Rucksack einmal auf und wieder zu, wenn sich etwas darin befindet. Legt den Rucksack ab und gebt mit „L“ eine Raumbeschreibung aus. Und tut die angeschaltete Lampe einmal in den Rucksack, geht in den Keller und schließt dann den Rucksack.

Unser Adventure hat schon etwas mehr zu bieten als noch vor kurzem. Immer, wenn sich etwas im Rucksack befindet und dieser offen ist, wird der Inhalt des Rucksacks aufgelistet: „In dem Rucksack siehst Du …“

Auch diese Antwort lässt sich individuell auf den Rucksack zuschneidern:

```
Darin   'Im Moment [ist 0] [liste 0] im Rucksack.'
```

Die genaue Bedeutung der eckigen Klammern wird später erläutert. Im Moment genügt es, zu wissen, dass **`[liste 0]`** die Auflistung aller Objekte im Nominativ (= 0) einfügt, und **`[ist 0]`** je nach Anzahl der Objekte in der Liste „ist“ oder „sind“ ausgibt.

Zu Beginn des Spiels können andere Objekte bereits im Rucksack plaziert werden, in dem man die Ortsangabe **`Ort in Rucksack`** angibt. (Achtung: Man ist natürlich verleitet, **`IM Rucksack`** zu schreiben, aber „im“ wird von T.A.G. nicht verstanden. Im Spiel unter T.A.M. ist dies jedoch möglich.)

Aufgabe 4

Erzeuge eine verschließbare Butterbrotdose, die sich im Rucksack befindet, und in der sich eine Tomate, ein hartgekochtes Ei und ein Sandwich befinden. (Oder was Du sonst gerne isst.)

Aufgabe 5

Ein Anorak wäre für eine Expedtion auch nicht schlecht. Erzeuge einen, der zu Beginn des Spiels im Rucksack ist.

Man kann die Objekte im Rucksack nur sehen, wenn dieser offen ist. Allgemeiner kann man Objekte in einem Behälter sehen, wenn er nicht geschlossen ist, denn ein Behälter muss nicht unbedingt verschließbar sein, und was in einem Eimer oder einem Karton ohne Deckel ist, kann man immer sehen.

Mit dem Attribut **`transparent`** kann man Behälter aber durchsichtig machen. Man kann dann Objekte im transparenten Behälter sehen, aber nicht anfassen.

Aufgabe 6

Definiere ein geschlossenes Einmachglas, in dem sich zur Abrundung unserer Vesper eine Essiggurke befindet. Man kann die Gurke natürlich durch das geschlosene Glas sehen.

Der Befehl **`essen`** ist kein Standard-T.A.G.-Befehl. Er ist aber in **`normal.adv`**, die wir ja in unserem Text-Adventure benutzen, enthalten. Zusätzlich steht uns also ein Attribut **`essbar`** zur Verfügung. Um in den Genuss unserer Marschverpflegung zu kommen, sollten wir also der Tomate, der Gurke und dem ganzen Zeug das Attribut **`essbar`** verpassen.

4.2 Ablagen und feste Objekte

Genauso, wie Objekte in anderen enthalten sein können, können sie auch auf anderen liegen. Dazu gibt es das Attribut **`Ablage`**:

```
Obj     Tisch
Name    'klein^ Holztisch' m
Adj     'klein' 'grob' 'wackelig'
Vor     'holz'
Subst   'tisch' m
Besch   'Grob zusammengezimmert und etwas
        wackelig.'
Attr    Ablage
Ort     Im_Haus
```

(Das Dach im Namen zeigt an, dass es sich beim vorangehenden Wort um ein Adjektiv handelt. Anstelle des Dachs wird die passende Endung eingefügt, je nachdem, in welchem Fall das Objekt ausgegeben werden soll. Das Adjektiv wird im Namen – genau wie bei den Vokabeln – ungebeugt angegeben.)

Damit haben wir einen Tisch, auf dem wir Dinge ablegen können. Um Sachen schon zu Beginn des Spiels auf den Tisch zu legen, geben wir **`Ort auf Tisch`** an. Das Pendant zu **`Darin`** ist **`Darauf`**.

III Jetzt, wo ein Tisch im Haus steht, ist die Beschreibung des Raums nicht mehr ganz korrekt. Anstatt „Es gibt keine Möbel“ sollte man vielleicht „nur wenige Möbel“ oder „Der Raum ist spärlich möbliert“ schreiben. Solche Nachbesserungen werden oft nötig, wenn andere Objekte ins Spiel kommen.

Habt Ihr mit dem Tisch schon herumgespielt und Sachen daraufgelegt und wieder weggenommen? Schön. Habt Ihr auch einmal versucht, den Tisch zu aufzuheben? Wenn, nicht, dann probiert es mal. Hoppla: „Du hast nun den Tisch.“ Das ist schon etwas eigenartig, oder?

Eigenartig vielleicht, aber richtig: Wenn nichts weiteres angegeben wird, sind alle Objekte beweglich. Das heißt, sie können aufgehoben und mitgeschleppt werden. Es gibt aber in Adventures viele Objekte, die man nicht aufheben kann, wie eben z.B. Möbel. Für solche Objekte gibt es das Attribut **`fest`**.

Also, dem Tisch **`fest`** verpassen, Adventure neu generieren und ausprobieren. Beim Betreten des Raums fällt sofort etwas auf: Der Tisch ist nicht mehr da. Falsch, er ist da, er wird nur nicht erwähnt. Mit „untersuche Tisch“ bekommt man nämlich eine Beschreibung des Tisches. Auch kann man Sachen auf den Tisch legen. Schaut man sich dann um, so erscheint die Liste aller Dinge, die auf dem Tisch liegen.

Feste Gegenstände werden also nicht in der Liste aller Objekte am Ende einer Raumbeschreibung erwähnt. Das wird so gemacht, weil sie üblicherweise bereits in der Raumbeschreibung vorkommen. Eine neue, angepasste Raumbeschreibung für **`Im_Haus`** wäre jetzt etwa:

```
Raum    Im_Haus
        ...
Besch   'Das kleine Haus wirkt irgendwie leer:
        Außer einem kleinen Tisch gibt es keine
        Möbel und die Wände sind kahl. Im Westen
        geht es wieder hinaus.'
```

Man kann aber, wenn man will, auch feste Objekte wie gewohnt anzeigen lassen. Dazu gibt man ihnen nicht das Attribut **`fest`**, sondern das Attribut **`immobil`**, was dasselbe ist wie **`fest`**, mit der Ausnahme, dass das Objekt in der Liste der Gegenstände auftaucht.

III Natürlich spricht nichts dagegen, den Tisch beweglich zu machen, zumal er laut Beschreibung klein und wacklig ist, aber im allgemeinen schleppt man so etwas ja nicht mit sich herum.

Aufgabe 7

Ändere die Raumbeschreibung im Keller und definiere die modrigen Regale. Benutze dabei

eine sinnvolle **Darauf**-Definition.

4.3 Das Inventar

Zeit, sich einmal damit zu beschäftigen, was der Spieler so alles mit sich herumtragen kann. Man kann sich eine Liste der Objekte, die der Spieler bei sich trägt, das sogenannte Inventar, mit „Inventar“ oder kurz „I“ ausgeben lassen.

Im Normalfall ist dies eine Liste mit Spiegelstrichen. Dinge, die in oder auf anderen Dingen sind, werden dabei eingerückt.

Der Spieler kann natürlich nur eine bestimmte Anzahl von Dingen mit sich herumschleppen. Diese Anzahl wird in einer Variable festgelegt, die **maxInv** heißt. Diese kann nur in einer Anweisung verändert werden, wie das geht, sehen wir im nächsten Kapitel.

Wenn nichts anderes angegeben wird, ist **maxInv** vier. Das heißt, der Spieler kann vier Gegenstände bei sich haben, mehr nicht. Allerdings zählen Gegenstände, die er angezogen hat und die sich in anderen Gegenständen befinden, nicht dazu, sondern nur die, die er direkt in seinen Händen hält. Insofern ist der Rucksack natürlich eine prima Sache: Man belastet den Spieler nicht, kann aber ziemlich viel mit sich herumschleppen.

Eine weitere Variable, **Inv**, gibt an, wieviele Objekte der Spieler gerade mit sich herumschleppt. Dieser Wert liegt natürlich zwischen 0 und **maxInv**.

Für diejenigen, die es gerne genau nehmen, bietet T.A.G. zwei weitere Konzepte, um Gegenstände zu handhaben: Gewicht und Volumen.

Jedem beweglichen Gegenstand kann ein Gewicht von 0 bis 255 gegeben werden. Dazu fügt man seiner Objektdefinition einfach die Zeile **Gew** *xxx* zu, wobei statt *xxx* natürlich der Wert des Gewichts eingesetzt werden soll. Die Maßeinheit spielt dabei keine Rolle, sie sollte nur einheitlich sein.

Der Spieler kann natürlich nur ein bestimmtes Gewicht mit sich tragen. Dieses „zulässige Höchstgewicht“ wird in der Variable *maxGew* gespeichert. Dabei wird natürlich das Gewicht der Dinge, die der Spieler angezogen hat und die sich in einem Behälter, den er trägt, befinden, miteinberechnet. Denn eine Ritterrüstung oder ein Goldnugget im Rucksack ziehen einen doch ganz schön runter.

Eine weitere Variable, **InvGew**, zeigt an, wieviel der Spieler noch mit sich schleppen könnte. Das heißt, **InvGew** ist **maxGew** minus dem Gewicht, das er mit sich schleppt. Ist **InvGew** null, ist der Spieler am Limit, ist es **maxGew**, so schleppt der Spieler nichts oder nur Gegenstände mit dem Gewicht 0 mit sich herum.

Neben dem Gewicht kann man jedem Gegenstand mit **Vol** *xxx* ein Volumen von 0 bis 255 zuordnen. Das Konzept des Volumens ist etwas anders als das des Gewichts, denn das Volumen ist nicht nur ein Maß für das physikalische Volumen, sondern auch für die

„Sperrigkeit“ eines Objekts. Die Variablen heißen analog zu **`InvGew`** und **`maxGew`** hier **`InvVol`** und **`maxVol`**.

Bei der Berechnung von InvVol werden wiederum angezogene und in anderen Objekten befindliche Gegenstände nicht mit einbezogen. Logisch, denn ein Gegenstand in einem Behälter vergrößert dessen Volumen nicht. Und ein aufgezogener Hut ist nicht sperrig.

Das Volumen der Gegenstände wird auch für die Schachtelung von Objekten ineinander benutzt. Das Volumen eines Behälters ist nicht nur sein Außenvolumen, sondern ebenfalls sein Fassungsvermögen. Das heißt, die Summe der Volumina aller Objekte in einem Behälter darf nicht größer sein als das Volumen des Behälters selbst. Dieses Konzept gilt auch für Ablagen, die zwar kein Volumen in dem Sinne haben, aber auch keine unbegrenze Kapazität. Hier ist also etwas Abstraktions- vermögen bei der Volumendefinition gefragt.

Wenn man die Definition von Volumen verwendt, verhindert man damit unmögliche Sachen, z.B., dass man einen großen Karton in ein kleines Ebenholzkästchen legen kann. Man muss sich allerdings die Mühe machen, und sinnvolle Volumina zu jedem Objekt finden. Achtung: Behälter sollten dann in jedem Fall ein Volumen erhalten, auch wenn sie **`fest`** oder **`immobil`** sind.

Wer zu keinem Objekt ein Gewicht oder ein Volumen definiert, lässt diese Konzepte einfach außer acht, so wie wir es bis jetzt getan haben. Die Beschränkung der beim Spieler getragenen Gegenstände greift jedoch immer, kann aber durch Setzen von **`maxInv`** auf einen usinnig hohen Wert, z.B. 200, umgangen werden.

Aufgabe 8

Definiere für alle Objekte, die bis jetzt in Deinem Testspiel sind, sinnvolle Volumen. Erzeuge einen Wanderstab, der nicht in den Rucksack passt, den man aber auf den Tisch legen kann.

× Auch ein Volumen oder ein Gewicht von Null können sinnvoll sein: Eine Feder oder eine Plastikkarte nehmen bestimmt keinen großen Platz ein, und die Antwort „Die Feder ist zu schwer/sperrig“ kann man nicht unbedingt nachvollziehen.

4.4 Sitze, Liegen, Standflächen

Auf manche Objekte kann man nicht nur andere Objekte setzen oder stellen oder legen, sondern auch sich selbst. Dazu gibt man ihm einfach eins oder mehrere der Attribute **`Sitz`**, **`Standfläche`** oder **`Liege`**.

Im Gegensatz zu Objekten wird beim Spieler unterschieden, ob er auf einem Gegenstand sitzt, steht oder liegt. Deshalb gibt es für jede dieser Haltungen ein Attribut. Ist der Spieler auf keinem anderen Objekt, sondern direkt in einem Raum, so ist seine Haltung jedoch egal. Meist wird einfach angenommen, dass er steht.

Also, ein neues Objekt:

```
Obj     Sofa
Name    'verschlissen^ Sofa' n
Vor     'alt' 'verschlissen' 'abgenutzt' 'rot'
Subst   'sofa' n  'couch' f  'chaiselongue' n
Subst   'diwan' m
Besch   'Dieses mittlerweile sehr mitgenommene
        Sofa war in seinen guten Zeiten einmal
        rot. Trotz der Abnutzungserscheinungen
        sieht es doch sehr gemütlich aus.'
Erst    'In einer Ecke des Raums steht ganz
        verstohlen ein altes, abgenutztes Sofa
        herum.'
Attr    Sitz Liege Immobil Ablage
Vol     50
```

Probiert das neue Sofa einmal aus. Setzt, stellt und legt Euch darauf (wenn es geht). Eine **`Liege`**, ein **`Sitz`** oder eine **`Standfläche`** müssen übrigens keine Ablage sein. Sie müssen auch nicht **`fest`** sein, der Spieler darf sie jedoch nicht bei sich haben, wenn er sich auf sie setzen will.

Aufgabe 9

Definiere einen Klappstuhl, der im Keller steht, und den der Spieler mit sich herumschleppen kann. Man kann sich auf ihn setzen, aber auch stellen!

4.5 Dekorationen

Dekorationen sind für das Spiel unwichtige, aber für die Atmosphäre wichtige Objekte. Zum Beispiel haben wir in unserer Beschreibung für **`Vor_dem_Haus`** einige Blumen erwähnt, die dem Spieler einfach nur ein Gespür für den Ort geben sollen. Vielleicht kommt der Spieler ja auf die Idee, diese Blumen zu untersuchen. Dann bekommt er aber „Ich kenne ‚Blumen' nicht." als Antwort.

Um den Spieler an dieser Stelle nicht zu enttäuschen, fügen wir eine Dekoration **`Blumen`** hinzu. Die Definition ist dieselbe wie bei einem Objekt, nur benutzt man statt **`Obj`** das Schlüsselwort **`Deko`**.

```
Deko    Blumen
Name    'wild^ Blumen' p
Adj     'wild' 'bunt' 'farbenfroh'
Subst   'blumen' p  'blume' f  'mohn' n
Subst   'margeriten' p
Ort     Vor_dem_Haus
Besch   'Es ist eine wilde Ansammlung farbenfroher
        Blumen, hauptsächlich Mohn und Margeriten.
        Obwohl sie wild gewachsen ist, sieht es
        fast aus wie ein richtiger Garten.'
```

Eine Deko kann sich während des Spiels nicht verändern. Sie kann Attribute und einen Zustand haben, aber es werden immer dieselben wie am Anfang sein. Dekos können keine Behälter oder Ablagen sein. Außerdem sind sie immer fest und werden nie extra erwähnt. Sie

sind nur da, um unwichtigen Objekten etwas mehr „Leben“ einzuhauchen. Viel mehr als sie anschauen kann man mit ihnen nicht machen.

Es gibt eine Besonderheit bei Dekos: Sie können in bis zu vier Räumen sein. Dazu gibt man einfach mehere Räume in der **Ort**-Zeile an. Eine Deko kann allerdings niemals in oder auf einem anderen Objekt sein.

Für den Trampelpfad definieren wir also:

```
Deko    Trampelpfad
Name    'Trampelpfad' m
vor     'trampel' 'fuß'
subst   'pfad' m  'weg' m
Ort     Lichtung   vor_dem_Haus
Besch   'Der Pfad ist eine staubige Spur im Gras,
        mehr nicht.'
```

Aufgabe 10

Definiere die Deko Wald/Bäume, um unsere Spielwelt ein wenig aufzuwerten. Und für die beiden Innenräume die Deko Wand.

× Es gibt noch eine weitere Möglichkeit, den Ort von Dekos zu beschreiben, nämlich per Raumattribut. Wenn man nach den Wort "Ort" ein gültiges Raumattribut angibt, so ist eine Deko in diesem Raum, wenn der Raum dieses Attribut besitzt. Stellt man dem Attribut einen Schrägstrich voran, so ist es umgekehrt: Die Deko ist im Raum, wenn der Raum das Attribut nicht hat.

Mit den Attributen, die wir jetzt kennen, macht es keinen Sinn, aber wenn wir z.B. ein Raumattribut aussen definieren würden, könnten wir folgende Definitionen benutzen:

```
Deko    Sonne
Ort     aussen
...

Deko    Zimmerdecke
Ort     /aussen
...
```

Zusammenfassung:

- Für bestimmte Arten von Objekte gibt es in T.A.G. bereits vordefinierte Attribute, wie z.B. **Behälter**, **Ablage**, **Sitz**, **Liege**, **Standfläche**.

- Um festzustellen, wieviel der Spieler bei sich tragen kann, gibt es drei verschiedene Konzepte: Die Anzahl, das Gewicht und das Volumen (Form) der Objekte

- Dekorationen sind Objekte, die sich während des Spiels nicht verändern. Sie dienen dazu, um in der Raumbeschreibung erwähnten Dingen ein wenig Atmosphäre zu verleihen.

Verweise:

- Handbuch, Kapitel 3.2: Objekte
- Handbuch, Kapitel 10: Verschiedene Arten von Objekten

5 Anweisungen

5.1 Ein ordentlicher Anfang

Bis jetzt wird unser Spieler ohne Kommentar ins Geschehen geworfen: Er bekommt die erste Raumbeschreibung um die Ohren geknallt, und los gehts. Das ist nicht die feine englische Art, deswegen wollen wir ihm einen ordentlichen Einstieg ins Spiel geben.

Dazu definieren wir eine *Aktion*. Eine Aktion ist eine Anzahl von Anweisungen, die nacheinander ausgeführt werden. Bis jetzt heben wir unsere Spielwelt nur mit den standardmäßig in T.A.G. vorhandenen Definionen erschaffen. Das wird sich in diesem Kapitel ändern, wenn wir unsere eigenen Regeln schreiben!

Aber zurück zum Anfang des Adventures. Es gibt eine Aktion Anfang, die zu Beginn des Spiels aufgerufen wird. (Sie wird auch aufgerufen, wenn der Spieler einen Neustart mit „Neu“ wünscht.) Diese Aktion definieren wir:

```
Aktion  Anfang
Ausf
  Text  '[f]D E R   T E S T[n]
        [x]Ein interaktives Versuchsgelände,
        (C) 2000 Cecilia McIntyre

        Du bist hier offensichtlich in eine
        ziemlich eigenartige Welt geraten. Am
        besten, du schaust dich einmal um und
        erkundest, was es mit dieser Gegend auf
        sich hat.'

  Sei maxInv 5   ! Ich kann 5 Teile tragen ...
  Sei maxVol 80  ! ... bis zu einem Volumen von 80

  geheZu vor_dem_Haus
EndeAusf
```

In dieser Aktion tauchen viele neue Elemente auf. Zunächst sehen wir, dass die Aktion mit **`Ausf`** beginnt und mit **`EndeAusf`** endet. Alles dazwischen ist bis jetzt Neuland.

Die erste Anweisung lautet **`Text`**, gefolgt von einem über mehrere Zeilen gehenden Text in einfachen Anführungszeichen. Diese Texte kennen wir ja schon, und die Anweisung **`Text`** heißt einfach nur „gib diesen Text formatiert auf dem Bildschirm aus.“ – wer hätte das gedacht?

Innerhalb des Textes tauchen wieder die kryptischen Zeichen innerhalb der eckigen Klammern auf, die wir von der Liste der Gegenstände in einem anderen Objekt bereits kennen. Dies sind sogenennte Textbefehle. Hier wird folgendes verwendet:

[f]	Fettdruck ein
[n]	Normaldruck ein, d.h. Fettdruck aus
[x]	Unformatierter Zeilennumbruch, d.h. eine neue Zeile.

Ein weiterer Textbefehl ist, auch wenn man das nicht gleich auf den ersten Blick sieht, die Leerzeile, die, wie wir wissen, einen Absatz bedeutet. Wenn nichts anderes definiert ist, hat der Absatz in T.A.M. eine Leerzeile ohne nachfolgende Einrückung. Eine andere Möglichkeit, einen Absatz zu schreiben, wäre **[#]** im Fließtext.

× Da die Texte zwischen Apostrophen stehen, muss ein Apostroph umschrieben werden. dazu kann man das Prozentzeichen **%** benutzen oder eine der Notationen **[,]** oder **["]**. Um ein Prozentzeichen zu schreiben, benutzt man **[%]**. Um eckige Klammern zu schreiben, müssen diese doppelt angegeben werden: **[[** oder **]]**.

Die Textbefehle werden intern als Sequenz von Zeichen, die mit einem Schrägstrich beginnen, abgelegt. Daher muss auch der Schrägstrich immer doppelt stehen: **//**.

Die nächsten Anweisungen heißen **sei**. Hier wird der Variablen maxInv der Wert 5 und der Variablen **maxVol** der Wert 80 zugewiesen. In vielen Programmiersprachen sehen solche Anweisungen so (oder so ähnlich) aus: **maxInv = 5**, aber T.A.G. geht hier einen eher rudimentären Weg.

Alles, was hinter dem Ausrufezeichen steht wird von T.A.G. übrigens nicht beachtet. Man nennt das einen Kommentar, und es kann nützlich sein, ab und zu einmal ein paar zusätzliche Bemerkungen einzufügen, damit man später in seinem eigenen Adventure noch durchblickt.

Die letzte Anweisung lautet **geheZu** und bewirkt, dass der Spieler zu einem bestimmten Raum bewegt wird. Danach wird die Raumbeschreibung ausgegeben.

Das hätten wir nicht unbedingt machen müssen, denn wenn der Spieler am Anfang noch nirgendwo ist, wird er automatisch in den zuerst definierten Raum gestellt und es wird ebenfalls die Raumbeschreibung ausgegeben. Aber es ist „sauberer“ so, und außerdem kann man die Zeile später schnell abändern, wenn der Spieler lieber doch im Wald beginnen soll.

5.2 Ins Geschehen eingreifen

Bis jetzt haben wir nur die bereits in T.A.G. vorhandenen Definitionen verwendet und konnten damit eine Spielwelt mit bestimmten Regeln schaffen. Ein eigenes Adventure benötigt aber zu diesen allgemein gültigen Regeln noch weitere, die der Autor selbst definieren kann. Das wollen wir jetzt machen.

Zum Beispiel könnte der Spieler auf die Idee kommen, die Taschenlampe zu öffnen, um an die Batterien zu kommen. Die Antwort darauf wäre „Du kannst doch keine Taschenlampe öffnen.“, was nicht besonders toll ist. Wir wollen dem Spieler stattdessen etwas anderes sagen, dazu erweitern wir die Definition der Taschenlampe:

```
Obj     Lampe
```

```
...
VorAusf
    (öffnen)
        Text 'Wozu? Die Batterien tun es noch,
            und es gibt nichts, wozu man sie
            sonst gebrauchen könnte.'
        Stop
EndeAusf
```

Die zusätzliche Definition ist wieder ein Ausführungsblock, er wird wie eine Aktion mit **EndeAusf** abgeschlossen, beginnt aber mit **VorAusf**. Eine **VorAusf** kann zu einem Objekt definiert werden, **VorAusf** bedeutet dabei (etwas holprig): „Führe diesen Block aus, bevor du den eigentlichen Befehl abarbeitest."

Das erste Element im Block ist ein Befehlsname, der in runden Klammern steht. Solche Angaben in runden Klammern bedeuten: Wenn der vom Spieler eingegebene Befehl **öffnen** ist, dann führe alles aus, was jetzt kommt, bis du einen anderen Befehl in runden Klammern oder **EndeAusf** erreichst.

Dann folgen zwei Anweisungen. Die **Text**-Anweisung kennen wir schon, sie gibt den folgenden Text aus. Die nächste Anweisung **Stop** bewirkt, dass der ganze Befehl **öffnen** sofort abgebrochen wird. Sagt der Spieler also „öffne Lampe", dann wird der Text mit den Batterien ausgegeben, und das war's. Die normale Prozedur zum Öffnen von Gegenständen wird nicht mehr aufgerufen, da vorher mit dem **Stop**-Befehl eingegriffen wurde. Würde das **Stop** vor dem **Text** stehen, so würde nicht einmal der Text angezeigt.

Bei „öffne Rucksack" oder einem anderen Objekt passiert natürlich nichts, denn da die **VorAusf** bei der Lampe definiert wurde, gilt sie auch nur für die Lampe.

× Da diese Kombination von **Text** und **Stop** häufig vorkommt, kann man sie abkürzen zu **Stop 'Wozu? ...'**. Man kann also den Ausgabetext direkt hinter der Anweisung **Stop** angeben.

Anstatt die Durchführung eines Befehls vorher abzufangen, kann man ihn auch nachträglich ergänzen. Die Definition hierfür heißt **NachAusf**:

```
Obj     Zettel
...
NachAusf
  (nehmen)
      Wenn /(Zettel bewegt)
          Text 'Der Zettel fühlt sich feucht an,
              als Du ihn aus dem taubenetzten Gras
              fischst.'
EndeAusf
```

Hier ist der Befehl „nimm Zettel" bereits erfolgreich ausgeführt worden, es wurde gecheckt, dass der Spieler den Zettel nicht schon hat, weitere Gegenstände aufnehmen kann, usw. Der Zettel befindet sich bereits beim Spieler. Nun wird nachträglich ein anderer Text ausgegeben. Dabei sollte man beachten, dass der erste Text in der **NachAusf** den Standard-Text bei erfolgreichem Ausführen ersetzt, alle weiteren Texte werden wie gehabt nacheinander ausgegeben.

Die Ausgabe des Textes ist hier allerdings an eine Bedingung geknüpft. Eine Bedingung in T.A.G. ist ein Ausdruck in runden Klammern. Die Bedingung heißt hier **(Zettel bewegt)**. **Bewegt** ist ein Attribut, das gesetzt wird, sobald der Spieler den Gegenstand aufhebt. Dieses Attribut wird auch benutzt, um festzustellen, ob die **Erst**-Beschreibung ausgegeben wird oder nicht.

Der Bedingung geht ein Schrägstrich voran. Schrägstriche bedeuten in T.A.G. Verneinungen. Die Bedingung ist also erfüllt, wenn der Zettel noch *nicht* bewegt wurde. Das Wort **Wenn** vor der Bedingung heißt: „Wenn diese Bedingung erfüllt ist, dann führe den nächsten Befehl aus, ansonsten überspringe ihn.“ Der Text wird also nur ausgegeben, wenn der Zettel zum ersten Mal aufgehoben wird. Danach besitzt er das Attribut **bewegt**, und die **Text**-Anweisung wird übersprungen.

Räume können übrigens auch eine **Vor**- und **NachAusf** haben. Sie wird immer dann aufgerufen, wenn sich der Spieler in diesem Raum befindet:

```
Raum    Im_Haus
...
NachAusf
    (gehen)
        Wenn /(aRaum besucht)
        Text '[#]Hier ist es irgendwie unheimlich.
            Du bist wahrscheinlich seit Jahren
            der Erste, der dieses Haus betritt.'
EndeAusf
```

Damit wird dem Spieler beim ersten Betreten des Hauses gesagt, dass es unheimlich hier ist. **aRaum** ist eine Variable, die den momentanen Aufenthaltsort beschreibt. Da dies in diesem Fall immer **Im_Haus** ist, könnte man auch schreiben **/(im_Haus besucht)**.

5.3 Bedingungen und Steuerstrukturen

Bedingungen spielen gerade bei Adventures eine wichtige Rolle: Der Geheimgang ist nur sichtbar, wenn Hebel A gezogen ist, Knopf B zweimal gedrückt wurde und der Spieler den Gegenstand C bei sich hat. So oder so ähnlich sehen viele Rätsel aus.

Die Bedingungen sind relativ selbsterklärend. Die wichtigsten sind:

(⟨*Objekt/Raum*⟩ ⟨*Attribut*⟩**)**
ist wahr, wenn das Objekt oder der Raum ein Attribut besitzt.

(⟨*Objekt*⟩ **in** ⟨*Ort*⟩**)**
ist wahr, wenn ein Objekt an einem bestimmten Ort ist. Der Ort ist dabei, wie mit der Ort-Zeile in der Objektdefinition angegeben. Wenn Ort ein Raum ist, wird **in** vorangestellt.

(⟨*Objekt*⟩ ⟨*Zustand*⟩**)**
ist wahr, wenn ein Objekt den Zustand hat.

(Proz ⟨*Zahl*⟩**)**
ist in Zahl Prozent aller Fälle wahr. Hier wird zufällig eine Zahl zwischen 0 und 100 bestimmt, wenn diese kleiner ist als Zahl, so ist die Bedingung wahr. (Proz 50) ist eine fifty-fifty-Chance.

(Licht_In ⟨*Raum*⟩**)**
ist wahr, wenn man im Raum sehen kann. Dies ist genauer, als **/(Raum dunkel)**, da Lichtquellen mit berücksichtigt werden.

All diese Bedingungen können mit einem Schrägstrich verneint werden.

Diese Bedingungen werden in sogenannten Steuerstrukturen benutzt. Eine haben wir schon kennengelernt:

Wenn (⟨*Bedingung*⟩**)** ⟨*Anweisung*⟩

Wenn (⟨*Bedingung*⟩**)**
⟨*Anweisung*⟩

Diese beiden Formen sind gleich und meinen: Wenn die Bedingung wahr ist, führe die Anweisung aus. Hier kann nur eine Anweisung stehen. Oft möchte man mehrere Anweisungen von einer Bedingung abhängig machen. Dann benutzt man

Wenn (⟨*Bedingung*⟩**) dann**
⟨*beliebig viele Anweisungen*⟩
Ende

Dies funktioniert genau wie die einfache Wenn-Bedingung, alles zwischen **dann** und **Ende** wird nur ausgeführt, wenn die Bedingung wahr ist.

Zu diesem Konstrukt gibt es eine Erweiterung. Manchmal möchte man eine Fallunterscheidung machen: Wenn das gilt, mache das, wenn nicht, etwas anderes. Dazu kann man die Wenn-dann-Bedingung mit **sonst** erweitern:

Wenn (⟨*Bedingung*⟩**) dann**
⟨*Anweisungen für erfüllte Bedingung*⟩
sonst
⟨*Anweisungen für nicht erfüllte Bedingung*⟩
Ende

Eine Sache, die in Adventures sehr häufig vorkommt, sind Bedingungen, die zum sofortigen Abbruch des Befehls führen, wenn sie nicht erfüllt sind. Zum Beispiel checkt **anziehen**, ob das Objekt ein Kleidungsstück ist, **nehmen**, ob man es mitmehmen kann, **liegen**, ob es eine Liege ist usw. Ist dies nicht der Fall, bekommt der Spieler einen Satz auf den Bildschirm, und er darf einen neuen Befehl eingeben. Dies würde im Moment so aussehen:

```
Wenn /(⟨Bedingung⟩) dann
    Text '⟨Text⟩'
    Stop
Ende
```

oder kürzer

```
Wenn /(⟨Bedingung⟩) Stop '⟨Text⟩'
```

Eine Alternative hierzu ist:

```
Bed (⟨Bedingung⟩) '⟨Text⟩'
```

Man muss nur aufpassen, dass hier die Bedingung nicht verneint ist. Die **Bed**-Bedingung ist die Bedingung zum Weitermachen, die **Wenn**-Bedingung die zum Abbrechen. Welche Schreibweise man benutzt, ist aber letztendlich egal.

Aufgabe 11 *

Erzeuge einen Pilz, der im Wald steht. Wenn man ihn zum ersten Mal aufhebt, soll ein Text ausgegeben werden, der sagt, wie er mit einem leisen „Plopp“ aus der Erde gezogen wird. Wenn man ihn isst, soll man in 20 Prozent aller Fälle vergiftet werden. (Die Anweisung, um den Spieler sterben zu lassen, heißt **gestorben**. Konsequenterweise wird damit auch der momentane Befehl abgebrochen.)

Mann kann mehrere Bedingungen verknüpfen. Dazu gibt es die Operatoren **und** und **oder**:

(⟨*Bedingung a*⟩**) und (**⟨*Bedingung b*⟩**)**
ist wahr, wenn beide Bedingungen erfüllt sind.

(⟨*Bedingung a*⟩**) oder (**⟨*Bedingung b*⟩**)**
ist wahr, wenn mindestens eine Bedingung erfüllt ist.

× Man kann in T.A.G. keine Bedingungen ineinander schachteln. Man kann auch immer nur einen Ausdruck verneinen. Folgende Bedingungen muss man also umschreiben:

/(a **und** b**)** → **/**a **oder /**b

/(a **oder** b**)** → **/**a **und /**b

5.4 Objekte manipulieren

Bis jetzt haben wir nie Sachen wirklich verändert, sondern immer nur dem Spieler gesagt, dass etwas nicht geht. Nun wollen wir anfangen, die Objekte wirklich zu manipulieren.

Wir definieren nun einen Lichtschalter im Keller:

```
Obj     Lichtschalter
Name    'Lichtschalter an der Wand' m
vor     'licht'
subst   'schalter' m
Ort     Keller
Attr    immob
Besch   'Das Symbol einer Glühbirne ziert den
        Schalter.'
Erst    'An einer Wand ist auf etwa ein Meter
        Höhe ein Schalter.'
VorAusf
    (drücken)
         Wenn (Keller dunkel) dann
             Text 'Der Raum wird von einem
                 eigenartigen, kalten Licht
                 durchflutet, das von den Wänden
                 herzukommen scheint.'
             AttrWeg Keller dunkel
         sonst
             Text 'Der Keller ist jetzt wieder so
                 stockfinster wie zuvor.'
           AttrHin Keller dunkel
         Ende
EndeAusf
```

Die Anweisungen **AttrHin** und **AttrWeg** geben dem Raum ein Attribut oder nehmen es ihm weg. Diese Anweisungen funktionieren auch bei Objekten:

```
ObjAttr gelesen

Obj     Zettel
...
VorAusf
    (lesen)
        Bed /(Zettel gelesen)
            'Das hast Du doch schon gelesen.'
EndeAusf

NachAusf
    (lesen)
        AttrHin Zettel gelesen
EndeAusf
```

Mit **ObjAttr** kann man eigene Attribute für Objekte definieren. Es gibt auch ein **RaumAttr** für zusätzliche Raumattribute. Mit dieser neuen Definition wird verhindert, dass der Spieler den Zettel zweimal liest. Zugegeben, kein realitätsnahes Beispiel, aber es zeigt, wie es geht: Beim ersten Mal ist der Zettel noch nicht gelesen. Der Text wird wie üblich angezeigt. Im Nachhinein bekommt der Zettel aber das Attribut **gelesen**. Beim nächsten Mal bricht dann die **VorAusf** den Befehl ab.

Aufgabe 12

Ändere den Pilz so ab, dass er nicht beim ersten Probieren tödlich ist, sondern erst beim zweiten Mal. Dann aber bestimmt. (Dazu sollte sichergestellt sein, dass der Spieler beim ersten Mal nur knabbert, damit der Pilz nicht verschwindet.)

Ein anderes Beispiel: Unser Rucksack soll zuerst geschlossen werden, bevor er aufgesetzt wird:

```
Obj     Rucksack
...
VorAusf
    (anziehen)
        Wenn (Rucksack offen) dann
            Text '(Du schließt den Rucksack erst,
                 bevor du ihn aufsetzt.)[#]'
            ObjZust Rucksack geschlossen
        Ende
EndeAusf
```

Hier wird dem Spieler mitgeteilt, dass er den Rucksack implizit zumacht, bevor er den Rucksack aufsetzt. Es ist eine Konvention, dies dem Spieler in runden Klammern mitzuteilen.

Die Anweisung **`ObjZust`** weist dem genannten Objekt dann einen neuen Zustand zu. Es gibt übrigens den Zustand **`normal`**, der bedeutet: kein besonderer Zustand.

Aufgabe 13 *

Ändere den Pilz, so dass er beim ersten Mal nur genommen werden kann, wenn der Spieler das Taschenmesser bei sich hat und dieses offen ist. Natürlich muss eine Meldung kommen, dass der Pilz abgeschnitten wurde.

Aufgabe 14 *

Definiere einen Blechkasten, der nicht auf herkömmliche Weise geöffnet und geschlossen werden kann. Stattdessen besitzt dieser Kasten einen Knopf, der ihn öffnet und schließt.

Wir haben nun den Zustand und die Attribute der Objekte verändert, eine Sache fehlt noch: der Ort. Hierzu heißt die Anweisung **`ObjNach`** ⟨*Objekt*⟩ ⟨*Ort*⟩:

```
Obj     Lichtschalter
...
VorAusf
    (drücken)
        Wenn (Keller dunkel) dann
            Text  'Der Raum wird von einem
                eigenartigen, kalten Licht
                durchflutet, das von den Wänden
                herzukommen scheint.'
            AttrWeg Keller dunkel
            Wenn (Ring in nirgendwo) dann
                Text '[#]Durch das gleichmäßige Licht
                    entdeckst du in einer Ecke des
                    Kellers einen Ring, den du vorher
                    mit deiner Funzel nie entdeckt
                    hättest.'
                ObjNach Ring Keller
            Ende
        sonst
            Text 'Der Keller ist jetzt wieder so
                stockfinster wie zuvor.'
```

```
            AttrHin Keller dunkel
        Ende
        Stop
EndeAusf

Obj     Ring
Name    'Ring' m
subst   'ring' m
Erst    'In einer Ecke des Kellers liegt ein Ring.'
Attr    Kleidung
```

Hier wird ein Ring in den Keller gelegt, den der Spieler entdeckt, wenn er den Lichtschalter drückt. Die Angabe des Ortes erfolgt wie in der **`Ort`**-Zeile der Objekt-Definition.

`Nirgendwo` ist ein Ort, den der Spieler nie betreten kann. Dort werden Gegenstände aufbewahrt, die zu Beginn des Spiels noch nicht verfügbar sind. Ein Gegenstand ist automatisch im **`nirgendwo`**, wenn die **`Ort`**-Zeile weggelassen wird.

× Es gibt einen zweiten Raum, den der Spieler nicht betreten kann. Er heißt **`Nirwana`** und dort kommen alle Gegenstände hin, die aus dem Spiel verschwinden.

Aufgabe 15 *

Definiere einen Wäscheschacht im Haus, so dass alles, was dort hineingelegt wird, automatisch in den Keller rutscht. Dazu muss die Pseudoaktion **`empfangen`** abgefangen werden. (Tip: Der Schacht muss nur **`im_Haus`** definiert sein.)

Aufgabe 16

Definiere einen Weidenkorb im Keller, den der Spieler nicht mitnehmen kann, und in den die Wäsche (oder was auch immer) aus dem Wäscheschacht rutscht.

Eine weitere Art, Objekte zu verändern, ist einfach ihren Ort zu vertauschen. Das ist besonders wirkungsvoll, wenn eines der Objekte im Nirgendwo ist. Ohne dass es der Spieler mitbekommt, bezieht er sich auf ein anderes Objekt. Die Anweisung hierfür lautet: **`Tausche`** ⟨*Objekt1*⟩ ⟨*Objekt2*⟩.

Aufgabe 17

Definiere eine Ming-Vase, die zu einem Scherbenhaufen wird, wenn der Spieler sie wirft oder zerstört.

Zusammenfassung:

- *Aktionen* bestehen aus einer Anzahl von Anweisungen, die nacheinander abgearbeitet werden. Die Definition dieser Anweisungen erfolgt in einem Ausführungsblock, der mit **`Ausf`** beginnt und mit **`EndeAusf`** abgeschlossen wird.

- Die Aktion **`Anfang`** wird zu Beginn des Spiels durchgeführt.

- Zu Objekten können die Ausführungsblöcke **`VorAusf`** und **`NachAusf`** definiert werden, um in den Ablauf einer Befehlsausführung einzugreifen.

• Bedingungen sind Ausdrücke in runden Klammern, die bestimmte Sachverhalte in der Spielwelt überprüfen. Sie werden oft in Steuerstrukturen wie **wenn**/**dann** benutzt.

• Mit den Anweisungen **ObjNach**, **ObjZust**, **AttrHin** und **AttrWeg** können Objekte außerhalb der in T.A.G. bereits vorhandenen Regeln manipuliert werden.

Verweise

• Handbuch, Kapitel 5: Programmieren der Ausführungsblöcke
• Handbuch, Kapitel 6: Die verschiedenen Aktionen im Spiel

6 Variablen

6.1 Vorhandene Variablen

Variablen sind Merker für Werte. In T.A.G. können Variablen auch Merker für Objekte, Räume und Befehle sein. Einige Variablen haben wir schon kennengelernt, nämlich die Inventar-Daten und **aRaum**. Es gibt aber noch weitere Variablen, die bereits vordefiniert sind.

Die wichtigsten sind wahrscheinlich die Objektvariablen. Die Variablen **aObj** und **aObj2** stehen für die Objekte, die im Befehl des Spielers erwähnt werden. Manche Befehle verlangen kein Objekt, und oft ist **aObj2** Null, d.h. nicht vorhanden. (Es gibt auch noch ein **aObj3**, aber es wird so gut wie nie verwendet.)

Die wichtigste Raumvariable ist **aRaum**, die immer angibt, wo sich der Spieler befindet. Ihr sollte nie explizit ein Wert zugwiesen werden, sondern sie sollte mit **GeheZu** automatisch angepasst werden.

Die meisten Variablen stehen nur für Zahlen, in T.A.G. können dies die Zahlen von 0 bis 255 sein. Diese Variablen heißen in T.A.G. (nicht ganz korrekt) *Flaggen*. Eine Flagge kann normalerweise nur zwei Zustände annehmen, nämlich gesetzt und nach gesetzt. In T.A.G. gilt eine Flagge als gesetzt, wenn ihr Wert größer als Null ist. Bereits benutzte Flaggen sind die bereits erwähnten **maxInv**, **maxGew**, **maxVol**, **Inv**, **InvGew** und **InvVol**.

× Zwei weitere Flaggen, **Minuten** und **Stunden**, geben die Zeit im Spiel an. Jeder Zug dauert eine Minute, und es wird still mitgezählt. Die meisten Adventures benutzen keine Zeitangeben, dies wird aber manchmal in Krimi-Adventures verwendet.

Manchmal muss man negative Zahlen oder Zahlen, die größer sind als 255, speichern. Dazu gibt es ein besonderes Zahlenformat, *Integer*. Integer-Zahlen können Werte von –2 Mrd. bis 2 Mrd. annehmen. Bereits definierte Integerwerte sind **Züge**, in dem angegeben wird, wieviele Züge der Spieler bereits gemacht hat, und **Pktzahl**, ein Integer, der nur gelesen werden kann, und der die Gesamtpunktzahl des Spielers enthält (s. Kap. 10.2).

× Ein weiterer Integer, **Nummer**, wird verwendet, wenn in der Angabe des Spielersanstelle eines Objekts eine Zahl angegeben wird, wie z.B. in „Wähle 112 auf Telefon“. **aObj** wäre in diesem Fall ein besonderes Objekt mit dem Namen **Zahl**.

6.2 Variablen definieren und ändern

Man kann sich natürlich weitere Variablen definieren. Dazu muss außerhalb eines Definitionsblocks eine der Anweisungen

ObjVar ⟨*ID*⟩ ⟨*Anfangswert*⟩
RaumVar ⟨*ID*⟩ ⟨*Anfangswert*⟩
Flagge ⟨*ID*⟩ ⟨*Anfangswert*⟩
Integer ⟨*ID*⟩ ⟨*Anfangswert*⟩

stehen. Die ID muss natürlich, wie immer, eindeutig sein. Der Anfangswert kann weggelassen werden, dann ist er automatisch Null (bzw. kein Raum oder kein Objekt, was auch mit Null gekennzeichnet wird).

Wie der Name schon sagt, sind Variablen genau das, nämlich variabel. Man kann eine Variable z.B. auf einen bestimmten Wert ändern, indem man in einem Ausführungsblock die Anweisung

sei ⟨*Variable*⟩ ⟨*Wert*⟩

benutzt. Wert kann hier ebenfalls eine Variable sein, dann haben die Variablen den selben Wert, nämlich den der zweiten Variable. Man muss nur aufpassen, dass man hier nicht Äpfel mit Birnen verwechselt: Objektvariablen können nur Werte haben, die ein Objekt sind. Raumvariablen können nur Räume speichern, und Flaggen und Integer nur Zahlen. Alle Variablentypen können jedoch den Wert 0 annehmen.

Es gibt einige weitere Anweisungen, die eine Variable relativ zu ihrem jetzigen Wert ändern:

inkr ⟨*Variable*⟩ ⟨*Wert*⟩
Erhöhung um ⟨*Wert*⟩

dekr ⟨*Variable*⟩ ⟨*Wert*⟩
Erniedrigung um ⟨*Wert*⟩

mult ⟨*Variable*⟩ ⟨*Wert*⟩
Multiplikation mit ⟨*Wert*⟩

div ⟨*Variable*⟩ ⟨*Wert*⟩
Division durch ⟨*Wert*⟩

mod ⟨*Variable*⟩ ⟨*Wert*⟩
Rest der Division durch ⟨*Wert*⟩

Diese braucht man nicht so häufig, außer **inkr** und **dekr** vielleicht. Bei **inkr** und **dekr** kann man den Wert auch weglassen, er ist dann eins.

Wenn man Flaggen nur als Schalter benutzt, kann man auch folgende Abkürzungen benutzen:

setze ⟨*Flagge*⟩
gibt der Flagge den Wert 1

lösche ⟨*Flagge*⟩
gibt der Flagge den Wert 0

Natürlich kann man Variablen auch vergleichen. Dazu gibt es die Bedingung

(⟨*Variable*⟩ **=** ⟨*Wert*⟩**)**

die dann erfüllt ist, wenn der Wert der Variable und der abgegebene Wert gleich sind. Wert kann auch hier eine andere Variable sein, aber man muss wieder aufpassen, dass man nicht „Äpfel mit Birnen vergleicht“.

Für Zahlenwerte gibt es auch die beiden Bedingungen **(x < y)** und **(x > y)**, die prüfen, ob **x** kleiner oder größer als **y** ist. Diese Bedingung kann man auch für Objekte und Räume verwenden, auch wenn sie keinen großen Sinn macht: Sie vergleicht dann ob Objekt/Raum **x** vor Objekt/Raum **y** definiert wurde, was aber im allgemeinen egal ist.

Aufgabe 18 *
Erweitere die Vase so, dass man sie nirgendwo abstellen kann, außer auf dem Tisch.

T.A.G. bietet außerdem die Möglichkeit, Variablen „zufällig“ zu belegen. Um zum Beispiel den Wurf eines Würfels zu simulieren, kann man

```
Zufall Augen 1 6
```

benutzen, was die Variable **Augen** mit einem Wert von eins bis sechs inklusive belegt. Jeder der sechs möglichen Werte ist gleich wahrscheinlich. Eine Variante der **Zufall**-Anweisung ist, anstatt einen Start- und Endwert eine Liste der möglichen Werte in Klammern anzugeben:

```
Zufall xObj (Messer Gabel Schere)
```

Auch hier treten alle möglichen Werte mit der gleichen Wahrscheinlichkeit auf.

6.3 Sachverhalte merken mit Variablen

Dazu ein Anwendungsbeispiel: Wir erzeugen jetzt einen Safe, der ein Rädchen hat, das immer auf eine Nummer von 0 bis 99 gestellt ist:

```
Flagge  code 66

Obj     Safe
Name    'Safe' m
vor     'stahl' 'geld' 'panzer'
subst   'safe' m  'tresor' m  'schrank' m
Ort     Keller
Zust    abgeschlossen
Attr    immobil Behälter
Besch   'Der Safe ist sehr stabil und hat ein
        kleines Rädchen an der Tür, das momentan
```

```
        auf [code] gestellt ist.'
Erst    'Ein schwach metallisch schimmernder
        Panzerschrank steht in einer Ecke des
        Kellers.'

Obj     Rädchen
Name    'Rädchen am Safe' n
vor     'stell'
subst   'rad' n  'rädchen' n
Ort     an Safe
Attr    Fest
Besch   'Das Stellrädchen ist momentan auf
        [code] eingestellt.'
```

So, wir haben eine Variable **code** definiert, die die Position des Stellrädchens angibt. Im Moment ist sie immer 66. Das wird auch mit **[code]** ausgegeben, wenn man den Safe oder das Rädchen untersucht. Das soll sich ändern:

```
Obj     Rädchen
...
VorAusf
    (drehen)
        Bed (Safe abgeschlossen)
            'Wieso? Der Safe ist bereits auf.'
        Zufall code 0 99
        Stop 'Du drehst das Rädchen auf [code].'
EndeAusf

Aktion *
Ausf
  Wenn (code = 23) und (Safe abgeschlossen) dann
      Text '[#]Vom Safe hörst du ein leises
          "Klick!"'
    ObjZust Safe geschlossen
  Ende
EndeAusf
```

So, nun gibt es eine Möglichkeit, das Rad zu verdrehen. Allerdings nur zufällig. Immerhin wird jetzt mit **[code]** der momentan eingestellte Code ausgegeben. Diese Eigenschaft des Rädchens merken wir uns mit der Flagge **code**.

In einer Aktion, die nach jedem Zug ausgeführt wird, wird nun gecheckt, ob der Code 23 ist und der Safe noch zu ist. Dann nämlich soll der Safe sich mit einem „Klick“ entriegeln. Das hätte man natürlich auch direkt in die **VorAusf** packen können. Aber im nächsten Kapitel wollen wir noch eine andere Möglichkeit, das Rädchen zu verstellen, vorstellen, da ist diese Vorgehensweise wahrscheinlich besser.

Übrigens: Aktionen, die mit einem Sternchen definiert werden, d.h., die in jedem Zug ausgeführt werden, benötigen keine ID.

6.4 Zeitabhängige Eregnisse

Man kann Variablen auch dazu benutzen, um zeitabhängige Ereignisse zu beschreiben. Nehmen wir dazu einmal an, unsere Taschenlampe hätte nur eine Brenndauer von 100 Zügen. Diese Brenndauer wird in der Variable **Brenndauer** gespeichert. Also:

```
Obj     Lampe
...
VorAusf
    (anmachen)
    Bed (Brenndauer > 0)
        'Du schaltest die Lampe ein, aber die
        Batterien sind leer, und so machst du sie
        wieder aus.'
    (öffnen)
    Stop 'Wozu? Die Batterien tun es noch, und es
        gibt nichts, wozu man sie sonst gebrauchen
        könnte.'
EndeAusf
```

Damit haben wir verhindert, dass der Spieler die Lampe einschaltet, wenn die Batterien leer sind. Nun müssen wir nur noch dafür sorgen, dass die Lampe in jedem Zug ihre Batterie etwas entlädt. Dazu kann man eine Aktion mit einem Sternchen definieren. Das Sternchen heißt „Führe diese Aktion nach jedem Zug aus, und zwar unabhängig davon, ob der Befehl ausgeführt wurde“. Das heißt, wenn der Spieler versucht, das Haus zu nehmen oder den Tisch zu essen, verliert die Lampe ebenfalls Leuchtkraft.

Also definieren wir die Aktion:

```
Aktion  Batterieverbrauch *
Ausf
    Wenn (Lampe ein) dann
        Dekr Brenndauer
        wenn (Brenndauer = 1 2 3 5 10)
            Text 'Die Lampe wird zunehmend
                 schwächer.'
        wenn (Brenndauer = 0) dann
            Text 'Mit einem letzen Flackern gibt
                 die Taschenlampe endgültig ihren
                 Geist auf.'
            ObjZust Lampe aus
        Ende
    Ende
EndeAusf
```

× Man kann mehrere Bedingungen desselben Typs zu einer abkürzen. So bedeutet **(x = 1 3 5)** dasselbe wie **(x = 1)** oder **(x = 3)** oder **(x = 5)**.

Bei verneinten Bedinungen wird mit **und** verknüpft: **/(aObj fest immobil)** heißt **/(aObj fest) und /(aObj immobil)**.

6.5 Textbefehle

Nun wird unser Adventure immer variabler, und das bringt es mit sich, dass die Ausgabe der Texte darauf angepasst wird. Wir haben die sogenannten Textbefehle einmal ganz kurz kennengelernt. Nun wollen wir sie uns einmal genauer anschauen.

Textbefehle sind Sequenzen, die innerhalb eines Ausgabetextes in eckigen Klammern stehen. **[x]**, **[#]**, **[f]** und **[n]** kennen wir bereits.

Am häufigsten verwendet werden wohl die Befehle zur Ausgabe von Objekten:

> **[der** ⟨*Objekt*⟩**]**
> **[den** ⟨*Objekt*⟩**]**
> **[dem** ⟨*Objekt*⟩**]**
> Ausgabe mit bestimmtem Artikel. Der Fall wird durch **der**, **den** und **dem** bereits festgelegt. Das ist auch der Grund, warum es hier „der“ und nicht „das“ Objekt heißt: nur für das Maskulinum sind die drei Artikel verschieden.
>
> **[ein** ⟨*Objekt*⟩**]**
> **[einen** ⟨*Objekt*⟩**]**
> **[einem** ⟨*Objekt*⟩**]**
> Ausgabe mit unbestimmtem Artikel. Namen im Plural haben keinen unbestimmten Artikel.
>
> **[kein** ⟨*Objekt*⟩**]**
> **[keinen** ⟨*Objekt*⟩**]**
> **[keinem** ⟨*Objekt*⟩**]**
> Ausgabe als Verneinung.
>
> **[**⟨*Objekt*⟩**]**
> Ausgabe des Objekts im Nominativ und ohne Artikel.

× T.A.G. verzichtet auf den Genitiv, da er komplizierte Regeln zur Bildung benötigt, und gibt sich mit der korrekten Ausgabe von Dativ und Akkusativ zufrieden, die sowieso häufiger vorkommen.

Damit kann man nun Sätze ausgeben, wie

```
Text '[Der aObj] verschwindet hinter [dem aObj2].'
```

und T.A.M. setzt dann die passenden Objekte ein:

> „Der schokoladenbraune Teddybär verschwindet hinter der Bettdecke.“

Etwas unschön ist es allerdings, wenn bei solchen allgemeinen Antworten Objekte eingesetzt werden, die im Plural stehen, und die im Nominativ („der“) ausgegeben werden sollen:

> „Die sechs Murmeln verschwindet hinter der Mauer.“

So etwas hätte unser Deutschlehrer gar nicht gern gesehen, deshalb gibt es auch Textbefehle zur Ausgabe von Verben:

[ist ⟨*Objekt*⟩**]**
ist „ist“, wenn Objekt im Singular ist, ansonsten „sind“.

[hat ⟨*Objekt*⟩**]**
gibt „hat“ oder „haben“ aus.

[wird ⟨*Objekt*⟩**]**
gibt „wird“ oder „werden“ aus.

[t ⟨*Objekt*⟩**]**
gibt „(e)t“ oder „en“ aus. Dies ist die Endung für schwache Verben.

[⟨*xxx*⟩**/**⟨*yyy*⟩ ⟨*Objekt*⟩**]**
Gibt ⟨*xxx*⟩ im Singular und ⟨*yyy*⟩ im Plural aus. Dies kann für unregelmäßige Verben benutzt werden: **[gibt/geben aObj]**.

Nun ist es etwas unschön, auch bei der Endung das Objekt angeben zu müssen. Daher gilt als Vereinfachung: Wenn das Objekt fehlt, wird das aus dem vorherigen Befehl eingesetzt.

Damit ändern wir den Text

```
Text '[Der aObj] verschwinde[t] hinter [dem aObj2].'
```

und erhalten immer einen korrekt konjugierten Satz. Etwas Aufwand vielleicht, aber bestimmt die Sache wert. Außerdem habe ich versucht, die Befehle so zu halten, dass der Satz nicht zerrissen wird. Denkt man sich im Satz oben die Klammern weg, bleibt ein ganz normaler Satz über.

Weitere wichtige Textbefehle sind:

[⟨*Flagge*⟩**]** gibt den Zahlenwert der Flagge aus.

[num ⟨*Flagge*⟩**]** gibt den Zahlenwert der Flagge aus, aber benutzt Zahlwörter für die Zahlen bis hundert.

[⟨*Zeit*⟩**]** gibt die Uhrzeit im Spiel im Format hh:mm aus.

[⟨*Aktion*⟩**]** ruft die Aktion aus dem Text heraus auf.

Ist der erste Buchstabe in der Klammer groß, so ist auch der erste Buchstabe der Ausgabe des Textbefehls groß. **[Der aObj]** und **[der aObj]** sind also verschieden. Bei allen weiteren Buchstaben ist, wie allgemein bei T.A.G., die Groß- und Kleinschreibung egal.

× Es gibt eine weitere Art von Textbefehlen, die aus zwei Zeichen in eckigen Klammern bestehen, und die Sonderzeichen darstellen. **[ss]** ist das deutsche scharfe s, **[^e]** ein e mit Zirkumflex, **[a"]** ein ä, **[<<]** spitze Anführungszeichen links usw. Diese Textbefehle sind nur dann nützlich, wenn man die passenden

Zeichen nicht auf seiner Tastatur hat, denn anstatt **[a"]** kann selbstverständlich auch einfach „ä“ geschrieben werden. (Mehr dazu im Handbuch)

6.6 Schleifen

Eine weitere wichtige Steuerstruktur sind Schleifen. Dabei werden bestimmte Ausführungen öfters wiederholt. Wie oft, das wird im allgemeinen mit einer Bedingung festgelegt. Die einfachsten Schleifen sind:

solange (⟨*Bedingung*⟩**)**
    ⟨*Anweisungen*⟩
Ende

wiederhole
    ⟨*Anweisungen*⟩
bis (⟨*Bedingung*⟩**)**

Diese Schleifen sind sehr ähnlich. Die Anweisung wird sooft wiederholt, bis die Bedingung wahr ist. Dabei muss natürlich in den Anweisungen etwas passieren, damit die Bedingung irgendwann wahr werden kann. Der Unterschied zwischen den beiden Schleifen ist, dass die Ausführung bei **wiederhole**-**bis** mindestens einmal abgearbeitet wird, bei **solange** nicht. (Wenn die Bedingung zu Beginn falsch ist, wird der Anweisungsblock bei **solange** direkt übersprungen.)

III Bei Schleifen ist ein wenig Vorsicht geboten: Wenn die Bedingung immer wahr ist, wird die Anweisung immer wiederholt. Man nennt dies eine *Endlos-Schleife*. Ein Beispiel wäre:

```
solange (1 < 5)
    Text '*'
Ende
```

Weitere Schleifen lassen eine Variable von einem Anfangs- zu einem Endwert laufen:

Schleife ⟨*Variable*⟩ ⟨*Anfang*⟩ ⟨*Ende*⟩
    ⟨*Anweisungen*⟩
Ende

Diese Schleifen heißen in den meisten Programmiersprachen **for**-Schleife. Sie sind ein einfaches Hochzählen einer Variable:

```
Schleife x 1 10
    Text '[num x]...[x]'
Ende
```

Anstatt einer Flagge kann eine Schleife auch eine Objekt- oder Raumvariable sein. Dann werden alle Objekte oder Räume abgearbeitet, ⟨*Anfang*⟩ und ⟨*Ende*⟩ müssen nicht angegeben werden:

```
Sei x 0
Schleife xObj
    Inkr x
Ende
Text 'Es gibt [x] Objekte in diesem Adventure.'
```

Eine nützliche Sache ist das Verbinden einer Schleife mit einer Bedingung, die an die Variable in runden Klammern angehängt wird:

```
Sei x 0
Schleife xRaum (xRaum dunkel)
    Inkr x
Ende
Text 'Es gibt [num x] dunkle Räume in diesem
    Adventure.'
```

Zusammenfassung:

• *Variablen* sind Merker für bestimmte Sachverhalte. Variablen in T.A.G. sind global, d.h. der Wert einer Variable ist von überall aus abrufbar.

• Es gibt Variablen für Zahlen, Objekte, Räume, Zustände, Richtungen und Befehle, also quasi zu jedem Element von T.A.G..

• Mit den Anweisungen **`sei`**, **`inkr`**, **`dekr`** usw. kann der Wert einer Variable verändert werden.

• Aktionen, die mit einem Sternchen **`*`** definiert werden, werden nach jedem Zug ausgeführt.

• Zur flexiblen Ausgabe von Variablen und Objekten stehen in T.A.G. eine Reihe von Textbefehlen zur Verfügung.

Verweise:

• Handbuch, Kapitel 4.2: Flaggen und Variablen
• Handbuch, Kapitel 5.1.2: Variablenzuweisungen
• Handbuch, Kapitel 6.2: Ablauf eines Zuges
• Handbuch, Kapitel 7.2: Anweisungen im Text
• Variablendefinitionen in **`karn.adv`**

7 Befehle

7.1 Eigene Befehle definieren

T.A.G. bringt die häufigsten Befehle wie **nehmen**, **gehen**, **legen**, **öffnen** usw. bereits mit. Die Datei **normal.adv** enthält weitere Befehlsdefinitionen wie drücken, ziehen, schlagen. Trotzdem benötigt man für ein eigenes Adventure eigene Befehle, die man in T.A.G. definieren kann.

Am einfachsten sind wohl Befehle, die nur aus einem Wort bestehen, wie zum Beispiel **Lage** oder **Inventar**. Die Definition eines solchen Befehls sieht folgendermaßen aus:

```
Bef     Diagnose
Name    'mich diagnostizieren'
Verb    'diagnose' 'd'
Ausf
    Jenach Gesundheit
    (1) Text 'Du stehst mit einem Bein im Grab.'
    (2) Text 'Du bist stark angeschlagen.'
    (3) Text 'Du bist leicht angeschlagen.'
    (4) Text 'Du bist relativ fit.'
    (5) Text 'Du bist gesund und munter.'
    Ende
EndeAusf
```

Ein Befehl hat also eine ID, einen Namen, verschiedene Verben und einen Ausführungsblock. Der Name wird zum Beispiel beim Rückgängigmachen eines Zugs benutzt. Mit den Verben können die Befehle aufgerufen werden. „Diagnose“ oder „D“ sind zwar keine Verben im eigentlichen Sinne, werden von T.A.G. aber so behandelt. Da Diagnose ein Befehl ohne Objekte ist, muss hier nichts weiteres angegeben werden.

Der Ausführungsblock enthält ein Konstrukt, das mit **jenach** beginnt und mit Ende aufhört. Dazwischen stehen Zahlen in Klammern und Text-Anweisungen. Dieses Konstrukt ist eine Vereinfachung von:

```
wenn (Gesundheit = 1)
    Text 'Du stehst mit einem Bein im Grab.'
wenn (Gesundheit = 2)
    Text 'Du bist stark angeschlagen.'
...
```

In den Klammern können auch mehrere Angaben stehen. Außerdem können statt Zahlenwerten auch Objekte oder Räume verglichen werden. Diese Schreibweise ist ähnlich wie die Schreibweise in den **Vor**- und **NachAusf**-Blöcken.

Aufgabe 19

Erzeuge einen Befehl, der den Spieler, wenn er das magische Wort „XYZZY“ sagt, auf magische Weise ins Haus teleportiert.

Die meisten Befehle verlangen allerdings Objekte. Welche Objekte das sind, muss dann in dem Befehl angegeben werden. Definieren wir also einmal den Befehl „drehe *x* auf *y*“, der auf unseren Safe gemünzt ist.

```
Bef     drehen_auf
Name    'drehen'
Verb    'drehe'
Syntax  dasObj auf dasObj (Allg)
Ausf
    Bed (aObj = Rädchen Safe)
        'Das kannst du nicht drehen.'
    Bed (aObj2 = Zahl)
        'Du kannst das Rädchen nur auf eine
        Nummer drehen.'
    Bed (Nummer > -1) und (Nummer < 100)
        'Das Rädchen kann nur auf Zahlen von 0
        bis 99 gedreht werden.'
    Sei Code Nummer
    Text 'Du hast das Rädchen auf [code] gedreht.'
EndeAusf
```

Dieser Befehl hat eine **Syntax**-Angabe, in der die benötigten Objekte angegeben werden. In dieser Zeile können folgende Angaben gemacht werden:

dasObj	Objekt im Akkusativ
demObj	Objekt im Dativ
nachRitg	Richtungsangabe („nach Norden“)

Direkt nach den Objekten können Zusatzangaben in Klammern gemacht werden, wie hier **(Allg)**. Jedes Wort, was nicht in Klammern steht und kein Schlüsselwort ist, wird als Präposition behandelt. In unserem Beispiel ist das **auf**.

× Nach einer Präposition muss ein Objekt folgen, da T.A.G. bei Präpositionen unterscheidet, ob sie zu einem Objekt oder zu einem Verb gehören. Präpositionen, die Bestandteil des Verbs sind, müssen in der **Verb**-Zeile deklariert werden, z.B. **'mache auf'**

Die Objekte heißen nach der Reihenfolge ihres Auftretens in der **Syntax**-Zeile **aObj**, **aObj2** und **aObj3**. Die Richtung heißt **aRitg** und ist eine Richtungsvariable. Die Reihenfolge, in der der Spieler die Objekte eingibt, ist egal:

> GIB DEM MANN DAS BUCH
aObj = Mann, **aObj2** = Buch

> GIB ES IHM
aObj = ihm (Mann), **aObj2** = es (Buch)

Zahl ist ein besonderes Objekt. Es bedeutet, dass der Spieler anstelle eines Objekts im Spiel eine **Zahl** eigegeben hat. Der Wert der **Zahl** wird in der Variable **Nummer** gespeichert.

Dieser Befehl kann offensichtlich nur auf das Rädchen angewandt werden. Deshalb wird alles im Ausführungsblock des Befehls definiert. Ein Befehl, der allgemeingültiger ist, hätte

dort nur eine allgemeine Antwort. Die speziellen Geschehnisse würden dann bei den Objekten in einer **`VorAusf`** definiert.

Aufgabe 20 **

Um Fehler und Ungereimtheiten in Spielen zu überprüfen, baut man während des Schreibens oft sogenannte Debug-Verben ein. Definiere das Verb „Status ⟨*Objekt*⟩“, das zu jedem Objekt, ob es sichtbar ist, oder nicht, eine Liste von Informationen ausspuckt, wie z.B. Aufenthaltsort, Gewicht, ob bereits benutzt oder nicht. (Tip: Für den Aufenthaltsort beim Befehl Räume ganz unten in **`normal.adv`** schauen. Man muss eventuell schon einmal ins Kapitel 8.3 schauen.)

7.2 Vokabular von Befehlen

Für Text-Adventures ist es wichtig, dass der Spieler seine Gedanken dem Computer in richtiger Sprache mitteilen kann. Natürlich gibt es Einschränkungen: Der Satz muss ein Befehlssatz sein, der möglichst einfach aufgebaut ist.

Damit der Spieler aber seine Idee umsetzen kann, sollte man sich also bei neuen Befehlen überlegen, wie diese heißen. Das Dumme ist nur, dass man denselben Befehl auf verschiedene Weise ausdrücken kann, insbesondere bei Befehlen, die zwei Objekte benötigen, ist das so. Dem Spieler wird aber meist nur eine Möglichkeit einfallen, und wenn es nicht klappt, wird er vielleicht seine Idee aufgeben, weil er denkt, der Parser verstünde das Verb nicht.

Synonyme zu Befehlen, die eine andere Satzstruktur haben, können in T.A.G. abgegeben werden, indem die Ausführung umgeleitet wird. Dabei gibt man keinen Ausführungsblock an, sondern nur das Wort **`Ausf`**, gefolgt von dem Befehl, zu dem umgeleitet wird.

Dazu ein Beispiel: Der Spieler kann sagen „Fülle Wasser ins Glas“ oder „Fülle das Glas mit Wasser“. Der Code sieht dann so aus:

```
Befehl  füllen
Name    'füllen'
Verb    'fülle'
Syntax  dasObj mit demObj
Ausf
    Text  'Dort kannst du nichts einfüllen.'
EndeAusf

Befehl  füllen_in
Name    'füllen'
Verb    'fülle'
Syntax  in dasObj dasObj
Ausf    füllen
```

Mit der Zeile **`Ausf füllen`** wird der Befehl **`füllen_in`** umgeleitet zu füllen. Dabei müssen die Objekte in der selben Reihenfolge definiert werden, und das ist auch der Grund, warum die Syntax-Zeile von **`füllen_in`** etwas komisch aussieht: Der Behälter kommt an

erster Stelle, der Inhalt an zweiter. (T.A.G. erlaubt die Eingabe der Objekte in beliebiger Reihenfolge, der Fall und die Präposition legen dann fest, was gemeint ist.)

Eben haben wir gesehen, dass man nach **dasObj** oder **demObj** zusätzliche Angaben in Klammern machen kann. Diese Angaben sind der Ort und ein Attribut, dass ein Objekt besitzen muss.

Die Ortsangabe muss erfüllt sein. Ist das Objekt am falschen Ort, so bricht der Zug vorzeitig mit einer Meldung des Parsers ab. Die Orte sind:

hier
Das Objekt ist im Raum und für den Spieler zugänglich. Wird kein Ort angegeben, so muss das Objekt hier sein.

beiMir
Das Objekt ist beim Spieler.

nichtBeiMir
Das Objekt ist im Raum, aber nicht beim Spieler.

inSicht
Das Objekt ist sichtbar im Raum, aber nicht unbedingt für den Spieler zugänglich, wie z.B. etwas, das in einem transparenten Behälter eingeschlossen ist.

mobil
Das Objekt ist im Raum und kann aufgehoben werden.

Inhalt
Das Objekt ist in einem anderen enthalten. Inhalt kann nur für **aObj** verwendet werden, und es bezieht sich immer auf **aObj2**.

Die Attributangabe erfolgt wahlweise. Sie muss nicht erfüllt sein. Sie dient nur dazu, eventuell vom Spieler weggelassene Objekte zu raten. Wenn es ein Objekt, das das angegebene Attribut besitzt, am Ort gibt, so wird es automatisch angenommen und dem Spieler in runden Klammern mitgeteilt:

```
> SETZ DICH
(auf den Louis-XV-Sessel)
Du lässt dich etwas unelegant auf dem filigranen Sessel nieder.
```

Wenn man ein Verb in ein anderes umleitet, sollte man aufpassen, dass man für jedes Objekt die passenden Zusatzangaben macht.

7.3 Befehle erweitern

In T.A.G. sind die häufigsten Befehle bereits implementiert. Sie funktionieren in der Regel, wie man es gerne hätte, aber trotzdem möchte man sie manchmal ändern.

Deshalb kann man jeden der bereits vordefinierten Befehle noch einmal definieren, und ihn so erweitern. Wenn zum Beispiel eine Blume im Spiel vorkommt, so möchte sie der Spieler automatisch „pflücken“ anstatt sie aufzuheben oder einfach zu nehmen (obwohl es eigentlich dasselbe ist).

Also erweitern wir den Befehl nehmen:

```
Bef     nehmen
Verb    'pflücke'
```

Das war's, und schon erkennt T.A.G. das Verb „pflücken“ als „nehmen“.

Man kann den Befehlen auch Ausführungsblöcke vor- oder nachschalten. Zum Beispiel unser neues Verb „drehe Rädchen auf Nummer“. Ein gutes Synonym hierfür wäre „stelle“ oder „setzte Rädchen auf Nummer“. Dieses Vokabular gehört aber bereits zum Befehl hineinlegen. Also erweitern wir den Befehl:

```
Bef hineinlegen
VorAusf
    wenn (aObj2 = Zahl) dann
        Ausf drehen_auf aObj aObj2
        Stop
    Ende
EndeAusf
```

Damit checken wir, ob das zweite Objekt eine Zahl ist. Wenn ja, wird **`drehen_auf`** ausgeführt, und der Befehl hineinlegen wird mit **`Stop`** abgebrochen.

7.4 Pseudobefehle

Eine Sache, die den Code für Objekte vereinfacht, aber auf Anhieb nicht immer leicht zu verstehen ist, sind Pseudobefehle.

Zu jedem Befehl kann ein Pseudobefehl definiert werden, z.B. mit

```
Pseudo  getroffen werfen
```

wobei werfen ein bereits definierter Befehl ist. Obwohl zu jedem Befehl ein Pseudobefehl erzeugt werden kann, machen Pseudobefehle nur Sinn, wenn ein Befehl zwei Objekte benötigt. Der Sinn des Pseudobefehls ist es nämlich nur, Aktionen in der **`Vor`**- oder **`NachAusf`** von Objekten abzufangen, wenn dieses Objekt das zweite erwähnte Objekt ist, also **`aObj2`**.

Es gibt eine Variable, **aBef**, die den durchzuführenden Befehl enthält. Dies ist die Variable, die in den **Vor**- und **NachAusf** in runden Klammern abgefragt wird. Sie kann nie einen Befehl enthalten, der auf einen anderen umgeleitet wurde, der Wert von **aBef** ist den der umgeleitete Befehl. Wenn der Befehl zwei Objekte benötigt, wird diese Variable vorübergehend auf den Pseudobefehl gesetzt. Der Pseudobefehl hat nur in **Vor**- und **NachAusf**-Blöcken eine Bedeutung.

Leutet der Befehl „Wirf Dart auf Scheibe“, so bedeuten:

(werfen)
Der Befehl ist **werfen**, das Objekt, das überprüft wird, ist aObj, also die Scheibe.

(getroffen)
Der Befehl ist ebenfalls **werfen**, aber das Objekt, das überprüft wird, ist **aObj2**, also der Dartpfeil.

Der Pseudobefehl ist also nur eine Krücke, um das Objekt auch in der runden Klammer ansprechen zu können.

Für einige T.A.G.-Befehle gibt es vordefinierte Pseudobefehle:

hineinlegen → **empfangen**
herausnehmen → **freigeben**
aufschließen → **aufschl_mit**
abschließen → **abschl_mit**

Aufgabe 21 *

Modifiziere den Tisch so, dass man nur die Vase daraufstellen kann. Bei allen anderen Objekten gibt es eine negative Antwort.

Zusammenfassung:

- Zusätzlich zu den in T.A.G. bereits vorhandenen Befehlen kann man eigene definieren. Die Definition erfolgt wie gewohnt in einem abgeschlossenen Block.

- Zu jedem Befehl müssen Verben und ein Ausführungsblock definiert werden. Wenn ein Befehl Objekte verlangt, so muss eine **Syntax**-Angabe gemacht werden, die den Satzbau beschreibt.

- Nach ihrer Reihenfolge in der **Syntax**-Angabe heißen die angegebenen Objekte **aObj**, **aObj2** und **aObj3**.

- Sätze mit verschiedenen Strukturen, die dasselbe bewirken können auf einen Befehl umgeleitet werden.

- DieStandard-T.A.G.-Befehle können erweitert werden.

- Pseudobefehle können benutzt werden, um Befehlsdurchführungen bei den Ausführungsblöcken von **`aObj2`** abzufangen.

Verweise:

- Handbuch, Kapitel 4.1: Befehle
- Handbuch, Kapitel 6.6: Neue Befehle einbauen
- Befehlsdefinitionen in **`normal.adv`** und **`karn.adv`**.

8 Klassendenken

8.1 Klassen von Objekten

In vielen Adventures kommen Objekte vor, die einander ähnlich sind, oder die zumindest ähnliche Eigenschafen haben. Beispiele sind hier die Schätze aus *Adventure* und *Zork*, Schriftrollen mit Zaubersprüchen, ID-Clips wie in *Starrider* und, und, und.

Solche Objekte lassen sich in so genannten *Klassen* zusammenfassen. Auf diese Weise muss man die allgemeinen Eigenschaften nur einmal definieren. Trotzdem kann jedes Objekt dieser Klasse individuelle Eigenschaften besitzen.

× Der Begriff der Klasse stammt aus der Objektorientierten Programmierung (OOP), die durch die Klassen sehr mächtig ist. Die Klassen von T.A.G. haben die wichtigsten Eigenschaften der OOP-Klassen, sind aber mehr auf reine Text-Adventure-Objekte zugeschnitten und daher nicht ganz so potent.

Eine Klasse wird genau wie ein Objekt definiert, nur beginnt der Block nicht mit **`Obj`**, sondern mit dem Schlüsselwort **`ObjKlasse`**. Für unser kleines Adventure wollen wir zum Beispiel die ganze Wegzehrung, die wir definiert haben, in einer Klasse zusammenfassen. Dazu definieren wir:

```
ObjKlasse Verpflegung
Name    'Verpflegung' f
Vor     'lebens' 'nahrungs'
Subst   'verpflegung' f  'essen' n  'mittel' p
Subst   'mittel' n
Ort     in Rucksack
Besch   'Hmm, sehr nahrhaft.'
Attr    essbar
```

Wenn wir das neue Adventure generieren und laufen lassen, gibt es nichts Neues. Es gibt kein Objekt Verpflegung im Rucksack. Eine Objektklasse beschreibt kein Objekt im Spiel, sondern ist lediglich eine Vorlage für Objekte.

Damit die Klasse wirksam wird, müssen wir also ein Objekt definieren, das dieser Klasse angehört. Dazu geben wir bei einer Objektdefinition die Klasse in Klammern an:

```
Obj     Butterbrot (Verpflegung)
```

Diese Zeile reicht bereits, da alle Eigenschaften von der Klasse übernommen werden: das Butterbrot heißt „Verpflegung“ und kann mit dem bei der Klasse angegebenen Vokabular angesprochen werden. Es befindet sich im Rucksack und ist essbar.

Der Nachteil ist im Moment jedoch, dass dieses Objekt keine individuellen Eigenschaften besitzt. Wenn wir mit dieser Methode noch einen Schokoriegel, eine Banane usw. definieren, sehen alle diese Dinge gleich aus, haben dasselbe Vokabular und treten nur als „Verpflegung“ in Erscheinung. Das ändert sich im nächsten Abschnitt.

8.2 Vererbung

Wie wir eben gesehen haben, übernimmt ein Objekt alle Eigenschaften der Klasse. Man spricht hier von Vererbung. Trotzdem können zu jedem Objekt, das einer Klasse angehört (man sagt dazu auch *Instanz der Klasse*), individuelle Eigenschaften gegeben werden. Je nach Eigenschaft geschieht die Verebrung aber auf verschiedenem Wege:

Ersetzende Verebung
Bei Eigenschaften, die jedes Objekt logischerweise nur einmal haben kann, werden die geerbten Eigenschaften durch die individuellen ersetzt. Das ist der Fall beim Namen, bei den Beschreibungstexten (**`Besch`**, **`Erst`**, **`Darauf`** usw.), bei Zuständen, Ortsangaben und allgemeinen Variablen wie **`Gew`**, **`Vol`** usw.

Erweiternde Vererbung
Andere Eigenschaften, wie z.B. das Vokabular und die Ausführungsblöcke werden dem Objekt einfach hinzugefügt. das Objekt besitzt also das Vokabular der Klasse und des Objekts.

Das klingt vielleicht etwas verwirrend, deshalb hier ein paar Beispiele für unsere Verpflegungsklasse:

```
Obj     Butterbrot (Verpflegung)
Name    'Butterbrot' n
Vor     'butter'
Subst   'brot' n  'schnitte' f
Ort     in Dose
```

Damit ist unser Objekt Butterbrot schon wesentlich individueller, es kann als Brot angesprochen werden und heißt auch nicht mehr „Verpflegung“, sondern „Butterbrot“. Es kann aber, dank der Vokabeldefinition der Klasse, als „Verpflegung“ angesprochen werden.

Weiter geht's:

```
Obj     Banane (Verpflegung)
Name    'Banane' f
Adj     'krumm' 'gelb'
subst   'banane' f  'frucht' f
Besch   'Die Banane ist eine krumme, gelbe Frucht,
        die hoffentlich der EU-Norm entspricht.
        (Oder besser: hoffentlich nicht.)'
```

Damit haben wir für die Banane die Standard-Beschreibung der Klasse ersetzt.

Ein Objekt übernimmt natürlich auch die Attribute der Klasse. Mit **`Attr`** kann man neue Attribute hinzufügen, mit einem vorangestellten Schrägstrich aber auch wieder wegnehmen:

```
Obj     Karotte (Verpflegung)
```

```
Name    'phosphoresziered^ Plastik-Möhre'
adj     'phosphoreszierend' 'leuchtend'
vor     'leucht' 'phosphor' 'plastik'
subst   'karotte' f  'möhre' f
Attr    /eßbar Lichtquelle
```

Etwas komplizierter ist das Vererben von Ausführungsblöcken. Um dies zu veranschaulichen, erweitern wir unsere Klassendefinition:

```
ObjKlasse Verpflegung
Name    'Verpflegung' f
vor     'lebens' 'nahrungs'
subst   'verpflegung' f  'essen' n  'mittel' p
subst   'mittel' n
Ort     in Rucksack
Besch   'Hmm, [der selbst] sieht sehr lecker aus.'
Attr    eßbar
NachAusf
    (essen)
        Wenn (Hunger > 5) dann
            dekr Hunger 5
        sonst
            lösche Hunger
        Ende
EndeAusf
```

Selbst ist eine Variable, die das tatsächlich angesprochene Objekt meint. (Wir erinnern uns, dass die Klasse selbst nicht als Objekt im Spiel auftaucht. In den Ausführungsblöcken der Klasse muss daher selbst benutzt werden.) In der Regel ist selbst aber mit **`aObj`** oder **`aObj2`** identisch.

Mit der **`NachAusf`** wird der **`Hunger`** getilgt. (Dabei wird angenommen, dass **`Hunger`** eine globale Variable ist.) Alle Verpflegungen verringern den Hunger um fünf Einheiten. Wir können diese Ausführung aber erweitern, wobei der individuelle Ausführungsblock des Objekts aber vor dem entsprechenden Block der Klasse ausgeführt wird:

```
Obj     Landjäger (Verpflegung)
...
NachAusf
    (essen)
    Text 'Der Landjäger ist ziemlich fettig, aber
        egal – du bist jetzt weniger hungrig.'
EndeAusf

Obj     fauler_Apfel (Verpflegung)
...
NachAusf
    (essen)
    Stop 'Bah! Der faule Apfel bekommt dir gar
        nicht und du spuckst die Überreste sofort
        aus.'
EndeAusf
```

Bei diesen beiden Objekten wird der Text des Essens nachträglich abgeändert. Beim Landjäger wird nachher allerdings die Reduzierung des Hungers durchgeführt, beim Apfel nicht, da hier die Handlung mit Stop abgebrochen wurde.

8.3 Variablen für Objekte

Zu Objekten können Variablen definiert werden, die allerdings nur Zahlenwerte von 0 bis 255 annehmen können. Dabei muss der Variablenbezeichner von allen anderen IDs verschieden sein. Verschiedene Objekte können aber dieselbe Variable besitzen. Diese Variablen werden im Objektblock mit

Var ⟨*Variablen-ID*⟩ ⟨*Wert*⟩

definiert, der Wert kann wie immer weggelassen werden, er ist dann Null. In Ausführungsblöcken spricht man die Variablen mit

⟨*Objekt*⟩**.**⟨*Variablen-ID*⟩

an. Dabei kann Objekt ein Objekt oder eine Objektvariable sein, und das macht die Variablen für Objekte natürlich besonders nützlich. Ein Objekt, bei dem eine Variable nicht definiert wurde, besitzt sie einfach nicht. Man kann sie dann nicht ändern, und wenn man sie lesen will, ist sie immer Null. Deshalb muss man vorher schon einmal nachfragen, ob ein Objekt eine Variable überhaupt besitzt. Das geht mit folgender Bedingung:

(⟨*Objekt*⟩ **besitzt** ⟨*Variablen-ID*⟩**)**

Diese Variablen sind natürlich nicht an eine Klassendefinition gebunden. Aber man kann sie im Zusammenhang mit Klassen gut einsetzen. Ist bei einer Klasse eine Variable definiert, so haben alle Objekte dieser Klasse auch diese Variable, und zwar mit dem bei der Klasse definierten Wert. Der Wert kann natürlich in der individuellen Objektdefinition überschrieben werden.

So können wir für unsere Verpflegungsklasse den Sättigungsgrad angeben, denn ein Baguette mit Schinken, Käse und Ei sättigt doch mehr, als eine Tomate.

```
ObjKlasse Verpflegung
...
Var Nährwert 5
NachAusf
    (essen)
        Wenn (Hunger > selbst.Nährwert) dann
            dekr Hunger selbst.Nährwert
        sonst
            lösche Hunger
        Ende
EndeAusf
```

Damit ist genau dasselbe erreicht wie vorher: Alle Verpflegungs-Objekte machen den Spieler um 5 Nahrungseinheiten satter. Aber wir können nun diese Sättigung leicht für verschiedene Objekte anpassen:

```
Obj     Landjäger (Verpflegung)
Var     Nährwert 7
...

Obj     Banane (Verpflegung)
...

Obj     BigMac (Verpflegung)
Var     Nährwert 1
...
```

Und so weiter. Die Banane als Durchschnittsverpflegung behält hier den Nährwert von fünf. (Ernährungsfanatiker würden natürlich Variablen für Eiweiß, Kohlehydrate und Fett definieren, aber als erste Näherung kann man sich wohl mit der Angabe eines allgemeinen Nährwerts zufrieden geben.)

Variablen für Objekte sind also sinnvoll, wenn mehrere Objekte – typischerweise natürlich welche, die zu einer Klasse zusammengefaßt sind – dieselbe Eigenschaft besitzen. Eine Variable nur für ein Objekt zu definieren, macht wenig Sinn, dann sollte man lieber eine globale Flagge benutzen.

Aufgabe 22 **

Definiere eine Klasse von Zaubertränken, die verschiedene Farben und eine der drei Wirkungen Heilung, Gift oder Sättigung haben.

Es gibt übrigens Variablen für Objekte, die wir bereits kennen, und die alle Objekte besitzen: ⟨*Objekt*⟩`.Gew` und ⟨*Objekt*⟩`.Vol`. Zu jedem Objekt können bis zu acht zusätzliche Variablen angegeben werden. Räume können in T.A.G. keine Variablen besitzen.

8.4 Ordnung halten mit Klassen

Objektklassen bieten einen weiteren, schönen Effekt: Man kann in Aufzählungen, wie sie z.B. in der Liste der Dinge in einem Raum oder einem Behälter auftauchen, Objekte einer Klasse zusammenfassen.

Dazu gibt es den Plural, der zusätzlich zum Namen definiert werden kann. (Da unsere Verpflegung individuelle Namen besitzt, muss der Name der Klasse nicht angegeben werden.) Wir ändern die Definition der Verpflegung auf

```
ObjKlasse Verpflegung
Plural  'Verpflegung' f 3
...
```

und schon sieht die Beschreibung des Rucksacks folgendermaßen aus:

> Im Moment sind etwas Verpflegung (ein Landjäger, ein Butterbrot und ein nicht mehr ganz frischer Apfel), eine Landkarte und ein Anorak im Rucksack.

Wenn die Liste lang ist, finde ich diese Darstellung übersichtlicher. Die 3 nach dem **f** für feminin bewirkt übrigens, dass es nicht „eine Verpflegung“, sondern „etwas Verpflegung“ heißt. (Außerdem sieht man hier, dass der **Plural** nicht unbedingt im Plural stehen muss.)

Es geht aber noch weiter. Stellen wir uns einmal eine Reihe von Bällen vor:

```
ObjKlasse Ball
Plural  'Bälle' p
subst   'ball' m  'bälle' p
...

Obj     blauer_Ball (Ball)
Name    'blau^ Ball' m
Adj     'blau'
...
```

Mit dieser und weiteren, ähnlichen Definitionen, sähe eine Liste folgendermaßen aus:

> In dem Korb siehst Du vier Bälle (einen blauen Ball, einen roten Ball, einen bunten Ball und einen schwarz-weißen Ball).

Diese Aufzählung hätte unser Deutschlehrer wohl mit einer roten Schlange und einem „A“ für Ausdrucksfehler bedacht. Um solch unschöne Wortwiederholungen bei ähnlichen Objekten zu vermeiden, kann man den Objekten der Klasse einen Listennamen (**lName**) geben:

```
Obj     blauer_Ball (Ball)
Name    'blau^ Ball' m
lName   'blau^' m
Adj     'blau'
...
```

Nun sieht die Aufzählung so aus:

> In dem Korb siehst Du vier Bälle (einen blauen, einen roten, einen bunten und einen schwarz-weißen).

Wenn beim **Plural** nach der Geschlechtsangabe nichts angegeben wird (oder eine 0), so werden die Objekte, die gruppiert werden, durchgezählt und das Zahlwort vorangestellt.

Wenn der Spieler nun „nimm den Ball“ eingibt, so erscheint die standardmäßige Nachfrage: "Welchen Ball? Den blauen, den roten, den bunten oder den schwarz-weißen?" Bei dieser Nachfrage werden auch die Listennamen, wenn vorhanden, benutzt.

Ist die Eingabe aber „Nimm einen Ball“, so sucht das Programm eigenmächtig einen aus, da wegen des unbestimmten Artikels angenommen wird, dass es dem Spieler egal ist, welchen. (Es wird wohl der zuerst erwähnte sein, also der blaue.) Man kann sogar „Nimm zwei Bälle“

oder „Hebe alle Bälle außer den bunten auf“ angeben. Das funktioniert natürlich nur bei Befehlen, die Mehrfachobjekte zulassen, wie **nehmen** und **hinlegen**. Die Eingabe mit unbestimmtem Artikel oder Zahlwort wird bei allen Objekten erkannt, die gemeinsames Vokabular haben, aber bei Klassen ist die Definition natürlich wesentlich einfacher.

Zum guten Schluss lassen sich mit Klassen gleiche Objekte zusammenfassen. Gleiche Objekte sind Objekte, die derselben Klasse angehören, und die kein individuelles Vokabular besitzen. Diese Objekte sind also nie einzeln ansprechbar, sondern immer nur als Objekt einer Klasse. Die Gruppierung funktioniert wie oben, nur, dass die Objekte nicht zusätzlich in Klammern aufgelistet werden:

```
ObjKlasse Silbermünze
Plural  'Silbermünzen'
...

Obj     S1 (Silbermünze)
Obj     S2 (Silbermünze)
Obj     S3 (Silbermünze)
Obj     S4 (Silbermünze)

ObjKlasse Kupfermünze
Plural  'Kupfermünzen'
...

Obj     K1 (Kupfermünze)
Obj     K2 (Kupfermünze)
```

Die Liste lautet hier z.B. „Hier sind vier Silbermünzen und zwei Kupfermünzen“. Obwohl die Münzen gleich sind, müssen sie unterschiedliche IDs besitzen. Für das Programm sind sie also doch verschieden.

Aufgabe 23 *

Erzeuge ein Gefäß mit einem Rostlösemittel. Erzeuge weiterhin einen Satz von verrosteten Münzen, die zunächst nicht unterscheidbar sind, sich aber nachdem sie in den Rostlöser getaucht wurden, in verschiedene Münzen verwandeln: eine Dublone, eine Pesete usw.

8.5 Raumklassen

So wie es Klassen für Objekte gibt, so gibt es auch Raumklassen. Hier werden der Name und die Ausgänge ersetzend vererbt, die Ausführungsblücke erweiternd. In der Praxis sind Raumklassen eigentlich nur interessant, wenn sie Ausführungsblöcke besitzen. Das Schlüsselwort für eine Raumklasse ist – wer weiß es? – **RaumKlasse**.

Als kurzes Beispiel soll hier eine Raumklasse für ein Labyrinth definiert werden, in der verhindert wird, dass der Spieler Gegenstände auf den Boden legt, um eine Karte zu zeichnen:

```
RaumKlasse Irrgarten
```

```
Name    'Irrgarten'
Std     Wand
Attr    Laby
Besch   'Der Irrgarten ist dunstig und neblig,
        dicke Schwaden ziehen über den Boden.'
NachAusf
    (legen)
        Text 'Bald wabern die Nebelschwaden über
            [den aObj], und [er] verschwinde[t]
            – für immer.'
NachAusf
```

Nun muss der Autor nur noch einzelne Räume erzeugen, die lediglich Informationen über die Ausgänge enthalten müssen. Die Definition von Raumklassen funktioniert analog zu den Objektklassen, wird aber weitaus weniger benutzt. Deshalb soll dieser kurze Absatz zu Raumklassen genügen.

Zusammenfassung:

- Um mehrere ähnliche Objekte zu implementieren, können *Objektklassen* definiert werden, die nicht im Spiel als Objekt auftauchen, aber als Vorbilder für tatsächliche Objekte dienen.

- Die Eigenschaften der Klasse werden auf Objekte dieser Klasse übertragen, man spricht hier von *Vererbung*.

- Ausführungsblöcke und Vokabular werden beim Vererben ergänzt, alle anderen Eiganschaften ersetzt.

- Zu jedem Objekt können bis zu acht Variablen definiert werden. Durch eine Definition derselben Variablen bei anderen Objekten kann man so flexibel auf diese Werte zugreifen.

- In T.A.G. können Klassen auch dazu benutzt werden, um Objekte bei Listen zu gruppieren oder um eine Anzahl von identischen Objekten zu beschreiben.

- Es können auch Klassen für Räume erzeugt werden.

Verweise:

- Handbuch, Kapitel 3.3: Klassen von Objekten und Räumen
- Handbuch, Kapitel 11: Aufzählungen und Beschreibungen
- Objektklasse Edelstein in karn.adv

9 Personen und Kommunikation

9.1 Personen sind Objekte

Zunächst einmal sind Personen (oder NPCs, *Non-Player Characters*, wie sie auch gerne genannt werden) in T.A.G. gewöhnliche Objekte. Sie besitzen nur zusätzlich das Attribut **`Person`**. Das bedeutet, dass man sie nicht aufheben kann und dass sie in der Raumbeschreibung gesondert am Schluss erwähnt werden. Eine Person unterscheidet sich damit nicht allzusehr von den anderen Objekten im Spiel, und all das, was eine Person ausmacht, nämlich Gespräche oder Verhaltensmuster, muss mehr oder weniger mühsam programmiert werden.

Aber eins nach dem anderen. Definieren wir also eine Person für unsere Haus- und Waldidylle, vielleicht einen Naturliebhaber.

```
Obj     Wanderer
Name    'Wanderer' m
vor     'wanders'
subst   'wanderer' m  'mann' m
Ort     Lichtung
Attr    Person transparent
Besch   'Der Wanderer ist ein gutgelaunter Kerl
        Ende Dreißig, der eine Kniebundhose und
        ein kariertes Hemd trägt und sich an den
        schönen Dingen des Lebens, das heißt der
        intakten Natur, erfreut.'
Dabei   'Der Wanderer trägt [liste 1] bei sich.'
Erst    'Ein Wandersmann sitzt hier und macht
        gemütlich Pause.'
```

Damit ist der erste Schritt getan. Der Wanderer befindet sich auf der Lichtung und ist einfach da. Mehr aber auch nicht. Die Passivität des Wanderers ändert sich aber im nächsten Abschnitt. (**`Dabei`** ist übrigens das Äquivalent von Darin und **`Darauf`** für **`Dinge`**, die eine Person **`bei`** sich hat.)

9.2 Reaktionen

Personen bringen natürlich Leben in ein Adventure: Sie reagieren auf Aktionen des Spielers, sie reden mit dem Spieler und wandern unter Umständen sogar auf eigene Faust umher. Das macht diese Personen interessant, aber auch nicht ganz einfach zu programmieren. Schon unwichtige, eher muffelige Zeitgenossen benötigen einige Zeilen, um ihr Verhalten zu beschreiben. Bei ausgereiften Charakteren, wie sie z.B. in Detektivadventures vorkommen, steigt der Aufwand um ein Vielfaches, und trotzdem wirken die NPCs manchmal immer noch stereotyp.

Kümmern wir uns erst einmal um die Reaktion der Person auf die Aktionen des Spielers. Dazu gibt es zwei eigene Ausführungsblöcke, **VorReakt** und **NachReakt**. Diese Blöcke funktionieren wie die altbekannten **Vor**- und **NachAusf**, mit dem Unterschied, dass keins der erwähnten Objekte die Person sein muss. Mit anderen Worten, diese Blöcke können eine Reaktion auf jede beliebige Handlung des Spielers sein.

Erweitern wir also die Definition um ein paar Verhaltensmuster.

```
Obj     Wanderer
...
VorReakt
    (gehen)
        Text '"Tschüss!", winkt mir der Wanderer
            hinterher.'
        Absatz
EndeAusf

NachReakt
    (gehen)
        Wenn (aRaum besucht) dann
            Text '[#]Der Wanderer nickt mir kurz
                zum Gruß zu.'
        sonst
            Text '[#]"Guten Tag!", sagt der
                Wanderer, "Schön, dass es noch
                mehr Freunde der Natur hier gibt.'
        Ende
    (hinlegen)
        Text '[#]"Wieso werfen Sie das weg?",
            fragt der Wanderer verärgert. "Tun
            Sie ihren Müll doch in einen Beutel
            oder Ihren Rucksack."'
    (singen springen fluchen)
        Text '[#]Der Wanderer schaut mich etwas
            mitleidig an.'
EndeAusf
```

Die Ausführungsblöcke **VorAusf** und **NachAusf** funktionieren bei Personen natürlich so, wie gehabt:

```
VorAusf
    (gegeben)
        Bed (aObj eßbar)
            '"Nein, danke!", sagt der Wanderer
            höflich.'
        Text '"Hmm, lecker, danke!", sagt der
            Wanderer, und kaum hat er [den aObj]
            in den Händen, verschlingt er [ihn]
            mit großem Appetit.'
        ObjNach aObj Nirwana
        Stop
EndeAusf
```

× Tatsächlich können alle Objekte die Blöcke **VorReakt** und **NachReakt** haben. So kann z.B. ein Rauchmelder auf das Anzünden eines Streichholzes reagieren oder ein stark riechender Käse alle

Schnupperversuche im Keim ersticken. Am häufigsten gebraucht werden diese Blöcke aber wohl bei Personen.

9.3 Gespräche führen

Natürlich kann man sich nicht nur anhören, was die Personen im Spiel zu sagen haben, sondern man kann mit ihnen auch reden. Üblicherweise geschieht dies mit einem der folgenden Befehle:

```
> Frage den Wanderer über die Bäume
> Erzähle ihm von dem Haus
> Wanderer, gib mir die Karte.
```

Die ersten beiden Möglichkeiten sind dabei gewöhnliche Befehle, die mit einer **VorAusf** abgefangen weden können:

```
VorAusf
    (fragen)
    Text 'Der Wanderer sagt:'
    Jenach aObj2
        (Bäume)
            Text '"Ganz klar, das sind Birken."'
        (Wanderer)
            Text '"Ich bin ein Wanderer.  Außerdem
                bin ich Kassierer beim  SGV."'
        (Haus)
            Text '"Ich weiß nicht, wem das  Haus
                gehört."'
        (Natur)
            Text '"Ich liebe die Natur."'
        ...
        (sonst)
            Text '"Ouh, da fragen Sie mich was."
                Er schüttelt den Kopf. "Aber
                darüber weiß ich nichts."'
    Ende
    Stop
EndeAusf
```

Dabei kann man natürlich über alles mögliche fragen, auch über Dinge, die sich nicht im Raum befinden. Manchmal muss man zusätzliche Dekos einfügen, um abstrakte Gesprächsthemen, die natürlich in keinem Raum sind, abzufangen:

```
Deko    Natur
Name    'Natur' f
adj     'frei'
subst   'natur' f  'umwelt' f
```

Die letzte oben genannte Möglichkeit ist eigentlich ein normaler Befehl, der nur an jemand anderes als den Spieler gerichtet ist. An wen er gerichtet ist, wird in der Variable **Akteur** gespeichert, die normalerweise den Wert 0 hat. Ist **Akteur** ein anderes Objekt, so tritt ein besonderer Ausführungblock in Kraft, die **BefAusf**. Sie ist wie die bekannten Blöcke

aufgebaut:

```
Obj     Wanderer
...
BefAusf
    (nicht_verst)
    Text '"Ich habe Sie nicht richtig verstanden",
        sagt der Wandersmann.'
    (gehen)
    Text '"Ich gehe, wohin ich will."'
    ...
    (sonst)
    Text '"Was sind Sie denn für ein Hektiker.
        Genießen Sie lieber die Natur, anstatt
        andere Leute herumzukommandieren", sagt
        der Wanderer erbost.'
EndeAusf
```

Dabei ist **nicht_verst** der Befehl Nr. Null, der bedeutet, dass die Eingabe des Spielers fehlerhaft war. Man sieht, dass Personen in der Regel die meisten Befehle nicht befolgen.

Wenn man versucht, einem Objekt Befehle zu erteilen, das keine **BefAusf** hat, erscheint einfach ein platter Satz wie „Der Tisch scheint nicht an einem Gespräch mit mir interessiert zu sein.“

Aufgabe 24 **

Definiere einen Troll, der im Haus ist, und der den Spieler nicht in den Keller lässt, bevor er ihm etwas Essbares gegeben hat. (Die Trolle in unserer Welt sind offenbar eher auf Nahrungsmittel als auf Wertsachen fixiert.) Befehle und Konversation bedenkt der Troll natürlich mit einem unverständlichen Knurren.

9.4 Eigenständige Handlungen anderer Personen

Manche Personen bleiben für die Dauer des Spiels am selben Platz, andere wiederum wandern scheint’s selbständig umher, und benutzen andere Objekte, genau wie der Spieler selbst. Dabei können die Handlungen der Person mehr oder wenig zufällig sein, oder aber einem vorgefertigten Schema folgen.

Egal, wie, diese Handlungen laufen unabhängig von der Eingabe des Spielers ab, und können so nicht in der NachReakt abgefangen werden. Solche eigenständigen Handlungen gehören in eine Aktion *****.

Zum Beispiel können wir unseren Wanderer nach einem Zufallsschema von der Lichtung vors Haus wandern lassen und umgekehrt:

```
Aktion *
Ausf
    wenn (proz 12) dann
        wenn (Wanderer in vor_Haus) dann
```

```
            wenn (Wanderer hier)
                Text 'Der Wanderer geht langsam
                nac Norden auf die Lichtung.'
            ObjNach Wanderer Lichtung
        sonst
            wenn (Wanderer hier)
                Text 'Der Wanderer schlendert
                nach Süden.'
            ObjNach Wanderer vor_Haus
        Ende
        Wenn (Wanderer hier)
            Text 'Der Wanderer ist soeben
            angekommen, er nickt mir freundlich
            zu.'
    Ende
EndeAusf
```

Diese Aktion ist relativ simpel, der Wanderer wandert zwischen zwei Räumen hin und her. Wann das geschieht, wird durch eine Zufallsbedingung gesteuert, hier in zwölf Prozent aller Fälle, also etwa alle acht Züge.

Aufgabe 25 **

Der Wanderer soll nun entlang einer festgelegten Route laufen, wobei er immer dieselbe Zeit am selben Ort verweilt. Zu bestimmten Zeitpunkten macht er bestimmte Bemerkungen. Dieses Verhaltensmuster soll einen geschlossenen Kreislauf bilden, so dass es sich ständig wiederholt.

Zusammenfassung:

- Andere Personen im Spiel (NPCs) werden von T.A.G. als Objekte behandelt.

- Es gibt zwei Ausführungsblöcke für Objekte, **`VorAusf`** und **`NachAusf`**, die es dem Objekt bzw. der Person erlauben, bei Befehlen zu intervenieren, in denen sie nicht direkt angesprochen wurden.

- Ein weiterer Ausführungsblock, **`BefAusf`**, steuert das Ausführen von Befehlen durch andere Personen, die in der Form „Person, Befehl“ gegeben werden.

- Allgemeine Verhaltensmuster, die unabhängig vom Befehl des Spielers sind, müssen in einer Aktion **`*`** definiert werden.

Verweise:

- Handbuch, Kapitel 6.5.2. Ausführungen von Objekten
- Handbuch, Kapitel 12. Andere Personen im Spiel
- Der alte Mann, die Zwerge, der Geist und die Meisterin in **`karn.adv`**

10 Verschiedenes und Ausblick

10.1 Der Spieler

Obwohl viele Dinge in T.A.G. als Objekt dargestellt werden – der Spieler nicht. Er wird eigentlich nur durch eine Anzahl von Variablen beschrieben: Sein momentaner Aufenthaltsort ist **aRaum**, die Anzahl der Dinge, die er bei sich trägt ist **Inv**. Objekte, die beim Spieler sind, sind **beiMir**, die er angezogen hat **angezogen**.

Das ist vielleicht nicht ganz konsequent, aber, wie man so schön sagt, wenn einem nichts Besseres einfällt, „historisch gewachsen“. Wenn man sich einmal daran gewöhnt hat, funktioniert es auch ganz gut so. Was aber unschön ist, ist, dass man dem Spieler keine **Vor**- und **NachAusf** verpassen kann. Deshalb müssen spezielle Dinge, die den Spieler betreffen, direkt bei der Befehlsdefinition abgefangen werden:

```
Bef     töten
Name    'töten'
Verb    'töte' 'bringe um' 'massakriere' ...
Syntax  dasObj (Person)
Ausf
    wenn (aObj = du) dann
        Text 'Na gut.'
        gestorben
    Ende
    Text  'Was soll diese rohe Gewalt?'
EndeAusf
```

10.2 Punkte

Nach wie vor wird der Erfolg des Spielers in Adventures oft mit Punkten gemessen. In T.A.G. gibt es dazu ein System, das für einzelne Aktionen vergebene Punktzahlen automatisch addiert. Dazu muss man sich nur überlegen, welche Aktion im Spiel wieviele Punkte gibt. Jede dieser Aktionen bekommt eine Nummer, die von 1 bis 255 gehen kann. Zu Beginn ist die Punktzahl für alle Aktionen null.

Wenn zum Beispiel das Öffnen des Safes fünf Punkte bringen soll, und wir diesem Ereignis die Punktenummer 1 geben, so ändert sich unsere Aktion zum Safe öffnen:

```
Aktion *
Ausf
    Wenn (code = 23) und (Safe abgeschlossen) dann
        Text '[#]Vom Safe hörst du ein leises
            "Klick!"'
        ObjZust Safe geschlossen
        Punkte 1 5
    Ende
```

```
EndeAusf
```

Die Anweisung **Punkte** bewirkt, dass für Ereignis 1 5 Punkt vergeben werden. Je nachdem, ob Punktestandsmeldung eingeschaltet ist oder nicht wird auch ein Text wie „Du hast gerade 5 Punkte bekommen“ ausgegeben.

Was aber, wenn wir den Safe schließen und dann wieder öffnen? Nichts, denn T.A.G. hat für dieses Ereignis bereits fünf Punkte vergeben und wird es nicht noch einmal tun. (Es gibt eine Möglichkeit, dem Spieler die fünf Punkte beim Schließen des Safes mit **Punkte 1 0** wieder wegzunehmen. In diesem Fall würde es beim Erneuten Öffnen wieder fünf Punkte geben. Aber diese Vorgehensweise ist wohl eher unüblich.)

Die Integer-Variable **Pktzahl** summiert die Einzelpunktzahlen aller Ereignisse. Sie kann nur gelesen, nicht geändert werden. Änderungen erfolgen immer über die Einzelpunktzahlen.

Dies ist der einfachste Fall der Punktevergabe. Das Punktesystem von T.A.G. besitzt weitere Möglichkeiten der Punktevergabe: Wie Punkte für Räume und Objekte vergeben werden oder wie man negative Punktzahlen vergibt, steht ausführlich im Kapitel 8.4 des T.A.G.-Handbuchs.

10.3 Das Ende des Spiels

Das Ende des Spiels wird erreicht, wenn der Spieler es wünscht und „Ende“ eingibt, wenn er stirbt oder wenn er gewinnt.

Die Anweisung **gestorben** lässt den Spieler sterben. Er bekommt gesagt, dass er tot ist, seine Punkte aufgelistet und wird dann gefragt, ob er aufhören, neu anfangen oder ein abgespeichertes Spiel laden möchte.

Mit der Anweisung gewonnen wird das Spiel gewonnen. „Du hast gewonnen“ und die erreichten Punkte werden ausgegeben und das Programm bricht ab. (Wer gerade gewonnen hat, möchte wohl nicht von vorne anfangen.) Unser Testspiel ist zu klein und hat auch keine richtige Aufgabe für den Spieler, so dass es im Moment nicht gewonnen werden kann. Wer will, darf das natürlich einbauen.

10.4 Standardtexte

Da T.A.G. bereits die wichtigsten Befehle mitbringt, gibt es auch bereits Texte, die diese Befehle ausgeben. Diese Texte passen nicht unbedingt zu jedem Adventure, und der Autor möchte sie wahrscheinlich an sein Opus anpassen.

Diese Texte, die so genannten *Standardtexte*, werden normalerweise von der Datei **tag.std** gelesen. Sie sind dort nacheinander aufgelistet, kurze Kommentare sagen, zu welchem Befehl

der jeweilige Antwortblock gehört. (Die ersten vierzig Antworten sind die Fehlermeldungen des Parsers.) Diese Texte können nach Belieben abgeändert werden, solange die Numerierung nicht geändert wird.

Man kann auch selbst eine Standard-Datei erzeugen, die natürlich den selben Aufbau wie **`tag.std`** haben muss. (Das heißt, Text Nr. 6 sollte in beiden Dateien in etwa dieselbe logische Aussage haben.) Dass eine eigene Datei verwendet wird, teilt man T.A.G. mit folgendem Befehl mit:

```
#STD 'test.std'
```

(Dabei muss anstelle von **`test.std`** natürlich der korrekte Dateiname stehen.)

Wer will, kann den gesamten Satz von Standardtexten auch in seine **`adv`**-Datei packen. Dann werden die Antworten als ein Block behandelt, der mit dem Schlüsselwort **`Standard`** beginnt.

10.5 Äußere Form

Text-Adventures, die mit T.A.G. erstellt wurden, kommen relativ schlicht daher. Es gibt wenige Farben, und jede Menge von Text, der in Schreibmaschinenschrift auf dem Bildschirm erscheint. Wenn es Grafik gibt, dann ist diese in einer fast vergessenen Stilrichtung gehalten, dem ASCII-ismus.

Was man also bei der äußeren Form von Texten verändern kann, ist das Format. Dazu gibt es einige Steuerflaggen, die mit einem Doppelkreuz beginnen:

`#Absatz` beschreibt das Absatzformat: 0 bedeutet keine Einrückung, keine Leerzeile zwischen Absätzen. 1 schreibt eine Leerzeile, 2, rückt Absätze am Anfang um eine Tabulatorweite ein und 3 macht beides. Die Voreinstellung ist 1.

`#Eingabe` gibt an, ob die Eingabe immer in Großbuchtaben (0) oder wie vom Spieler eingegeben (2) erscheint. Dasselbe gibt es auch in Fettdruck (1 oder 3).

`#Raumname` gibt das Format des kurzen Raumnamens vor der langen Raumbeschreibung an. Ist dieser Wert vier, so wird kein Raumname vor der langen Raumbeschreibung ausgegeben. Das ist die Voreinstellung. Werte bis vier bedeuten, der Raumname wird in einer Zeile (0), in einem Absatz (2) und als Fettdruck (1 bzw. 3) vorangestellt

`#Besch` gibt knappe (0) oder ausführliche (1) Raumbeschreibungen aus.

`#Inv` schaltet zwischen dem kompakten Listenformat (0) und der Klartext-Liste (1) für das Inventar hin und her. (Die Befehle „i lang“ und „i

quer“ aus **normal.adv** benutzen diese Flagge.)

Die einzige Textauszeichnung, die man in T.A.G. vergeben kann, ist Fettdruck. Fett bedeutet hier allerdings, dass eine andere, intensivere Farbe zur Textausgabe verwendet wird, mehr nicht. Wer Wörter ohne Fettdruck betonen möchte, kann sich an die e-Mail-Konvention halten, und sie in ***Sternchen*** oder _**Unterstriche**_ einschließen.

Zum guten Schluss gibt es einen Block zur Umgebungsdefinition. Hier können der Name des Spiels, eine Kennung, die für abgespeicherte Spielstände verwendet wird, und die Textfarben definiert werden. Für unser Testbeispiel sähe das etwa so aus:

```
Umgebung
Name    'Der Test'
Kennung 'test'
Text    %hellgrau %blau
Fett    %weiß %blau
Zeile   %rot %hellgrau
```

(**%weiß**, **%blau** usw. sind hier Systemkonstanten für die Farben.) Da Farben immer Geschmackssache sind, kann der Spieler diese in der T.A.M. ausschalten, wenn er möchte.

10.6 Ausblick

Wenn Ihr Euch durch dieses kurze Tutorium gearbeitet habt, und ab und zu mit dem Kopf nicken konntet, dann seid Ihr jetzt fit genug, um ein eigenes Adventure auf die Beine zu stellen. Dabei werdet ihr wahrscheinlich ab und zu auf Probleme stoßen. Dann schaut Euch das Handbuch an, insbesondere Teil II.

Wer etwas weiter in die Theorie von T.A.G. eindringen will, sollte sich Buch III des Handbuchs zu Gemüte führen. Dort werden weitere Beispiele behandelt, die man bestimmt auf sein eigenes Adventure hin abwandeln kann. Die Grundlage zum Verständnis des dritten Teils dürfte mit diesem Tutorium geschaffen worden sein.

Also, es bleibt mir nichts anderes übrig, als Euch ein frohes Schaffen zu wünschen. Wenn ihr absolut nicht weiterwißt, oder Euch gar Probleme mit T.A.G. oder T.A.M. auffallen (etwas, das sich leider nicht ausschließen läßt), dann schreibt es mir unter **martin.oehm@gmx.de**.

Zusammenfassung:

- Der Spieler in T.A.G. ist kein Objekt, sondern wird durch Variablen beschrieben.

- T.A.G. verfügt über ein ereignisabhängiges Punktesystem. Mit der Angabe **Punkte** ⟨*Ereignis-Nr*⟩ ⟨*Punktezahl*⟩ können leicht Punkte vergeben werden.

- Mit **gestorben** kann man den Spieler sterben lassen, mit **gewonnen** gewinnt er das Spiel.

• Die Standardtexte in **tag.std** können angepasst oder komplett neu geschrieben werden, solange sie dieselben Nummern wie in **tag.std** haben.

• Mit Formatvariablen läßt sich das äußere Erscheinungsbild der Ausgabe in T.A.M. ändern. Ein Umgebungsblock definiert allgemeine Konstanten.

• Ab hier müßt Ihr selbst weitermachen. Empfohlene Lektüre ist Teil III des T.A.G.-Benutzerhandbuchs. Empfohlene Methode ist *learning-by-doing*.

Verweise:
• Handbuch, Kapitel 13: Der Spieler
• Handbuch, Kapitel 7: Erstellen der Texte
• Handbuch, Kapitel 8: Allgemeines
• Handbuch, Kapitel 16: Extras
• Handbuch, Kapitel 17: Allgemeine Tips und Tricks
• Der Spieler in **karn.adv**
• Die Texte in **tag.std** und **karn.std**

Anhang A:

Lösungen zu den Aufgaben

1

Einfach folgende Zeilen in die Raumblöcke einfügen:

```
Raum    Vor_dem_Haus
...
Rein    Im_Haus

Raum    Im_Haus
...
Raus    Vor_dem_Haus
```

Rein und **Raus** sind in **normal.adv** definiert.

2

```
Raum    Vor_dem_Haus
Name    'Vor dem Haus'
Std     nur_N_oder_O
NO      Gegen_Wand
SO      Gegen_Wand
O       Im_Haus
N       Lichtung
Besch   ...

Raum    Im_Haus
Std     Gegen_Wand
W       Vor_dem_Haus
Name    'In dem kleinen Haus'
Besch   ...

Antwort nur_N_oder_O
Besch   'Dort ist das Gebüsch zu dicht. Es geht
        nur nach Norden oder nach Osten ins Haus.'

Antwort Gegen_Wand
Besch   'Dort ist leider eine Wand.'
```

3

```
Obj     Taschenmesser
Name    'Taschenmesser' m 5
vor     'taschen' 'klapp'
subst   'messer' m
Zust    geschlossen
Ort     BeiMir
Besch   'Dein Taschenmesser hat einen
        Hirschhorngriff und ist im Moment
```

```
            [Taschenmesser.Zust].'
```

Einer der Zustände **offen**, **geschlossen** oder **abgeschlossen** bewirkt, dass man diesen Gegenstand öffnen und schließen kann.

Die 5 hinter dem **m** für Maskulinum bewirkt, dass das Messer nicht „das Taschenmesser“, sondern „dein Taschenmesser“ heißt. Mehr dazu im Handbuch.

Der Ausdruck in eckigen Klammern bedeutet, dass der Zustand im Klartext, also „offen“ oder „geschlossen“, je nach dem momentanen Zustand des Messers ausgegeben wird. (Das ist ein Vorgriff auf Kapitel 6.)

4

```
Obj     Dose
Name    'Brotdose' f
vor     'brot' 'blech'
subst   'dose' f
Ort     in Rucksack
Zust    geschlossen
Attr    Behälter

Obj     Tomate
Name    'Tomate' f
subst   'tomate' f
Ort     in Dose
Attr    essbar
Besch   'Die Tomate ist rot, rund, bißfest und sie
        hat oben einen kleinen grünen Kranz. das
        unterscheidet sie nicht von anderen
        Tomaten. (Wer jetzt hier eine Bemerkung
        über Tomaten aus Holland erwartet hat –
        Pech gehabt.)'

Obj     Ei
Name    'hartgekocht^ Ei' n
adj     'hart' 'gekocht' 'hartgekocht'
vor     '8–Minuten'
subst   'ei' n
Ort     in Dose
Attr    essbar
Besch   'Hartgekocht, acht Minuten ungefähr.'
```

5

Einfach ein Objekt mit dem Attribut **Kleidung** definieren:

```
Obj     Anorak
Name    'Anorak' m
adj     'rot'
vor     'wind'
subst   'anorak' m  'jacke' f
Ort     in Rucksack
Attr    Kleidung
```

```
Besch   'Eine rote, wind- und wetterfeste Jacke
        aus Mega-Duro-Tex [[tm]].'
```

6 Damit man die Gurke sehen kann, muss man einen Behälter definieren, der **transparent** ist:

```
Obj     Einmachglas
Name    'Einmachglas' n
vor     'einmach' 'einweck'
subst   'glas' n
Ort     in Rucksack
Attr    Behälter transparent
Zust    geschlossen
Besch   'Von meiner Oma.'
Darin   'In der trüben Brühe im Einmachglas
        schwimm[t 0] [liste 0].'

Obj     Gurke
Name    'Gewürzgurke' f
vor     'gewürz'
subst   'gurke' f  'gürkchen' n
Ort     in Einmachglas
Attr    essbar
Besch   'Diese Spezialität aus dem Spreewald ist
        grün und picklig, aber trotzdem nicht
        abstoßend.'
```

7

```
Raum    Keller
...
Besch   'Dieser Keller riecht modrig, die feuchten
        Wände sind mit Moos bewachsen. Das, was
        von den verfaulten Regalen übriggeblieben
        ist, ist meist leer.

        Eine intakte Holztreppe führt zurück nach
        oben.'
Attr    dunkel

Obj     Regale
Name    'halbverfault^ Regale' p
adj     'faulig' 'faul' 'verfault' 'modrig'
subst   'regale' p  'regal' n
Ort     Keller
Attr    fest
Darauf  'Auf einem der verfaulten Regale [ist]
        [liste 0].'
```

8 Nun gut, das ist wohl eher eine Fleißarbeit. Exemplarisch sollen hier der Tisch, der Rucksack und der Wanderstab betrachtet werden:

```
Obj     Tisch
...
Vol     80

Obj     Rucksack
...
Vol     40
Gew     3

Obj     Wanderstab
Name    'Wanderstab' m
vor     'wander'
Subst   'stab' m  'stecken' m
Ort     beimir
Gew     2
Vol     60
Besch   'Der Wanderstab ist gut einen Meter
        sechzig lang. Zwei Plaketten sind an den
        Stab genagelt: ein Edelweiß und eine mit
        der Aufschrift "Internationale
        Volkswanderung Kahler Asten 1982"'
```

Klar, der Trick ist, dass der Stab ein größeres Volumen hat als der Rucksack, aber kein größeres als der Tisch.

9

```
Obj     Klappstuhl
Name    'Klappstuhl' m
vor     'klapp'
Subst   'stuhl' m
Attr    Sitz Standfläche
Zust    offen
Ort     Keller
Besch   'Ein blauer Klappstuhl aus der Serie
        "Luleå".'
```

10

```
Deko    Bäume
Name    'Bäume' p
Subst   'baum' m 'bäume' p 'eichen' p 'birken' p
Subst   'eiche' f 'birke' f
Ort     Lichtung
Besch   'Die Bäume hier sind hauptsächlich Eichen
        und Birken. (Das hättest Du aber schon aus
        der Raumbeschreibung erfahren können, du
        Naseweis.)'

Deko    Wand
Name    'Wand' f
subst   'wand' f  'wände' p 'decke' f  'boden' m
Ort     Im_Haus Keller
```

```
Besch   'Die Wand ist ein vertikales, ebenes
        Konstrukt. Es gibt hier vier Wände. Die
        begrenzenden Ebenen oben und unten heißen
        Decke und Boden.'
```

11

```
Obj     Pilz
Name    'eigenartig^ Pilz' m
adj     'eigenartig' 'grün' 'weiß' 'giftig'
vor     'gift' 'knollenblätter'
subst   'pilz' m 'kappe' f 'hut' m 'punkte' p
Ort     Lichtung
Besch   'Der Pilz hat eine grüne Kappe mit
        kleinen, weißen Punkten.'
Erst    'Ein eigenartiger grüner Pilz wächst
        mitten auf der Lichtung.'
VorAusf
    (essen)
    Wenn (proz 20) dann
        Text 'Du beißt herzhaft in den Pilz. Aber
            erstens schmeckt der rohe Pilz
            ziemlich bescheiden und zweitens ist
            er hochgiftig.'
        Gestorben
    Ende
EndeAusf
NachAusf
  (nehmen)
  Wenn /(Pilz bewegt)
      Text 'Mit einem leisen "Plopp!" ziehst du
          den Pilz aus dem lockeren Waldboden.'
EndeAusf
```

12

```
ObjAttr probiert

Obj     Pilz
...
VorAusf
    (essen)
    Wenn (Pilz probiert) dann
        Text 'Du beißt herzhaft in den Pilz. Aber
            erstens schmeckt der rohe Pilz
            ziemlich bescheiden und zweitens ist
            er hochgiftig.'
        Gestorben
    sonst
        Text 'Du knabberst ein wenig an dem Pilz.
            Irgenwie schmeckt er eigenartig, aber
            du kannst nicht genau sagen, warum.'
        AttrHin Pilz probiert
    Ende
```

```
    Stop
EndeAusf
```

13

```
Obj     Pilz
...
VorAusf
    (nehmen)
    Bed (Taschenmesser beiMir)
        'Der Pilz sitzt fest im Boden, du kannst
        ihn nicht einfach herausziehen.'
    Bed (Taschenmesser offen)
        'Gute Idee, aber das Messer ist noch zu.'
    ...
EndeAusf
NachAusf
    (nehmen)
     Text 'Du schneidest den Pilz mit deinem
          Taschenmesser sauber knapp über dem
          Boden ab.'
EndeAusf
```

14

```
Obj     Blechkasten
Name    'Blechkasten' m
vor     'blech'
subst   'kasten' m  'kiste' m
Ort     Keller
Attr    Behälter
Zust    geschlossen
Besch   'Vorne auf dem Blechkasten ist ein großer,
        roter Knopf.'
Darin   '[Liste 0] [ist] im Blechkasten.'
VorAusf
  (öffnen schließen öffnen_mit aufschließen)
        Stop 'Das geht so nicht.'
EndeAusf

Obj     roter_Knopf
Name    'rot^ Knopf auf dem Blechkasten' m
adj     'rot' 'groß'
Subst   'knopf' m
Ort     auf Kasten
VorAusf
    (drücken)
    Wenn (Blechkasten offen) dann
        ObjZust Blechkasten geschlossen
        Text 'Klick! Der Kasten schnappt
            wieder zu!'
    sonst
        ObjZust Blechkasten offen
        Text 'Klick! Der Kasten schnappt auf!'
```

```
    Ende
    Stop
EndeAusf
```

15

```
Obj     Wäscheschacht
Name    'Wäscheschacht' m
vor     'wäsche'
subst   'schacht' m
Ort     Im_Haus
Attr    immobil Behälter
Besch   'Schwarz und großmäulig, so schluckt der
        Wäscheschacht alles, was man ihm gibt.'
Erst    'In der Nordwand ist eine gähnende,
        schwarze Öffnung – ein Wäscheschacht.'
NachAusf
    (empfangen)
    Text 'Du legst [den aObj] in den Schacht, und
        [er] rutsch[t] bald außer Sichtweite.'
        ObjNach aObj Keller
EndeAusf
```

16 Geschenkt, oder? Zuerst der Korb:

```
Obj     Wäschekorb
Name    'Weidenkorb' m
vor     'weiden' 'wäsche'
subst   'korb' m   'körbchen' n
Attr    Liege Sitz In_Obj Behälter
Vol     100
Ort     Keller
Besch   'Der Korb ist für einen Wäschekorn
        verhältnismäßig groß. (War vielleicht als
        Gondel für einen Fesselballon gedacht.)'
VorAusf
    (nehmen)
    Stop 'Der Korb ist sehr groß und sehr schwer.
        Lass ihn besser stehen.'
EndeAusf
```

Dann eine Änderung beim Wäscheschacht:

```
Obj     Wäscheschacht
...
NachAusf
    (empfangen)
    Text 'Du legst [den aObj] in den Schacht, und
        [er] rutsch[t] bald außer Sichtweite.'
    ObjNach aObj in Wäschekorb
EndeAusf
```

17 Dazu definieren wir zwei Objekte:

```
Obj     Mingvase
Name    'Ming-Vase' f
Adj     'schön' 'wertvoll' 'prachtvoll'
Adj     'unbezahlbar' 'teuer' 'chinesisch'
Adj     'unermeßlich' 'sündhaft' 'wunderschön'
Adj     'selten' 'rar' 'verziert' 'einmalig'
vor     'ming'
Subst   'vase' f  'rarität' f
Vol     4
Attr    Behälter
Ort     auf Tisch
Besch   'Diese Ming-Vase ist eine der letzten aus
        der zweiten Ming-Dynastie und ist eine
        unbezahlbare Rarität. Jeder halbwegs
        gescheite Archäologe würde sich nach
        diesem Stück die Finger lecken.'
VorAusf
    (zerstören werfen)
        Text 'Die Vase zersplittert in tausend
            Scherben. Gut gemacht!'
        Tausche Mingvase Scherben
        ObjNach Scherben aRaum
EndeAusf

Obj     Scherben
Name    'tausend Scherben' p 2
adj     'ehemalig' 'einstig'
vor     'ex'
subst   'scherben' p   'vase' f
Besch   'Diese Scherben waren eine der letzten
        Vasen aus der zweiten Ming-Dynastie und
        waren eine unbezahlbare Rarität. Jeder
        halbwegs gescheite Archäologe würde sich
        den Idioten in die Finger wünschen, der
        sie zerstört hat.'
```

18

```
Obj     Mingvase
...
VorAusf
    (hinlegen hineinlegen)
        Bed (aObj2 = Tisch)
            'Dort sollte ich ein so wertvolles
            Artefakt nicht abstellen.'
EndeAusf
```

19

```
Bef     xyzzy
```

```
Name    'Das Wort "xyzzy" sagen'
Verb    'xyzzy'
Ausf
    Bed /(aRaum = Im_Haus)
        'Nichts passiert...'
    Text 'Dir wird kurz schwindlig, und deine
        Umgebung verschwimmt. Du befindest Dich
        plötzlich woanders:[#]'
    GeheZu Im_Haus
EndeAusf
```

Hier können natürlich keine Befehle in runden Klammern angegeben werden, da es sich um den Hauptausführungsblock des Befehls selbst handelt.

20

```
Bef     debug_status *
Name    'status'
Verb    'status'
Syntax  dasObj (Allg)
Ausf
    Text '[aObj]

        Gewicht: [aObj.Gew]
        [x]Volumen: [aObj.Vol] [x]'
    Wenn (aObj fest immobil)
        Text 'unbeweglich[x]'
    Wenn (aObj Behälter)
        Text 'Behälter[x]'
    Wenn (aObj Ablage)
        Text 'Ablage[x]'
    Wenn /(aObj normal)
        Text '[Zust aObj][x]'

    | Aus normal.adv geklaut
    Text 'Ort:'
    Wenn (aObj BeiMir) Text  'in der Hand'
    Wenn (aObj Angezogen) Text  'angezogen'
    Wenn (aObj in Nirwana) Text 'im Nirwana'
    Wenn (aObj in Nirgendwo) Text  'nirgendwo'
    MutterObj xObj aObj
    Wenn (xObj = 0) dann
        StammRaum xRaum aObj
        Wenn (xRaum > 0) und (xRaum < Nirwana)
            Text  '[xRaum]'
    sonst
        Wenn (aObj auf xObj) Text 'auf [dem xObj]'
        Wenn (aObj in xObj)  Text 'in [dem xObj]'
        Wenn (aObj bei xObj) Text 'bei [dem xObj]'
        Wenn (aObj an xObj)  Text 'an [dem xObj]'
      Ende
    Text '[x]'

    Wenn (aObj bewegt) Text 'Bereits bewegt.[x]'
    Text '[x]'
```

```
EndeAusf
```

Der Stern nach dem Befehlsnamen definiert ihn als Steuer- oder Meta-Befehl. Er gehört dann nicht zum Spiel selbst, verbraucht keinen Spielzug und kann nicht rückgängig gemacht werden. Andere Meta-Befehle sind z.B. „Ende“, „speichern“, „Knapp“, „Punkte“ oder „Manuskript“.

Der Zusatz **(Allg)** in der **Syntax**-Zeile heißt, dass das Objekt an jedem beliebigen Ort sein kann, der Spieler muss es nicht sehen können. Dieser Zusatz wird im Allgemeinen für Gesprächsthemen oder ähnliches benutzt.

21

```
Obj     Tisch
...
VorAusf
    (empfangen)
    Bed /(aObj = Scherben)
        'Der Tisch ist auschließlich für intakte
        Mingvasen gedacht.'
    Bed (aObj = Mingvase)
        'Der Tisch ist auschließlich für die
        Mingvase gedacht.'
EndeAusf
```

22

```
ObjAttr Wirkung_bekannt

ObjKlasse Zaubertrank
Plural  'Zaubertränke' p
vor     'zauber'
subst   'trank' m  'elixier' n
Var     Wirkung 2
Besch Ausf
    Wenn (selbst Wirkung_bekannt) dann
        Text '[Der selbst]'
        Jenach selbst.Wirkung
        (1)       Text '[hat] eine heilende
                  Wirkung.'
        (2)       Text '[ist] ein Gift.'
        (3)       Text 'still[t] den Hunger.'
        (sonst)   Text '[hat] eine mir unbekannte
                  Wirkung.'
        Ende
    sonst
        Text 'Du hast noch nicht herausgefunden,
            was [der selbst] bewirkt.'
    Ende
EndeAusf
VorAusf
    (trinken)
```

```
    Text 'Du trinkst [den selbst].'
    Jenach selbst.Wirkung
    (1)     Text 'Du fühlst dich besser.'
            Inkr Gesundheit
    (2)     Text 'Du fühlst dich auf einmal
                ziemlich schlecht.'
            Dekr Gesundheit
            Wenn (Gesundheit = 0) dann
                Text 'In der Tat so schlecht, dass
                du auf den Boden fällst, und dir
                den Magen hälst. Kurz darauf
                stirbst du an dem Gift.'
                Gestorben
            Ende
    (3)     Text 'Du fühlst dich satt.'
            Wenn (Hunger > 0) Dekr Hunger
    (sonst) Text 'Du fühlst dich... Ja, wie
            eigentlich? (Hey, du zweitklassiger
            Programmierer, Wirkung kann nur 1, 2
            oder 3 sein!)'
    Ende
    ObjNach selbst Nirwana
    AttrHin selbst Wirkung_bekannt
    Stop
EndeAusf

Obj     blauer_Trank (Zaubertrank)
Name    'blau^ Zaubertrank' m
lName   'blau^' m
Adj     'blau'
Var     Wirkung 1
Ort     beiMir

Obj     milchig_Trank (Zaubertrank)
Name    'milchig^ Zaubertrank' m
lName   'milchig^' m
Adj     'milchig'
Var     Wirkung 3
Ort     in Safe
```

Die Hauptsache ist hier wohl die Vor-Ausführung für **`trinken`**, die die Wirkung der Heiltränke benutzt. Mit dieser Klassendefinition fallen dann die einzelnen Objektblöcke relativ übersichtlich aus. Als Vokabular muss hier nur die Farbe angegeben werden.

Zur Abrundung gibt es ein paar Extras: Anstatt eines Strings kann man für Besch auch einen Ausführungsblock angeben. Das funktioniert auch bei **`Text`**, **`Erst`** und den Listen **`Darin`**, **`Darauf`** und **`Dabei`** und bei **`Besch`** und **`Name`** für Räume.

Die Bewandtnis von **`Plural`** und **`lName`** wird genau im Abschnitt 8.4. des Tutoriums beschrieben.

Wie man die Wirkung der Tränke herausfindet, ohne sie zu trinken, wird hier leider nicht beschrieben.

23

```
ObjKlasse rostige_Münze
Name    'rostig^ Münze' f
Plural  'rostig^ Münzen' p
Adj     'rostig'
Subst   'münze' f  'münzen' p
Besch   'Man kann nicht erkennen, was für eine
        Münze es einmal war. Ein guatemaltekischer
        Quetzal? Ein Maria-Theresien-Taler? Oder
        eine Öresund-Öre von 1782? - Wer weiß.'

Obj     rostig1 (rostige_Münze)
Ort     beimir

Obj     rostig2 (rostige_Münze)
Ort     in Wäschekorb

...

Obj     Krügerrand
Name    'krügerrand' m
Vor     'krüger'
Subst   'rand' m  'münze' f
Besch   'Unter einem springenden Gnu steht der
        Wahlspruch "Soli Deo Gloria".'

Obj     Halfpenny
Name    'Halfpenny' m
Adj     'halb'
Vor     'half'
Subst   'penny' m
Besch   'Obwohl es nur ein Halfpenny ist, hat er
        die Form eines ganzen Kreises. Das
        britische Wappen prangt auf der
        Rückseite.'

Obj     Lösungsmittel
Name    'Glas mit Lösungsmittel' n
Vor     'lösungs' 'rost'
Ort     Keller
Subst   'glas' n  'gefäß' n  'mittel' n
Subst   'chemikalie' f
Besch   'In dem Glas schwimmt eine türkise
        Chemikalie, die relativ aggressiv aus/-/
        sieht. Laut Etikett ein Rostlösemittel.'
VorAusf
    (empfangen)
    Bed (aObj rostige_Münze)
        'Es ist ein Rostlösemittel. Du tust am
        besten nur rostige Sachen hinein.'
    Bed (Handschuhe angezogen)
        'Ohne Handschuhe packe ich nicht in diese
        Brühe!'
    Jenach aObj
        (rostig1) Sei xObj Krügerrand
```

```
        (rostig2) Sei xObj Halfpenny
        (rostig3) Sei xObj Zloty
        (rostig4) Sei xObj Zechine
        ...
    Ende
    Text 'Du hältst die rostige Münze für einige
        Zeit in die türkise Lösung. Nachdem der
        gröbste Rost weg ist, siehst du, dass die
        Münze [ein xObj] ist.'
    Tausche aObj xObj
EndeAusf

Obj     Handschuhe
Name    'Paar Handschuhe' n
subst   'handschuhe' p  'handschuh' m  'paar' n
Attr    Kleidung
Ort     angezogen
Besch   'Knallgelb, ziemlich dick, und aus Gummi.'
```

Mit der Münzenklasse werden zunächst alle rostigen Münzen erzeugt. Parallel dazu werden die individuellen Münzen erzeugt, die aber im nirgendwo sind. Eine rostige Münze tauscht mit der individuellen den Ort, wenn sie ins Rostlösebad gehalten wird. Dabei ist die Münze, die anfangs bei mir ist, immer der Rand, die im Korb immer der Halfpenny und so weiter.

Die Handschuhe sind natürlich nur zum Schutz des Spielers gedacht und haben mit der ursprünglichen Aufgabenstellung nichts zu tun.

24

```
Flagge  Troll_bezahlt

Obj     Troll
Name    'Troll' m
Subst   'troll' m
Ort     Im_Haus
Attr    Person
Besch   'Der Troll ist häßlich, so wie man es von
        einem Troll erwartet.'
Erst    'Ein grimmiger Troll steht hier mitten im
        Raum.'
Dabei   'Er spielt mit [liste 2].'
VorReakt
    (gehen)
    Bed /(aRitg = r) oder (Troll_bezahlt)
        '"Halt, Freundchen!", grunzt der Troll,
        "Wenn Du dort hinunter gehen willst, musst
        du mir was zum Naschen geben."'
EndeAusf
VorAusf
    (gegeben)
        Bed (aObj Verpflegung)
            '"Das ist nichts zum Essen. Ich will
            es nicht", schnarrt der Troll.'
        ObjNach aObj bei Troll
```

```
        Text '"Danke!", sagt der Troll relativ
            freundlich.'
        Setze Troll_bezahlt
        Stop
    (fragen erzählen)
        Stop 'Der Troll brummelt nur vor sich hin.'
EndeAusf
BefAusf
    (sonst)
        Text 'Der Troll brummt unverständliches
            Zeug.'
EndeAusf
```

25

```
Flagge Dämon

Aktion *
Ausf
    Inkr Dämon
    Jenach Dämon
    (4)     Wenn (Wanderer hier)
            Text 'Der Wanderer geht nach Süden'
            ObjNach Wanderer Vor_dem_Haus
            Wenn (Wanderer hier)
            Text 'Der Wanderer kommt aus dem Wald
                geschlendert.'
    (6)     Wenn (Wanderer hier)
            Text '"Schöne Blumen!", fühlt sich der
                Wandersmann ermuntert zu sagen.'
    (12)    Wenn (Wanderer hier)
            Text 'Der Wanderer betritt das Haus.'
            ObjNach Wanderer Im_Haus
            Wenn (Wanderer hier)
            Text 'Der Wanderer kommt von draußen
                herein.'
    ...
    (46)    Wenn (Wanderer hier)
            Text 'Der Wanderer saugt demonstrativ
                die frische Waldluft ein.'
    Ende
    wenn (Dämon = 48) lösche Dämon
EndeAusf
```

So funktioniert’s: Für jeden Zug, in dem etwas geschehen soll, einen **`Jenach`**-Block schreiben. Bei der Textausgabe muss man natürlich darauf achten, dass der Wanderer am selben Ort ist, wie der Spieler.

Wenn **`Dämon`** bis 48 hochgezählt wurde, fängt das ganze Spiel von vorne an. Dabei sollte sichergestellt sein, dass der Wanderer wieder auf der Lichtung ist.

Zählvariablen, die jeden Zug hoch- oder herunterzählen, heißen *Dämonen*, und deshalb habe ich unsere Flagge auch so genannt. Die meisten Systeme haben ein ausgereifteres Konzept dazu, aber was einfach ist, muss ja nicht unbedingt schlecht sein. Wichtig ist, dass man daran

denkt, den Dämon in jeder Runde um eins zu erhöhen.