AOQML Tutorial

(Stand: 30. Mai 2008)

Inhaltsverzeichnis

AOQML Tutorial....3
Grundsätzliches zum Aufbau – XML (geändert am 30.05.2008)....3
Die Startszene und Szenenwechsel....4
Vorbedingungsprüfung und und Schlussszenen (geändert 17.05.2008)....5
Szenenteile einfügen (neu am 14.05.2008)....6
Ausrüstung abfragen, aufnehmen und ablegen (neu am 14.05.2008)....6
Zufallsauswahl (geändert am 29.05.2008)....7
Fallunterscheidungen (neu am 14.05.2008)....8
Proben (geändert am 29.05.2008)....8
Proben-Erleichterungen durch Ausrüstung (neu am 14.05.2008)....10
Gespeicherte Proben und stille Proben (geändert am 29.05.2008)....10
Variablen (neu am 14.05.2008, geändert am 19.05.2008)....11
SQL Ausdrücke (neu am 29.05.2008)....12
Kämpfe (neu am 30.05.2008)....13

AOQML Tutorial

Grundsätzliches zum Aufbau – XML (geändert am 30.05.2008)

In diesem Tutorial soll Schritt für Schritt ein kleines Quest mit AOQML aufgebaut werden. AOQML ist eine HTML-ähnliche Sprache, die für Antamar Zufallsbegegnungen (ZB) und Quests entwickelt wurde und derzeit in Antamar implementiert wird, es bedeutet *Avesjünger und Ordenskrieger Quest Modelling Language*.

Wie gesagt, ist AOQML ähnlich zu HMTL, genau genommen XHTML. Es ist also grundsätzlich erst einmal Text, der auch ausgegeben wird, und dazwischen sind sogenannte *Tags* eingestreut, welche den Ablauf und die Darstellung beeinflussen. Z.B. bedeutet `<p>Dieses ist ein Absatz.</p>`, dass "*Dieses ist ein Absatz.*" als eigenständiger Absatz dargestellt wird. Das `<p>`-Tag erzeugt also Absätze. Und jedes Tag hat einen Anfang, eben `<p>`, und ein Ende, hier also `</p>`.

Start-Tag und Ende-Tag müssen immer paarig sein. Sie dürfen verschachtelt werden, aber nicht gekreuzt. Folgendes ist also falsch:`<p>... <challenge ...>...</p>...</challenge>`.

Des weiteren wird jede Szene mit dem die Spieler in einem Quest konfrontiert werden in ein eigenes Dokument geschrieben. Eine reine Zufallsbegegnung wäre also genau genommen ein Quest mit einem einzigen solchen Dokument. Jedes solche Dokument enthält zunächst einen Rahmen aus `<scene>...</scene>`-Tags. Hier ein Beispiel:

entenfamilie.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<p>Du siehst eine Ente mit Küken.</p>
</scene>
```

Dieses Beispiel wäre also praktisch eine ganz simple Zufallsbegegnung, die nur den einen Satz "Du siehst eine Ente mit Küken." ausgibt, sonst nichts.

Ein XML Dokument, welches auf der obersten Stufe genau ein (öffnendes und schließendes) Tag hat (hier `<scene>...</scene>`, und bei dem alle Tas paarig sind und das noch ein paar weitere Bedinungen erfüllt, nennt sich *wohlgeformtes XML-Dokument*. `<?xml version="1.0" encoding="UTF-8"?>` ist außerdem eine XML-Steueranweisung und nicht etwa ein zweites Tag auf oberster Ebene, daher wird diese auch nicht geschlossen, wie es für XML-Tags notwendig ist.

Hätte ein Tag gar keinen Inhalt, kann man statt `<p>...</p>` auch kurz `<p/>` schreiben. Die AOQML-Engine fügt dann in einigen Fällen einen passenden automatischen Inhalt ein.

An jeder Stelle, an der ein Tag stehen kann, kann auch ein Kommentar stehen:

kommentar.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
...
<!-- Dieses ist ein Kommentar, der sich auch über mehrere Zeilen hinziehen kann.
Sogar Tags sind erlaubt: <p>bla</p> und werden hier ignoriert.
-->
...
</scene>
```

TIPP: Die XML Syntax kann auch online überprüft werden: http://www.validome.org/xml/

Soweit der der grundsätzliche Aufbau eines jeder Quest-Szene. Als nächstes wollen wir uns der Startszene zuwenden.

Die Startszene und Szenenwechsel

Jedes Quest wird in einem eigenen Unterverzeichnis abgelegt. Die Startszene heißt dabei immer `start.xml` und ist natürlich wie oben beschrieben als gültiges XML Dokument aufgebaut Die Startszene markiert zunächst das Quest als „gestartet“, was mit dem `<quest>`-Tag geschieht.

Als nächstes wollen wir dem Spieler einen Text ausgeben. Hier verwenden wir als Kennzeichnung für einen Absatz einfach das `<p>`-Tag von XHTML.An einigen Stellen wollen wir direkte Rede anzeigen, hierfür verwenden wir das `<q>`-Tag von XHTML.

Zuletzt wollen wir dem Spieler ermöglichen, sich zwischen zwei Folgeszenen zu entscheiden. Damit ist dann auch die Startszene beendet, denn nun muss die Ausgabe erst dem Spieler angezeigt werden und dieser muss seine Wahl treffen, bevor es weiter gehen kann.

start.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<quest status="running"/>
<p>Gerade hast du dich mit einem frisch gezapften Bier niedergelassen, als dir die
Blicke auffallen, mit denen dich ein ebenfalls im Schankraum sitzender, wohlhabend
aussehender Bürger mustert. Er scheint nervös zu sein, vor allem als sein Blick auf
deine Waffen fällt, rafft sich dann aber dennoch auf und tritt zu dir. </p>

<p><q>Ähm, also, ja... Seid ihr zufällig an einem Auftrag interessiert? Es könnte
gefährlich werden, aber ich bin bereit, euch gut zu bezahlen.</q></p>
<ul>
    <li><choice target="ausruhen">Nö, keine Lust.</choice>
              Du bist hier um dich auszuruhen, nicht um irgendwas gefährliches zu machen.
</li>
    <li><choice target="anhoeren">Tja, warum nicht.</choice>
              Erst mal hören, was er will. </li>
</ul>
</scene>
```

Die Auswahlmöglichkeiten, die mit <choice>-Tags ausgezeichnet werden, haben wir außerdem etwas hübscher in einer Liste mit `<ul>` und `<li>` dargetellt, wie wir sie aus XHTML kennen.

Der Name der Folgeszene wird jeweils im Attribut `target` des `<choice>`-Tags angegeben. Das muss der Dateiname sein, nur ohne die Extension `.xml`. Unterverzeichnisse sind allerdings möglich und können in großen Quests der Strukturierung dienen. Als Trennzeichen im Pfad muss dann ein „`/`“ verwendet werden. Daraus generiert die AOQML-Engine dann Links. Und der Text, der zwischen dem öffnenden und schließenden `<choice>`-Tag steht, wird als Link-Text angegeben.

> Gerade hast du dich mit einem frisch gezapften Bier niedergelassen, als dir die Blicke auffallen, mit denen dich ein ebenfalls im Schankraum sitzender, wohlhabend aussehender Bürger mustert. Er scheint nervös zu sein, vor allem als sein Blick auf deine Waffen fällt, rafft sich dann aber dennoch auf und tritt zu dir.
>
> *"Ähm, also, ja... Seid ihr zufällig an einem Auftrag interessiert? Es könnte gefährlich werden, aber ich bin bereit, euch gut zu bezahlen."*
>
> - Nö, keine Lust. Du bist hier um dich auszuruhen, nicht um irgendwas gefährliches zu machen.
> - Tja, warum nicht. Erst mal hören, was er will.

Angezeigt würde diese Szene in etwa so:

Vorbedingungsprüfung und und Schlussszenen (geändert 17.05.2008)

Hinter der Auswahl „Nö , keine Lust“ soll sich in unserem Beispiel ein Ende des Quests befinden:

ausruhen.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<p>Enttäuscht geht der Mann wieder, und du kannst in Ruhe dein Bier trinken. </p>
<quest status="aborted"/>
</scene>
```

Hier passiert nicht viel. Der eine Absatz wird angezeigt und das Quest wird als vom Spieler abgebrochen markiert. Ein so beendeter Quest kann dem Helden wieder angeboten werden.

Die Möglichkeiten, ein Quest zu beenden sind:

- `rejected:` Die Vorbedingungen zum Quest sind nicht erfüllt.
  (Diese Variante kann nur in der Startszene verwendet werden.)
- `aborted`: Der Spieler hat sich entschieden, das Quest abzubrechen.
  Ein solches Quest kann dem Helden ggf. erneut angeboten werden.
- `ended`: Das Quest ist in einer Weise zu Ende gekommen, bei der es Sinn macht, dass dem Helden dieses Quest erneut angeboten werden kann.
- `finished`: Das Quest wurde auf eine Weise beendet, bei der es keinen Sinn macht, es dem Helden erneut anzubieten. D.h. diesem Helden wird dieses Quest nie wieder angeboten werden.
- `failed:` Das Quest wurde nicht erfolgreich beendet, kann aber auch nicht wieder angeboten werden.

In jedem der o.g. Fälle wird das Quest sofort beendet, d.h. der Rest der Szene nicht mehr angezeigt.

Oft werden am Ende Abenteuerpunkte vergeben. Das geschieht so:

questende-beispiel-1.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<p>Stolz blickts du auf dieses Abenteuer zurück ...</p>

<set attribute="AP" inc="10"/>
<quest status="finished"/>

</scene>
```

Natürlich geht das jederzeit auch mitten im Quest.

Falls Abenteuerpunkte aber zwischendurch nur festgelegt werden soll, die Vergabe aber erst beim Abschluss des Quests erfolgen soll, kann dies derzeit mithilfe von `<retain>` und `<replay>` realisiert werden. Irgendwo im Quest werde hier z.B. 5 AP für später aufgehoben:

ap-vergabe-beispiel-start.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<quest status=“running“/>
...
<retain name=“AP-1“ id=“AP-1a/>
...
</scene>
```

Obiger Code initialisiert quasi ein Unterprogramm mit dem Namen AP-1 ohne Inhalt. Das ist wichtig, damit beim Abruf in der Schlussszene kein Fehler auftritt, falls diese AP niemals vergeben

werden, weil der entsprechende Zweig nicht durchlaufen wurde. Jedes `<retain>`-Tag muss außerdem eine in der Szene eindeutige ID besitzen, die mit `id="..."` angegeben wird. Darüber wird der Zweig wiedergefunden, wenn er mit `<replay>` abgespielt werden soll.

ap-vergabe-beispiel-irgendwo.xml

```
...
<retain name="AP-1" id="AP-1b"><set attribute="AP" inc="5"/></retain>
...
```

Die vorstehenden Anweisungen kommen dann irgendwo im Laufe des Quests vor.

ap-vergabe-beispiel-ende.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
<p>Stolz blickst du auf dieses Abenteuer zurück ...</p>

<replay name="AP-1"/>
<quest status="finished"/>

</scene>
```

Und hier, in der Schlussszene, erfolgt dann tatsächlich die AP-Vergabe, indem der vorher gemerkte Programmcode zur Ausführung gebracht wird.

Szenenteile einfügen (neu am 14.05.2008)

Oftmals kommt es vor, dass ein und dieselbe Teilszene in mehreren Szenen vorkommt. Das kann einfach ein längerer Absatz sein, der erläutert, was man sieht. Oder auch Programmcode.

include-beispiel.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
...
<include target="tuer"/>
...
</scene>
```

Im obigen Ausschnitt aus einer AOQML-Szene wird die Szene `tuer.xml` eingefügt. Dies verhält sich genau so, als wenn die in `tuer.xml` innerhalb des dortigen `<scene>`-Tags befindlichen Anweisungen direkt an dieser Stelle in `include-beispiel.xml` stünden.

Das `<include>`-Tag ist damit auch ein typisches Beispiel für ein Tag ohne Inhalt. Daher wurde es sofort mit `.../>` geschlossen.

Ausrüstung abfragen, aufnehmen und ablegen (neu am 14.05.2008)

Jeder Held trägt diverse Ausrüstungsgegenstände mit sich herum. Diese können in AOQML abgefragt werden, es können Ausrüstungsgegenstände aufgenommen (z.B. gefunden) werden und abgelegt (z.B. verloren, abgenommen) werden. Die dazu verwendeten Tags sind `<has>`, `<take>` und `<drop>`. Das folgende Beispiel zeigt ihre Verwendung:

ausruestung-beispiel.xml

```
...
<has item="Schwert>
	<success min="2">
		Du schenkst ihm eines deiner Schwerter. <drop item="Schwert"/>
	</success>
	<failure>
		Er schenkt dir ein Schwert. <take item="Schwert"/>
	</failure>
</has>
...
```

Hier ist zu beachten, dass die Schreibweise des Gegenstandes 100%ig mit dem Namen in der Datenbank übereinstimmen muss. Man kann auch SQL-Wildcards benutzen, also „%" für eine Folge beliebiger Zeichen und „?" für ein einzelnes beliebiges Zeichen. Für <has> gibt es speziell noch die Möglichkeit, mehrere Item-Namen mit einem logischen Und „+" oder einem logischen Oder „|" zu verknüpfen. Dies ist hier dargestellt:

ausruestung-beispiel-2.xml

```
...
<has item="%Messer|%Dolch>
	<success>
		Gut, dass du etwas zum schneiden dabei hast.
	</success>
	<failure>
		Leider kannst du kein Messer oder etwas ähnliches bei dir finden.
		Und große Waffen nützen dir für diese Arbeit auch nichts.
	</failure>
</has>
...
```

Zufallsauswahl (geändert am 29.05.2008)

Oft möchte man zufällige Elemente einbringen, um das Spiel abwechslungsreicher zu gestalten. Wir kennen das von den ZB in Antamar. Das ist mit der Kombination aus <random> und <case> realisierbar:

zufalls-beispiel.xml

```
...
<p>Du siehst
<random>
	<case>einen Hasen über ein Feld hoppeln.</case>
	<case>eine kleine Kuhherde auf einer Weide.</case>
	<case>am Waldrand ein Reh stehen.</case>
</random>
</p>
...
```

Eine völlig andere Art von Zufalllsauswahl sind nummerische Wertebereiche, zB. für die Anzahl von Elementen:

zufalls-bereich-beispiel.xml

```
...
Du findest einige Flaschen Wein <take item="Bitterwein" count="2...5"/>.
...
```

Und das kann man auch würfeln lassen:

zufalls-würfel-beispiel.xml

```
...
Du findest einige Flaschen Wein <take item="Bitterwein" count="$[2W6]"/>.
...
```

Fallunterscheidungen (neu am 14.05.2008)

variablen-beispiel-1.xml

```
...
<p>Plötzlich wirst du von hinten angesprochen: <q>
<switch attribute=“gender“>
	<case value=“male“>Holder Recke,</case>
	<else>Edle Dame,</else>,
</switch>
könnt Ihr mir vielleicht sagen, wo ich hier einen Schmied finde?</q></p>
...
```

Dabei wird immer nur der erste passende Unterzweig ausgeführt. Falls keiner passt, und kein <else>-Zweig angegeben ist, wertet die <switch>-Anweisung also zu nichts aus, was duchaus in einigen Fällen gewollt sein kann.

Proben (geändert am 29.05.2008)

Jetzt wird es spannend: Unser Held soll eine Probe ablegen. Zunächst ein einfaches Beispiel mit einer Sinnenschärfe-Probe:

einfache-probe.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>
...
<p>Du siehst
<challenge talent="Sinnenschärfe">
	<failure>Bäume, Bäume und noch mehr Bäume!</failure>
	<success>
		Der Wald ist voller Bäume, doch dir fällt auf, dass sie an einer
		bestimmten Stelle zu deiner Rechten in Reih und Glied stehen.
		Zwischen ihnen scheint ein seltsames Licht zu leuchten.
	</success>
</challenge>
</p>
...
</scene>
```

Mit dem <challenge>-Tag wird also eine Probe eingeleutet und je nachdem, ob sie gelungen ist oder nicht, wird der Unterzweig <failure> oder <success> ausgeführt. Wichtig ist hier, dass zwischen dem <challenge>-Tag und den Untertags <success> bzw. <failure> keine Zwischenebene eingezogen werden darf. Also dazwischen dürfen keine anderen XML-Tags stehen!

Aber Proben können auch komplexer sein, so kann z.B. die Probe mit dem modification-Attribut erschwert oder erleichtert werden. Und innerhalb der <success> und <failure> Tags können auch andere Anmoweisungen stehen.

einschleichen.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<scene>

<p>Betont unauffällig treibst du dich ein wenig im Gasthaus der Streunerin herum und
wartest, bis sie es verlässt. Dann, in einem unbeobachteten Moment, schlenderst du die
Treppe nach oben zu den Gästezimmern. Es ist ruhig, auf dem Flur ist niemand zu sehen,
aber dennoch solltest du erst einmal leise vorgehen, damit keine anderen Gäste oder das
Personal auf dich aufmerksam werden. </p>

<challenge talent="Schleichen" mod="+5">
       <success><include target="tuer"/></success>
       <failure min="-2"><include target="ausrede"/></failure>
       <failure><include target="flucht"/></failure>
</challenge>

</scene>
```

Dem aufmerksamen Leser fällt sicherlich noch etwas auf: Es gibt zwei `<failure>` -Zweige. Der erste ist mit dem Attributwert `min="-2"` versehen und wird daher nur ausgeführt, wenn die TaP (nach der Probe übrigen Talentpunkte) größer oder gleich (eben „min“) -2, also -1 oder -2, betragen.

Es wird immer genau der erste und nur der erste passende Unterzweig ausgeführt. Bei -1 oder -2 TaP wird also nur der mittlere Zweig ausgeführt, und der letzte Zweig übersprungen, obwohl er alleine betrachtet auch passen würde.

Im negativen Bereich muss man mit größer/kleiner oder gar minimal/maximal etwas verdreht denken. Andererseits können bei misslungener Probe (`<failure>`-Zweigen) Daher kann man die Werte alternativ auch doppelt““ umdrehen, aus „min“ wird „max“ (und andersherum) und aus negativen Zahlen werden positive. D.h. die Probe von oben ist identisch mit:

einschleichen2.xml

```
...
<challenge talent="Schleichen" mod="+5">
       <success><include target="tuer"/></success>
       <failure max="2"><include target="ausrede"/></failure>
       <failure><include target="flucht"/></failure>
</challenge>
...
```

Und auch das ändert nichts:

einschleichen2.xml

```
...
<challenge talent="Schleichen" mod="+5">
       <success><include target="tuer"/></success>
       <failure max="2"><include target="ausrede"/></failure>
       <failure min="3"><include target="flucht"/></failure>
</challenge>
...
```

Und immer daran denken: Nur erste passende Zweig wird ausgeführt!

Mit dem Attribut `quality` können außerdem auch Proben auf Eigenschaften (MU, KL; IN, CH, FF, GE, KO, KK) abgelegt werden:

eigenschafts-probe.xml

```
...
<challenge Quality="Charisma" mod="-2">
      <success>
            Die Frau erzählt dir, dass der Mann in Richtung Firum abgereist ist<.
      /success>
      <failure>Die Frau guckt dich nur genervt an.</failure>
</challenge>
...
```

Proben-Erleichterungen durch Ausrüstung (neu am 14.05.2008)

Oft macht es Sinn, Proben zu erleichtern, wenn der Held bestimmte Gegenstände bei sich führt. Hierfür muss man den Gegenstand abfragen und je nach Ergebnis verschiedene Modifikatoren angeben:

schlossknacker-beispiel.xml

```
...
<has item="Satz Dietriche">
    <success>
        <challenge talent="Schlösser Knacken" mod="-2">
            <failure><include target="_mist"/></failure>
            <success><include target="_zimmer"/></success>
        </challenge>
    </success>
    </failure>
        <challenge talent="Schlösser Knacken" mod="+0">
            <failure><include target="_mist"/></failure>
            <success><include target="_zimmer"/></success>
        </challenge>
    </failure>
</has>
...
```

Um die Fälle für Proben-Erfolg und -Misserfolg nicht doppelt codieren zum müssen, habe ich diese Teile ausgelagert und mit `<include>` eingefügt.

Eine andere Variante wird im Folgenden Abschnitt gezeigt.

Gespeicherte Proben und stille Proben (geändert am 29.05.2008)

Probenergebnisse können auch gespeichert werden. Dazu gibt man einfach einen Namen an:

gespeicherte-probe.xml

```
...
<challenge Quality="Charisma" mod="-2" name="Charisma1">
...
<challenge name="Charisma1">
      <success>
            Die Frau erzählt dir, dass der Mann in Richtung Firum abgereist ist<.
      /success>
      <failure>Die Frau guckt dich nur genervt an.</failure>
</challenge>
...
```

Dabei steht der Ergebnis nicht nur in der aktuellen Szene, sondern bis zum Ende des Quests zum Abruf bereit bzw. bis es überschrieben wird. Das Ergebnis kann natürlich auch direkt ausgewertet werden und später dann auch so oft man es braucht.

Des weiteren kann die Probe ausführlich (Vorgabe: *Schleichen-Probe+5: -5 TaP*), nur der Herausforderungs-Teil (*Schleichen-Probe+5*) ohne das eigentliche Ergebnis oder gar nicht (versteckte Probe) angezeigt werden:

versteckte-probe.xml

```
...
<!-- Anzeige mit Ergebnis durch Vorgabewert -->
<challenge Quality="Charisma" mod="-2"/>
...
<!-- Anzeige mit Ergebnis explizit -->
<challenge Quality="Charisma" mod="-2" show="result"/>
...
<!-- Anzeige ohne Ergebnis -->
<challenge Quality="Charisma" mod="-2" show="challenge"/>
...
<!-- versteckte Probe -->
<challenge Quality="Charisma" mod="-2" show="none"/>
...
```

Natürlich können auch stille Proben gespeichert werden, oder gespeicherte Proben still abgerufen werden etc.

Variablen (neu am 14.05.2008, geändert am 19.05.2008)

Variablen sind benannte Speicherbereiche, die an einer Stelle mit Werten belegt werden können, die an einer anderen Stelle wieder abgerufen werden können. Im folgenden Beispiel wird einmalig eine abfällige Anrede für den Helden bzw. die Heldin bestimmt, die dann mehrfach verwendet werden kann:

variablen-beispiel-1.xml

```
...
<store name="AbfaelligeHeldenAnrede">
      <switch attribute="gender">
            <case value="male">Bürschchen</case>
            <else>Weibsstück</else>
      </switch>
</store>
...
<p><q>Hey, <fetch name="AbfaelligeHeldenAnrede"/>, was suchst du da in meinen Taschen?
</q></p>
...
```

Falls der erste Buchstabe des Variablen-Inhalts in einen Großbuchstaben umgesetzt werden soll, z.B. an einem Satzanfang, kann dies mit `<fetch ... initcaps="true"/>` erreicht werden. Ein Defaultwert kann außerdem zwischen `<fetch ...>` und `</fetch>` angegeben werden.

Natürlich kann hier auch andersherum verschachtelt werden, was z.B. sinnvoll ist, wenn mehrere Variablen von der selben Fallunterscheidung gesetzt werden sollen:

variablen-beispiel-2.xml

```
...

<switch attribute="gender">
      <case value="male">
            <store name="FreundlicheHeldenAnrede">edler Recke</store>
            <store name="AbfaelligeHeldenAnrede">Bürschchen</store>
      </case>
      <else>
            <store name="FreundlicheHeldenAnrede">edle Dame</store>
            <store name="AbfaelligeHeldenAnrede">Weibsstück</store>
      </else>
</switch>
</store>
...
```

ACHTUNG: In diesem Fall blieben die Variablen undefiniert, falls keiner der Fälle im `<switch>` zutrifft. Undefinierte Variablen wirken zwar zunächst wie leere Zeichenketten, Ihre Verwendung kann aber zu einem Fehler führen, wenn sie z.b. als Talentname in einer Probe verwendet werden.

Eine undefinierte Variable ist auch nicht identisch mit einer leeren, das scheint nur auf den ersten Blick so!

start.xml

```
...
<quest status=“running“>
<store name=“FreundlicheHeldenAnrede“/>
<store name=“FreundlicheHeldenAnrede“><fetch name=“FreundlicheHeldenAnrede>
       Den Zwölfen zum Gruße</fetch></store>
<store name=“AbfaelligeHeldenAnrede“<fetch name=“AbfaelligeHeldenAnrede/>
       Hey, du wertloses Stück Dreck</fetch></store>>
...
```

Die Variable `AbfaelligeHeldenAnrede` enthält hinterher den Wert „*Hey, du wertloses Stück Dreck“* Die Variable `FreundlicheHeldenAnrede` ist hinterher aber leer. Denn der Defaultwert zwischen `<fetch ...>` und `</fetch>` wirkt nur bei unitialisierten, nicht aber bei leeren Variablen.

Variablen können auch in Attributwerten verwendet werden. In diesem Fall erfolgt der Abruf der Variable „name“ mit `${name}`. Dies ist im folgenden Beispiel gezeigt:

variable-in-attributen-beispiel.xml

```
...
<random>
    <case>
       <store name="Waffe">Schwert</store>
       <store name="Waffen">Schwerter</store>
    </case>
    <case>
        <store name="Waffe">Säbel</store>
        <store name="Waffen">Säbel</store>
    </case>
</random>

<take item="${Waffe}" count="1...3">Du findest einige <fetch name="Waffen"/>.</take>
...
```

SQL Ausdrücke (neu am 29.05.2008)

Manchmal braucht man einen einzelnen Wert aus der Antamar-Datenbank, für den aber so eine spezielle Suchanfrage benötigt wird, dass es keinen Sinn macht, dafür eine AOQML-Syntax bereitzustellen.In diesem Fall kann mit `$(...)` man auch direkt einen SQL-Ausdruck verwenden:

sql-beispiel.xml

```
...
<take item="$(name FROM ant_ware WHERE preis -le 50 AND name!='EDIT'
       AND NOT att_selten AND NOT att_rare)"/>
...
```

Da „<“ und „>“ in XML reservierte Zeichen sind, und man diese als „<“ bzw. >“ schreiben müsste, was sehr unübersichtlich ist, können alternativ die folgenden Zeichenfolgen verwendet werden:

- -lt für < (kleiner als)
- -le für <= (kleiner als oder gleich)
- -eq für = (gleich – nur der vollständigen Einheitlichkeit halber)
- -ne für != (ungleich – nur der vollständigen Einheitlichkeit halber)
- -ge für >= (größer als oder gleich)
- -gt für > (größer als)

Dieses Feature sollte mit Vorsicht verwendet werden!

Kämpfe (neu am 30.05.2008)

Es sollte zwar in Mini-Quests, die ja auch für Nicht-Kämpfer sein sollen, immer Pfade ohne Kampf geben, aber wenn der Held sich entsprechend anstellt, wird ein Kampf oft nicht zu vermeiden sein. Daher kann man in Quests natürlich auch Kämpfe aufsetzen:

kampf-beispiel.xml

```
...
  <fight>
    <friends
      <!-- z.B. Gruumsh 1:1 aus der DB -->
      <npc npcid="67"/>
    </friends>

    <rivals>
      <!-- z.B. ein Hauptmann aus der DB als Salina mit einem Säbel -->
      <npc npcid="34" name="Salina" gender="female" weapon="1717"/>
    </rivals>

    <victory>
      <include target="kampf-gewonnen"/>
    </victory>

    <escape>
      <include target="kampf-gefluechtet"/>
    </escape>

    <defeat>
      <include target="kampf-verloren"/>
    </defeat>
  </fight>
...
```

Kämpfe werden also mit dem Tag <fight> durchgeführt, Gegner mit <rivals> und Freunde mit <friends> hinzugefügt (ggf. auch mehrere). Im einfachsten Fall nimmt man sich einen Gegner aus der Datenbank, hier mit der ID 70. Das ist auch als Basis derzeit immer notwendig. Den so geladenen Gegner kann man dann noch abwandeln, indem man seinen Namen und ein Geschlecht setzt und ihm eine andere Waffe gibt (wieder per Datenbank-ID). Männlich (male) ist dabei der Vorgabewert, kann also auch weggelassen werden. Weitere Optionen (z.B. Fluchtverhalten) werden später folgen.

Wenn der Held gesiegt hat, wird der <victory>-Zweig ausgeführt, ist er geflüchtet, wird der <escape>-Zweig ausgeführt, hat er verloren, wird der <defeat>-Zweig ausgeführt. Möglichkeiten, um festzustellen, was mit den Gegnern im Kampf geschehen ist, kommen noch.