

Mission Director für Anfänger

Eine Einführung in die
Entwicklung von Missionen
für die X3 Spielereihe von
Egosoft.

v2.2



Inhalt:

Einführung	3
Mission Director - Vorbereitungen	4
XML Editoren	5
Erstellen einer neuen Missionsdatei	6
XML Formatierung	7
<gues>	7
<cue>	8
<condition>	8
Variablen	9
<timing>.....	11
Einheiten bei Zeit und Entfernung	12
Operatoren	12
<action>	13
Beispiel 1	14
Testen der ersten Mission	14
'In Space' Missions Tutorial	15
Text ID und Sprachdatei	44
Mission Briefing	46
Actor und Create_offer	49
Eine nützliche Variable	51
Benutzen von Sound Effekten	52
Testen der Mission	52
Fragen und Antworten	55
Anhang	
Anhang 1: Instanzen	56
Anhang 2: Library Cues	57
Anhang 3: Dateien aus .cat/.dat entpacken	59
Anhang 4: Links	61
Nachwort	62

Einführung

In den früheren Spielen der X-Reihe waren Missionen (Plot und non-Plot) nur umständlich zu entwickeln, schwer zu testen und waren wegen den Coding-Anforderungen von KC anfällig für Fehler. Das Ergebnis waren hängende Plots und wenige Variationen an non-Plot Missionen.

Das Ziel des Mission Director's (MD genannt) ist es, eine Benutzerfreundliche Methode anzubieten um Missionen zu schreiben, dabei aber für Nicht-Programmierer leicht Erlernbar zu sein. Dafür wurde eine XML-basierte Entwicklungsumgebung gewählt um eine einfache ‚Plug-in‘ Funktionalität und einen leichten Einstieg für Nicht-Programmierer bieten zu können. Hier kommen Sie ins Spiel...

Bereits mit wenigen Befehlen können Events erstellt werden, die zusammen genommen, eine ganze Mission bilden die der Spieler angehen kann. Dabei kann jedes Objekt, wie Schiffe oder Stationen, genutzt werden.

Es ist relativ einfach sich mit dem MD vertraut zu machen, und erfordert am Anfang nur sehr wenig Verständnis der ‚Daten-Logik‘ um Missionen zu schreiben. Mit etwas Übung wird das Verständnis dahinter wachsen, und damit auch die Komplexität des ‚Codes‘, und somit auch die Missionen die Sie in der Lage sind zu entwickeln.

Die Struktur der Missionsdateien ist sehr Geradlinig und ist somit einfach zu Testen und kann geändert werden, ohne Angst haben zu müssen, den ganzen darunterliegenden Code unbrauchbar zu machen. So kann man ein einfaches Event erstellen und es sich In-Game anschauen und Testen, etwas daran ändern, und schließlich In-Game den MD resetten und die Änderungen direkt beobachten. Diese neue Flexibilität ist eines der Schlüsselfunktionen des MD und hat dafür gesorgt das sich die Entwicklungszeiten neuer Missionen wesentlich verkürzt haben.

Beispiele

Es wurden, seit erscheinen des MD, bereits eine Menge Missionen geschrieben (zu finden im Director Ordner innerhalb der .Cat/.Dat Dateien). Ein einfacher Weg herauszufinden wie etwas funktioniert ist es sich diese Missionen anzuschauen, sie zu verändern und zu schauen was In-Game passiert. Das wird ihnen nicht nur helfen die einzelnen Befehle kennenzulernen, diese Missionen dienen auch als nützliche Bibliothek gefüllt mit Beispielen wie man seine Vorstellungen umsetzen kann.

Besonders Hilfreich ist es sich Missionen anzuschauen die der eigenen ähnlich sind, ihr könnt diese verändern oder einzelne Abschnitte für eure eigene Mission kopieren. Das sollte nicht als Nachahmen verstanden werden, sondern als Kompliment für den eigentlichen Entwickler. Diese Anleitung gibt ihnen eine Einführung in die Schlüsselemente des MD bis hin zu einigen Schritt-für-Schritt Anleitungen einfacher Missionen. Wenn Sie diesen Anleitungen folgen sollten Sie eine bessere Vorstellung bekommen wie der MD funktioniert, und sind schließlich bereit ihre eigenen Missionen in Angriff zu nehmen.

Zielgruppe

Auch wenn etwas Erfahrung im Programmieren helfen kann, ist sie jedoch keinesfalls erforderlich. In der Tat kann sie sogar hinderlich sein: MD ist Ereignis-basiert anstatt Objekt-basiert, Programmierer werden hier Umdenken müssen.

Diese Anleitung geht davon aus das Sie nur wenig oder gar keine Erfahrung im Programmieren haben, sie wurde für Anfänger geschrieben. Es wird aber angenommen das Sie Erfahrung mit dem X-Universum haben. Bleibt zu hoffen das diese Anleitung allen von nutzen ist, die diese benutzen.

Mission Director - Vorbereitungen

Der MD wird gleichzeitig mit dem Script-Editor aktiviert wenn der Spielname in ‚Thereshallbewings‘ geändert wird. Sie benötigen einen Ordner mit dem Namen ‚director‘ in ihrem Spielverzeichnis, hier kommen alle Missionen hin die sie Spielen möchten. Zum ändern, oder zum erstellen neuer Missionen, brauchen sie allerdings noch die in Bild 1 gezeigten Dateien.



Bild 1 - EgoSoft\X3 Terran Conflict\director' Ordner

Das Spiel wird versuchen alle in diesem Verzeichnis befindlichen .XML Dateien zu laden, egal wie viele es sind. Um sicherzustellen das es keine Konflikte gibt sollten alle in den Dateien vorhanden Cue's einen Einzigartigen Namen haben.

Cues, ihre Namen und verwendung wird in einem Späteren Kapitel beschrieben.

Die 6 Dateien aus Bild 1 sind in .Cat/.Dat Dateien in ihrem Spielverzeichnis, wie sie diese Dateien extrahieren und in den Ordner bekommen wird im [Anhang 3](#) erklärt.

Bei Albion Prelude ist der Ordner im Addon Unterverzeichnis (X3 Terran Conflict\addon\director) zu finden.

Hier ein Überblick wofür, einige, der Dateien gebraucht werden:

director.html

Das ist eine wirklich nützliche Datei, nicht nur für Anfänger. Sie enthält eine Liste aller Befehle des MD, und jede Menge nützlicher Variablen. Am anfang der Datei kann man die Liste Filtern, falls man etwas bestimmtes sucht. Diese Datei wird mit den Patches von EgoSoft immer auf dem neusten Stand gebracht sollte sich etwas am MD ändern.

Nachteil dieser Datei ist, dass sie nur mit dem Internet Explorer (IE) angezeigt werden kann. Sollten sie eine Nachricht vom IE bekommen dass ActiveX geblockt wird, machen sie mit der Maus einen Rechts-Klick auf die Nachricht und erlauben sie den geblockten Inhalt.

director & dirschema XSL stylesheets und director cascading stylesheet
Diese Dateien beinhalten Format Informationen für die XML und HTML Dateien in diesem Ordner. Diese wird man nie selbst öffnen müssen, hauptsächlich sie sind vorhanden.

director & dirobjdb XML schema Dateien

Die Dateien wird man auch nie selbst öffnen. Sie beinhalten die ganzen Namen, look-up Funktionen und sonstige Dinge, die im XML Editor angezeigt werden. In der dirobjdb.xml sind die ganzen ‚typename‘ von allen Objekten in X3 gespeichert. Das beschleunigt das Arbeiten enorm da man nicht alles von Hand raus suchen muss.

XML Editoren

Es gibt einige Freie und Kommerzielle XML Editoren im Internet. Bei den Kostenlosen kenne ich 2 brauchbare. (In dieser Anleitung wird VWD benutzt)

XMLPad3 Pro (nur Englisch):

Dieser bietet einen großen Funktionsumfang, z.B.: Tree-view, Table-View und eine Menge anderer Kleinigkeiten die das Arbeiten einfach machen. Leider wird dieser nicht mehr weiter entwickelt, und auch das Wiki ist Off-line. Für Microsoft Allergiker, trotz fehlender Anleitung, die wohl beste Alternative. Leider hat XMLPad seine Ecken und Kanten wodurch er im Allgemeinen nicht ganz Rund läuft.

Visual Web Developer Express (VWD):

Trotz (weniger) fehlender Funktionen empfehle ich diesen Editor, da er am wenigsten Probleme verursacht und bereits von vielen Usern benutzt wird. Leider wird seit der 2010 Version ein „Windows Live ID Konto“ benötigt um das Programm zu registrieren. Wer das nicht möchte kann versuchen ein .iso der 2005 oder 2008 Version zu finden, die sind genauso gut aber brauchen keine Registrierung.

Welcher Editor auch benutzt wird, er sollte mindestens XMLschemas (nicht DTD) und stylesheets unterstützen, besser wäre wenn er auch Syntax hervorhebung, Auto-Vervollständigen und Tree-View kann.

Fragen zur Bedienung oder bei Problemen des Editors bitte in Foren stellen die sich mit dem jeweiligen Editor befassen. Das ES Forum kann nicht für alles Support bieten.

Erstellen einer neuen Missionsdatei

In diesem Kapitel werfen wir einen Blick auf den Prozess bei der Erstellung einer Mission. Dazu brauchen wir eine neue Datei und sehen uns erstmal die Grundlegenden Funktionen an. Die wichtigsten Schlüsselemente werden dabei erklärt, und wir erhalten so schonmal unsere erste kleine Beispiel Mission.

Anstatt den XML Editor zu starten und eine neue Datei zu erstellen, suchen wir uns eine vorhandene Datei aus dem director Ordner des Spielverzeichnis. Ob diese aus einer .Cat/.Dat kommt oder aus einem heruntergeladenem Script ist egal.

Diese Datei öffnen wir jetzt mit dem XML Editor, und löschen alles zwischen dem auf Bild 2 gezeigten Text und der Zeile: `</director>` am Ende der Datei. Anders als auf dem Bild, sollte Director NICHT unterstrichen sein.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="director.xml" type="text/xml" ?>
<director name="test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="director.xsd">
```

Bild 2 - Der Datei Header

Speichern sie diese Datei jetzt mit einem neuen Namen (Speichern unter), z.B. template.xml ab. Diese Datei kann als Start für jede neue Mission genommen werden. Denken sie aber daran das die Datei mit der sie jetzt arbeiten wollen im director Verzeichnis liegen muss, speichern sie eventuell ein zweites mal mit einem anderen Namen. Achtung: Durch ‚Speichern unter‘ ist die Datei jetzt eventuell in einem anderen Ordner, falls sie die woanders gespeichert haben. Wo wir gerade dabei sind, Speichern sie oft und machen sie Backup’s in einem anderen Ordner. Wenn etwas schief läuft können sie so jederzeit eine ältere Version wiederherstellen.

Eine fertige template.xml sollte zusammen mit diesem Guide in dem .zip Archive gewesen sein, Sie brauchen sich diese Datei also nicht selbst erstellen. :)

Fehlersuche bei der Arbeit mit MD Dateien

Falls sie versuchen eine Datei zu bearbeiten die sich nicht im director Ordner befindet, werden sie feststellen das keine Schema-Daten vorhanden sind (siehe Bild 3). Es gibt 2 Wege die man jetzt gehen kann. Erstens, kopieren sie die Datei in den director Ordner. Oder zweitens: Kopieren sie die 6 Dateien (siehe MD-Vorbereitung) in den Ordner in dem die Datei liegt, das ist nützlich falls sie einen Arbeitsordner benutzen wollen.



Bild 3 - Keine Schema Daten

Bild 4 - Fehlende director.xsd

Denken sie daran nach einem Update des Spiels die neue .Cat/.Dat Datei auf neuere Dateien (siehe MD - Vorbereitung) zu überprüfen. Kopieren sie die, falls neuere vorhanden sind, in den director UND, falls genutzt, ihren Arbeitsordner.

XML Formatierung

Unabhängig vom verwendeten XML Editor gibt es ein paar Grundlegenden Regeln beim arbeiten mit XML.

Die tags (<XYZ>) sind wichtig um den Aufbau der Funktionen erkennen zu können.

```
<haupt_node>
  <erster_subnode>
    <zweiter_subnode>
      <geschachtelte_funktion eigenschaft=""/>
    </zweiter_subnode>
  </erster_subnode>
</haupt_node>
```

Es ist Wichtig zu Wissen das XML Tags immer mit einem / geschlossen werden müssen. Das kann man auf zwei Arten machen. Ersten: Wenn ein Tag keine Unterelemente besitzt kann es am ende der Zeile geschlossen werden, zum Beispiel: <geschachtelte_funktion/> oder <find_sector x="0" y="0" name="kingdomend"/>. Und, wenn es Unterelemente besitzt, wie die obigen <haupt_node> oder <erster_subnode> wird es durch einen zweiten Tag NACH den Unterelementen geschlossen. Dieser zweite Tag hat den selben Namen hat jedoch das / am Anfang, z.B. </haupt_node>

<CUES>

Der <cues> Tag sagt dem XML Parser das als nächstes ein <cue> Sub-node kommt, hier können auch mehrere <cue> Sub-nodes auf der selben Ebene sein. Ein <cue> kann auch einen <cues> Sub-node haben, der wiederum <cue> Sub-node(s) enthält. Auf diese Weise ist es möglich auf die unterschiedlichsten Begebenheiten zu reagieren, z.B. das je nach Spielschiff unterschiedliche Gegner zu bekämpfen sind , oder bei niedrigem Kampfrang keine Gegner kommen sollen.

```
<?xml version="1.0"?>
<?xml-stylesheet href="director
] <director name="test" xmlns:xsi
  <cues></cues>
- </director>
```

Bild 5 - <cues>

Wenn man < drückt erscheint ein Fenster mit den möglichen Tags, dabei wird durch schreiben eines weiteren Zeichens das erste passende markiert, und kann durch drücken von > automatisch fertig geschrieben werden.

TIP: Man kann die Tags auch mit der Maus, Mousrad oder des Scroll-Balkens wählen.

Wenn der Cursor zwischen <cues> und </cues> ist (Bild 5), brauch man nur ‚return‘ drücken und der Cursor wandert zur nächsten Zeile, wo man den nächsten Tag schreiben kann.

</cues> wird dabei direkt in die übernächste Zeile gesetzt.

<CUE>

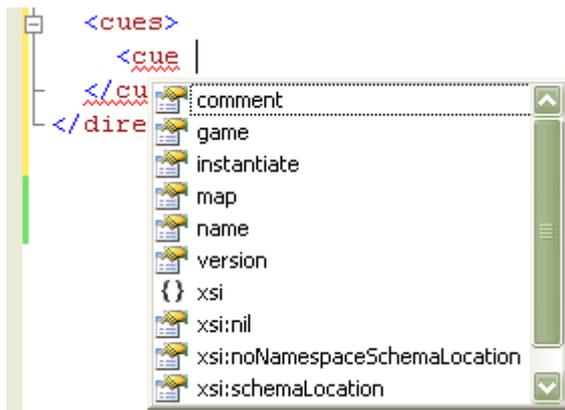


Bild 6 - <cue>

Mit dem <cue> Tag wird die Umgebung festgelegt die für das folgende Ereignis gelten soll. Dazu werden die Eigenschaften benutzt wie sie in Bild 6 zu sehen sind. Die unteren XSI Einträge werden allerdings nicht benutzt und können ignoriert werden.

Die Eigenschaft ,instantiate‘ ist ein wenig komplexer und wird im [Anhang 1](#) genauer erklärt.

Der <cue> Node kann bis zu 4 Sub-nodes haben, diese sind:

Sub-Node	Bemerkung
<condition>	Wenn nicht angegeben startet dieser cue sofort.
<timing>	Wenn nicht angegeben startet der cue sofort und einmal.
<action>	Wenn nicht angegeben passiert nix, außer es sind ein oder mehrere sub-cues vorhanden.
<cues>	Enthält sub-cues innerhalb dieses cue's. Wenn nicht angegeben wird der cue nach <action> beendet.

Die Tabelle zeigt die 4 Schlüssel Sub-nodes mit denen Sie arbeiten werden. <action> ist das Ereignis, das nach einer bestimmten Zeit (<timing>) ausgeführt wird wenn die Voraussetzungen (<condition>) erfüllt sind. Sub-cues von <cues> werden nach <action> ausgeführt, und haben ihre eigenen <condition>, <timing> und <action> Nodes

<CONDITION>

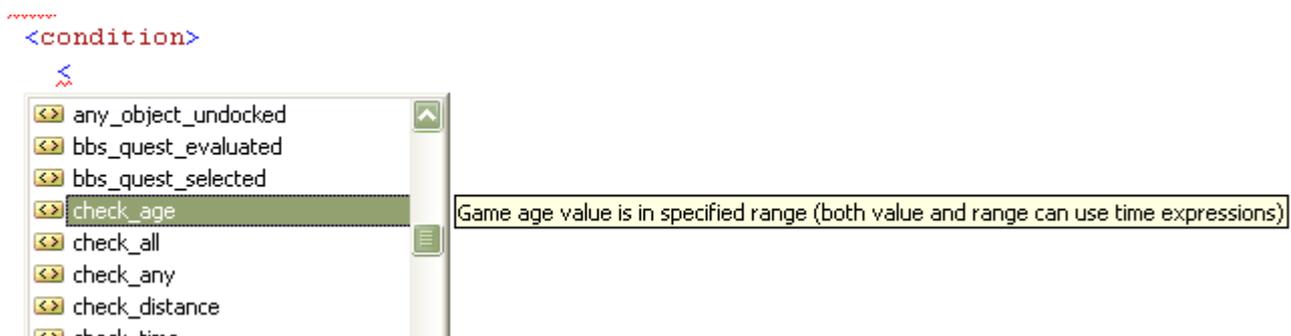


Bild 7 - Condition hat sehr viele Eigenschaften

Im <condition> Node werden, eine oder mehrere, Voraussetzungen definiert die erfüllt sein müssen damit das Ereignis gestartet wird. Das können einfache abfragen von Werten sein

z.B. wie viel Credits der Spieler hat oder wieviel Zeit seit Spielbeginn vergangen ist. Oder auch etwas Komplexere Abfragen, z.B. in welchem Sektor der Spieler gerade ist oder wie nah der Spieler an einem bestimmten Objekt ist.

Eine besondere Klasse von Vorraussetzungen basiert auf Ereignissen im Spiel z.B. ein Objekt wurde anvisiert, Angegriffen, zerstört oder der Sektor wurde gewechselt.

Vorraussetzungen können in Listen kombiniert werden wo alle Vorraussetzungen erfüllt sein müssen, irgendeine erfüllt sein muss, oder eine Kombination aus beiden Varianten.

Wenn mehrere Vorraussetzungen angegeben werden, müssen diese geschachtelt werden: Mit `<check_all>` wenn alle erfüllt werden sollen und `<check_any>` wenn irgendeine zutreffen soll.

Bei `<check_all>` ist es gut zu Wissen das diese der Reihe nach abgearbeitet werden. Wenn die erste Voraussetzung nicht stimmt, werden die anderen gar nicht erst geprüft.

Wenn Sie Ereignisse und Werte im selben Cue abfragen, setzen Sie Ereignisse nach oben um diese zuerst abzufragen. Ereignisse werden normalerweise einmalig ausgelöst, während die Werte jedesmal abgefragt werden wenn die Datei gelesen wird.

Vorraussetzungen eines Cue's können auch vom Status eines anderen Cue's abhängig sein, z.B. das ein bestimmter Cue beendet sein muss (`<cue_is_complete cue="MeinCue"/>`). Für den Haupt-cue empfiehlt es sich oftmals etwas einfaches, wie die Spielzeit (Bild 7), zu nehmen.

Welche Vorraussetzungen Sie auch nehmen, sobald sie erfüllt sind werden nach `<timing>` die Aktionen (`<action>`) ausgeführt.

Hier ein kleines Beispiel in dem der Cue ausgelöst wird wenn im Spiel zwischen 5 und 7

```
<condition>
```

```
  <check_age value="{player.age}" min="5s" max="7s" />
```

```
</condition>
```

Zeitangaben können durch anhängen von s,m oder h in Sekunden(s), Minuten(m) und Stunden(h) angegeben werden.

Variablen

Im obigen Beispiel wird ihnen `{player.age}` in der ‚Value‘ Eigenschaft aufgefallen sein. Das ist eine Variable und sie steht für eine Information die es im Spiel gibt, in diesem Fall für die Spieldauer. Es gibt im MD viele festgelegte Variablen, z.B. `{player.age}`, `{player.ship}` und `{player.money}`.

`{player.sector.race.name}` steht für das Volk dem der Sektor gehört in dem der Spieler gerade ist, egal wo er gerade ist.

Andere Variablen sehen ähnlich aus, es gibt aber auch Variablen die ein zusätzliches Element haben, das auf ein Benutzerdefiniertes Objekt bezogen ist. Diese Objekte werden nach einem @ angehängt. Zum Beispiel: `{object.pilot@object}`.

Wenn sie ein Schiff erstellen und brauchen den Namen des Piloten, dann ersetzen sie ‚object‘ hinter dem @ durch ihr Schiff: `{object.pilot@meinSchiff}`. Ein Beispiel wo das nützlich ist, wäre: `<incoming_message author="{object.pilot@meinSchiff}" text="Hilfe!"/>`

Eine Wichtige Sache beim erstellen und benutzen von Variablen ist ob sie Global oder Lokal sind. Globale Variablen sind für alle XML Missionen die zur Zeit Aktiv sind sichtbar, egal aus welcher Datei sie stammen. Dagegen sind Lokale Variablen nur in der Mission sichtbar in der sie erstellt wurden.

```
<cue name="eingegner">          <- Der cue in dem wir ein Schiff erstellen
.....
<create_ship name="gegner"      <- Hier wird das objekt in eine Globalen Variable gesetzt
<create_ship name="this.gegner" <- Und hier in eine Lokale Variable
```

Das ‚this.‘ wird nur benutzt wenn die Variable im selben Cue benutzt wird in dem sie erstellt wurde. Soll die Variable in einem anderen Cue benutzt werden muss ‚this‘ durch den Namen des Cue’s ersetzt werden in dem die Variable erstellt wurde. Im oberen Beispiel würde der Aufruf in einem anderen Cue z.B. so aussehen:

```
<incoming_message author={object.pilot@eingegner.gegner} text="{1081,16}" />
```

Wir haben also den Cuenamen ‚eingegner‘ und, durch einen Punkt getrennt, den Objektnamen ‚gegner‘

zusätzliche Informationen zu Variablen

Für Lokale Variablen wird also der Name des Cues benutzt, für Plot Missionen oder andere Missionen die nur einmal ausgeführt werden reicht das normalerweise aus. Für Missionen die mehrmals starten sollen (in mehreren Instanzen - siehe Anhang 1) ist es manchmal nützlich eine Bestimmte Instanz finden zu können. Beim suchen der Cuenamen wird der MD als erstes die aktuelle Cue Instanz anschauen, dann die darüberliegende Cue Instanz, und so weiter bis zur höchsten Instanz in der Cuestruktur. Dabei wird er auch alle Cues (und ihre Cub-cues) durchsuchen die auf der selben Ebene liegen.

Sollte der MD nichts finden wird er versuchen alle Cue Strukturen abzusuchen, wird aber vielleicht nicht immer etwas finden. (Tatsächlich wird bei ‚instantiated‘ Cues immer die Master Instanz gefunden.)

Diesen Prozess kann man abkürzen und den letzten Abschnitt verhindern, indem man ‚this‘ anstelle des Cuenamens verwendet wenn der aktuelle Cue gemeint ist, oder ‚parent‘ wenn der darüberliegende Cue gemeint ist.

<timing>

Nachdem die <condition> erfüllt sind kann man mit <timing> bestimmen wie lange gewartet werden soll bevor der Cue abgearbeitet wird. In <timing> legt man auch fest wie oft der Cue ausgeführt werden soll, oder ob er als Intervall (z.B. alle 5 Sekunden) ausgeführt wird.

Alleangaben können als fester Wert angegeben, oder aus einem gewissen Bereich zufällig gewählt werden.

Es kann nur einen <timing> Node geben, kann aber auch ganz weggelassen werden. In dem Fall wird der Cue ohne Verzögerung ausgeführt.

sub-node	Bemerkung
<count>	Wie oft der Cue ausgeführt wird
<time>	Wie lange gewartet wird bevor der Cue ausgeführt wird.
<interval>	Wie lange zwischen einzelnen Ausführungen gewartet wird.
<params>	Zusätzliche Parameter, diese werden in einer Library festgelegt.

Bild 8 ist ein Beispiel in dem Zufällig eine Wartezeit zwischen 5 und 10 Sekunden gewählt wird.

```
<timing>
  <time min="5s" max="10s" />
</timing>
```

Bild 8 - ein <timing> Beispiel

Bild 9 zeigt ein Beispiel in dem der Cue zufällig zwischen 50 und 100 mal ausgeführt wird. Wer möchte kann man mit den gezeigten 5 Profilen das Zufalls-Ergebnis beeinflussen. Das in Bild 9 markierte ‚increasing‘ erhöht die Chance größere Werte zu bekommen.

```
<timing>
  <count min="50" max="100" profile="" />
</timing>
cue>
ies>
:ctor>
```

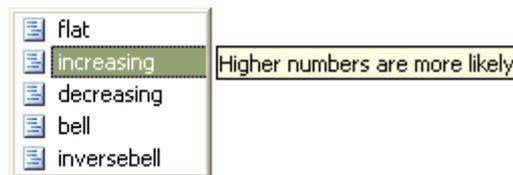


Bild 9 - Profile

Bild 10 zeigt ein Beispiel in dem alle 5 Sekunden der Cue gestartet wird, solange bis 60 Sekunden vergangen sind.

```
<timing>
  <time min="5s" max="60s" />
  <interval min="5s" />
</timing>
```

Bild 10 - Beispiel für <interval>

Einheiten bei Zeit und Entfernung

Wenn nur die Zahl angegeben wird, rechnet MD in Millisekunden bzw. mit den Spiel eigenen Einheiten (game-units) . Da man aber meistens mit größeren Spannen zu tun hat gibt es hier eine Tabelle mit den Möglichkeiten:

Formel	Beispiel	Bemerkung
Entfernung + m	100m	Meter
Entfernung + km	{value@this.xpos}km	Kilometer
Zeit + d	5d	Tage (engl. Days)
Zeit + h	12h	Stunden (engl. Hours)
Zeit + m	{value@this.zeit}m	Minuten
Zeit + s	({value@this.zeit}*10)s	Sekunden
m:s.ms	1:30.500	Minuten:Sekunden.Millisekunden
h:m:s.ms	1:30:0.0	Stunden:Minuten:Sekunden.Millisekunden

Bei den unteren beiden Zeitangaben sollte man darauf achten das keine der Zahlen mit 0 beginnt, auch sollten die Millisekunden(ms) als Zahl angesehen werden und nicht als Dezimalstellen hinterm Komma (Sprichwörtlich, hier ist es ein Punkt). Außerdem können hiermit keine Tage angegeben werden.

Die maximale Wert für die Zeit beträgt 24 Tage, nur {player.age} kann größer Werte haben dafür werden Millisekunden ignoriert.

Operatoren

Beim arbeiten mit Variablen und Werten können einfache Mathematische Operatoren benutzt werden. Hier ein Beispiel indem geprüft wird ob der Spieler mindestens 1000 Credits mehr als in der Variable hat:

```
<check_value value="{this.geldvorher}" min="{player.money}+1000"/>
```

Alle Werte im Format ,Number' (anstatt ,Integer') können Operatoren benutzen.

Operator	Beispiel	Beschreibung
x+y	{value@this.wert1}+{value@this.wert2}+2	Addieren
x-y	{value@this.wert1}-{value@this.wert2}	Subtrahieren
x*y	{counter@myloop}*10	Multiplizieren
x/y	100/{value@this.wert}	Dividieren
(x)	(100+{value@this.wert})/10	Klammern werden zuerst behandelt

Dabei wird zuerst Berechnet was in Klammern gesetzt ist, dann * / und zum Schluss + - Bei mehreren gleichen Operatoren von Links nach Rechts.

<action>

Nachdem die <condition> zutreffen und <timing> das OK gegeben hat, kann in <action> schließlich etwas passieren. Das kann ein einzelner Befehl sein, etwa dem Spieler Geld geben, eine Nachricht anzeigen oder ein Schiff erschaffen. Die Befehle können aber auch in Listen kombiniert werden wo alle Befehle ausgeführt werden, irgendeiner ausgeführt wird, oder auch eine Kombination aus beiden Varianten. Denken sie daran das durch <timing> der gesamte <action> Teil eventuell mehrmals ausgeführt wird.



Bild 11 - Einige der <action> sub-nodes

Wie bei <condition> können die Listen verschachtelt werden um auch Komplexere oder Alternative Aktionen zu erhalten. Wichtig zu erwähnen ist das diese <action> Listen kein Programm sind. Eine normale Programmierung mit Schleifen und Bedingungen, wie ein Programmierer es gewohnt ist, gibt es in diesem Sinne nicht. Für Komplexere Dinge werden <condition> und <timing> benutzt solche Dinge zu realisieren, für ein paar einfachere Dinge gibt es aber Alternativen.

In der Regel wird in jedem Cue ein <action> Node sein, und zwar höchstens einer. Sollen mehrere Aktionen verwendet werden müssen diese in einem Sub-node geschachtelt werden. <do_all> wenn alle Befehle ausgeführt werden sollen oder <do_any> wenn irgendeiner ausgeführt werden soll. Ohne diese Sub-nodes kann nur ein Befehl benutzt werden. Wie vorher schon angedeutet können <do_all> und <do_any> auch untereinander geschachtelt werden um Komplexere Missionen zu schreiben. Das funktioniert übrigens auch in den <condition> Sub-nodes <check_all> und <check_any>, das wird später aber noch genauer erklärt. Zusätzlich gibt es noch <do_if>, wo die Aktion nur ausgeführt wird wenn ein Wert mit einem anderen Übereinstimmt. Und <do_choose>, was eine erweiterte version von <do_if> ist. Sub-nodes von <do_choose> sind <do_when> und <do_otherwise>, welches Aktionen beinhaltet die ausgeführt werden falls keine der <do_when> abfragen richtig waren.

```
<action>
  <do_any>
    <incoming_message author="Shipboard Computer" text="Hello World"/>
    <incoming_message author="{1323,119}" text="Welcome to MD"/>
    <incoming_message author="{1323,119" text="{1278,305)"/>
  </do_any>
</action>
```

In diesem Beispiel wird <do_any> benutzt um eine der 3 Nachrichten an den Spieler zu senden, welche Nachricht geschickt wird ist dabei Zufällig. Die ‚author‘ Eigenschaft von <incoming_message> ist der im Spiel gezeigte Absender und ‚text‘ ist die eigentliche Nachricht. Beide können direkt als Text angegeben werden, jedoch ist es besser eine Text ID zu benutzen. Nachträgliche Änderungen und Übersetzungen sind damit wesentlich einfacher, das wird im Kapitel „[Text ID und Sprachdatei](#)“ erklärt.

Beispiel 1

Mit den bisher gezeigten Beispielen lässt sich schon ein kleines Lauffähiges MD script basteln. Schauen wir es uns einmal im ganzen an:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="director.xsl" type="text/xsl" ?>
<director name="test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="director.xsd">
  <cues>
    <cue name="bs_evaluator" game="all" version="1" comment="Ein Kommentar was im cue passiert">
      <condition>
        <check_age value="{player.age}" min="5s"/>
      </condition>
      <timing>
        <time min="5s" max="10s"/>
      </timing>
      <action>
        <do_any>
          <incoming_message author="Schiffscomputer" text="Hallo Welt"/>
          <incoming_message author="{1323,119}" text="Willkommen beim MD"/>
          <incoming_message author="{1323,119}" text="{1278,305}"/>
        </do_any>
      </action>
    </cue>
  </cues>
</director>
```

Testen der ersten Mission

Nachdem sie die Mission gespeichert haben, überzeugen sie sich das sie auch im director Ordner des Spielverzeichnisses vorhanden ist. Nochmal zur Erinnerung, verwenden sie für ihren Cue keine Namen die bereits in anderen Missionen verwendet werden.

Zeit das Spiel zu starten. Zum testen wählen wir ‚Neues Spiel‘ und dann ‚Selbsterstelltes Spiel‘ auf einer beliebigen Map. Ändern Sie ihren Spielernamen zu ‚Thereshallbewings‘, und wenn sie das obere Beispiel verwenden sollten sie kurz darauf eine der 3 Nachrichten bekommen.

,In-Space‘ Missions Tutorial

Auch wenn diese Anleitung nicht gedacht ist um den Umgang mit einem XML Editor zu lernen, hier ein paar Tipps bevor es richtig losgeht:

- 1.: Wenn sich das Popup-Menü mit den Möglichen Nodes schließt, kann man es mit STRG+Space (CTRL+Space) wieder hervorholen
- 2.: Im Normalfall braucht man sich um die XML-Formatierung keine Gedanken machen. Sollte doch mal ein Tab zuviel oder zuwenig vorhanden sein, kann man das über ‚Bearbeiten/Dokument Formatieren‘ berichtigen lassen.
- 3.: Bei großen Missionen kann man einzelne Nodes oder Cues mit den -/+ auf der linken Seite auf- und zuklappen.
- 4.: Wenn man bei Fehlern den Mauszeiger kurz auf dem Rot oder Blau Unterstrichenen Text läßt kommt ein Tooltip mit hinweisen wo der Fehler liegt.
- 5.: Wenn man den Mauszeiger kurz auf einem Befehl o.A. läßt kommt ein Tooltip mit einer Beschreibung.
- 6.: Bei den Popup-Listen mit den Möglichen Befehlen bekommt man eine Beschreibung wenn man einen Eintrag markiert.

In diesem Kapitel werden Sie Schritt für Schritt durch die Entwicklung einer Mission geführt, dazu gibt es zu allen wichtigen Elementen eine genaue Beschreibung. Diese Mission basiert auf einem alten X:Tension favoriten und heißt ‚Docking Race Bet‘.

Das erste Schlüsselement jeder Datei ist der Header. (Siehe Bild 2, Seite 6)

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

In dieser Zeile wird die verwendete Zeichenkodierung festgelegt. Oft wird hier ‚iso-8859-1‘ verwendet, da diese Kodierung aber keine Umlaute beherrscht verwenden wir ‚utf-8‘. Das spart Zeit und Ärger, da ein einzelnes ä, irgendwo in der Datei, dafür sorgen kann das die ganze Mission nicht geladen wird.

```
<?xml-stylesheet href="director.xml" type="text/xsl" ?>
```

Diese Zeile legt das verwendete stylesheet für XML fest, das bestimmt wie die Datei formatiert wird.

```
<director name="template" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="director.xsd">
```

In dieser langen Zeile wird festgelegt wo die Schemadatei zu finden ist, die in dieser Datei benutzt wird. Diese beinhaltet z.B. alle Befehle so das der Editor weiß welche es gibt und wie sie benutzt werden. Die ‚name‘ Eigenschaft vom <director> Tag wird im Spiel übrigens nicht verwendet. Wenn diese Datei im Browser o.Ä. geöffnet wird steht der Name ganz oben auf der Seite oder auch im Titel. Dieser Name ist für die Datei selbst, und hat mit den Cue Nodes nichts zu tun. Trotzdem muss der <director> Tag am Ende der Datei geschlossen werden, das ist die letzte Zeile jeder MD Datei (</director>).

Die nächsten paar Zeilen sind eigentlich Selbsterklärend und für Offiziell Entwickelte Missionen Obligatorisch. Der `<documentation>` Node verrät Entwicklern und Testern vom wem die Mission geschrieben wurde (`author+alias`), welche Version es ist (`version number`) und worum es geht (`description`). Falls man Feedback oder Kommentare abgeben möchte ist das über `,contact` möglich. Zusätzlich gibt es noch das Datum (`date`) und den Aktuellen `,Status`

```
<documentation>
  <author name="Fred Bloggs" alias="Froggs" contact="froggs@egosoft.com"/>
  <content name="Docking Race Bet" description="Der Spieler wird von einem Jägerpi-
  loten gefragt einen Rennen bis zur nächsten Handelsstation zu machen"/>
  <version number="1.0" date="2006-06-15" status="beta test"/>
</documentation>
```

Einige Cue Namen in dieser Anleitung haben eine vorangestellte Seriennummer. Das muss man zwar nicht machen, stellt aber Sicher das keine Konflikte mit anderen Missionsdateien entstehen die ähnliche Bezeichnungen verwenden. Eine beliebte Methode bei der Namensgebung ist z.B. das verwenden von Abkürzungen. z.B.: `<cue name="bs1_name"/>` oder einfach nur `name="bs1"`, `,bs1` steht dabei für **BeiSpiel 1**. Einige setzen z.B. zusätzlich noch ihre Initialen davor, zum Beispiel: `<cue name="tt_bs1_name"`. Schauen sie sich auch ruhig einmal an wie das in den Missionen des Spiels gehandhabt wird.

Beim ersten Cue, auch Top-level oder Main-cue genannt, ist es absolut notwendig das dieser einen einzigartigen Namen hat. Bei den Sub-cues ist es nur wichtig das sie in innerhalb ihres Parent-cues (dem übergeordneten Cue) einzigartig sind, als Entwickler sollten Sie aber versuchen es sich zur Gewohnheit zu machen und überall ein solches Schema bei der Namenswahl zu benutzen.

<cues>

Der nächste Node wird immer `<cues>` sein. Wenn sie `<cues>` schreiben wird der schließen Tag `</cues>` automatisch dran gehängt, durch drücken von Return geht der Cursor in die Position für den nächsten Sub-node. Gut zu Wissen ist das innerhalb `<cues>` mehrere `<cue>` sein können, die ihrerseits auch `<cues>` Sub-nodes haben können. Bleiben wir aber erstmal bei der Aktuellen Mission.

<cue>

Der erste `<cue>` in jeder Datei ist der Top-Level Cue; alle anderen `<cue>` und `<cues>` sind diesem untergeordnet. Sozusagen sind die Nodes `<condition>`, `<timing>` und `<action>` in diesem Cue der Auslöser der die ganze Mission erst startet.

<cues>

```
<cue name="S01M25S00prepare">
```

Wie bereits erwähnt ist es Wichtig einen einzigartigen Namen zu benutzen. Außer `name` kann `<cue>` aber noch andere Eigenschaften haben die für die Mission gelten sollen:

`game` = legt fest in welchen Spielmodi die Mission verfügbar ist. (`all`, `plot`, `noplot`, `custom`)
Wenn es nicht angegeben ist wird automatisch `,all` (alle) benutzt.

map = Legt fest welche map die Mission benutzt (? custom map ?)

instantiate = ‚static‘ erstellt eine Kopie (Instanz) des Cue’s. Wenn es nicht angegeben ist wird keine Kopie erstellt. Eine genauere Beschreibung finden sie in [Anhang 1](#).

library = Markiert den Cue als Library, das heißt das `<condition>`, `<timing>` und `<action>` erst ausgeführt werden wenn der Cue, an anderer Stelle, über `ref=“curname“` aufgerufen wird.

ref = Hiermit kann ein als Library markierter Cue benutzt werden. Eine genaue Beschreibung zur verwendung von `ref` und `library` gibt es im [Anhang 2](#).

comment = Hier kann man Kommentieren was im Cue passiert. ‚comment‘ gibt es bei allen Nodes und ist sehr nützlich. Gewöhnen Sie sich an ihre Missionen gut und viel zu Kommentieren, das hilft bei der Fehlersuche und man verliert nicht so schnell den Überblick.

Kommen wir zu den 3 Wichtigsten Elementen jedes Cue’s:

`<condition>`

Um unsere Mission zu Starten braucht es ein paar vorraussetzungen die erfüllt sein müssen.

```
1 <condition>
2   <check_all>
3     <object_changed_sector comment=“Erst diesen Prüfen, wenn dieses Ereignis nicht
4       stattfindet braucht der Rest gar nicht erst geprüft zu werden“/>
5     <match_object class=“fighter“ comment=“M3, M4 oder M5“/>
6     <check_value value=“{player.age}“ min=“1m“/>
7     <!--
8     <check_age min=“10m“ chance=“(72000/{player.time})+1“ comment=“Dies redu-
9       ziert die Chance das die Mission angeboten wird von 100% nach 10 Minuten auf 1% nach
10      20 Stunden“/>
11     -->
12     <object_has_equipment comment=“Wenn ein Landecomputer vorhanden ist, kann das
13       Rennen los gehen“>
14       <ware typename=“SS_WARE_TECH241“ min=“1“/>
15     </object_has_equipment>
16     <check_value value=“{player.money}“ min=“{reward.money@veryeasy.
17       XXXT}+500“/>
18     <check_value value=“{player.sector.race}“ exact=“{lookup.race@argon}“/>
19   </check_all>
20 </condition>
```

Schauen wir uns den `<condition>` Node mal Reihe für Reihe an:

- 1 & 15: Die Öffnen und Schließen Tags für den ganzen `<condition>` Teil. Das war einfach
- 2 & 14: `<check_all>` erlaubt es uns mehrere Bedingungen zu stellen, die alle erfüllt werden müssen.
- 3: Die erste Bedingung die erfüllt werden muss ist `<object_changed_sector>`. Das ist ein Ereignis das eintritt wenn das Objekt den Sektor gewechselt hat oder andockt(?). Wenn kein Objekt angegeben ist wird das Spielerschiff als Objekt genutzt.
- 4: Diese Bedingung wird erfüllt wenn das Spielerschiff ein Jäger ist (Fighter - M3,M4,M5).
- 5: `{player.age}` Hier prüfen wir ob seit Spielbeginn mindestens eine Minute vergangen ist. Diesen Wert nehmen wir nur zum Testen der Mission, damit wir nicht lange warten müssen.
- 6&8: Alles zwischen `<!--` und `-->` wird vom XML Parser Ignoriert. Auf diese Weise kann man vorübergehend einen Teil des Codes auskommentieren sozusagen ausschalten, oder längere Kommentare zur Mission zwischen den nodes plazieren.
- 7: Solange wir die Mission Testen ist diese Bedingung auskommentiert. Wenn soweit alles funktioniert werden wir Zeile 5 löschen und diese Zeile als Bedingung nehmen. Eine Beschreibung finden sie in der `,comment'` Eigenschaft.
- 9-11: Hier prüfen wir ob der Spieler einen Landecomputer hat. Im `<ware>` Sub-node stellt man über die `,typename'` Eigenschaft ein was auf dem Objekt vorhanden sein muss. Der `<object_has_equipment>` Node kann auch mehrere `<ware>` Sub-nodes haben, falls das Objekt mehrere Dinge an Bord haben soll. Bei `,typename'` finden sie den Landecomputer als `"SS_WARE_TECH241"` in der Popup-Liste, woran man gut sehen kann warum man seinen Code gut Kommentieren sollte. Später weiß man sonst nicht mehr was diese Typenamen genau bedeuten. Wenn kein Objekt angegeben ist wird wieder Automatisch das Spielerschiff geprüft.
- 12: Diese Zeile prüft ob der Spieler mindestens das Geld für die Wette +500 Credits auf dem Konto `{player.money}` hat. Nur für den Fall das der Spieler die Wette verliert. `{reward.money@veryeasy.XXXT}` ist eine Variable mit der man einen Bonus passend zur Mission und dem Schwierigkeitsgrad erfragen kann. Weitere solcher vor-Balancierten Belohnungen finden sie in der `director.html`
- 13: Die letzte Bedingung vergleicht zwei Variablen: `{player.sector.race}` steht für die interne ID des Volkes dem der Sektor gehört wo der Spieler sich gerade befindet, und `{lookup.race@argon}` ist die Interne ID der Argonen. Kurz gesagt der Spieler muß in einem Argonischen Sektor sein.

Dieser `<condition>` Node hatte ein paar Einträge mehr, denken sie daran das bei `<check_all>` alle Bedingungen erfüllt sein müssen.

TIP: Wenn sie einen `<condition>` Node möchten in dem mehrere Bedingungen erfüllt werden sollen, aber von manchen nur eine erfüllt sein muß, setzen sie innerhalb des `<check_all>` Nodes zusätzlich noch ein `<check_any>`.

Andersherum können Sie in einem `<check_any>` Node auch mehrere Blöcke mit `<check_all>` erstellen in welchem dann mehrere Bedingungen sind.

`<timing>`

In diesem Top-Level Cue ist der `<timing>` Node ziemlich einfach:

```
<timing>  
  <time min="3s" max="5s"/>  
</timing>
```

Irgendwann zwischen 3 und 5 Sekunden nachdem alle Bedingungen erfüllt wurden geht es mit dem `<action>` Node weiter.

`<action>`

Der `<action>` Node hat ein paar Sub-nodes mehr. Schauen wir sie uns, Zeile für Zeile, mal

```
1 <action>  
2   <do_all>  
3     <find_gate nearest="1" name="this.gate">  
4       <distance max="10km"/>  
5     </find_gate>  
6     <find_station name="this.finish" class="trade" dockingallowed="1"  
7       findobject="{player.ship}" max="1" typename="SS_DOCK_A_TRADE"/>  
7     <set_value name="this.reward" exact="{reward.money@veryeasy.  
8       XXXT}/100)*100" comment="das /100 und *100 benutzen wir um eine Runde  
9       Zahl zu bekommen ;)/>  
8     <load_text fileid="8737"/>  
9   </do_all>  
10 </action>
```

1&10: Die öffnen und schließen Tags des `<action>` Nodes.

2&9: Die öffnen und schließen Tags des `<do_all>` Nodes, ähnlich wie `<check_all>` sollen alle Sub-nodes ausgeführt werden.

- 3-5: Dieser `<find_gate>` Node soll das nächste (`nearest="1"`) Sprungtor finden, das im Umkreis von 10 Kilometern (`distance="10km"`) sein muss. Sobald es gefunden wird speichern wir es in einer Variable mit dem Namen `"this.gate"`, this wird benutzt weil die Variable nur in diesem Cue benutzt werden soll (lokale Variable). Zur Erinnerung: Würden wir sie nur `,gate'` nennen wäre es eine Globale Variable die von allen Missionen, aus allen Dateien gesehen und benutzt werden könnte.
- 6: Als nächstes wollen wir eine Station finden, die wir in der Variable `"this.finish"` speichern. Diese muss zur Klasse `,trade'` (z.B. Handelsstation) gehören, und weil wir nur die Argonische Handelsstation finden wollen geben wir den `,typename'` an (`SS_DOCK_A_TRADE`). Natürlich muss der Spieler (`{player.ship}`) dort auch Landen können (`dockingallowed="1"`).
- 7: Mit `<set_value>` können alle möglichen Werte (`exact="wert"`) in einer Variable gespeichert werden. In diesem Fall verwenden wir eine Formel um den Wert zu berechnen und abzurunden. `{` und `}` wird benutzt um MD zu sagen das es sich um eine Variable handelt während in `()` eingeschlossene Ausdrücke zuerst berechnet werden sollen.
- 8: Über `<load_text>` laden wir eine Sprachdatei, `Fileid="8737"` heißt das wir die Datei `„8737-L049“` benutzen wollen, dort stehen alle in dieser Datei vorkommenden Texte. Innerhalb dieser Mission geben wir dann nur eine TextID an, die auf einen Eintrag in dieser Datei verweist. Eine genauere Beschreibung finden sie im Kapitel `„TextID und Sprachdatei“`.

Damit wäre der Top-Level Cue soweit fertig. Wir haben die Bedingungen festgelegt um die Mission zu starten, haben nach einer Station und einem Tor gesucht die als Zielpunkte dienen und wir haben den Gewinn festgelegt.

Im nächsten Cue, der ein Sub-cue von diesem sein wird, schauen wir ob eine Station und ein Tor gefunden wurden, und falls nicht, kümmern wir uns darum was passieren soll. Der `<condition>` Node ist auch ein gutes Beispiel einer `<check_all>` `<check_any>` verschachtelung.

Aber zuerst: Was macht einen Cue zu einem Sub-cue? Wie schon erwähnt können Cues untereinander verschachtelt werden, auf diese Weise können wir die Mission von einem Ereignis zum nächsten führen. Dazu erstellen wir einen weiteren `<cues>` Node BEVOR wir den aktuellen Cue schließen, er ist dem Top-Level Cue untergeordnet (sub = Unter). `<cues>` kommt dabei direkt nach dem `</action>` Tag, zum Beispiel:

```
</action>
  <cues>
    <cue name="blah">
```

Wie man sehen kann ist innerhalb von `<cues>` mindestens ein `<cue>` Node, es können aber genauso gut mehrere sein. Diese liegen dann auf der selben Ebene und jeder einzelne `<cue>` Node kann wiederum seine eigenen Sub-cues besitzen.

<condition>

Schauen wir uns den zweiten Cue etwas genauer an, angefangen mit dem <condition> Node:

```
1 <cues>
2   <cue name="reset">
3     <condition>
4       <check_all>
5         <cue_is_complete cue="S01M25S00prepare"/>
6         <check_any>
7           <object_exists negate="1" object="S01M25S00prepare.gate"/>
8           <object_exists negate="1" object="S01M25S00prepare.finish"/>
9         </check_any>
10      </check_all>
11    </condition>
```

1: Wie gerade schon gesagt, beginnen wir einen Sub-cue mit <cues>, der nächste Tag muss ein <cue> sein

2: Wie versprochen... Wir nennen diesen Cue ,reset‘ (neustart), was eine gute Beschreibung dessen ist was im Cue passiert.

3&11: Die öffnen und schließen Tags des <condition> Nodes

4&10: <check_all> alle Bedingungen müssen erfüllt werden.

5: Die erste Bedingung prüft ob der Cue mit dem Namen "S01M25S00prepare" fertig ist, also alle Bedingungen erfüllt und alle Aktionen ausgeführt wurden. Das ist unser Top-Level Cue.

6&9: Der <check_any> Node innerhalb von <check_all> heißt das nur einer der angegeben Bedingungen erfüllt werden muss. Hier heißt <check_any> also ungefähr: ,und wenn einer von beiden‘

7&8: Hier prüfen wir ob eine Station oder ein Tor gefunden wurde, tatsächlich prüfen wir ob sie nicht (negate="1") gefunden wurden. Wenn nur eine der Beiden nicht gefunden wurde, wird die Mission neu gestartet.

Sie wundern sich vielleicht warum die (Lokalen) Variablen S01M25S00prepare und nicht this benutzen. this wird NUR in dem Cue verwendet in dem die Variable erstellt wurde, in untergeordneten Cues ersetzen wir this durch den Namen des Cues aus dem sie stammt. Wenn die Variable aus dem darüberliegenden Cue kommt, kann auch parent verwendet werden.

<timing> und <action>

Wir haben jetzt geprüft ob der Top-Level Cue beendet wurde, und ob die Station oder das Tor gefunden wurden. Wurde einer der beiden Objekte nicht gefunden, würde die Mission nicht funktionieren, und es geht über <timing> mit dem <action> Node weiter um die Mission neuzustarten.

```
<timing>  
  <time min="1s"/>  
</timing>
```

Das ist wieder ein einfacher <timing> Node, eine Sekunde nachdem die <condition> Wahr ist, wird die <action> gestartet.

```
<action>  
  <reset_cue cue="S01M25S00prepare"/>  
</action>  
</cue>
```

Das ist nicht der größte <action> Node, mit Blick auf die <condition> allerdings einer der Wichtigeren. Mit <reset_cue> wird der angegebene Cue neugestartet, seine <condition> müssen also erneut erfüllt werden. In unserem Fall ist es der Top-Level Cue, es beginnt also die gesamte Mission von vorne.

Wir erinnern uns das eine der Bedingungen ein <object_changed_sector> Ereignis war, der Spieler muss also erst in einen anderen Sektor fliegen. Erst dann wird erneut nach einer Station und einen Tor gesucht, das geschieht dann solange bis beide gefunden wurden.

Der </cue> Tag nach dem <action> Node schließt diesen Cue, es gibt also keine Sub-cues. Der nächste Cue kommt direkt nach dem </cue> Tag, diese beiden Cues sind auf der selben Ebene, es sind somit beide Sub-cues von ‚S01M25S00prepare‘. Die Ebene wird erst durch den </cues> Tag geschlossen, der in unserer Mission aber erst später kommt.

Dieser Cue ist etwas größer, aber wie vorher auch, teilen wir ihn in handliche Stücke.

```
1 <cue name="selectOpponent" comment="wähle einen zufälligen typename ähnlich dem  
   Spielschiff">  
2   <condition>  
   <cue_is_complete cue="S01M25S00prepare"/>  
3 </condition>
```

1: Der Cue und sein Name, im Kommentar wird schon in etwa verraten was hier geschieht.

2-3: Die einzige Bedingung ist das der Parent Cue (S01M25S00prepare) seine <condition> und <action> Nodes beendet hat.

Der `<action>` Node sieht auf dem ersten Blick etwas gemeiner aus, wir gehen ihn aber wieder Zeile für Zeile durch.

```
1 <action>
2   <do_choose>
3     <do_when value="{player.ship.class}" exact="{lookup.class@m3}">
4       <set_value name="selectOpponent.typename" exact="{lookup.type@{random.type
@SS_SH_A_M3|SS_SH_A_M3_1|SS_SH_A_M3_2|SS_SH_A_M3_3}}"/>
5     </do_when>
6     <do_when value="{player.ship.class}" exact="{lookup.class@m4}">
7       <set_value name="selectOpponent.typename" exact="{lookup.type@{random.type
@SS_SH_A_M4|SS_SH_A_M4_1|SS_SH_A_M4_2|SS_SH_A_M4_3}}"/>
8     </do_when>
9     <do_otherwise comment="kein M3 und kein M4 also vermutlich ein M5">
10      <set_value name="selectOpponent.typename" exact="{lookup.type@{random.type
@SS_SH_A_M5|SS_SH_A_M5_1|SS_SH_A_M5_2|SS_SH_A_M5_3}}"/>
11    </do_otherwise>
12  </do_choose>
13 </action>
14 </cue>
```

1&13: Die `<action>` Tags sollten sie mittlerweile kennen.

2&12: Alle Aktionen in diesem Cue sind in einem `<do_choose>` Sub-node, das erlaubt es uns zusätzliche Bedingungen zu stellen damit eine Aktion ausgeführt wird. Eine gute Beschreibung wäre: Wenn A=B ist führe C aus, und wenn nicht D.

3&5: der `<do_when>` Sub-node behandelt die Bedingung die erfüllt sein muss damit die geschachtelte Aktion ausgeführt wird. In diesem Fall: ob das Spielerschiff (`player.ship`) zur Klasse M3 gehört.

4: Wenn das Spielerschiff ein M3 ist, wird diese Aktion ausgeführt. Die in `,exact=‘` benutzte Variable sieht etwas komplizierter aus, auch weil hier eine Variable innerhalb einer Variable verwendet wird. (Zur Erinnerung: Variablen sind von `{ }` umschlossen). `,Lookup.type‘` gibt uns die Interne ID eines Typenamen das hinter `@` angegeben ist. `,random.type‘` wählt zufällig einen der Typenamen, die hinter `@` angegeben sind, aus. Die einzelnen Typenamen werden dabei durch ein `|` voneinander getrennt. Das Ergebnis wird dann mit `<set_value>` in `selectOpponent.typename` gespeichert, dies wird das Schiff unseres Gegners sein.

6-8: Dieser `<do_when>` Node ist wie der obere, nur das diesmal ein M4 benutzt wird.

9-11: Hier benutzen wir `<do_otherwise>`, nur wenn die Bedingungen von den `<do_when>` Nodes nicht erfüllt werden, wird dieser benutzt. Die Aktion selbst ist die selbe wie in den `<do_when>` Nodes, nur mit einem M5.

Vielleicht haben sie bemerkt das in allen `<set_value>` Sub-nodes die Variable `,selectOpponent.typeName'` verwendet wird. Hier wird die Interne ID des Typenamens gespeichert, die wir mit `,lookup.yxz'` erfragt haben. Im nächsten Cue benutzen wir diese um das Schiff unseres Gegners zu erstellen.

14: Da wir keine Sub-cues benötigen wird dieser Cue hier geschlossen.

Der nächste Cue liegt wieder auf der selben Ebene wie die letzten beiden, ist also auch ein Sub-cue vom Top-Level Cue `,S01M25S00prepare'`. Wir nennen ihn: `,createopponent'`

```
<cue name="createopponent">
```

Wie immer gehen wir diesen Cue Stück für Stück durch, angefangen mit dem `<condition>` Node. Nur für den Fall das es niemandem auffällt, dieser Cue hat keinen `<timing>` Node. Es geht, nachdem die `<condition>` erfüllt wurden, also direkt zur `<action>`.

```
1 <condition>
2   <check_all>
3     <cue_is_complete cue="selectOpponent"/>
4     <cue_is_complete cue="S01M25S00prepare"/>
5     <object_exists object="S01M25S00prepare.gate"/>
6     <object_exists object="S01M25S00prepare.finish"/>
7   </check_all>
8 </condition>
```

Die Bedingungen die erfüllt werden müssen sind fast genauso wie die Bedingungen aus dem `,reset'` Cue.

1&8: Die `<condition>` Node tags

2&7: `<check_all>` alle Bedingungen müssen zutreffen.

3: Der Cue `,selectOpponent'` muss beendet sein

4: Der Cue `,S01M25S00prepare'`, unser Top-Level Cue, muss ebenfalls beendet sein

5&6: Das Tor und die Station nach denen wir im Top-Level Cue gesucht haben, müssen Existieren und gefunden sein.

So, wenn alles gut läuft haben wir ein Schiff für unseren Gegner rausgesucht, haben ein Tor und eine Station gefunden und überprüft ob diese als Objekte im Spiel auch tatsächlich vorhanden sind. Jetzt können wir, wie der Cuenamen schon verrät, anfangen einen Gegner zu erstellen gegen den der Spieler dann ein Rennen fliegen kann.

Achtung: Im `<action>` Node ist diesmal einiges los, wenn sie noch Kaffee oder Popcorn brauchen holen Sie das besser bevor Sie weiterlesen. :)

```

1 <action>
2   <do_all>
3     <create_ship name="this.opponent" typename="{value@selectOpponent.typename}"
4         racellogic="0" race="argon" class="{player.ship.class}" highlight="1">
5       <position min="5km" max="6km" object="S01M25S00prepare.gate"/>
6       <equipment loadout="default"/>
7       <command command="follow" commandobject="{player.ship}"/>
8       <pilot name="{random.pilot.argon}" race="argon"/>
9     </create_ship>
10    <set_value name="this.tunings"
11        exact="{player.ship.maxspeed}/({object.basespeed@this.opponent}/10)-10"/>
12    <add_equipment object="this.opponent">
13      <ware typename="SS_WARE_TECH213" exact="-100"
14        comment="Sicherheitshalber alle Engine Tunings entfernen"/>
15      <ware typename="SS_WARE_TECH213" min="{value@this.tunings}-1"
16        max="{value@this.tunings}+1" comment="Engine Tunings"/>
17    </add_equipment>
18    <do_if negate="1" value="{object.class@this.opponent}" exact="{lookup.class@
19      m5}" comment="außer M5, die können keinen Sprungantrieb nutzen">
20      <add_cargo object="this.opponent">
21        <ware typename="SS_WARE_ENERGY" min="15" max="50"
22          comment="Energiezellen"/>
23      </add_cargo>
24      <add_equipment object="this.opponent">
25        <ware typename="SS_WARE_WARPING" exact="1" comment="Sprungantr."/>
26      </add_equipment>
27    </do_if>
28    <ask_question name="bet" author="{object.pilot@this.opponent}" text="{8737,1}"/>
29  </do_all>
30</action>

```

Die einfachen zuerst:

1&24: Die `<action>` Node tags.

2&23: `<do_all>` Sub-node Tags damit alle Aktionen ausgeführt werden.

3&8: Wir benutzen `<create_ship>` und seine Sub-nodes um die Eigenschaften unseres Gegners festzulegen und speichern ihn als Objekt in `,this.opponent'`. Im Cue `,selectOpponent'` haben wir bereits das Model (`typename`) seines Schiffes festgelegt. `racellogic="0"` verhindert das der Pilot sich für ein normales Völkerschiff hält, in diesem fall ein (`race=`) Argone mit der selben Klasse (`class`) wie das Spielerschiff. Durch `highlight="1"` wird der Gegner auf der Sektorkarte hervorgehoben und erscheint, in der Objektliste, unter dem Spielerschiff.

- 4: Das Schiff wird zwischen 5 und 6 Kilometer (**min** / **max**) Entfernung zum zuvor gesuchten Sprungtor (**object=**) erstellt. Zur Erinnerung: ‚**this**‘ wird nur in dem Cue benutzt in dem die Variable erstellt wurde. Deswegen haben wir ‚**this**‘ durch den Namen des Cues ersetzt in dem wir das Tor gesucht haben.
- 5: Hier spendieren wir unserem Gegner die Ausrüstung für sein Schiff. Über ‚**loadout**‘ können wir ihm ganz einfach einen Satz mitgeben. Dabei kann man aus mehreren Stufen wählen wieviel er bekommen soll:
- 1.: **minimum** - einen 1MW Schild, einen Satz Laser, keine Raketen, keine Tunings und wenig Software.
 - 2.: **default** - Beste Schilde (mit geringer chance das es etwas weniger ist), Volle Bewaffnung (mit einer geringen Chance auf ‚unübliche‘ Laser), einige Raketen, etwas tuning und etwas bessere Software.
 - 3.: **maximum** - Beste Schilde, Volle Bewaffnung, Raketen, und eine ähnliche Ausrüstung wie default
- 6: Im **<create_ship>** Node können wir ihm gleich ein Kommando geben, so das unser Gegner direkt etwas zu tun hat. In diesem Fall soll er dem Spieler (**{player.ship}**) folgen (**follow**).
- 7: Im **<pilot>** Node geben wir an welchen Namen (**name=**) er haben soll und welchem Volk (**race=**) er angehört. **{random.pilot.argon}** gibt uns einen Zufälligen Argonischen Namen, meistens wird hier ein Name direkt als Text, oder noch besser als Text ID angegeben.
- 9: Hier wollen wir in der Variable **{this.tunings}** den in ‚**exact=**‘ berechneten Wert speichern. Das ist die Höchstgeschwindigkeit vom Spielerschiff (**player.ship.maxspeed**) geteilt durch ein Zehntel der Grundgeschwindigkeit (**object.basespeed**) des Gegnerschiffs (**this.opponent**) minus 10. Dieser Wert kann als Multiplikator für die Tunings des Gegnerschiffs genutzt werden.
- 10&13: Über den **<add_equipment>** Sub-node können wir dem Schiff zusätzliche Erweiterungen geben.
- 11: In dieser Zeile entfernen wir 100 Engine Tunings vom Schiff des Gegners, das müßten theoretisch alle Tunings des Schiffes sein.
- 12: Und weil wir Großzügig sind bekommt das Gegnerschiff eine zufällige Anzahl Tunings zwischen **{value@this.tunings}-1** und **{value@this.tunings}+1** zurück.
- 14&21: Hier benutzen wir einen **<do_if>** Sub-node, ähnlich dem **<do_choose>** muss eine Bedingung erfüllt sein damit die geschachtelten Aktionen ausgeführt werden. Wir überprüfen ob das Gegnerschiff ein M5 ist, durch **negate=“1“** wird die Abfrage ‚umgedreht‘. Wodurch die Bedingung nur erfüllt wird wenn das Schiff kein M5 ist. **{object.class@this.opponent}** ist die Interne ID der Klasse des Gegnerschiffes und **{lookup.class@M5}** die Interne ID der Klasse mit der wir das Schiff vergleichen.

15-17: Wenn das Gegnerschiff kein M5 ist, bekommt es hier zwischen 15 und 50 Energiezellen.

18-20: Zusätzlich bekommt das Schiff einen Sprungantrieb. Der wird für das Rennen zwar nicht gebraucht, ist aber eine Elegante Art das Schiff hinterher verschwinden zu lassen. Das ist auch der Grund warum es kein M5 sein soll, die können in der Regel keinen Sprungantrieb tragen.

22: Die letzte Aktion in diesem Cue ist ein `<ask_question>` Node der wir den namen ,bet‘ geben, hier wird der Spieler gefragt ob er ein Rennen machen möchte. Über den Namen (bet) können wir in den folgenden Sub-cues die vom Spieler gegebene Antwort abfragen. Das ganze wird dem Spieler als , eingehende Nachricht‘ geschickt. Als Absender (author) geben wir den Namen unseres Gegners an, für den eigentlichen Text eine TextID. Man kann den Text zwar auch direkt angeben, aber für Übersetzungen in andere Sprachen und für nachträgliche Korrekturen ist es einfacher eine TextID zu benutzen. So ziemlich alle veröffentlichten Mission benutzen TextIDs. Wie man diese benutzt wird im Kapitel „Text ID und Sprachdatei“ Erklärt

Hui, das war jetzt eine ganze menge Stoff. Dafür haben wir unseren Gegner und die Anfrage zum Rennen ist auch schon an den Spieler gegangen. Bevor es weiter geht schauen wir uns aber noch die gestellte Frage etwas genauer an:

```
Ich bin {object.pilot@createopponent.opponent} von den Argonen. Wie ich sehen kann  
haben Sie ein schnelles Schiff!\nSollen wir um {value@S01M25S00prepare.reward}  
Credits wetten das ich schneller an der {object.name@S01M25S00prepare.finish} bin  
als Sie?\nDamit sie nicht lange suchen müssen, das ist die große Station in der Mitte des  
Sektors.\n[center][select value="ja"]Los geht's![/select][center]\n[center][select value="nein"]Nein Danke![/select][center]
```

Gehen wir erstmal die Variablen durch: `{object.pilot@createopponent.opponent}` ist der Name des Piloten unseres Gegnerschiffes, diesmal benutzen wir den Cuenamen anstatt ,this‘ um sicherzustellen das die Variable auch gefunden wird. Der Gewinn (`value@S01M25S00prepare.reward`) und das Ziel (`object.name@S01M25S00prepare.finish`) kommen aus dem Top-level Cue, auch diese beiden benutzen den Cuenamen anstelle von ,this‘. `[select value]` ist der Name mit dem wir die Antwort später abfragen können, und z.B. ,Nein Danke‘ der im Spiel angezeigte Text. `[center]` sorgt dafür das die Knöpfe in der Nachricht mittig ausgerichtet sind.

Wenn wir `<ask_question>` benutzen brauchen wir mindestens einen Sub-cue, normalerweise aber Zwei - jeweils einen für die ,ja‘ Antwort und einen für ,nein‘. Jeder Cue prüft dabei auf seine Antwort und führt dann entsprechende Aktionen aus. Alternativ kann auch ein Sub-cue mit `<do_choose>` oder `<do_if>` benutzt werden.

Schauen wir jetzt was passiert wenn wir auf die Anfrage von `this.opponent` antworten.

Da die jeweiligen Aktionen der Antworten Sub-cues sind schreiben wir als nächstes `<cues>` gleich unterhalb des `</action>` Tags. Und danach öffnen wir den ersten Sub-cue:

```
</action>
  <cues>
    <cue name="noBet">
```

Wie erwartet wird im `<condition>` Node geprüft ob die Antwort ‚nein‘ war.

```
<condition>
  <question_answered question="bet" answer="no" />
</condition>
```

Beachten sie das ‚bet‘ in die ‚question=‘ Eigenschaft eingetragen ist, das ist der Name den wir der Frage (`ask_question`) im übergeordneten Cue (Parent-cue) zugewiesen hatten. Wenn also die Antwort nein war folgt der `<action>` Node:

```
<action>
  <do_all>
    <incoming_message author="{object.pilot@createopponent.opponent}"
                      text="{8737,9}" />

    <cancel_cue cue="closetoplayer" />
  </do_all>
</action>
```

Der Text von `text="{8737,9}"` ist „Ich hätte wissen müssen das jemand mit so einem Seelenverkäufer sich auf kein Rennen einläßt. Flieg schnell nach Hause, bevor der Kahn auseinander fällt!“

Im vergleich zum vorherigen Cue ist dieser recht überschaubar, es wird eine Nachricht an den Spieler geschickt und der Cue ‚closetoplayer‘ wird abgebrochen. Das wäre der Cue der die ‚ja‘ Antwort behandelt, es wird also die gesamte Mission abgebrochen. Übrigens werden, wenn man einen Cue abbricht, auch alle dazugehörigen Sub-cues abgebrochen.

Bevor wir uns dem ‚closetoplayer‘ Cue zuwenden gibt es in diesem noch einen ganz kleinen Sub-cue:

```
<cues>
  <cue ref="clean_up">
    <params>
      <param name="time" value="10s" />
    </params>
  </cue>
</cues>
```

Dieser kleine Cue, hübsch eingepackt zwischen `<cues>` Tags, ist ein Sub-cue von ‚noBet‘ und wie alle Sub-cues kommt auch dieser direkt nach dem `</action>` Tag. Wie Sie sich vielleicht erinnern wird durch die `ref=` Eigenschaft eine Library aufgerufen. Man kann es sich so vorstellen als würde man den Library Cue Kopieren und an dieser Stelle einfügen. Mit dem `<params>` Node und seinen untergeordneten `<param>` Nodes kann man, einen oder mehrere, Parameter an die Library übergeben. Wie der Name verrät, benutzen wir die Library um

unsere Mission aufzuräumen, sprich unseren Gegner aus dem Spiel zu entfernen. Wie das genau gemacht wird sehen wir später wenn wir uns dem ‚clean_up‘ Cue widmen. Eine Erklärung zur Verwendung von Librarys finden sie auch in [Anhang 2](#).

Der nächste Cue ist, genau wie ‚noBet‘ ein Sub-cue von ‚createOpponent‘, damit dieser nicht aus versehen ein Sub-cue von ‚noBet‘ wird müssen wir diesen erst mit </cue> schließen.

```
</cue>
<cue name="closetoplayer">
```

Wie Sie sehen können ist dies der Cue der in ‚noBet‘ abgebrochen wird (mit allen seinen Sub-cues) wenn auf die vorhin gestellte Frage mit ‚nein‘ geantwortet wurde. Für diesen Cue sind wir aber Optimistisch und Hoffen das die Antwort ‚ja‘ war. Genau wie im vorherigen Cue sind die <condition> recht einfach, nur das diesmal auf die ‚ja‘ Antwort geprüft wird.

```
<condition>
  <question_answered question="bet" answer="yes" />
</condition>
```

An dieser Stelle werden die Unterschiede aber wieder größer, wir benutzen einen <timing> Node. Dieser sagt uns der <action> Node zwischen 1 und 3 Sekunden nach erfüllen der <condition> ausgeführt wird.

```
<timing>
  <time min="1s" max="3s"/>
</timing>
```

Als nächstes: <action>:

```
<action>
  <do_all>
1    <incoming_message author="{object.pilot@createopponent.opponent}"
                                   text="{8737,2}" popup="1"/>
2    <create_object name="this.marker" typename="SS_SPECIAL_MARKER"
                                   class="special">
      <position min="3km" max="5km" object="{player.ship}"/>
    </create_object>
3    <set_target object="this.marker"/>
4    <set_command object="createopponent.opponent" command="moveposition"
                                   commandobject="this.marker">
      <position object="this.marker"/>
    </set_command>
  </do_all>
</action>
```

Hier wird wieder `<do_all>` verwendet, diesen Node kennen sie bereits. Ich gehe davon aus das Sie mittlerweile mit den wichtigsten Nodes (`< cues>`, `< cue>`, `< condition>`, `< timing>`, `< action>`, `< do_all/any>` und `< check_all/any>`) vertraut sind. Deswegen werde ich ab hier nur noch die Zeilen erklären in denen auch etwas passiert.

1: Als erstes bekommt der Spieler eine Nachricht. Anders als die ‚Eingehenden Nachricht‘ mit ihrem bekannten Sound, wird diese dem Spieler sofort angezeigt (`popup=“1“`). Der ‚author‘ ist wieder der Pilot unseres Konkurrenzschiffes, und als Text eine TextID mit der Nachricht:

Großartig! Ich schalte meine Freund/Feind Kennung aus, dann kannst du mich als roten Punkt auf deinem Radar sehen. Bequemer geht’s nicht. Aber nicht auf mich schießen, ich bin nicht dein Feind! Fliege jetzt zu den markierten Koordinaten, denk‘ dran das du auf 200 Meter an die Markierung heranfliegen musst.

2: In dieser Zeile wird die Markierung erstellt und sie heißt: ‚`this.marker`‘. Zur Erinnerung, in den Sub-cues wird es als ‚`closetoplayer.marker`‘ verwendet. Für den ‚`typename`‘ scrollen wir in der Popup-Liste ein Stück nach unten um ‚`SS_SPECIAL_MARKER`‘ zu finden, da es ein Spezialobjekt ist, ist als Klasse (`class=`) ‚`special`‘ angegeben. Der `< position>` Sub-Node gibt an das das Objekt zwischen 3 und 5 Kilometer entfernung vom Spielerschiff erstellt werden soll.

3: Mit dem `< set_target>` Node wird die Markierung vom Spielerschiff als Ziel anvisiert. Nochmal zur Erinnerung, wir benutzen `this.marker` weil wir noch im selben Cue sind wo dieses Objekt erstellt wurde.

4: Nachdem der Spieler sich hoffentlich auf den Weg zur Markierung gemacht hat, schicken wir jetzt auch das Gegnerschiff (`object=`) dorthin. Das Kommando (`command=`) ‚Fliege zu Position‘ hat als Ziel (`commandobject=`) die vorhin erstellte Markierung.

Schauen wir mal was in der Mission bis jetzt passiert. Wir haben eine Station gesucht (das Ziel) und ein Sprungtor, in dessen Nähe das Rennen beginnt. Wenn beide gefunden sind wird ein Gegner erstellt, der den Spieler zu einer Wette herausfordert. Wenn der Spieler ablehnt wird die Mission mit allem drum und dran abgebrochen, und wenn er darauf eingeht wird eine Markierung (unsere Startlinie) erstellt wo beide Kontrahenten sich sammeln sollen.

Wie früher schon angedeutet werden alle Aktionen, die auf die Frage (`bet`) folgen, in Sub-cues der jeweiligen Antwort behandelt. Genau wie das abbrechen der Mission ein Sub-cue der ‚nein‘ Antwort war, wird das Rennen jetzt in Sub-cues der ‚ja‘ Antwort weitergeführt.

Zur Abwechslung schauen wir uns den nächsten Cue mal im ganzen an:

```

<cues>
  <cue name="coward">
    <condition>
      <check_all>
        <object_changed_sector/>
      </check_all>
    </condition>
    <timing>
      <time min="1s"/>
    </timing>
    <action>
      <do_all>
        <incoming_message author="{object.pilot@createopponent.opponent}"
                          text="{8737,114}" popup="1"/>
        <destroy_object object="closetoplayer.marker"/>
        <cancel_cue cue="inposition"/>
      </do_all>
    </action>
  </cue>
</cues>

```

Dadurch können sie jetzt gut den Aufbau des Cues erkennen und wie die Nodes `<condition>`, `<timing>` und `<action>` auf der selben Ebene liegen. Beachten Sie das `<condition>` und sein `<check_all>` Sub-node nur eine Bedingung prüfen. Die `<check_all>` Tags werden in einem solchen Fall zwar nicht zwingend benötigt, ist aber dennoch Praktisch sie anzugeben falls man im nachhinein noch etwas hinzufügen möchte. `<object_changed_sector>` kennen wir aus dem Top-level Cue unserer Mission, dieses Ereignis wird ausgelöst wenn der Sektor gewechselt wird. Und weil kein Objekt angegeben ist, ist das Spilerschiff der auslöser. Wenn, aus welchem Grund auch immer, der Spieler den Sektor verlässt, wird eine Sekunde danach der `<action>` Node ausgeführt. In diesem gibt es 3 Aktionen: Als erstes wird dem Spieler eine Nachricht geschickt, „Feigling, Ich hab gesehen wie du geflüchtet bist! Das war's mit der Wette.“. Danach wird mit `<destroy_object>` die Markierung entfernt, beachten Sie das der Cuenamen anstatt `,this'` verwendet wird. Und als letzte Aktion wird der Cue `,inposition'`, mit allen in ihm vorhanden Sub-cues abgebrochen.

Sollten sie sich an dieser Stelle wundern, warum wir so viele Cues Abbrechen oder Neustarten, sollten Sie wissen das die Spiel-Engine in jeder Sekunde alle Missionen aus den XML Dateien durchsucht. Dabei wird jedesmal in der gesamten Datei überprüft ob die `<condition>` Nodes erfüllt werden, und falls ja geht's mit den Sub-cues weiter, solange bis die Datei komplett überprüft wurde. Wenn man die nicht benötigten Cues nicht abbricht werden die `<condition>` solange überprüft bis sie eventuell erfüllt werden, und `<action>` wird ausgeführt auch wenn die Mission eigentlich schon beendet sein müsste.

Kommen wir jetzt zum nächsten Cue, der ein Sub-cue von `,coward'` ist:

```

<cues>
  <cue ref="clean_up" comment="ab hier 10 Sekunden und der Gegner springt in
    einen anderen Sektor um entfernt zu werden"/>
    <params>
      <param name="time" value="10s"/>
    </params>
  </cue>
</cues>
</cue> <- Dieser Tag schließt den ,coward' cue

```

Dieser kleine Cue dürfte ihnen schon Bekannt sein, über die `ref=` Eigenschaft wird eine Library mit den Namen `clean_up` aufgerufen. Der Kommentar im Code beschreibt in diesem Fall ganz gut was hier passieren soll. Beachten sie auch den zusätzlichen `</cue>` Tag am ende, dieser schließt den übergeordneten (Parent-) Cue `,coward'`. Der nächste Cue wird damit auf der selben Ebene sein wie `,coward'`, ein genauso ein Sub-cue von `,closetoplayer'` sein:

```

<cue name="opponentready" comment="Dieser cue wird von der KI benötigt">
  <condition>
    <check_all>
1      <cue_is_complete cue="closetoplayer"/>
2      <object_position object="createopponent.opponent" max="200m">
        <position object="parent.marker"/>
      </object_position>
    </check_all>
  </condition>
  <action>
3   <set_command object="createopponent.opponent" command="none"/>
  </action>
</cue>

```

Der `<condition>` Node ist der umfangreichste, gehen wir die Punkte kurz durch:

- 1: Die `<action>` vom Parent-Cue muss beendet sein.
- 2: Das Objekt `,createopponent.opponent'` darf nicht weiter als 200m von `parent.marker` entfernt sein. Parent benutzen wir weil `,marker'` im parent-cue (dem übergeordneten cue) erstellt wurde, genausogut könnte man `,closetoplayer.marker'` schreiben, es würde beides funktionieren.

Sobald beide Bedingungen erfüllt sind geht es mit dem `<action>` Node weiter:

- 3: Hier ersetzen wir das `,Fliege zu Position'` Kommando durch `,kein' (none)`. Der Gegner hat somit nichts zu tun bis der Spieler eingetroffen ist.

Der nächste Cue ist auch wieder auf der selben Ebene wie die letzten Beiden und ebenso ein Sub-cue von ‚`closetoplayer`‘. Die `<condition>` sind ähnlich dem letzten Cue, nur prüfen wir diesmal ob der Gegner und der Spieler in der Nähe der Markierung sind:

```
<cue name="inposition">
  <condition>
    <check_all>
1      <cue_is_complete cue="closetoplayer"/>
2      <object_position object="createopponent.opponent" max="200m">
          <position object="parent.marker"/>
        </object_position>
3      <object_position max="200m">
          <position object="parent.marker"/>
        </object_position>
    </check_all>
  </condition>
  <action>
    <do_all>
4      <incoming_message author="{object.pilot@createopponent.opponent}"
                          text="{8737,8}" popup="1"/>
5      <set_command object="createopponent.opponent" command="none"/>
6      <set_relation object="createopponent.opponent">
          <relation relation="enemy" object="{player.ship}"/>
        </set_relation>
7      <set_target object="S01M25S00prepare.finish"/>
    </do_all>
  </action>
```

Wie im letzten Cue prüfen wir ob:

- 1: die Aktionen im ‚`closetoplayer`‘ Cue beendet wurden.
- 2: und ob der Gegner höchstens 200m von der Markierung Entfernung ist
- 3: Zusätzlich prüfen wir ob auch der Spieler höchstens 200m entfernt ist. Wie früher schon erwähnt wurde, wird der Spieler als Objekt genommen wenn die ‚`object=`‘ Eigenschaft nicht angeben ist.
- 4: Wenn dann beide Schiffe nah genug an der Markierung sind wird die `<action>` ausgeführt. Der Spieler bekommt eine Popup Nachricht vom Gegner: ‚`Gut, da wir jetzt in Position sind - Starten wir den Countdown...`‘
- 5: Wir sagen dem Gegner das er stehenbleiben soll indem wir ihm das Kommando ‚`Kein`‘ (`none`) geben.

6: Mit `<set_relation>` setzen wir die Beziehung (`relation=`) zwischen dem Gegnerschiff (`object=`) und dem Spieler (`object=`) auf Feindlich (`enemy`), so wie es in der Nachricht aus dem übergeordneten Cue stand (Freund/Feind Kennung).

7: Als letzte Aktion in diesem Cue bekommt der Spieler die Zielstation anvisiert.

Vielleicht haben sie sich gewundert warum das Gegnerschiff in diesem Cue ebenfalls das ‚kein‘ Kommando bekommt: Im vorherigen Cue wurde geprüft ob der Gegner bis auf 200m an die Markierung herankommt. Für den Fall das der Gegner schneller als der Spieler dort ist, wird sein Schiff gestoppt. Sollten allerdings beide in etwa Zeitgleich dort ankommen, wissen wir nicht welche Aktionen in welchem Cue ausgeführt werden, es stellt also sicher das der Gegner auch wirklich stehen bleibt.

Die Nachricht aus diesem Cue gibt schon einen Tip was im nächsten Cue, einem Sub-cue von ‚`inposition`‘ passieren soll. Wir starten einen Countdown:

```
< cues >
  < cue name="countdown" comment="zählt runter von 4 nach 1" >
    < condition >
1      < cue_is_complete cue="inposition" />
    < /condition >
    < timing >
2      < count exact="4" />
3      < time min="2s" />
4      < interval exact="1s" />
    < /timing >
    < action >
      < do_all >
5        < set_value name="this.counter" exact="5-{index@this}" comment="Die Zahl
          kann nicht im Text Berechnet werden, also benutzen wir eine Variable" />
6        < play_subtitles author="{object.pilot@createopponent.opponent}"
          text="{value@this.counter}!" />
7        < play_sound soundid="924" />
      < /do_all >
    < /action >
  < /cue >
```

Dieser Cue ist recht einfach, auch wenn der `<action>` Node nicht so aussieht.

1: Diese Zeile hatten wir bereits öfter, der Cue ‚`inposition`‘ muss beendet sein.

Der `<timing>` Node dagegen sieht etwas anders aus, er enthält diesmal `<count>`, `<time>` und `<interval>` Nodes. Schauen wir uns die einzelnen Sub-nodes von `<timing>` mal an:

2: `<count exact="4" />` bedeutet das der gesamte `<action>` Node genau 4 mal ausgeführt wird.

3: `<time min="2s"/>` bedeutet, wie wir es kennen, das die `<action>` mit 2 Sekunden verzögerung ausgeführt wird. (In diesem Fall nur der erste Durchgang)

4: `<interval exact="1s"/>` Dies ist die Verzögerung für die nachfolgenden durchgänge. Die `<action>` wird, im Interval, jede 1 Sekunde ausgeführt.

Bevor wir uns dem `<action>` Node zuwenden, erinnern wir uns das dieser 4 mal ausgeführt wird, insbesondere bei der Verwendung von `<set_value>`. Zum besseren Verständnis gehen wir die Aktionen aber, wie gehabt, einzeln durch.

5: Hier geben wir der Variable `,this.counter'` einen Wert von 5 minus `{index@this}`. Das heißt, das jedesmal wenn der `<action>` Node ausgeführt wird `{index@this}` um 1 erhöht wird, diese Variable zählt die einzelnen Durchgänge (angefangen bei 1). Das gesamt Ergebnis ist dann ein Countdown der, bei 4 angefangen, mit jeder Sekunde runterzählt. Wie `,comment='` schon verrät kann die Berechnung nicht Direkt im Text angegeben werden, deswegen dieser kleine Umweg über eine Variable.

6: In dieser Zeile geben Wir den Countdown als Untertitel auf dem Bildschirm aus, als Sprecher (`,author'`) ist dabei der Gegner angegeben.

7: Schließlich geben wir noch einen Alarm (ID 924 = alert) dazu, der jede Sekunde ertönt. Auf diese weise hat der Spieler zusätzlich noch einen akkustischen Anhaltspunkt.

Damit ist dieser Cue auch schon fertig, er bekommt auch keine Sub-cues. Der nächste Cue ist damit wieder ein Sub-cue von `,inposition'`, wo abgefragt wurde ob der Spieler bei der Markierung ist. Diesmal wollen wir abfragen ob der Spieler einen Frühstart hinlegt:

```
<cue name="headstart">
  <condition>
    <check_all>
1      <cue_timer cue="inposition" min="3s" max="6s"/>
2      <object_position min="220m">
          <position object="closetoplayer.marker"/>
        </object_position>
    </check_all>
  </condition>
  <action>
    <do_all>
3      <incoming_message author="{object.pilot@createopponent.opponent}"
                                     text="{8737,16}" popup="1"/>
4      <destroy_object object="closetoplayer.marker"/>
5      <cancel_cue cue="countdown"/>
6      <cancel_cue cue="coward"/>
7      <cancel_cue cue="go"/>
    </do_all>
  </action>
```

- 1: Der `<cue_timer>` Node überprüft die Zeit die der angegeben Cue (`inposition`) Aktiv war, in diesem Fall zwischen 3 und 6 Sekunden.
- 2: Hier wird geprüft ob der Spieler mindestens 220 Meter von der Markierung entfernt ist. Zur Erinnerung, wenn kein Objekt angegeben ist wird das Spielerschiff geprüft. Sollte der Spieler schummeln und zu Früh starten folgt somit der `<action>` Node:
- 3: Als erstes bekommt der Spieler eine Nachricht: ‚Du Ehrenloser Betrüger! Ich hab‘ genau gesehen wie du zu Früh losgeflogen bist, die Wette kannst du vergessen.‘
- 4: Dann wird die Markierung entfernt, die wir als Startpunkt erstellt hatten.
- 5-7: Zuletzt werden noch die Cues ‚countdown‘, ‚coward‘ und ‚go‘ mitsamt ihren Sub-cues abgebrochen. Damit ist die Mission endgültig beendet.

Der nächste Cue wird ein Sub-cue von ‚headstart‘ und sollte ihnen bereits bekannt sein. Wir rufen die Library ‚clean_up‘ auf um die, in dieser Mission, erstellten Objekte loszuwerden.

```

<cues>
  <cue ref="clean_up" comment="In 10 Sekunden wird unser Gegner den Sektor
                                wechseln und wird zerstört">
    <params>
1     <param name="time" value="10s"/>
    </params>
  </cue>
</cues>
</cue>      <- Hier wird der „headstart“ cue geschlossen.

```

Die Library, und was dort genau passiert, erkläre ich gegen Ende der Mission. Sie ist der letzte Cue in dieser Datei. Sehen wir uns aber erstmal an wie die kühnen Piloten das Rennen beginnen:

```

<cue name="go">
  <condition>
1   <cue_is_complete cue="inposition"/>
  </condition>
  <timing>
2   <time min="7s"/>
  </timing>
  <action>
    <do_all>
3   <play_subtitles author="{object.pilot@createopponent.opponent}" text="{8737,10}"/>
4   <play_sound soundid="923"/>
5   <set_command object="createopponent.opponent" command="dock"
                                commandobject="S01M25S00prepare.finish"/>
6   <destroy_object object="closetoplayer.marker"/>
    </do_all>
  </action>

```

- 1: Die `<action>` soll erst ausgeführt werden wenn `,inposition‘` seine Aktionen beendet hat. Das Rennen kann also erst starten wenn die Teilnehmer ihre Position eingenommen haben.
- 2: Im `<timing>` Node setzen wir eine verzögerung von 7 Sekunden.
- 3: Wenn die Bedingungen erfüllt sind kann nach 7 Sekunden das Startsignal kommen, wir benutzen die Untertitel um `,LOS!‘` zu sagen
- 4: Passend dazu soll auch ein Alarmton gespielt werden. (`soundid="923"`)
- 5: Hier geben wir dem Gegner das Kommando auf der Zielstation zu Landen, das Rennen hat also endlich begonnen.
- 6: Als letztes wird die Markierung entfernt, diese wird nicht mehr gebraucht.

Der nächste Cue ist ein Sub-cue von `,go‘`:

```

<cues>
  <cue name="annoyme">
    <condition>
1     <cue_is_complete cue="go"/>
    </condition>
    <timing>
2     <count min="3" max="6"/>
3     <time min="30s" max="60s"/>
    </timing>
    <action>
    <do_all>
4     <set_value name="this.txtid" min="21" max="40"/>
5     <incoming_message popup="1" temporary="1" author="{object.pilot@create-
        opponent.opponent}" text="{8737,{value@this.txtid}}"/>
    </do_all>
    </action>
6 </cue>

```

- 1: Eine einfache Bedingung die Sie mittlerweile schon gut kennen sollten, der Cue `,go‘` muss seine `<action>` beendet haben.
- 2: Der Zähler (`count`) in diesem `<timing>` Node sorgt dafür das die `<action>` zwischen 3 und 6 mal ausgeführt wird. Die genaue Anzahl ist zufällig.
- 3: Nach einer zufälligen Zeit, zwischen 30 und 60 Sekunden, nachdem das Rennen begonnen hat wird die `<action>` ausgeführt.
- 4: Hier speichern wir in einer Variable eine zufällige Zahl zwischen 21 und 40, dabei sollten wir bei jedem durchgang eine andere Zahl bekommen.

5: Das resultat aus dem <timing> Node und der <set_value> Zeile ist, das wir dem Spieler nach einiger Zeit (30 bis 60 Sek.) 3 bis 6 unterschiedliche Nachrichten schicken können. Der Eintrag **temporary="1"** bedeutet das diese Nachricht nicht in den Empfangenen Nachrichten gespeichert wird. Beachten Sie auch das ‚value@this.txtid‘ als Variable innerhalb einer Variable zusätzlich in {} gesetzt ist.

Das war schon das Ende dieses Cues, als nächstes einen weiteren Sub-cue von ‚go‘:

```

<cue name="winner">
  <condition>
    <check_all>
1      <object_is_docked dockobject="S01M25S00prepare.finish"/>
2      <object_is_docked dockobject="S01M25S00prepare.finish"
                                object="createopponent.opponent" negate="1"/>
    </check_all>
  </condition>
  <action>
    <do_all>
3      <cancel_cue cue="loser"/>
4      <cancel_cue cue="coward"/>
5      <cancel_cue cue="annoyme"/>
6      <play_sound soundid="1007"/>
7      <incoming_message author="{object.pilot@createopponent.opponent}"
                                text="{8737,3}" popup="1"/>
8      <set_relation object="createopponent.opponent">
          <relation relation="friend" object="{player.ship}"/>
        </set_relation>
9      <reward_player>
          <money exact="{value@S01M25S00prepare.reward}"/>
        </reward_player>
    </do_all>
  </action>

```

1&2: Die <condition> Sub-nodes sind fast gleich. Im ersten wird geprüft ob der Spieler am Ziel angedockt hat, wie Sie sich vielleicht erinnern brauchen wir den Spieler nicht extra anzugeben. Im anderen Node wird der Gegner geprüft, wie ‚negate =“1“‘ vielleicht schon verrät darf dieser nicht angedockt sein.

Wenn also der Spieler das Rennen gewonnen hat, kann die <action> losgehen:

3-5: Die Cues ‚loser‘, ‚coward‘ und ‚annoyme‘ können mitsamt ihren Sub-cues abgebrochen werden.

6: Der Sound 1007 wird abgespielt. ((unused) Ding Da Dong Stations Bekanntmachung Variation 2))

- 7: Hier erhält der Spieler eine Popup-Nachricht mit dem Text: „Ich bin {object.pilot@createopponent.opponent} von den Argonen. Du hast gewonnen! Ich überweise dir {value@S01M25S00prepare.reward} Credits, aber dafür spendierst du noch einen Drink in der Bar!“
- 8: Unser Gegner ist ein guter Verlierer und setzt in diesem Sub-node seine Freund/Feind Kennung wieder auf Freund.
- 9: In diesem Node bekommt der Spieler schließlich seinen Gewinn, diesen haben wir ganz am Anfang im Top-level Cue festgelegt.

Während unsere Piloten den Sieg des Spielers feiern, kümmern wir uns noch um einen Sub-cue: Dem bereits bekanntem aufruf der ‚clean_up‘ Library:

```

<cues>
  <cue ref="clean_up" comment="Diesmal warten wir 5 Minuten bevor wir
    das Schiff des Gegners verschwinden lassen.">
    <params>
      <param name="time" value="5m"/>
    </params>
  </cue>
</cues>
</cue>    <- Mit diesem Tag wird der ‚winner‘ cue geschlossen

```

Wie bei den vorherigen malen räumen wir mit der Library hinter uns auf, diesmal warten wir allerdings 5 Minuten. Derzeit werfen wir einen Blick darauf was passiert sollte der Spieler verlieren:

Auf dem ersten Blick gibt es keine großen Unterschiede zwischen dem ‚loser‘ und dem ‚winner‘ Cue. Selbst der Sub-cue wo wir die Library aufrufen hat den selben ‚time‘ Parameter.

```

<cue name="loser">
  <condition>
    <check_all>
1      <object_is_docked dockobject="S01M25S00prepare.finish" negate="1"/>
2      <object_is_docked dockobject="S01M25S00prepare.finish"
        object="createopponent.opponent"/>
    </check_all>
  </condition>
  <action>
    <do_all>
3      <cancel_cue cue="winner"/>
4      <cancel_cue cue="coward"/>
5      <cancel_cue cue="annoyme"/>
6      <play_sound soundid="1007"/>

```

```

7   <incoming_message author="{object.pilot@createopponent.opponent}"
                                     text="{8737,4}" popup="1"/>
8   <set_relation object="createopponent.opponent">
9     <relation relation="friend" object="{player.ship}"/>
    </set_relation>
    <reward_player>
10  <money exact="-{value@S01M25S00prepare.reward}"/>
    </reward_player>
    </do_all>
  </action>
  <cues>
11  <cue ref="clean_up" comment="Nach 5 Minuten fliegt unser Gegner davon um
                                     entfernt zu werden">
    <params>
      <param name="time" value="5m"/>
    </params>
  </cue>
</cues>
</cue>      <- Dieser Tag schliesst den 'loser' cue.

```

1&2: Der einzige Unterschied zum ‚winner‘ Cue ist das diesmal der Spieler mit ‚negate=“1“‘, geprüft wird, es darf also nur der Gegner angedockt sein.

3-6: In diesen Zeilen ist der Unterschied das der ‚winner‘ Cue (und seine Sub-cues) abgebrochen wird anstatt des ‚loser‘ Cues.

7: Natürlich bekommt der Spieler auch wieder eine Nachricht, der Inhalt ist allerdings etwas anders: “Hier spricht {object.pilot@createopponent.opponent} von den Argonen. Das war knapp aber ich war der Schnellste! Ich bin der Beste! Wo ist mein Geld?! \n\nWunderbar, na komm, ich spendier noch einen Drink.”

8&9: Auch wenn der Spieler verliert setzt der Gegner seine Beziehung zu ihm wieder auf ‚Freund‘, er ist also wieder Blau. Und nicht wegen dem Drink :)

10: Im <reward_player> Node passiert das genaue Gegenteil zum ‚winner‘ Cue, die Credits werden dem Spieler abgezogen. Darum haben wir am Anfang auch geprüft ob der Spieler genug Credits dabei hat.

An dieser Stelle können sie tief durchatmen, und sich Gratulieren das Sie bis hier durchgehalten haben. Sie haben jetzt schon fast die gesamte Mission fertig. Es müssen noch ein paar Cues geschlossen werden und dann werden wir endlich die Library behandeln, die schon so oft angesprochen wurde. Das wichtigste beim Arbeiten mit Librarys ist zu beachten das sie in übergeordneten Cues, oder zumindest auf der selben Ebene liegen müssen. Ob sie am Anfang der Datei, irgendwo in Mitte oder am Ende plaziert werden, spielt dabei keine Rolle. So ist die Library aus diesem Beispiel am Ende der Datei, liegt aber ‚über‘ den Cues in denen sie, über die ‚ref=‘ Eigenschaft ‚verwendet wird. Dazu müssen aber erst noch ein paar Cues geschlossen werden:

```

    </cue> <- Dieser Tag schliesst den „go“ cue.
  </cues>
  </cue> <- Dieser Tag schliesst den „inposition“ cue.
</cues>
  </cue> <- Dieser Tag schliesst den „closetoplayer“ cue.
</cues>
</cue> <- Und dieser Tag schliesst den „createopponent“ cue.

```

Nachdem diese 4 Cues geschlossen sind befinden wir uns wieder auf der selben Ebene wie die Cues ,reset‘, ,selectOpponent‘ und ,createopponent‘. Auf dieser Ebene wollen wir unsere Library plazieren.

Soweit es diese Mission betrifft, kommt jetzt der letzte Cue... Endlich. Der ,clean_up‘ Cue selbst hat allerdings noch 2 Sub-cues.

```

1 <cue name="clean_up" library="1">
  <timing>
2   <time min="{param@time}" />
  </timing>
  <action>
  <do_all>
3   <find_sector name="this.jumpsec" exact="1" />
4   <find_gate name="this.jumpgate" nearest="1">
    <sector sector="this.jumpsec" />
    </find_gate>
5   <set_command object="createopponent.opponent" command="jumpsector"
      commandobject="this.jumpgate">
    <sector sector="this.jumpsec" />
    </set_command>
  </do_all>
</action>

```

1: Um aus einem Cue eine Library Cue zu machen, und so dafür zu sorgen das er vom Parser nicht wie die anderen Cues ausgeführt wird, benutzen wir die **library="1"** Eigenschaft. Als Namen geben wir ihm ,clean_up‘, diesen haben wir auch beim aufrufen über ,ref=‘ benutzt.

2: Im <timing> Node benutzen wir bei <time> die Variable {param@time}, die wir beim aufrufen als Parameter angegeben haben.

3: Als nächstes suchen wir einen Sektor mit einem Sprung Entfernung und speichern ihn in ,this.jumpsec‘.

4: Hier suchen wir ein Sprungtor im gerade gesuchten Sektor und nennen es ,this.jumpgate‘.

5: Als letzte Aktion geben wir dem Gegnerschiff das Kommando in den gesuchten Sektor zu springen. ,commandobject‘ ist dabei das, gerade gesuchte, Tor aus dem das Schiff geflogen kommt.

```

< cues >
  < cue name="destroy opponent">
    < condition >
1      < object_changed_sector object="createopponent.opponent" />
    < /condition >
    < timing >
2      < time min="1s" />
    < /timing >
    < action >
3      < do_all >
        < destroy_object object="createopponent.opponent" />
      < /do_all >
    < /action >
  < /cue >

```

Sie Merken schon, auf den letzten Metern geht es schneller. Wir sind beim ersten Sub-cue:

1: Sobald der Gegner den Sektor gewechselt hat, was wir ihm gerade befohlen haben, geht es schon weiter mit <timing>.

2: Eine Sekunde nach dem Sektorwechsel wird er...

3: ...zerstört

Aber noch sind wir nicht ganz fertig:

```

  < cue name="restart" comment="Nach einer Std. gehts von vorn los">
    < timing >
1      < time min="1h" />
    < /timing >
    < action >
2      < reset_cue cue="S01M25S00prepare" />
    < /action >
  < /cue >
< /cues >
< /cue >  <- Hier wird der „clean_up“ library cue geschlossen
< /cues >

```

1: Eine Stunde nachdem der Gegner zerstört wurde...

2: wird der Top-Level Cue neugestartet und alles geht von vorne los :)

Jetzt müssen nur noch die letzten, oder besser gesagt die ersten Nodes geschlossen werden.

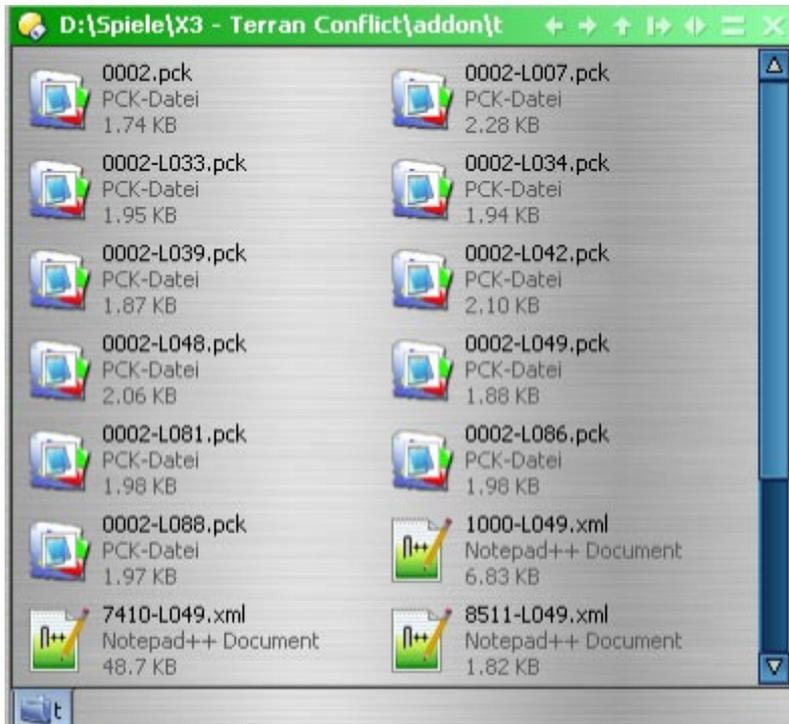
```

  < /cue >  <- Der „S01M25S00prepare“ cue wird geschlossen
< /cues >
< /director >  <- Und hier wird die ganze Datei geschlossen

```


Text ID und Sprachdatei

In der Beispielmision werden dem Spieler ein paar Nachrichten geschickt, die jeweils als Text IDs angegeben wurden. Dieser Text muss natürlich auch irgendwo gespeichert werden, ansonsten bekommt man die bekannten Readtext Fehler. Dafür wird eine Sprachdatei verwendet die sich im ‚t‘ Ordner des Spielverzeichnis befindet (siehe Bild 12).



Die .pck Dateien sind gepackte .xml Dateien und gehören zum Original Spiel.

Bild 12 - der t Ordner

Damit die Sprachdatei verwendet werden kann muss sie erst vom Spiel geladen werden:

```
<action>  
<load_text fileid="8737"/>
```

Wenn Sie jetzt eine TextID, wie {8737,1}, benutzen heißt das, das der Text auf „Seite“ 8737 unter der ID 1 zu finden ist, und in der Sprachdatei 8737-L049.xml gespeichert ist. Die ersten 4 Ziffern stehen für die „Seite“, die letzten 4 Zeichen stehen für die Sprache (z.B. L049 = Deutsch; L044 = Englisch). In der Datei selbst können auch mehrere „Seiten“ angegeben werden. Das der Dateiname mit der Seitenzahl beginnt ist zwar keine Pflicht, sollte aber so gemacht werden. Auf diese Weise kann es nicht so schnell passieren das unterschiedliche Mission (und MSCSI scripte) die selbe Sprachdatei verwenden wollen, ausserdem hilft es beim suchen von readtext Fehlern. Sehen wir uns die Sprachdatei für unsere Mission mal genauer an, angefangen mit dem Dateiheder:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<language id="49">
```

Die erste Zeile kennen wir schon aus der Missionsdatei, hier wird die verwendete Zeichenkodierung festgelegt. in der zweiten Zeile wird, zusätzlich zum Dateinamen, die Sprache festgelegt. 49 steht für Deutsch (oder 44 für Englisch), wer schonmal eine Englische Mod für ein Deutschsprachiges Spiel installiert hat wird diese Zeile kennen. <language> ist ausserdem der öffnen Tag für diese Datei und muss, wie <director> in der Missionsdatei, am Ende wieder geschlossen werden. Schauen wir jetzt wie es weiter geht:

```
<page id="8737" title="Docking Race Bet" descr="Beispiel Texte" voice="no">
  <t id="1">Ich bin {object.pilot@createopponent.opponent} ...</t>
  <t id="2">Großartig! Ich schalte meine Freund/Feind Kennung aus....</t>
  <t id="3">Ich bin {object.pilot@createopponent.opponent} ...</t>
  <t id="4">Hier spricht {object.pilot@createopponent.opponent} ...</t>
  <t id="8">Gut, da wir jetzt in Position sind - Starten wir den Countdown...</t>
  <t id="9">Ich hätte wissen müssen das jemand mit so einem ...</t>
  <t id="10">LOS!</t>
  ...
  ...
</page>
<page id="8738" .....
  <t id="1">text blah blah</t>
  ...
  ...
</page>
</language>
```

Der <page> Node öffnet eine Seite (engl. page).

,ID=' legt die ,Seitenzahl' für diese Seite fest.

,Title=' ist der Titel dieser Seite, z.B. der Name der Mission.

,descr=' ist eine Beschreibung was in dieser Seite zu finden ist.

,voice=' gibt an ob diese Texte vertont sind und abgespielt werden wenn man diese TextID benutzt. In unserem Fall gibt es keine Vertonung (no).

Der gesamte Text kommt dabei in eine Zeile, um den Überblick nicht zu verlieren habe ich sie etwas gekürzt. Hier gibt es nur ,id=' und den dazugehörigen Text, mit seinen Start und End Markierungen. Nach den Texten kommen schliesslich die schliessen Tags </page> und </language>. Wie im Beispiel zu sehen können auch mehrere Seiten vorhanden sein. Im Text können außer den Variablen auch Sonderzeichen genutzt werden um das Erscheinungsbild anzupassen, zum Beispiel:

[center]Text[/center] zentriert den angegeben Text

\n = fängt eine neue Zeile an

\033x = Färbt den Text ein. Um die Farbe zu bestimmen muss x mit einem Buchstaben ersetzt werden, zum Beispiel:

- G=Grün
- R=Rot
- Y=Gelb (eng. Yellow)
- X=Farbe zurücksetzen

Mission Briefing

Eine Elegante Art den Spieler durch die Mission zu führen ist das verwenden eines Missionbriefings. Mit dem Briefing geben Sie dem Spieler sozusagen ein Logbuch in dem steht worum es geht und was als nächstes zu tun ist. Dabei wird z.B. angegeben zu welcher Station der Spieler fliegen soll, wer sein Ziel ist oder auch was die darauffolgenden Schritte der Mission sind. Die im Briefing angezeigten Objekte (Stationen;Schiffe;Personen) müssen vor erstellen des Briefings vorhanden sein. Oftmals ist es nützlich diese Unzerstörbar zu machen um sicher zugehen das die Mission nicht hängen bleibt:

```
<set_invincible object="MeinBriefing.station01" invincible="1"/>
```

Denken sie daran die Objekte mit `invincible="0"` wieder zu Normalisieren sobald sie nicht mehr gebraucht werden. Wenn Sie alle Objekte gefunden oder erstellt haben können sie im `<action>` Node das Briefing erstellen. Empfehlenswert ist es das Briefing in einem Sub-cue zu erstellen, so kann man über `<condition>` prüfen das alle Objekte vorhanden sind und gegebenenfalls eine Alternative anbieten.

```
<create_briefing background="Bild01.tga" title="Titel" text="Text" cue="MeinBriefing">  
  <mission level="easy"/>  
  <reward money="5000"/>  
  <timer duration="5h"/>  
<objectives>
```

Background ist das Hintergrundbild das im Briefing angezeigt wird, im Albion Plot wird z.B. `albion_planet.tga` verwendet. Das Bild muss sich im `director\images` Ordner befinden damit MD es sehen kann. Die im Originalspiel benutzten Bilder sind in den Cat/Dat Dateien und brauchen nicht extra entpackt zu werden. Der **Titel** steht, etwas größer, ganz Oben im Briefing, z.B. der Name der Mission. **Text** ist eine kurze Beschreibung um was es in der Mission geht. Und **cue** ist der Cue in dem das Briefing erstellt wird.

Der `<mission>` Sub-node gibt den Schwierigkeitsgrad an, in diesem Fall `easy` (Einfach).

`<reward money="">` kann angegeben werden um anzuzeigen wieviel Credits der Spieler für die Mission bekommen würde. Das kann für die Kapitel auch jeweils einzeln angegeben, oder weggelassen werden. Anstatt **money** kann hier über „**other**“ auch ein Text stehen, was als Belohnung zu erwarten ist.

Über `<timer>` kann die gesamte Mission ein Zeitlimit bekommen, in diesem Fall hat der Spieler 5 Stunden um die Mission abzuschließen. Die Zeit kann auch über „**end**“ und „**start**“ angegeben werden.

Genau wie `<reward>` zählen die Angaben für die gesamte Mission, sollen die einzelnen Kapitel eine Belohnung oder ein Zeitlimit bekommen muss das im `<set_objective>` Node angegeben werden. Der `<objectives>` Node enthält in seinen Sub-nodes die einzelnen Kapitel der Mission:

```

<objectives>
  <objective step="1">
    <build>
      <ware exact="1" typename="SS_DOCK_TR_HQ"/>
    </build>
  </objective>
  <objective step="2">
    <buy>
      <ware min="10" typename="SS_MISSILE_TORP"/>
    </buy>
  </objective>
</objectives>
</create_briefing>

```

Mit „step=“ wird das Kapitel festgelegt, hiermit werden die Einträge später angesprochen um sie z.B. als beendet zu markieren. Die Sub-nodes von <objective> legen fest was im Briefing zu Lesen ist, dafür gibt es mehrere Möglichkeiten:

Node	Anmerkung
build	Baue die Angegebene Station
buy	Kaufe die angegebene Ware
deliver	Liefere die Angegebene Ware an die angegebene Station
destroy	Zerstöre das angegebene Objekt
dockat	Docke an dem angegebenen Objekt
find	Finde das angegebene Objekt
flyto	Fliege zum angegebenen Objekt, Sektor oder Position
follow	Verfolge das angegebene Schiff
kill	Töte den angegebenen Actor
protect	Beschütze das angegebene Objekt
sell	Verkaufe die angegebene Ware
talkto	Rede mit dem Angegebenen Actor
custom	Damit kann man sich einen Eintrag selbst zusammen stellen

Danach nur noch mit </create_briefing> den Node schliessen, und das Briefingfenster ist fertig. Zur Anzeige des Aktiven Kapitels und dem Gelben Marker, der anzeigt wohin der Spieler soll, benötigt man für das Aktuelle Kapitel noch einen <set_objective> Node:

```

<set_objective cue="createbriefing" active="1" title="Titel" noabort="1">
  <briefing cue="MiniBriefing" step="1"/>
  <mission level="easy"/>
  <reward money="5000"/>
  <build>
    <ware exact="5" typename="SS_DOCK_TR_HQ"/>
  </build>
</set_objective>

```

Cue ist der Cue in dem dieser `<set_objective>` Node plaziert ist, über diesen wird das Kapitel später wieder geschlossen.

Active markiert dieses Kapitel als Aktiv, sprich die Gelbe Markierung wird auf das Ziel dieses Kapitels zeigen.

Titel ist die Überschrift für dieses Kapitel, und **noabort**="1" heisst das das Kapitel nicht abgebrochen werden kann.

Der `<briefing>` Node legt fest welches Kapitel (**step**=) dies ist, und zu welchem Briefing (**cue**=) das Kapitel gehört.

Level="easy" zeigt an was für ein Schwierigkeitsgrad das Kapitel hat.

Mit **reward** wird die Belohnung angezeigt, hier kann auch ein `<timer>` gesetzt, oder beide weggelassen werden. Die Funktionsweise ist die selbe wie beim Briefing, genau wie beim folgenden `<build>` Node. Hier kann, oder sollte, der Node benutzt werden den man auch im Briefing angegeben hat.

Das erstellen des Briefings mitsamt seines ersten Kapitels ist somit fertig. Kapitel ist vielleicht auch das falsche Wort: Es wäre besser sich die Kapitel einfach als Punkte auf der Missionsliste vorzustellen, sie können nämlich direkt mehrere Punkte mit weiteren `<set_objective>` Nodes aktivieren um den Spieler wählen zu lassen was er, oder sie, als erstes machen möchte. Ansonsten aktivieren Sie die Kapitel nach Bedarf, das heißt sobald sie benötigt werden.

Hat man schließlich das Ziel eines Kapitels erreicht müssen diese als Beendet markiert werden:

```
<remove_objective cue="createbriefing" status="complete"/>
```

Cue ist der Cue in dem das Kapitel aktiviert wurde, der selbe name wurde auch beim `<set_objective>` Node angegeben.

Über **Status** wird angegeben warum das Kapitel beendet wird: **Complete** heisst das Ziel wurde erfüllt, es kann aber auch abgebrochen (**aborted**) werden oder der Spieler kann versagt (**failed**) haben.

Wenn alle Kapitel beendet sind muss schliesslich auch das Briefing entfernt werden:

```
<remove_briefing cue="createbriefing"/>
```

Hierzu gibt es nicht viel zu sagen, **cue** ist der Cue in dem das Briefing erstellt wurde. Wie bei den Kapiteln wurde der name auch im `<create_briefing>` Node angegeben. Leider ist das erstellen des Briefings mit seinen Kapiteln noch keine Mission, ob die Ziele erreicht wurden muss wie gewohnt über `<condition>` und `<action>` geprüft und bearbeitet werden.

Actor und create_offer

Möchten sie das ihre Missionen an den Stationen angeboten werden, oder Schiffe die der Spieler anfunken soll mit dem Plot Symbol (Blaues Buch) markiert werden, müssen sie `<create_offer>` benutzen. Dazu muss als erstes ein Actor erstellt werden:

```
<create_actor name="this.person" character="{8737,4}" face="203"
  race="boron" voice="211" invincible="1" object="MiniBriefing.station01"
  location="crew"/>
```

Den Actor speichern wir in `this.person`, darüber können wir den Actor später benutzen. **Character** ist sein Name, und **race** das Volk dem der Actor angehört.

Das bei Gesprächen angezeigte Video wird über `,face'` festgelegt, die Stimme über `,voice'`. Wenn man für `,face'` einen Eintrag im Popupmenü markiert wird eine kurze Beschreibung eingeblendet an der man sehen kann um wem es sich handelt.

`Invincible="1"` macht den Actor unsterblich.

Das Schiff oder die Station wo der Actor sich befindet wird über `,object'` angegeben, und `,location'` legt fest was der Actor an Bord macht. Zur auswahl steht **Pilot**, **Crew**, Passagier (**passenger**) oder Produkt (**product**).

Die Zeile die sein Leben schreibt, hiermit ist ein Borone zum Leben erweckt und kann auf der angegebenen Station bereits angesprochen werden. Zur Auswahl stehen bis jetzt aber nur ein paar Allgemeine Fragen, die bei allen Personen erscheinen. Um diesen Actor ein paar eigene Sätze mitzugeben, und um ein Symbol über der Station anzuzeigen, benötigen wir eine weitere Zeile:

```
<create_offer cue="Cue_offer" actor="this.person" conversation="MeineMission"
  discipline="XXXXP"/>
```

Wie beim Briefing ist `,cue'` der Cue in dem sich dieser `<create_offer>` Node befindet. Den `,actor'` haben wir gerade erstellt, und wird hier über seine `,name'` Eigenschaft angegeben. Die Sätze die der Borone sagen soll stehen in der `conversations.xml` und werden über **conversation** angegeben. Dazu kommen wir gleich noch. Das über der Station eingeblendete Symbol geben wir in `,discipline'` an, hier gibt es 5 Möglichkeiten:

Code	Symbol	Bemerkung
TXXX	C	Trade
XFXX	Fadenkreuz	Fight
XXBX	Bausteine	Build
XXXT	Glühbirne	Think
XXXXP	Buch	Plot

Als letztes muss das Gespräch mit diesem Actor in der `conversations.xml` eingetragen werden, diese Datei ist im `,t'` Ordner und muss vorher erst aus den `Cat/Dat` Dateien entpackt werden. Wie das funktioniert wird im [Anhang 3](#) erklärt.

```

<conversation name="MeineMission" description="Hier kommt der Text einer Mission">
  <player text="{8737,40}">
    <npc text="{8737,41}">
      <player text="{8737,42}">
        <npc text="{8737,43}">
          <player_all>
            <player text="{8737,44}." outcome="accept">
              <npc text="{8737,45}"/>
            </player>
            <player text="{8737,46}">
              <npc text="{8737,50}"/>
            </player>
          </player_all>
        </npc>
      </player>
    </npc>
  </player>
</conversation>

```

Der Aufbau ist ziemlich einfach, Der Spieler (**player**) und der Actor (**npc**) wechseln sich nacheinander, mit ihren Kommentaren, ab. **<player_all>** wird verwendet wenn der Spieler mehrere Antworten geben kann. **<player_any>** oder **<npc_any>** kann benutzt werden wenn aus mehreren Texten zufällig einer gewählt werden soll, im Prinzip genauso wie bei **<do_all/any>** oder **<check_all/any>**. Der erste Eintrag (**{8737,40}**) ist dabei bereits im Comm Menü zu sehen. Achtung: Durch einen Bug wird der NPC-Text nicht angezeigt wenn er direkt in der conversations.xml steht, eine Sprachdatei ist also unbedingt erforderlich. Über **,outcome‘** kann man im MD script die gegebene Antwort abfragen:

```

<condition>
  <conversation_completed actor="Cue_offer.person" answer="accept"
    conversation="MeineMission"/>
</condition>

```

Wie Sie sehen können wird der **,actor‘** und die verwendete **,conversation‘** angegeben, **,answer‘** ist dabei die Antwort die mit **,outcome‘** übergeben wurde.

Wird diese Antwort nicht gegeben, weil der Spieler z.B. das Angebot ablehnt, kann das Gespräch immer wieder geführt werden, bis der Cue mit der obigen **<condition>** ausgeführt wird und das (Gesprächs-) Angebot entfernt werden kann. Dazu benutzen wir:

```

<remove_offer actor="Cue_offer.person" conversation="MeineMission"
  discipline="XXXXP"/>

```

Die benutzten Angaben sollten ihnen bekannt sein: der **actor**, die **conversation** und die Art der Mission. Wenn der Actor nicht mehr gebraucht wird sollten Sie ihn ebenfalls entfernen, ansonsten bleibt er für den rest des Spiels auf der Station:

```

<destroy_actor actor="Cue_offer.person"/>

```

Mit dem **Actor** als einzige Angabe ist das schnell erledigt, R.I.P. Mr Tentacle.

Eine nützliche Variable

Die Variable `{group.object.select@group}` ist sehr praktisch um den Spieler schnell ein Objekt aus einer Liste von mehreren Objekten wählen zu lassen. Gehen wir davon aus das sich der Spieler aus mehreren Stationen im Sektor eine aussuchen soll, um dort z.B. anzudocken. Dazu müssen wir die Stationen zuerst in einer Gruppe zusammenfassen:

```
<action>
  <do_all>
    <find_station class="station" race="{player.sector.race.name}" max="5"
                                     group="FindeStation" multiple="1">
      <sector sector="{player.sector}" />
    </find_station>
  </do_all>
</action>
```

Im `<find_station>` Sub-node suchen wir 5 Stationen die zum Volk gehören dem auch der Sektor gehört in dem der Spieler gerade ist. `,group='` ist der Name der Gruppe, `multiple="1"` legt fest das wir mehrere Station finden wollen und `,max='` ist die Anzahl an Stationen die wir finden wollen.

`<ask_question>` mit `{group.object.select@FindeStation}` im Text sorgt dafür das sich der Spieler eine der Stationen aussuchen kann.

```
<action>
  <do_all>
1    <do_if min='1' value='{group.object.count@Foundstations}'>
2    <ask_question name="Frage01" ...
      ...author="{random.pilot@{player.sector.race.name}}" text="Such dir eine der...
      ...Stationen aus\n\n[center]{group.object.select@FindeStation}[/center]" />
    </do_if>
  </do_all>
</action>
```

1: Die Frage soll nur gestellt werden wenn mindestens eine Station gefunden wurde.

2: Wenn ja, wird die Frage gestellt. Die Frage stellt ein zufälliger (`author`) Bewohner aus dem Volk dem der Sektor gehört. `{group.object.select@FindeStation}` fügt dem Text soviele Schaltflächen hinzu wie Stationen in der Gruppe sind.

```
<condition>
  <question_answered question="Frage01" />
</condition>
```

Normalerweise benötigen wir für die Antworten von `<ask_question>` einzelne Sub-cues, hier prüfen wir nur ob die Frage überhaupt beantwortet wurde. Um die gewählte Station benutzen zu können, müssen wir die Antwort erst zu einem Objekt umwandeln:

```

<action>
  <do_all>
    <set_object name="this.gewählt" value="{question.answer@Frage01}" />
    <set_target object="this.gewählt" />
  </do_all>
</action>

```

Über `<set_object>` speichern wir die gewählte Station als Objekt in der Variable `this.gewählt`, mit dieser können wir die Station jetzt ganz normal benutzen. Beachten Sie den Namen der Frage hinter dem `@` Zeichen. In diesem Beispiel wird mit `<set_target>` die Station anvisiert.

Benutzen von Soundeffekten

Wie Sie in der Beispiel Mission gesehen haben, kann man alle möglichen Soundeffekte benutzen. So haben wir z.B. mit `<play_sound soundid="924" />` einen Alarmton ausgegeben. Wenn sie im XML-Editor die `soundid` Eigenschaft benutzen sehen sie in der Populiste eine Endlose Reihe an Sound IDs, markieren sie dort einen Eintrag um eine Beschreibung zu erhalten (Bild 13).

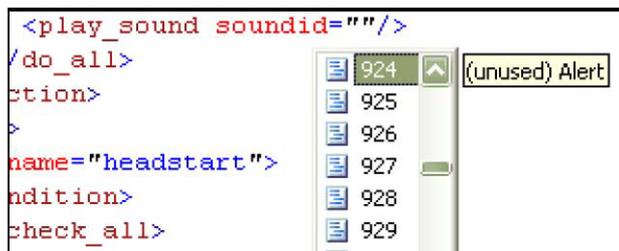


Bild 13 - Sound IDs

Die Liste der Sound IDs und ihre Beschreibung kann auch in der `director.html` gefunden werden.

Testen der Mission

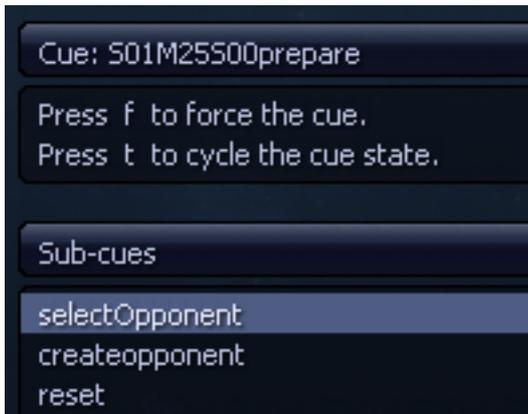
Es ist unumgänglich das Sie ihre Mission, im Laufe der Entwicklung, immer wieder testen müssen. Dadurch kann man am besten sehen wie `<condition>` und `<action>` die Mission beeinflussen, passen sie dann die Mission an oder Korrigieren Sie sie. Um bei diesem Prozess nicht immer das Spiel beenden zu müssen, gibt es das...

Mission Director Menu



Das MD Menü erreicht man über die Optionen, wählen sie dort ‚Spiel‘ aus. Der Eintrag erscheint nur wenn der Spielname vorher in „Thereshallbewings“ geändert wurde (beachten Sie das nur T groß geschrieben ist).

In diesem Menü haben Sie jetzt mehrere Möglichkeiten:



Die Hotkeys müssen zuerst in den Steuerungsoptionen eingestellt werden, die Einträge befinden sich im Script-Editor Tab.

Markieren Sie zuerst den cue den sie testen möchten, bevor sie den Hotkey drücken.

1: Force cue:

Erzwingt das Ausführen des markierten Cues, <condition> werden Ignoriert. Sinnvoll um Änderungen in <timing>,<action> Nodes oder Sub-cues zu testen

2: Cycle cue state:

Wechselt den aktuellen Status des Cues: Beendet (complete), Abgebrochen (canceled) und Inaktiv (inactive), letzterer wartet das die Bedingungen von <condition> erfüllt werden. Benutzen sie diesen Hotkey um die <condition> eines Cues zu testen.



Sollten sie mit alt-tab zum Editor wechseln um Änderungen vorzunehmen, müssen die Dateien im Spiel erst neu eingelesen werden. Dafür gibt es 3 Funktionen im MD:

1: Refresh cues:

Hiermit werden die Dateien im ‚director‘ Ordner neu eingelesen, die Cues behalten dabei ihren Status. Möchten sie ihre geänderte Mission erneut Testen verwenden sie zusätzlich den Hotkey ‚force cue‘

2: Diese Funktion liest die conversation.xml, zum testen der Änderungen benutzen Sie ‚force cue‘ bei dem Cue indem diese verwendet wird.

3: Reset Mission Director:

Mit dieser Funktion starten Sie den MD neu, dabei werden sämtliche Dateien neu eingelesen und alle Missionen werden auf ‚inactive‘ gesetzt. Diese Funktion ist nicht 100% sicher, so fehlen nach einem reset bei mir die Gespräche aus der conversations.xml, und manchmal werden <condition> nicht mehr erfüllt. Einmal ins Hauptmenü zurück um ein neues Spiel anzufangen behebt diese Fehler, es muss also nicht immer der Code schuld sein wenn etwas nicht läuft. ACHTUNG: Verwenden sie das nicht bei einem ‚guten‘ Spielstand, Sie müssten alle Plots von vorne beginnen.

Denken sie daran das bei Verwendung dieser Funktionen die Missionen durcheinander geraten! Ich empfehle einen neuen Spielstand zu verwenden (Selbsterstelltes Spiel) und während des Testens nicht zu speichern. Benutzen sie ein Cheat-Script um die Dinge zu erstellen, die zum auslösen der `<condition>` benötigt werden. Cheaten zerstört zwar den Spielspaß, aber beim testen spart man eine Menge Zeit.

Isoliertes Testen

Wenn sie in `<condition>` mehrere Bedingungen stellen und sich nicht sicher sind warum die `<action>` nicht ausgeführt wird, ist es nützlich einzelne Zeilen zu testen. Das kann man auf zwei Arten machen: Entweder indem man Zeile für Zeile (mit `<!--` und `-->`) Auskommentiert bis man die Problemzeile gefunden hat, oder indem man den fraglichen Cue in eine neue Datei kopiert und zum testen Zeile für Zeile löscht. Achten Sie darauf den Cuenamen zu ändern damit die beiden Dateien sich nicht in die quere kommen.

Verwenden von Debugnachrichten

In einer Mission kann es viele Cues geben die man benutzt um die Mission weiterzuführen, ohne das In-Game zu sehen ist was genau passiert. Um da nicht den Überblick zu verlieren kann man sich selbst Nachrichten schicken, die einen kurzen Text anzeigen wo die Mission gerade ist oder welchen Inhalt eine Variable gerade hat. Zum Beispiel:

```
<show_help force="1" text="Sektor gewechselt - weiter mit sub-cue"/>
```

Das zeigt nur ein paar Stichworte als Hilfe oben rechts

```
<play_subtitles author="debug" text="{sector.name@curname.GesuchterSektor} erreicht"/>
```

Hier wird als Untertitel der erreichte Sektor angezeigt.

director.dmp

Dies scheint zur Zeit nicht zu Funktionieren, zumindest habe ich es nicht hinbekommen. Sollte sich das ändern, werde ich es hier einfügen.

Fragen und Antworten

- F: Ich möchte das meine Mission das Spiel als Modified markiert, damit Spieler kein „Thereshallbewings“ eingeben müssen, geht das?
- A Erstellen sie eine leere Textdatei mit dem Namen „modified.txt“, diese gehört mit in den director Ordner. Das Mission Director Menü wird damit allerdings nicht Sichtbar.
- F: Hat noch jemand eine Frage?

Anhang

Im Anhang finden Sie Themen die zu Lang, und auch zu Umfangreich sind, um sie im Hauptteil der Anleitung zu erklären.

Anhang 1 - Instanzen

Die Instanziierung ändert das Verhalten des Cues nachdem die `<condition>` erfüllt werden. Wenn KEINE Instanz erzeugt wird, wird der Cue wenn er abgearbeitet ist als Beendet (complete) markiert und wird in dem Spielstand auch nicht mehr starten. Wenn allerdings eine Instanz erzeugt wird, wird eine Kopie des Cues (und seiner Sub-cues) erzeugt. Nach beenden des Cues wird dann die Kopie als Beendet Markiert, aber das Original läuft weiterhin, und es wird, sobald die `<condition>` erneut erfüllt werden, eine neue Kopie erstellt.

Instanzen sollten Sie nur mit `<condition>` benutzen die einmalig (oder zumindest sehr selten) erfüllt werden oder ein Ereignis beinhalten. Instanzen sollten NICHT benutzt werden, wenn einfache `<condition>`, wie die Abfrage der Spielzeit, benutzt werden, es wird dann alle paar Millisekunden eine neue Instanz erzeugt und das Spiel würde kurz darauf abstürzen. Häufig werden Instanzen in Zusammenhang mit Ereignissen benutzt, z.B. einem Sektorwechsel wie bei den Zufallsmissionen die der Spieler an den Stationen annehmen kann.

Eine Instanz ist eine Kopie des Cues und aller Sub-cues, dabei muss man beachten das diese ‚Instanz-Sicher‘ sind. Instanz-Sicher bedeutet das man Vorsichtig sein muss wie man Variablen, Objekte, Cuenamen, u.s.w. verwendet, und das man darauf achtet das vom Original keine neue Kopie erstellt wird solange die Instanz noch läuft. Ausserdem wird die Instanz nicht einfach als Beendet (complete) markiert wenn der Cue mit allen seinen Sub-cues beendet ist, die Kopie hört förmlich auf zu existieren, kann also nicht mit `<cue_complete>` abgefragt werden.

Es ist auch möglich den Sub-cue einer Instanz als Instanz zu starten, das klingt erstmal Kompliziert, bietet aber viele Möglichkeiten. Z.B. können sie einen Cue als Instanz starten der, im Laufe der Mission, nach jedem Sektorwechsel eine bestimmte Aktion ausführt, auch wenn die Mission bereits eine Kopie ist.

Anhang 2 - Library Cues

Es ist möglich wiederverwendbare Library Cues zu benutzen. Zum Beispiel: Sie erstellen an mehreren Stellen in ihrer Mission ein M5. Dann brauchen sie den Code nicht jedesmal Kopieren und Einfügen, stattdessen schreiben Sie einen Cue in dem ein M5 erstellt wird und machen diesen zu einer Library. Dadurch erspart man sich nicht nur das Einfügen des immer gleichen Codes, man erleichtert sich auch die Fehlersuche oder das nachträgliche Ändern des Schiffes. Eine Library wird nicht wie normale Cues ausgeführt, sondern nur wenn sie über `,ref=` aufgerufen wird.

```
<cue name="MeineLibrary" library="1">
  ...
</cue>

<cue name="MeinErsterCue">
  ...
  <cues>
    <cue ref="MeineLibrary" />
  </cues>
</cue>

<cue name="MeinZweiterCue">
  ...
  <cues>
    <cue ref="MeineLibrary" />
  </cues>
</cue>
```

Im oberen Beispiel wird der Cue „MeineLibrary“ vom Parser ignoriert weil er als Library markiert ist, sein Inhalt wird aber in beiden Sub-cues ausgeführt wo die Library mit `ref="MeineLibrary"` aufgerufen wird.

Manchmal ist es nützlich die `<condition>` in der Library vorzugeben, bei jedem Aufruf müssen diese dann erfüllt werden. Möchten sie für eine bestimmte Situation lieber spezielle `<condition>` verwenden, können sie die Library trotzdem nutzen in dem sie die Schlüsselnodes des Librarycues direkt aufrufen. Zum Beispiel:

```
<cue name="MeinErsterCue">
  <condition>
  ...
</condition>
  <action ref="MeineLibrary" />
</cue>

<cue name="MeinZweiterCue">
  <condition>
  ...
</condition>
  <action ref="MeineLibrary" />
</cue>
```

Librarys sind sehr nützlich, aber was wenn sie Cues möchten die nicht ganz die selben sind, sondern nur fast? Wenn zum Beispiel das erstellte M5 je nach Missionsziel einem anderen Volk angehören soll. Anstatt das Volk nach dem aufrufen der Library jedesmal einzeln zu ändern, können Sie einen Parameter benutzen der an die Library übergeben wird:

```
<cue name="MeineLibrary" library="1">
  <action>
    ...
    <create_ship ...=" race="{param@ZielVolk}" ...=" >
    ...
  </create_ship>
  ...
</action>
</cue>

<cue name="MeinErsterCue">
  ...
  <cues>
    <cue ref="MeineLibrary">
      <params>
        <param name="ZielVolk" value="{value@MeinErsterCue.MeinVolk}" />
      </params>
    </cue>
  </cues>
</cue>

<cue name="MeinZweiterCue">
  <condition>
    ...
  </condition>
  <action ref="MeineLibrary">
    <params>
      <param name="ZielVolk" value="{lookup.race@boron}" />
    </params>
  </action>
</cue>
```

Beachten Sie wie „MeinErsterCue“ die Library im ganzen als Sub-cue benutzt, während „MeinZweiterCue“ nur den `<action>` Node der Library benutzt.

Der Parameter „ZielVolk“ wird in beiden Varianten übernommen und in der Library verarbeitet. Möchten sie Parameter auf eine Variante begrenzen, können Sie das durch `{param.cue@ZielVolk}` und `{param.action@ZielVolk}` erreichen. In dem Fall wird der Parameter nur übernommen wenn der Aufruf auf die angebene Weise erfolgt ist.

Eine Library kann sovielen Parameter verwenden wie Sie möchten. Die übergebenen Werte können direkt (z.B. „1“ oder „m5“) oder als Variablen angegeben werden.

Die Library kann ein Top-level Cue der Datei sein, oder auch ein Sub-cue. Bei letzterem ist die Library allerdings nur für Cues (und deren Sub-cues) sichtbar die auf selber Ebene liegen, aber nicht für Cues die auf höherer Ebene sind. Soll ein Top-level Cue auf eine Library verweisen muss die Library Global sichtbar sein, z.B. wenn die Library selbst ein Top-level Cue ist.

Um einen Cue als Library benutzen zu können, braucht diese nicht unbedingt als Library angegeben zu sein. Sie können einen normalen Cue genauso über „ref=“ ausleihen, vorausgesetzt der Cue ist im sichtbaren Bereich (die selbe Ebene oder in einem übergeordneten Cue).

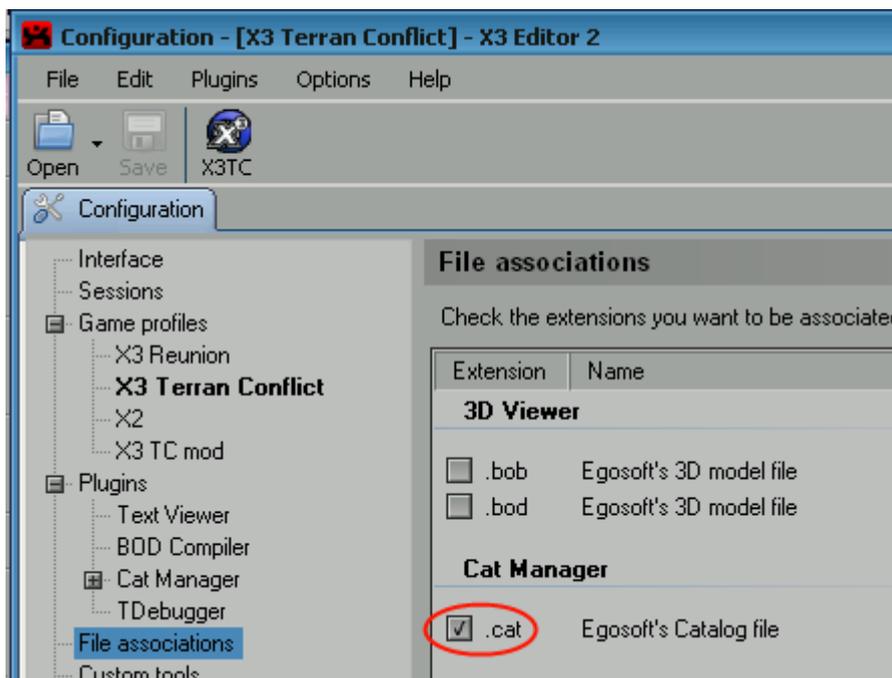
Anhang 3 - Dateien aus .cat/.dat entpacken

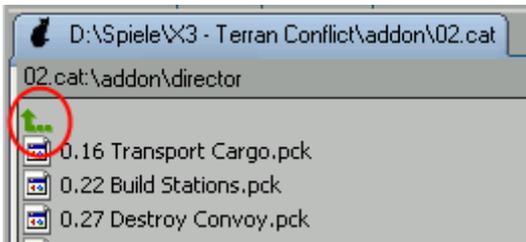
Zum erstellen neuer Missionen oder auch dem hinzufügen neuer **converstions** werden Dateien benötigt die in Cat/Dat Dateien gepackt sind. Zum entpacken wird der X3 Editor 2 von doubleshadow benötigt.



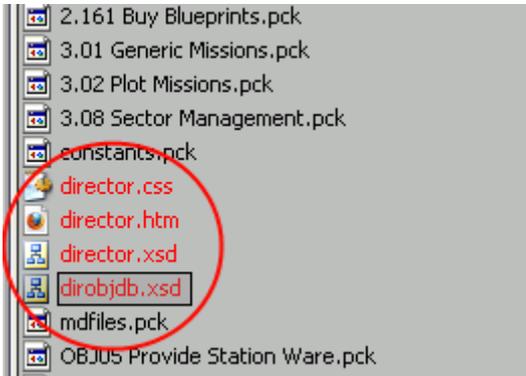
Wenn die .cat Dateien als Katzen zu sehen sind, sollte ein Doppelklick den X3 Editor 2 starten.

Sollte sich der X3 Editor 2 nicht durch einen Doppelklick auf die Cat Datei öffnen, können sie das unter „Options/Configuration“ einstellen. Setzen dafür den im unteren Bild markierten Haken. Sollten sie die Cat Dateien für ein anderes Programm benötigen können sie die Cat auch über „Open“ öffnen.



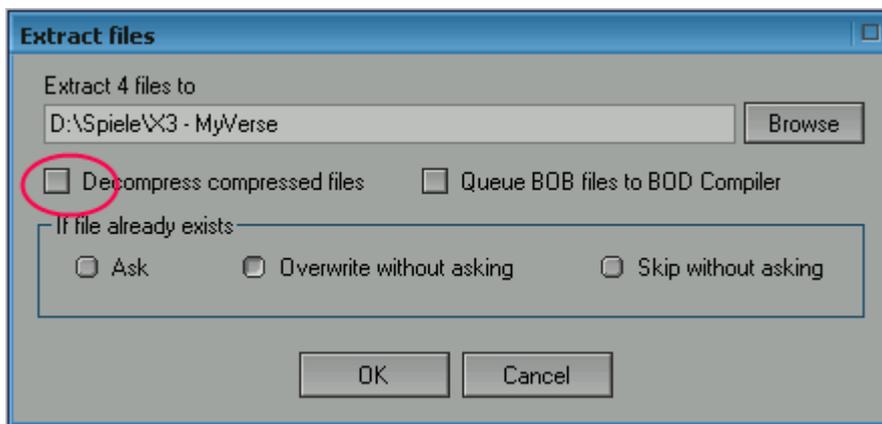


Wenn die Cat Datei geöffnet ist, lässt sie sich im X3 Editor 2 wie mit einem normalen Dateibrowser durchsuchen. Über den Pfeil wechselt man zum Übergeordneten Ordner.



Benutzen sie die STRG (CTRL) Taste um mehrere Dateien zu Markieren. Diese werden dann Rot eingefärbt.

Haben sie die gesuchten Dateien gefunden und markiert, drücken sie „Extract files“ in der Toolbar um die Dateien zu extrahieren. Über Browse können sie den Ordner wählen in dem die Dateien gespeichert werden sollen. Beachten sie den Haken bei „Decompress compressed files“ (Siehe unteres Bild), für die 6 Dateien die für's schreiben von Mission gebraucht werden darf dieser NICHT gesetzt sein. Wenn sie z.B. die conversations.xml aus dem „t“ Ordner entpacken wollen, muss der Haken gesetzt werden. Diese Datei ist zusätzlich noch Komprimiert, gut zu erkennen an der Dateiendung (.pck).



Nachdem sie auf OK gedrückt haben werden die Dateien entpackt.

Beachten sie das Dateien eventuell in mehreren Cat/Dat vorhanden sind, die jeweils höhere Cat/Dat ist in dem Fall die neuste Datei. Suchen sie bestimmte Dateien, fangen sie in den höheren Cat/Dat an zu suchen. Wollen sie dagegen z.B. alle Dateien aus dem director Ordner entpacken, fangen sie bei 01.cat an und arbeiten sie sich zur höchsten Nummer durch. Ältere Dateien werden so überschrieben. (Nur wenn im „Extract files“ Dialog „Ask“ oder „Overwrite without asking“ gesetzt ist, bei „skip without asking“ werden alte Dateien NICHT überschrieben.)

Damit Sie nicht lange suchen müssen, hier noch eine Tabelle in welcher Cat die wichtigsten Dateien zu finden sind:

Datei	Ordner	TC (v3.2)	AP (v2.5.3)
director.css	director	04.cat	02.cat
director.htm	director	04.cat	02.cat
director.xsd	director	13.cat	02.cat
director.xsl	director	10.cat	01.cat
diobjdb.xsd	director	13.cat	02.cat
dirschema.xsl	director	10.cat	01.cat
conversations.xml	t	12.cat	03.cat

Anhang 4 - Links

[Egosoft](#) - Die Heimat des X-Universums.

[Terran Conflict / Albion Prelude - Scripts und Modding](#) - Das Forum rund ums Modding für TC und AP. Hier finden sie Scripts, Mods oder Präsentieren sie hier ihre eigene Mission.

[Allgemeine MD-Fragen](#) - Hier können sie allgemeine Fragen zum Mission Director stellen.

[Kompatibilitätsliste](#) - Hier können sie sehen welche TextID Seiten bereits von anderen Scripts der Community verwendet werden. Am Anfang des Threads ist auch erklärt wie sie ihre Mission hier eintragen lassen können.

[Modding Tools von doubleshadow](#) - Hier finden sie den X2 Editor 2 und weitere X-Tools.

[Visual Web Developer](#) - Der empfohlene XML-Editor.

[Mission Director Basics](#) - Videotutorials für Anfänger von Ketraar (Englisch)

Nachwort

Seid X-BtF bin ich jetzt schon im X-Universum unterwegs, darum das erste Wort auch an die Jungs und Mädels von Egosoft: DANKE.

Dank auch an doubleshadow, seine X_Tools haben einige Dinge erst Möglich gemacht, und vieles vereinfacht.

Speziell zu diessem Guide auch Danke an Mark Wilson, der den Englischen Original Guide geschrieben hat. Und auch an Bernd, der mir mitten in der Nacht, auf'm Sonntag, noch die Erlaubnis gemailt hat den XMDGuide zu Übersetzen und zu erweitern. Müsst ihr wenig Freizeit haben, hoffen wir das es nur am kommenden X:Rebirth liegt. :)

Natürlich auch Danke an die Scripter und Modder mit ihren Ideen und Bemühungen X noch Interessanter zu gestalten.

Zu guter letzt Danke an die gesamte Community, die mit ihren Kommentaren für Leben auf dem Forum sorgen.

Ich hoffe der Guide ist euch so Hilfreich wie er es für mich war, und viel Spaß beim schreiben neuer Missionen.

Gruß
tero