ARMA
ARMED ASSAULT

EDITING GUIDE

SE-Version 1.02

EDITING GUIDE

by Mr-Murray

Vorwort

Der Inhalt dieser Editieranleitung wird dir das Leben im Armed Assault Editor wesentlich angenehmer gestalten. Du hast hiermit die Möglichkeit auch ohne Programmierkenntnisse schnell und einfach anspruchsvolle Missionen für Armed Assault zu erstellen.

Ganz richtig, ohne Programmierkenntnisse! Gewisse Vorkenntnisse aus dem Vorgänger Operation Flashpoint wären natürlich von Vorteil, sind aber nicht zwingend notwendig. In dieser Anleitung werden dir die Teilbereiche des Editors erläutert und anhand von Beispielen näher erklärt. Dazu werden dir die nahezu unbegrenzten Möglichkeiten aufgezeigt, die du in diesem Spiel und dem Editor hast.

Du wirst mit ein bisschen Geschick, Ideenreichtum und Kreativität deine eigenen Szenarien umsetzen und dank der Kameramöglichkeiten und dem Einbinden eigener Sounddateien Missionen erstellen, die schon fast mit einem Hollywoodfilm zu vergleichen sind und den Spieler in seinen Bann ziehen werden.

Dynamische Missionen erstellen, bei denen bei jedem Start das Wetter oder die Uhrzeit anders ist, dass Erfüllen von Missionszielen bei jedem Spielablauf verschieden abläuft und dass bestimmte Einheiten oder der Spieler bei jedem Neustart an einer anderen Position der Insel startet, sollte hiermit für dich kein Problem mehr sein.

Jetzt liegt es nur noch an deinem Ideenreichtum, deiner Kreativität und natürlich dir, gute Missionen erstellen zu können. Ein Drehbuch, Szenario oder eine Story für deine Mission ist hier nicht enthalten, dass musst du dir schon selbst ausdenken. Ansonsten hast du mit dem Editor und dieser Anleitung alles, was du brauchst um deine Ideen umzusetzen. Und wenn mal was nicht klappt, Editor aus, Spiel an und einfach mal entspannt in den Krieg ziehen.

Dieses Spiel baut als Operation-Flashpoint-Nachfolger mit eigens entwickelter Programmiersprache auf seinen Vorgänger auf. Es sind zwar hier und da Änderungen erfolgt und viele Neuheiten eingeflossen, aber vom Grundkonzept ist alles nahezu gleich geblieben. Der Editor ist, wie auch bei seinem Vorgänger, mit gleicher übersichtlicher und benutzerfreundlicher Oberfläche zu bewundern. Und auch die Missionsordner und der Inhalt dieser, sind vom Grundsatz her gleich geblieben. Doch lies, probier und editier dich selbst mit Hilfe dieser Anleitung durch die Welt von Armed-Assault.

Viel Erfolg und Spaß mit dem Editor wünschen BI, Morphicon und Mr-Murray

Anmerkungen

Diese Anleitung ist als Einführung in den Armed Assault Editor gedacht und soll gerade dem Editieranfänger den Umgang mit dem Editor erleichtern. Hier aufgeführte Skripte oder Abläufe sind frei erfunden und können selbstverständlich weiterentwickelt und verbessert werden.

Natürlich gibt es noch weit mehr Möglichkeiten im Editierbereich von Armed Assault als hier im Buch erläutert, wobei ich hier auf die offizielle Wiki

http://community.bistudio.com/wiki

verweisen möchte, die immer auf dem aktuellen Stand ist, was Editing, Scripting und sonstiges rund um Armed Assault betrifft. Hier wurde lediglich das grundlegendste mit eingebunden und ein paar Blicke über den Tellerrand gegeben, die dich anregen und auf eigene Ideen bringen sollen. Sämtliche Skripte aus Kapitel 6 stehen im offiziellen Forum für dich (www.forum.german-gamers-club.eu) zum Download bereit.

Danksagungen

Ich möchte mich mit diesem Editing Guide bei allen Operation Flashpoint Fans, die dem Spiel bis heute treu geblieben sind und natürlich bei Bohemia Interactive, ohne die es ein Spiel dieses Umfangs und somit diese Anleitung gar nicht geben würde, bedanken.

Mein nächster Dank richtet sich an das Morphicon-Team. Hier besonders an Morris Hebecker und Alexander Schmidt für die tatkräftige Unterstützung beim Vertrieb dieses Buches.

Ein weiterer Dank geht an das gesamte Mapfact-Team, BadAss, Chneemann, Flashpoint_K, JörgF., Kriegerdaemon, LockheedMartin$ch, MCPXXL, OneManGang, Silola, Sniping-Jack, Raedor, Lester und Wüstenfuchs und unseren helfenden Händen MemphisBelle, Simba, Marco-Polo-IV, Parvus, Sgt.Ace und SNKMan, welche mich in den letzten Jahren tatkräftig unterstützt haben.

Wobei hier ein besonderer Dank Raedor, Chneemann und den offiziellen BIS-Betatestern gilt, die mir bei diversen Problemen mit Armed Assault tatkräftig zur Seite standen. Ein weiterer ganz besonderer Dank geht an MemphisBelle und Dan für die hervorragende englische Übersetzung, Andre Scheufeld für die Vermittlung des Kontaktes zu Morphicon und Andreas Holzwart, Rastavovich sowie Wolle für den hervorragenden Support.

Des Weiteren bedanke ich mich noch bei allen Freunden und Bekannten und besonders bei meiner Familie und meiner Freundin, die mich in der Zeit der Realisierung dieses Werkes tatkräftig unterstützt und motiviert haben.

Euer Sascha “Mr-Murray” Hoffmann

Community Screenshot Contest

Im Rahmen der Neuauflage dieses Buches entschied ich kurzerhand die Community auf irgend eine besondere Art mit ins Buch zu integrieren. Da es leider nicht viele Möglichkeiten dafür gibt, entschied ich mich, nach Rücksprache mit Morphicon dazu, dass dies am besten mit einem Bilderwettbewerb erfolgen könnte.

Ich bedanke mich hiermit nochmal bei allen Teilnehmern, die am Wettbewerb teilgenommen haben und auch bei Armed-Assault.de für die Realisierung und Umsetzung des Wettbewerbs auf selbiger Seite.

Im gesamten Buchverlauf sind die Ergebnisse des Wettberwerbs zumeist auf den Kapitelverzeichnissen, aber auch in den Kapitel, zu bewundern.

Die drei nun folgenden Bilder stellen die Plätze 1 bis 3 dar, welche von der Community als beste Bilder gevotet wurden. Die Gewinner sind keine anderen als:

Platz 1: Marcus-Ergalla (Aljosha Rall)

Platz 2: Mr. Burns (Andreas Schmitz)

Platz 3: Stoned Boy (Frank Nobis)

Platz 1: Marcus-Ergalla

Platz 2: Mr Burns

Platz 3: Stoned Boy

Inhaltsverzeichnis

Kapitel 1: Der Einstieg

Kapitel 2: Die Dateien

Kapitel 3: Die Waffen – Fahrzeuge – Einheiten – Objekte

Kapitel 4: Die Mission

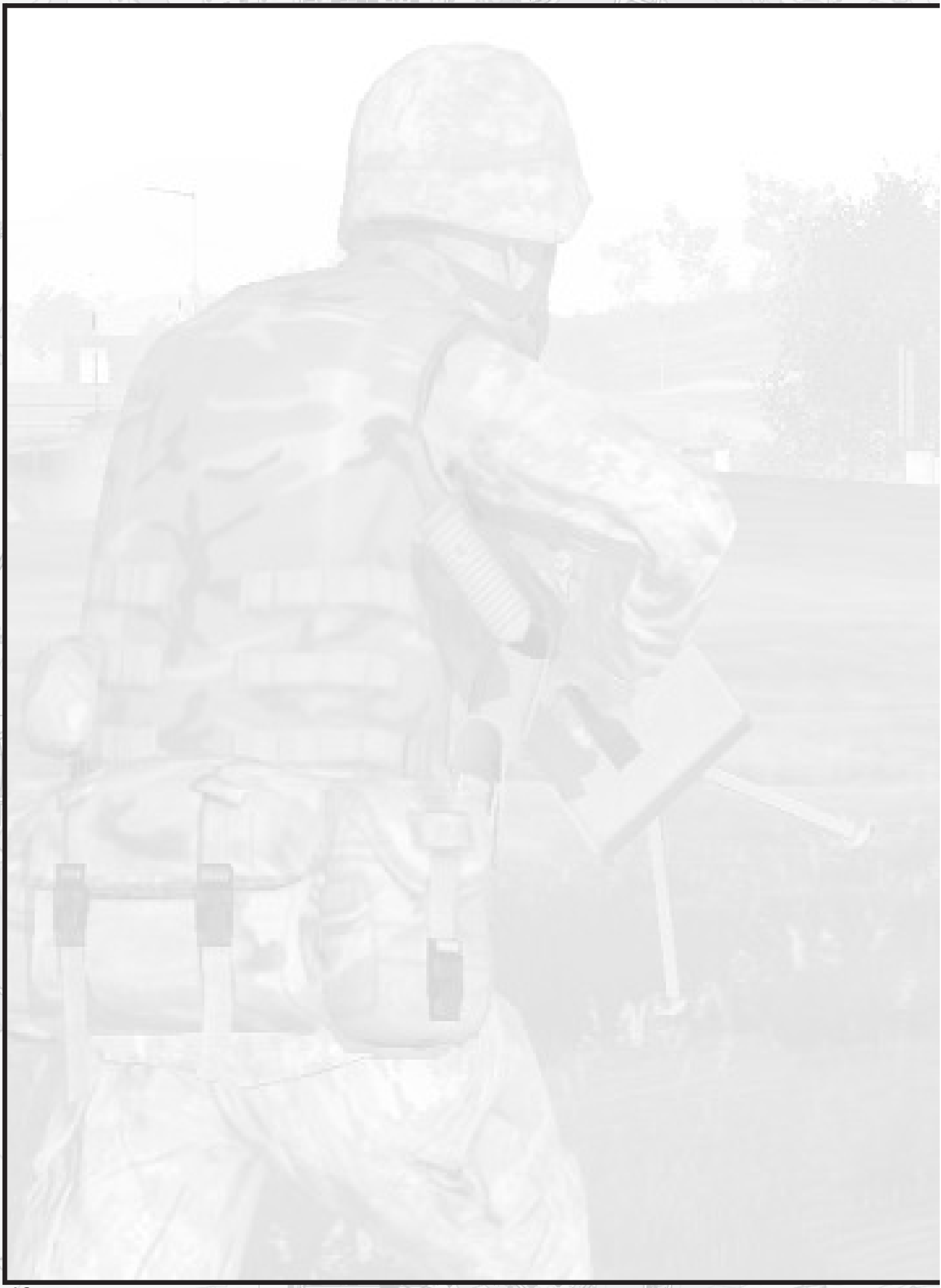
Kapitel 5: Das Missionszubehör

Kapitel 6: Die Missions Specials

Kapitel 7: Multiplayer

Kapitel 8: Das Camscripting

Kapitel 9: Scripting



Kapitel 1
- Der Einstieg -

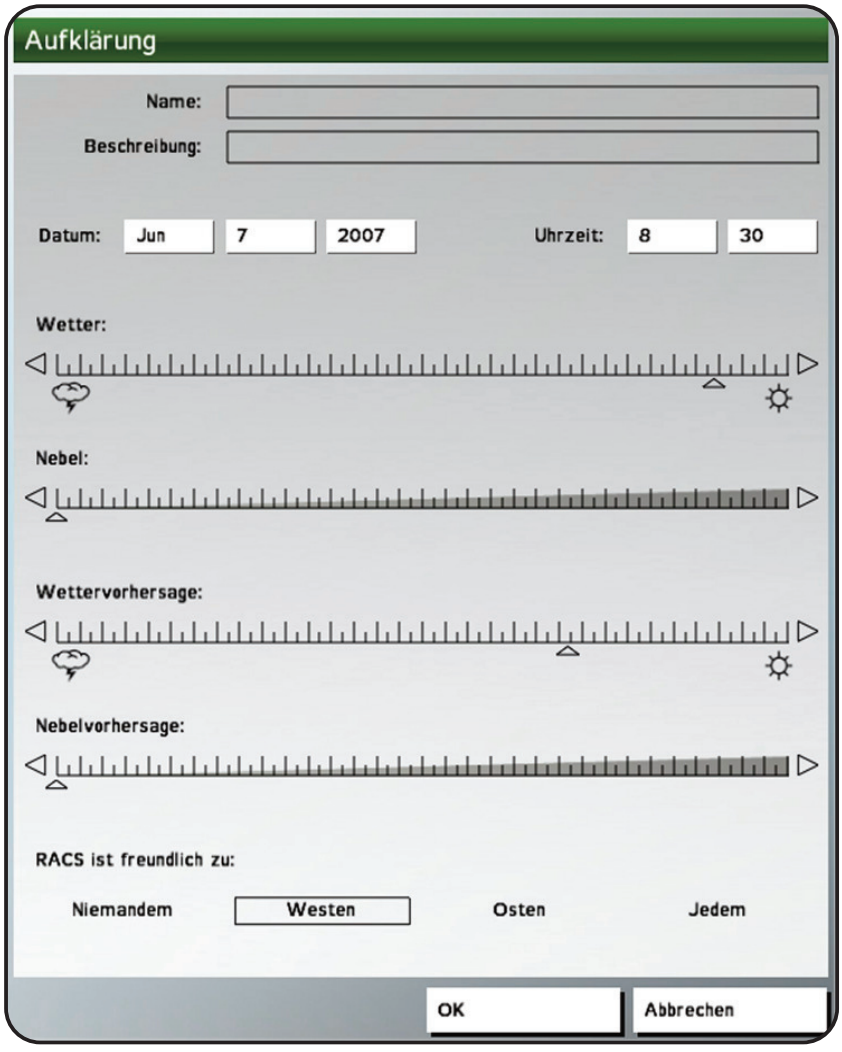
Dieses Kapitel soll dir zunächst mehr Übersicht und Durchblick über die Oberfläche des Editors verschaffen und dich auf die weiteren Kapitel vorbereiten. Mit Hilfe dieses Kapitels wirst du einen sicheren Umgang mit der Oberfläche des Editors erlangen und erste gute Ergebnisse erzielen. Es erläutert dir zunächst die Hauptfunktionen der einzelnen Bereiche.

Arctic (Ronny Krischker)

1.1 - Die Oberfläche

Die Oberfläche des Editors ist, wie man sieht, recht überschaubar und sehr benutzerfreundlich aufgebaut. In Zusammenarbeit mit der Maus, den Pfeil- und F-Tasten hat man die Möglichkeit relativ schnell die einzelnen Bereiche und Unterbereiche anzuwählen um damit zu arbeiten.

Im Bereich Info definiert man Dinge wie Wetter, Datum, Uhrzeit, Jahreszeit und auf welcher Seite die Widerstandseinheiten kämpfen. Des Weiteren gibt man hier den Namen der Mission und darunter eine kurze Beschreibung der Mission mit an. Hier ist es möglich ein Startwetter festzulegen und anzugeben, wie es sich im späteren Missionsverlauf ändern soll. Der Nebel lässt sich hierbei unabhängig von dem Wetter einstellen. Das Verstellen der Jahreszeiten hat unter anderen auch Änderungen auf den Wasserstand und die Helligkeit zu den verschiedenen Uhrzeiten zur Folge. Wie auch im echten Leben, sind die Tage im Sommer länger als im Winter.



Nachrichtendienst

Name der Mission
Beschreibung der Mission

Datum und Uhrzeit

Wettervorhersage
Gegenwärtiges Wetter

Gegenwärtiger Nebel

Späteres Wetter

Späterer Nebel

Seite der Racs(Widerstand)

F-Tasten

Mit Hilfe der F-Tasten F1 bis F6 wählt man die Untermenüs an. Dieser Abschnitt soll zunächst eine grobe Erläuterung der F-Tasten darstellen, welche im Laufe des Kapitels noch einzeln und näher erklärt werden.

F1 ermöglicht es Einheiten, Fahrzeuge und Objekte auf die Karte zu setzen und individuell einzustellen.

F2 hat gleich zwei Funktionen. Zum einen kann man damit gleich ganze Gruppen auf die Karte setzen und zum anderen dient F2 als Verbindungstool, mit welchem man Einheiten und Auslöser verbinden kann.

F3 Mit F3 setzt man Auslöser, welche man sehr flexibel und für viele Aktionen verwenden kann. Als Beispiel soll hier das Funkmenü dienen, welches unter anderem ja auch in den Auslösern definiert wird.

F4 Mit der F4-Taste weist man Einheiten oder auch einer Logik Wegpunkte zu, welche sie dann nacheinander abarbeiten und je nach Definition an gewissen Stellen Aktionen durchführen oder auslösen

F5 ist eine Funktion, welche gerne übersehen wird, obwohl sie sehr nützlich ist. Sie ermöglicht es Wegpunkte und Auslöser auf einander abzustimmen. Demnach würde der nächste Wegpunkt erst angelaufen werden, wenn der damit verbundene Wegpunkt oder Auslöser bereits ausgelöst wurde.

F6 Die F6-Taste führt ins Untermenü der Marker, mit welchem die Karte sehr taktisch gestaltet werden kann und dem Spieler ein wenig mehr Übersicht über die Mission gibt.

Einsatz

Im oberen Abschnitt wählt man zunächst den Arbeitsbereich aus. Hier hat man die Auswahl zwischen Einleitung, Einsatz und Abspann gewonnen und Abspann verloren. Es stehen also vier Karten pro Mission zur Verfügung, welche genutzt werden können. Dies ist sehr vorteilhaft, denn es spart eine Menge Performance, da die Einleitungs- bzw. Abspann-Einheiten und -Objekte nicht auf der eigentlichen Missionskarte sind. Zusätzlich hat man somit auch gleich mehr Übersicht beim Editieren auf der Hauptkarte.

Ein weiterer Punkt ist, dass der Spieler beim mehrfachen Spielen der Mission das Intro irgendwann nicht mehr sehen mag und dann die Möglichkeit hat, dieses per Leertaste wegzuklicken. Würde das Intro auf der Hauptkarte erstellt, wäre ihm dies nicht möglich und er müsste die Sequenz jedes Mal bis zum Ende schauen, was auf Dauer etwas demotivierend sein kann.

Laden

Mit Laden lädt man, wie der Name schon sagt, seine Missionen in den Editor. Natürlich muss diese auch im Ordner **C:\EigeneDateien\ArmA\Benutzer\Missions** vorhanden sein. Sprich, man muss zunächst seine Mission abgespeichert haben. Denn nach der Installation des Spiels ist dieser Ordner noch leer!

Fertige Missionen, welche im ArmA-Hauptmenü zum Spielen ausgewählt werden können, kann man ohne ein PBO-Tool nicht in den Editor laden. Diese müssen dazu erst mit einem solchen Tool entpackt werden.

Zusammenführen

Zusammenführen steht auch für Importieren. Man kann hiermit eine andere Mission bzw. die Einheiten, Objekte, Auslöser, Wegpunkte, Marker usw. aus einer anderen Mission importieren. Importiert wird hierbei alles, was auf dieser Karte zu finden ist, jedoch nicht der Ordnerinhalt.

Das Zusammenführen ist sehr nützlich, wenn man beispielsweise eine sehr komplexe Mission, die eine Weile zum Laden benötigt, editiert. Wenn man nun an einer Position noch ein paar Objekte platzieren und ausrichten möchte, müsste man teilweise ewig warten bis die Mission zur Vorschau startet. Aus diesem Grund ist es sinnvoll eine zweite Map anzulegen, welche man dann als Mission2 abspeichert und dort die Objekte setzt und ausrichtet. Ist man damit fertig, importiert man diese Mission danach einfach auf seine Hauptmissionskarte.

Eine weitere gute Möglichkeit ist, wenn man sich Vorlagen anlegt. Also zum Beispiel Karten auf welchen nur bestimmte Dinge vorgefertigt sind. Zum Beispiel ein selbsterstelltes Lager oder ähnliches. Man kann dies dann immer auf seine aktuelle Map importieren ohne es nochmal neu erstellen zu müssen.

Speichern
Mit Speichern wird die Mission abgespeichert. Hierbei kann man auswählen, ob die Mission zunächst als Editormission abgespeichert oder schon als fertige Mehrspieler- bzw. Einzelspielermission exportiert werden soll. Während Editormissionen im Verzeichnis **C:\EigeneDateien\ArmA\Missions** hinterlegt werden, findet man seine exportierten Mehrspielermissionen im Spieleverzeichnis unter **MP-Missions** und seine Einzelspielermission im Verzeichnis **Missions** wieder. Auf dem Bild sieht man den Ordner Missions unter „Eigene Dateien".

Zurücksetzen
Mit Zurücksetzen säubert man die Karte. Die Karte wird damit wieder in ihren Urzustand versetzt. Dabei wird lediglich alles was auf der Karte ist gelöscht. Der Missionsordner samt Inhalt bleibt aber weiterhin vorhanden.

ID´s zeigen
Mit dem Button ID´s zeigen lässt man sich die ID´s der Objekte auf der Karte anzeigen. Denn jedes Objekt auf der Karte hat eine ID, mit welcher man es ansprechen kann. Dies macht es möglich dem Objekt Schaden zuzufügen oder zu prüfen, ob es noch am Leben oder, anders gesagt, noch existent ist.

Texturen zeigen
Hiermit lassen sich die Texturen einer Karte anzeigen. Jede Variante, also mit Texturen anzeigen oder ohne, hat gewisse Vor- und Nachteile beim Editieren. Letztendlich muss jeder für sich entscheiden, wie er lieber editieren möchte.

Vorschau
Mit Vorschau kommt man in die Missionsvorschau, also direkt in die Mission, die man gerade editiert und kann sich so sein Editierergebnis anschauen und testen.

Weiter
Hiermit wechselt man die letzte Vorschau zurück. Änderungen, die man bis dahin im Editor vorgenommen hat, sind dann noch nicht sichtbar.

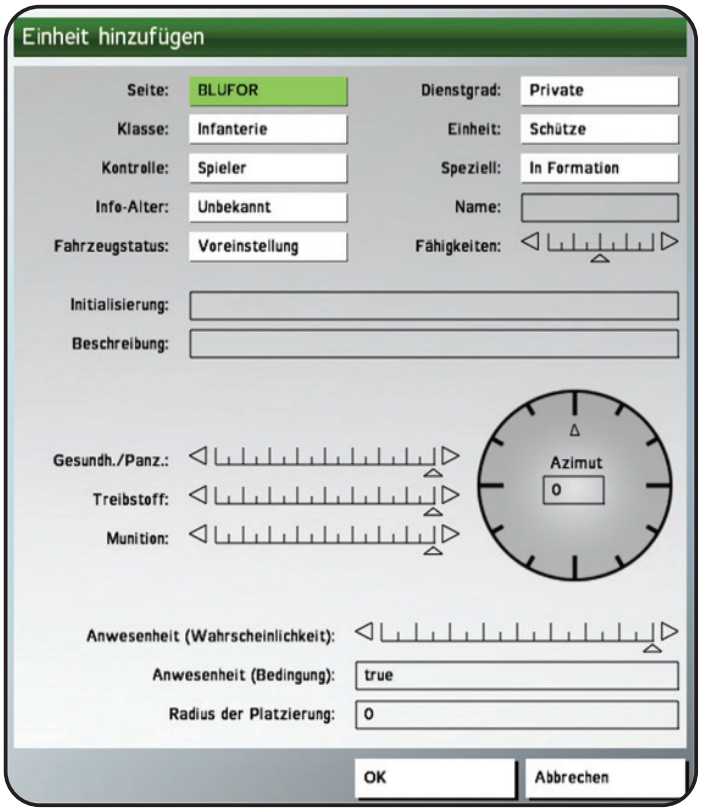
Beenden
Mit Beenden verlässt man den Editor und kommt zurück zum Hauptmenü.

1.2 - Einheiten einfügen (F1)

Mit der Taste F1 oder per Mausklick wählt man das Untermenü **Einheit** an, um eine Einheit, ein Objekt, ein leeres Fahrzeug, eine Logik oder ähnliches einzufügen. Hier ist es möglich sämtliche Einstellungen für jede Einheit individuell vorzunehmen.

Seite	Auswahl der Seite
Osten	Osteinheiten
Westen	Westeinheiten
Widerstand	Widerstand
Zivil	Zivilisten
Leer	Leere Vehikel
Logik	Logik

Klasse	Art der Einheit
Luft	Helikopter, Flugzeug
Munition	Waffen- u. Munition
Gepanyert	Panzer
Auto	Autos, Krad, Lkw
Infanterie	Soldaten
Minen	Minen
Objekte	statische Objekte
Schiff	Boote
Sounds	Sounds
Statisch	Geschütze, MG´s
Unterstützung	Support-Lkw´s



Kontrolle - Spieler oder Spielbar

Hier wählt man aus, ob man Spieler dieser Einheit sein soll oder ob diese Einheit spielbar bzw. nicht spielbar, also eine KI ist.

Spielbare Einheiten werden beispielsweise für Multiplayermissionen benötigt, bei denen ja für die Spieler mehrere Einheiten zur Auswahl stehen müssen. Für Singleplayermissionen ist Playable nötig, wenn man den Characterswitch nutzen möchte. Der Spieler kann dann in der fertigen Mission zwischen den spielbaren Soldaten umherschalten und diese einzeln steuern und in Position bringen.

Player	Spieler
Playable	Spielbar
Non Playable	Nicht Spielbar

Bei Fahrzeugen hat man zusätzlich die Auswahl, als was der Spieler im Fahrzeug sitzen soll und kann dies somit sofort festlegen.

Player as Driver	Spieler als Fahrer
Player as Pilot	Spieler als Pilot
Player as Gunner	Spieler als Schütze

Fahrzeugstatus - Einstellung des Fahrzeugs

Default	Voreinstellung
Locked	Abgeschlossen
Unlocked	Nicht abgeschlossen

Die Syntax für externes Ansteuern über ein Skript oder eine Initzeile sieht wie folgt aus:

Name lock true	Fahrzeug verschlossen
Name lock false	Fahrzeug offen

Dienstgrad - Dienstgrad der jeweiligen Einheit
Hier wird der Rang bzw. Dienstgrad der jeweiligen Einheit eingestellt. Die Einheit mit dem höchsten Dienstgrad ist automatisch der Gruppenführer der Gruppe.

Private	Soldat
Corporal	Gefreiter
Sergant	Feldwebel
Lieutnat	Leutnant
Captain	Hauptmann
Major	Major
Colonel	Oberst

Die Syntax für das Vergeben eines Ranges sieht dabei wie folgt aus:

Player setRank "Sergant"

Einheit - Art der Einheit
Nachdem man bei **Klasse** festgelegt hat, ob das Objekt ein Soldat oder Fahrzeug ist, kann man hier die Art des Objektes auswählen. Je nach **Klassen**-Auswahl ist hier die Unterauswahl anders. Bei **Infanterie** hat man die Auswahl der Soldaten vom Grenadier bis zum Scharfschützen, bei **Fahrzeuge** die Auswahl zwischen verschiedenen Fahrzeugen und so weiter.

Speziell - Besonderheit einer Einheit

Hier lassen sich gleich verschiedene Einstellungen vornehmen, die beim Editieren gerne übersehen werden. Denn hier ist es nicht nur möglich einzustellen, ob ein Flugzeug oder Helikopter beim Start fliegt, sondern noch weit mehr.

Keine	Würde man sich eine Einheit mit dem **Spezial** „**Keine**“ unterstellen und sie irgendwo weit weg auf die Karte stellen, würde sie sich, wenn sie weiß wo sich der Leader befindet, auf den Weg zu ihm begeben. Anders bei Formation…
In Formation	Ist beim **Spezial** der unterstellten Einheit „**In Formation**“ ausgewählt, wird die Einheit gleich, egal wo sie auf der Karte steht, an ihrer Formationsstelle neben dem Leader stehen.
Im Laderaum	Setzt man eine Gruppe auf eine Karte, von der eine Einheit ein Fahrzeug ist (muss nicht der Leader sein), und wählt bei allen anderen Einheiten bei **Spezial** „**Im Laderaum**“ aus, werden alle Einheiten der Gruppe in dem Fahrzeug sitzen.
Fliegen	Ein Flugzeug oder ein Heli sind bei Missionsstart gleich in der Luft.

Name - Name der Einheit

Hier wird der Name der jeweiligen Einheit oder des Objektes angegeben. Dieser ist sehr wichtig, wenn man diese Einheit per Skript oder ähnliches anprechen möchte.

Fähigkeiten - Fähigkeiten einer Einheit

Hier definiert man die Fähigkeiten einer Einheit. Diese bestimmt, wie gut diese schießen, reagieren und so weiter. Die Fähigkeiten werden mit einem Wert zwischen **0** und **1** definiert. **0** steht für weniger gut und **1** für sehr gut.

In Syntaxform schaut das Ganze wie folgt aus:

Name1 setSkill 0.8

Name1 setUnitAbility 0.6

Es ist auch möglich einen Zufallsskill einzustellen. Dazu müsste man in der Initialisierungszeile der Einheit folgende Syntax eingeben:

this setSkill (Random 0.6)

Jetzt würde der Einheit ein Zufallsskill im Bereich von **0** bis **0.6** zugewiesen.

Azimut - Die Blickrichtung einer Einheit

Hier legt man beim Einstellen der Einheit eine grobe Blickrichtung fest und kann somit auch gleich den Blickrichtungswert sehen. Die Blickrichtungswerten sind, wie im realen Leben auch, von **0** bis **360** Grad definiert. Bestätigt man nun mit OK, sieht man die Einheit mit der ausgewählten Blickrichtung auf der Karte.

Wenn man diese im Nachhinein nochmal drehen möchte, weil sie doch nicht in die richtige Richtung schaut, geht das auch in Kombination mit der Maus und der Shift-Taste. Dazu bleibt man auf der Karte und klickt sich nicht per Doppelklick ins Einheiten-Untermenü, sondern wählt die Einheit an, indem man sie nur ein Mal anklickt. Jetzt hält man die Shift-Taste und die linke Maustaste gedrückt und bewegt die Maus. Die Einheit wird sich nun mit Hilfe der Mausbewegungen frei drehen lassen.

Zum Drehen von mehreren Einheiten oder Objekten gilt das gleiche Prinzip. Dazu markiert man alle zu drehenden Einheiten und bewegt dann die Maus mit gedrückter linker Maustaste.

Eine andere Möglichkeit eine Einheit beispielsweise in einer Sequenz auszurichten wäre:

Name1 setDir Wert

Name1 setFormDir Wert

Hierzu kann man den Wert aus dem **Einheiten-Untermenü** mit der Tastenkombination **Strg +** C kopieren und für **Wert** einfügen.

Initialisierung - Die Initialisierungszeile

Jede Einheit bzw. jedes Objekt hat eine Initialisierungszeile. Befehle, welche dort eingetragen werden, werden beim Missionsstart sofort ausgeführt. Von hier aus kann man unter anderem Skripte starten oder wie oben bei **Fähigkeiten** eine Zufallsskillsyntax angeben. Generell lohnt es sich eine **Init.sqs** im Missionsordner zu erstellen, welche die Initalisierungsdatei der Mission darstellt und ohne angeben einer Syntax beim Missionsstart automatisch ausgeführt wird. Eine nähere Erklärung der **Init.sqs** findet man im **Kapitel 2.5 – Die Init.sqs**.

Sogar während einer laufenden Mission ist es möglich einen Eintrag in der Initialisierungszeile einer Einheit vorzunehmen und ausführen zu lassen. Dazu nutzt man zunächst den setVehicleInit-Befehl und zum Aufrufen dann processInitCommands.

Player setVehicleInit "Player say 'Sound1' "; processInitCommands;

Beschreibung - Die Infozeile

In dieser Zeile gibt man einen Namen oder eine kleine Beschreibung zur jeweiligen Einheit an. Diese Beschreibung wird dann angezeigt, wenn die Einheit als switchbare Einheit definiert wurde und man das Switchmenü öffnet.

Gesundheit/Panzerung - Der Gesundheitsstatus

Mit dem Schieberegler lässt sich der Gesundheits- bzw. Panzerungsstatus einer Einheit festlegen. Einheiten können also schon verletzt auf die Karte gestellt werden oder eben Fahrzeuge, wenn sie als Wrack dienen sollen, ohne jegliche Panzerung. Der Wert wird ebenfalls wieder von **0** bis **1** definiert und lässt sich auch extern des Einheiten-Untermenüs mit einer Syntax festlegen. Diese schaut wie folgt aus:

```
Name setdammage 1
```

Nun hätte die Einheit, der Panzer oder das Objekt einen Schadenswert von **1** und ist tot oder zerstört. Diesen Wert kann man natürlich wieder zurücksetzen und mit

```
Name setdammage 0
```

wäre die Einheit nun wieder geheilt oder repariert. Selbstverständlich können auch Zwischenwerte verwendet werden. Mit **0.5** wäre die Einheit halt nur zur Hälfte geheilt bzw. repariert usw.

Support:

Supportfahrzeuge wie Treibstoff-, Munitions- und Reparaturfahrzeuge können mit einem Wert von **0** bis **1** versehen werden. Ist dieser **0,** kann man diese nicht mehr zum Tanken, Reparieren oder Aufmunitionieren nutzen. Mit der Syntax

```
Name setRepairCargo 1
```

weist man einem Reparatur-Lkw einen Reparaturwert zu.

Treibstoff - Der Tankstatus

Hier stellt man die Treibstoffmenge ein, die ein Fahrzeug haben soll. Diese lässt sich ebenfalls extern mit einer Syntax ansteuern, welche wie folgt aussieht:

```
Name setfuel 0        steht für einen leeren Tank
Name setfuel 1        steht für einen vollen Tank
```

Support:

```
Name setFuelCargo 0.3    weist Tank-Lkw einen Füllwert zu
```

Munition - Der Munitionsstatus

Einstellung der Munitionsmenge, die die Einheit beim Missionsstart haben soll.

Support:

```
Name setAmmoCargo 0.7    weist Munitions-Lkw einen Füllwert zu
```

Anwesenheit - Wahrscheinlichkeit der Anwesenheit

Hier lässt sich die Wahrscheinlichkeit der Anwesenheit einer Einheit in Prozent einstellen. Stellt man den Regler auf die Mitte, wird die Einheit mit einer Wahrscheinlichkeit von 50% auf der Karte sein. Dies bringt sehr viel Dynamik ins Spiel, da man nie vorhersagen kann, ob die Einheit oder Einheiten wirklich auch auf der Karte sind.

Anwesenheit - Bedingung der Anwesenheit

Einheit ist nur anwesend, wenn eine Bedingung erfüllt ist. Diese Bedingung wird hierbei beim Missionsstart abgefragt. Gibt man hier CadetMode ein, wird die Einheit nur im CadetMode auf der Karte sein.

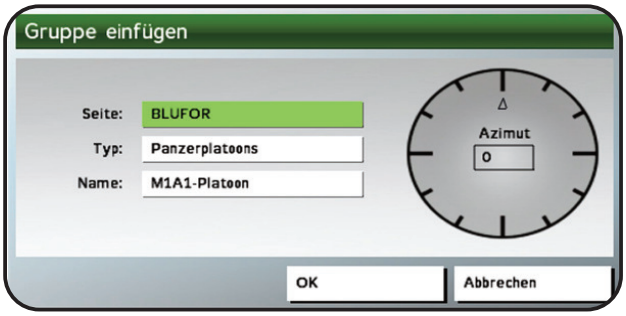
Radius der Platzierung - Platzierungsradius der Einheit

Hier stellt man den Radius ein, in welchem die Einheit beim Spielstart stehen soll. Diese Einheit wird dann bei jedem Spielstart irgendwo in dem Radius erscheinen.

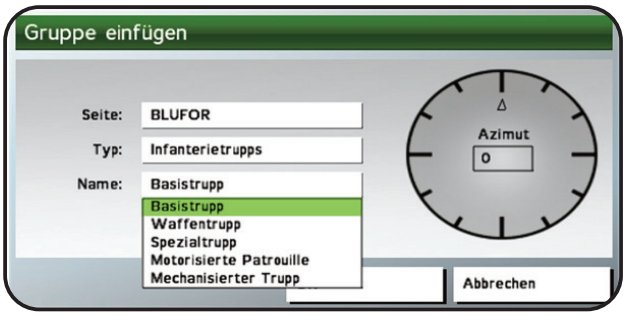
1.3 - Gruppen einfügen (F2)

Mit der Taste F2 wählt man das Untermenü **Gruppen** an, welches es einem ermöglicht gleich ganze Gruppen einzufügen. Dies spart eine Menge Zeit, da man nicht jede Einheit einzeln einsetzen muss. Hier gibt es für jede Seite vordefinierte Gruppen, welche man dann einfügen kann. Natürlich ist es auch möglich diese Gruppen nach belieben zu bearbeiten. Eine Gruppe zu vergrößern, zu verkleinern oder alle Einheiten einer Gruppe individuell einzustellen und zu bewaffnen ist sehr leicht zu handeln.

Nachdem man sich bei **Seite** für eine Seite entschieden hat, hat man bei **Typ** die Wahl zwischen einer Infanterie-, Panzer, oder Helikoptergruppe, welche man so sehr schnell einfügen kann.



Bei **Name** hat man dann die Möglichkeit sich eine von fünf Typen auszusuchen.



Hier hat man zum Beispiel bei West und Ost die Auswahl zwischen:

Basistrupp	Gemischte Infanteriegruppe
Waffentrupp	Kleinere Infanteriegruppe
Spezialtrupp	Spezialeinheit
Motorisierte Patrouille	Infanteriegruppe mit Fahrzeug
Panzergrenadiertrupp	Infanteriegruppe mit Schützenpanzer

Eine ähnliche Auswahl hat man neben den **Infanterietruppen** natürlich auch unter den **Panzerplatoons** oder den **Luftgeschwadern**. Bei den Air Squadrons muss man darauf achten, dass bei den einzelnen Vehikeln das **Spezial** auf **Fliegen** steht, wenn diese gleich zu Missionsbeginn fliegen sollen.

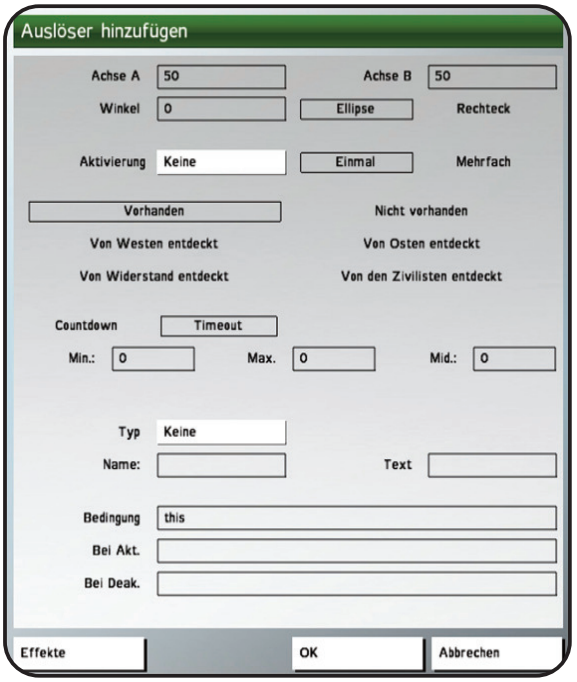
Mit **Azimut** stellt man hierbei wieder, wie auch bei anderen Einheiten, die Blickrichtung der Gruppe ein.

1.4 - Auslöser einfügen (F3)

Den Auslöser kann man als Ein- bzw. Ausschalter oder Prüfutensil benutzen. Er ermöglicht unter anderem auch das Einfügen der Funksprüche von Alpha bis Juliett. Von der Funktionsweise her unterscheiden sich Auslöser und Wegpunkte bis auf ein paar Extras nicht sonderlich. Der Auslöser ermöglicht es im Missionsablauf gewisse Aktionen zu starten oder zu stoppen. Mit Hilfe des Untermenüs Effekte hat man sogar die Möglichkeit Sounds, Musik, Ressourcen einspielen zu lassen oder auch Videoanimationen zu starten. Mit der Aktivierungszeile hat man zusätzlich die Möglichkeit beim Betreten des Auslöserbereiches Skripte oder ähnliches zu starten oder beim Verlassen des Auslöserbereiches bei Deaktivierung wieder zu stoppen.

Zudem kann man den Auslöser auch als Lineal benutzen, wenn man z.B. mehrere Objekte gerade aneinander setzen möchte. Dazu würde man z.B. bei **Achse A** den Wert 100 und bei **Achse B** den Wert 1 angeben. Jetzt hat man ein prima Lineal oder Abstandsmesser.

Achse A/Achse B	Wirkungsbereich
Winkel	Winkel
Ellipse/Rechteck	Form der Fläche
Einmal/Mehrfach	Anzahl der Auslösung
Aktivierung durch	Westen Osten Widerstand Zivilisten Spiellogik Jeder Funk A–J Erobert von Westen Erobert von Osten Erobert von Widestand



Art der Aktivierung

Vorhanden	Auslöser wird aktiviert, wenn bei **Aktivierung** zum Beispiel West ausgewählt wurde und eine westliche Einheit den Auslöserbereich betritt.
Nicht vorhanden	Auslöser wird deaktiviert, wenn bei **Aktivierung** zum Beispiel West ausgewählt wurde und keine westliche Einheit mehr in dem Auslöserbereich ist.
Entdeckt durch	Entdeckt durch Westen, Osten, Widerstand oder Zivilisten. Wird man im festgelegten Auslöserbereich durch eine im Auslöser festgelegt Seite entdeckt, werden die Aktionen ausgeführt, die im Auslöser definiert sind.

Countdown/Timeout - Zähler

Hier wird festgelegt, ob der Auslöser gleich bei Auslösung und erst nach Ablauf einer festgelegten Zeit ausgelöst werden soll. Durch die Möglichkeit einen Min, Mid und Max-Wert einzugeben hat man auch hier wieder eine gewisse Dynamik. Während der Min-Wert für die mindeste Wartezeit steht, stehen der Mid-Wert für den Mittelwert und der Max-Wert für den Maximalwert der Wartezeit. Durch Angabe aller drei Werte kann man die Auslösung dem Zufall überlassen. Gibt man überall den gleichen Wert ein, so wird der Auslöser nach Ablauf der angegebenen Zeit auslösen.

Typ - Auslösertyp

Keine	Keine
Von Westen bewacht	Geschützt durch Westen
Von Osten bewacht	Geschützt durch Osten
Von Widerstand bewacht	Geschützt durch Widerstand
Schalter	Schalten
Ende 1 bis 6	Ende einer Mission
Verlieren	Verloren am Ende

Text - Der Auslösertext

Den Auslösertext kann man im Editor lesen, wenn man mit der Maus über das Auslöserfähnchen fährt. Er soll die Orientierung auf komplexen Karten erleichtern. Man muss so nicht jeden Auslöser wieder öffnen und schauen was darin definiert ist, sondern erkennt ihn so am selbst festgelegten Namen schnell wieder.

Des Weiteren wird hier der Text für das Funkmenü definiert. Wenn man also mehrere der Funksprüche verwendet, kann man diese hier entsprechend benennen und man sieht in der späteren Mission besser welcher Funkspruch für welche Aktion vorgesehen ist.

Zum Beispiel:

Auslöser 1:	**Aktivierung:**	Radio Alpha
	Text:	Artillerie anfordern
Auslöser 2:	**Aktivierung:**	Radio Bravo
	Text:	Unterstützung anfordern

Beim Funkgerät auf der Kartenansicht würde jetzt Artillerie anfordern und Unterstützung anfordern stehen, was ein gewisses Maß an Übersicht gewährleistet.

Name - Der Auslösername

Der Name eines Auslösers ist wichtig, wenn man diesen auf irgendeine Art ansprechen möchte. Wenn man ihn beispielsweise versetzen oder löschen will. Mehr zum Versetzen oder Löschen eines Auslösers gibt es auf Seite 30 zu lesen.

Bedingung - Die Bedingung

Das Angeben einer Bedingung ermöglicht es einen Auslöser sozusagen auf „Stand by" zu setzen oder ihn etwas prüfen zu lassen. Dabei würde der Auslöser erst starten, wenn diese Bedingung erfüllt ist. Die Angabe von Bedingungen kann hierbei ganz verschieden aussehen. Zum Einen kann man hier die **Variante Verwendung einer Variablen** und zum Anderen das **Prüfung einer Bedingung** verwenden.

Verwendung einer Variablen

Gibt man in der Aktivierungszeile eine Variable an, wird der Auslöser erst ausgelöst, wenn diese erfüllt, also true ist. Wenn in der Aktivierungszeile beispielsweise die Variable **Angriff** steht, wartet der Auslöser, bis **Angriff** auf **true** geschaltet wurde.

Irgendwo auf der Karte muss dazu bei **Aktivierung** eines Auslösers, eines Wegpunktes oder in einem Skript die Variable Angriff also zunächst auf **true** gesetzt werden, damit dieser aktiviert wird. Das macht man mit folgender Syntax:

Angriff=true

Der Auslöser wurde nun aktiviert und startet die Aktionen, die darin definiert wurden.

Prüfung einer Bedingung

Eine andere Variante ist die Prüfung, ob eine Bedingung erfüllt ist. Zugegeben, eine Variable ist ja auch eine Bedingung, die geprüft und erfüllt werden muss. Hierbei ist eine Bedingung im vorstellbaren Sinne gemeint, welche man quasi als vorstellbare Aktion definieren kann. Zum Beispiel, dass Soldat1 nicht mehr lebt oder der Spieler(Player) im Fahrzeug Jeep1 sitzt. Dies würde dann wie folgt aussehen:

Bedingung: Spieler sitz in Jeep1:

Player in Jeep1 oder **Vehicle Player == Jeep1**

Bedingung: Soldat1 lebt nicht mehr:

Not alive Soldat1 oder **!(alive Soldat1)**

Bei Aktivierung

In dieser Zeile kann man fast alles definieren, was ausgeführt werden soll, wenn der Auslöser ausgelöst wird. Das Starten eines Skriptes, eine Bedingung auf true schalten usw.. Es ist also möglich so ziemlich jede Syntax hier anzugeben, die Armed Assault zu bieten hat. Natürlich gibt es auch Grenzen, aber da kann man ja dann auf ein Skript ausweichen. Auch der Übersicht halber, ist es oftmals besser auf ein Skript auszuweichen, wenn hier zum Beispiel unzählige Befehle oder ähnliches angegeben werden.

Bei Deaktivierung

Natürlich kann man bei einem Auslöser auch eine Aktion starten lassen, wenn dieser deaktiviert wird. Als Beispiel dient hier mal die Variante mit einem Actionmenüeintrag. Wenn der Spieler den Auslöserbereich betritt, soll er einen Actionmenüeintrag namens **Eintrag** bekommen und wenn er den Bereich wieder verlässt, soll er wieder gelöscht werden. Dazu vergibt man:

```
bei Aktivierung:     ID=Player addAction ["Eintrag","skript.sqs"]
bei Deaktivierung:   Player removeAction ID
```

Auslöser versetzen oder löschen

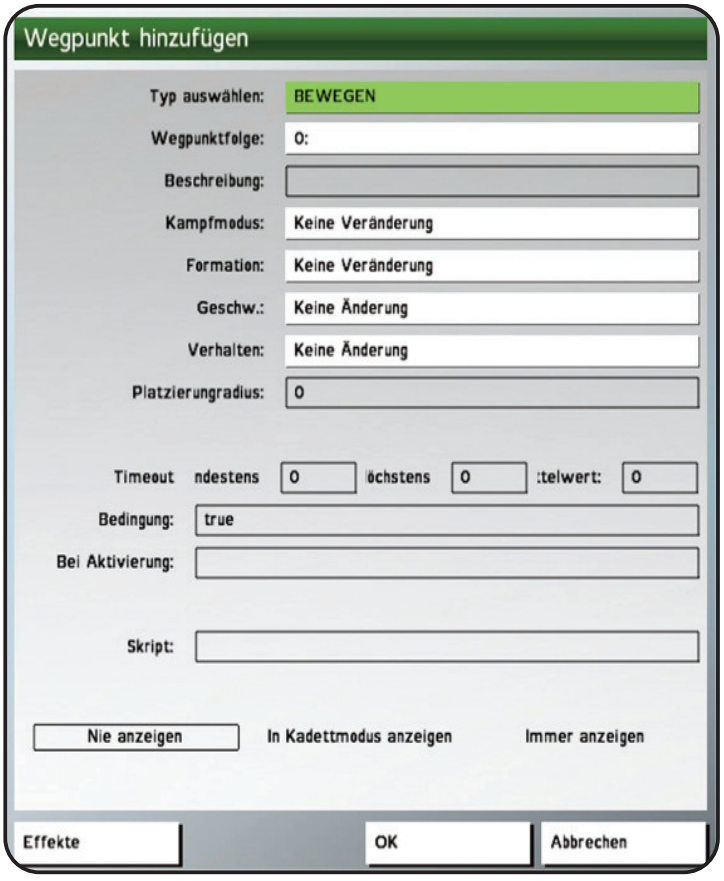
Auslöser lassen sich bei Bedarf verschieben oder auch löschen. Dazu muss der Auslöser mit einem Namen versehen werden, um ihn direkt ansprechen zu können. Dazu werden unter anderem folgende Befehlszeilen verwendet:

Auslösername setpos getpos Name

Oder in Koordinatenform:

Auslösername setpos [x,y,z]

1.5 - Wegpunkte einfügen (F4)



Mit dem Wegpunkten legt man nicht nur die Route fest, auf welcher sich eine Einheit bewegen soll, sondern auch Einstellungen wie Verhalten, Formation, Kampfmodus und Geschwindigkeit.

Ein Wegpunkt kann von der Funktionsweise her mit einem Auslöser verglichen werden.

Wenn die jeweilige Einheit ihren Wegpunkt erreicht hat, wird eine Aktion ausgeführt bzw. begibt sie sich weiter zu ihrem nächsten Wegpunkt.

Hinzu kommen aber noch eine Menge weiterer Definitionsmöglichkeiten, wie zum Beispiel Soundeffekte oder Musik am jeweiligen Wegpunkt abzuspielen oder ähnliches.

Typ Auswählen - Die Aktionen
Hier lassen sich für jeden Wegpunkt individuelle Einstellungen festlegen, welche eine Einheit beim Erreichen des Wegpunktes ausführen soll.

Move	Bewegen
Destroy	Zerstören
Get in	Einsteigen
Seek and destroy	Suchen und Zerstören
Join	Anschließen
Join and lead	Anschließen und führen
Get out	Aussteigen
Cycle	Wiederholen
Load	Laden
Unload	Entladen
Transport unload	Transport entladen
Hold	Halten
Sentry	Aufklären
Guard	Bewachen
Talk	Sprechen
Scripted	Geskriptet
Support	Unterstützen
Get in next	In Nähestes einsteigen
Released	Entlassen

Aufklären Besonderheit
Weist man einer Einheit/Gruppe einen solchen Wegpunkt zu, wird sie bis zu diesem Punkt laufen und dort auf Feindkontakt warten, bevor sie zum Nächsten weiterläuft.

Wegpunktfolge
Beim Anklicken dieser Option, kann man sämtliche Wegpunkte einer Einheit mit jeweiliger Aktion auf einem Blick sehen und im Nachhinein sogar die Abfolge ändern, indem man einfach eine andere Zahl auswählt.

Beschreibung
Dies ist die Beschreibung eines Wegpunktes. Was hier eingetragen wird, wird dann später nur im Cadet-Mode angezeigt werden und erleichtert dem Anfänger das Spielen und die Orientierung in einer großen und komplexen Mission.
Beispiel: Zerstören Sie das Ziel

Kampfmodus
Hier definiert man den Kampfmodus, den eine Einheit oder Gruppe haben soll.

Nie schießen	Blue
Nicht schießen	Green
Nicht schießen, Angriff nach eigenem Ermessen	White
Feuer eröffnen	Yellow
Feuer eröffnen, eigenständ. Angriff	Red

Die Syntax, um das Verhalten per Auslöser oder Skript festzulegen, schaut hierbei wie folgt aus:

Name setCombatMode "Blue"

Verhalten
Hier lässt sich das Verhalten einer Einheit oder Gruppe definieren, wobei man folgende Auswahlmöglichkeiten hat:

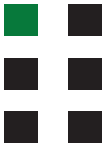
Careless	Achtlos
Safe	Sicher
Aware	Wachsam
Combat	Kampf
Stealth	Tarnung

Natürlich lässt sich das Verhalten auch wieder per Auslöser oder Skript definieren. Die Syntax hierfür lautet:

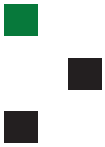
Name setBehaviour "Careless"

Formation
Diverse Situationen auf dem Schlachtfeld erfordern gewisse Formationen, welche in der jeweiligen Situation von Vorteil sind. Diese schauen wie folgt aus, wobei der Leader hier in grün zu sehen ist:

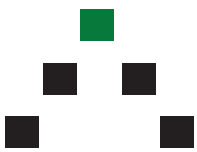
Columm - Kolonne



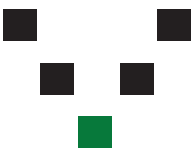
Staggered Columm - Gestaffelte Kolonne



Wedge - Keil



Vee - V-Förmig

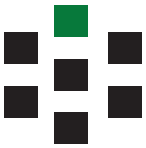


Echelon Left - Staffel links

Echelon Right - Staffel rechts

Line - Reihe

Delta - 3-Reihig



Kolonne (Kompakt)

Natürlich kann man einer Gruppe auch eine Formation per Auslöser oder Skript zuweisen. Das Ganze schaut dann in Sytaxform wie folgt aus:

Name SetFormation "Line"

Geschwindigkeit

Hiermit wird die Geschwindigkeit einer Einheit am jeweiligen Wegpunkt definiert. Zum Beispiel ist es möglich, einer Einheit zum Wegpunkt 1 eine langsame und zum Wegpunkt 2 eine schnellere Geschwindigkeit zu geben. Dabei hat man die Auswahl zwischen drei Varianten.

Normal / Limited / Full

Hierbei ist es auch wieder möglich das Ganze in einem Auslöser oder Skript zu definieren. Syntax:

Name SetSpeedMode "Limited"

Platzierungsradius

Der Radius eines Wegpunktes bringt wieder ein wenig mehr Dynamik ins Spiel. Hierbei wird in dem Bereich der festgelegt wird eine Zufallsposition bestimmt, die bei jedem Missionsstart anders ist. Dies bedeutet dass, wenn man dort zum Beispiel den Wert 100 angibt, ein Wegpunktradius von 100 Metern definiert wird, in dem der Wegpunkt dynamisch gesetzt wird.

Timeout - Verzögerung

Die Verzögerung bis zur Auslösung in Sekunden. Gibt man bei allen drei Positionen den gleichen Wert an, so ist die Auszeit dem Wert entsprechend lang. Durch das Angeben verschiedener Werte wird der jeweilige Wegpunkt per Zufallszeit aktiviert. Zufallszeit zwischen Min und Max mit Tendenz zum Mittelwert.

Min	Mindestzeit bis zur Aktivierung
Max	Maximale Zeit bis zur Aktivierung
Mid	Mittelwert

Das Verwenden der Zufallszeit bringt wieder einen gewissen Grad an Spannung ins Spiel, da man nie genau vorhersagen kann, wann die Einheiten auf der jeweiligen Stelle der Karte ankommen. Kombiniert mit dem **Platzierungsradius** und der **Bedingung der Anwesenheit** hat man sogar noch mehr Dynamik. Denn nun weiß man nicht, ob es die Einheit gibt, wohin sie Punktgenau läuft und wann sie kommt. Gerade dynamische Missionen haben einen hohen Spannungs- und Wiederspielgrad. ArmA besitzt anhand solcher Möglichkeiten die besten Vorraussetzungen dafür, seine Missionen so dynamisch wie möglich zu gestalten.

Bedingung

Das Angeben einer Bedingung ermöglicht es einen Wegpunkt sozusagen auf „Stand by" zu setzen oder ihn etwas prüfen zu lassen. Dabei würde der Wegpunkt erst aktiviert, wenn diese Bedingung erfüllt ist. Das Verwenden einer Bedingung ist bereits unter **Bedingung** beim Auslöser erklärt.

Bei Aktivierung

In dieser Zeile kann man fast alles definieren, was ausgeführt werden soll, wenn der Wegpunkt ausgelöst wird. Das Starten eines Skriptes, eine Bedingung auf true schalten usw. usw. Es ist also möglich so ziemlich jede Syntax hier anzugeben, die Armed Assault zu bieten hat. Natürlich gibt es auch Grenzen, aber da kann man ja dann auf ein Skript ausweichen. Auch der Übersicht halber, ist es oftmals besser auf ein Skript auszuweichen, wenn hier zum Beispiel, unzählige Befehle angegeben werden oder ähnliches.

Skript

Diese Zeile ermöglicht die Verwendung von Syntaxes, wie sie sonst nur in Skripten verwendet werden können.

Wegpunkt zeigen

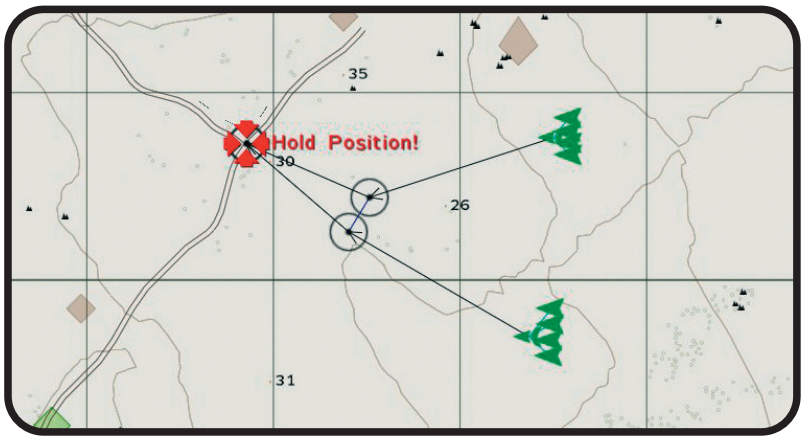
Wegpunkte lassen sich anzeigen oder verbergen. Hierbei kann man festlegen, ob sie nur im Cadet-Mode, generell oder gar nicht sichtbar sind.

Never Show	Wird nie angezeigt
Show in Cadet Mode	Nur im Cadet Mode
Always Show	Wird immer angezeigt

Anschließen und führen

Man setzt dazu zunächst zwei Gruppen auf die Karte, die an irgendeiner Position zu einer Gruppe verschmelzen sollen. Jede Gruppe bekommt jetzt einen Wegpunkt irgendwo auf der Karte. Bei dem einen Wegpunkt definiert man nun **Anschließen** und bei dem anderen Wegpunkt **Anschließen und führen**.

Jetzt müssen die beiden Wegpunkte synchronisiert werden. Dazu ist die **Synchronisation (F5)** notwendig, mit welcher die beiden Wegpunkte nun aufeinander abgestimmt werden. Wenn nun beide Gruppen ihre Wegpunkte erreicht haben und noch am Leben sind, werden sie zu einer Gruppe. Vorausgesetzt die Gruppen sind nicht zu groß. Die maximale Gruppengröße in Armed Assault beträgt inklusive Leader 144 Einheiten. Auf der Karte sieht das Ganze dann etwa so aus:



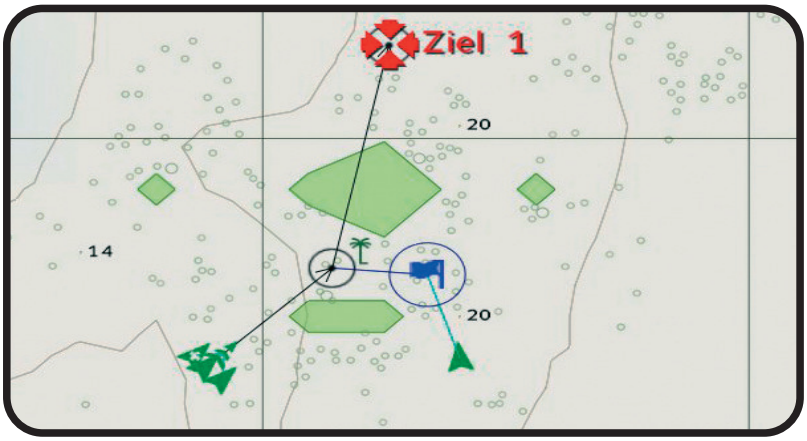
1.6 - Sychronisieren (F5)

Die Möglichkeit Wegpunkte mit Wegpunkten oder Wegpunkte mit Auslösern zu synchronisieren, wird gerne übersehen. Dank dieser Funktion kann man sich nämlich verschiedene Variablen sparen und man sieht auf einem Blick, wer wann wie starten darf.

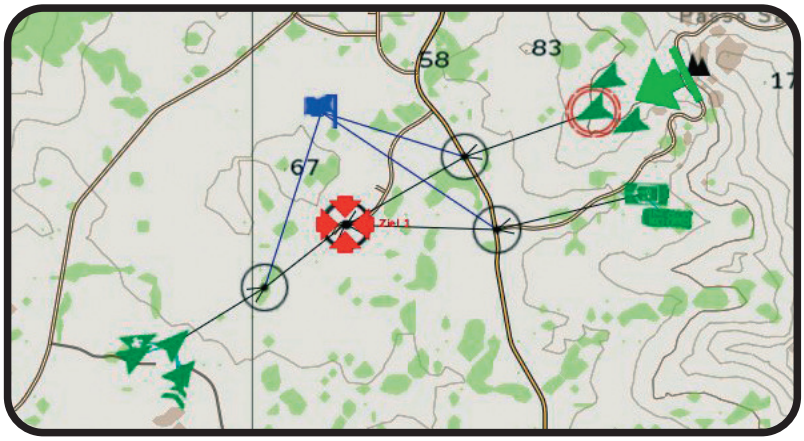
Auf dem Bild sieht man 2 Gruppen, die aus verschiedenen Richtungen kommen, aber das gleiche Ziel angreifen sollen. Jetzt soll die eine Gruppe am Wegpunkt verweilen, bis die andere Gruppe ihren Wegpunkt erreicht hat. Dazu wählt man mit F5 das **Synchronisieren** an, klickt mit der linken Maustaste auf den Wegpunkt der Gruppe 1, hält die Maus gedrückt und zieht einen Strich zum Wegpunkt der Gruppe 2. Die Synchronisation wird dann mit einer blauen Linie angezeigt. Sobald die Gruppe 1 nun ihren Wegpunkt erreicht hat, wird sie warten, bis Gruppe 2 ihren Wegpunkt erreicht hat.

Gleichermaßen läuft dies bei der Auslöser-Wegpunkt-Kombination. Die Gruppe würde erst weiterlaufen, wenn der Auslöser ausgelöst wurde. Man muss also nicht bei dem Wegpunkt einer Gruppe eine Variable (z.B. **Grp1go**) angeben und dann in der Aktivierungszeile des Auslösers **Grp1go=true** angeben, damit die Gruppe losläuft, wenn der Auslöser ausgelöst wird.

Die F5 -Variante ist doch viel einfacher und schneller. Ob der Auslöser dabei durch eine Einheit, ein Objekt oder Funk aktiviert wird ist dabei egal.



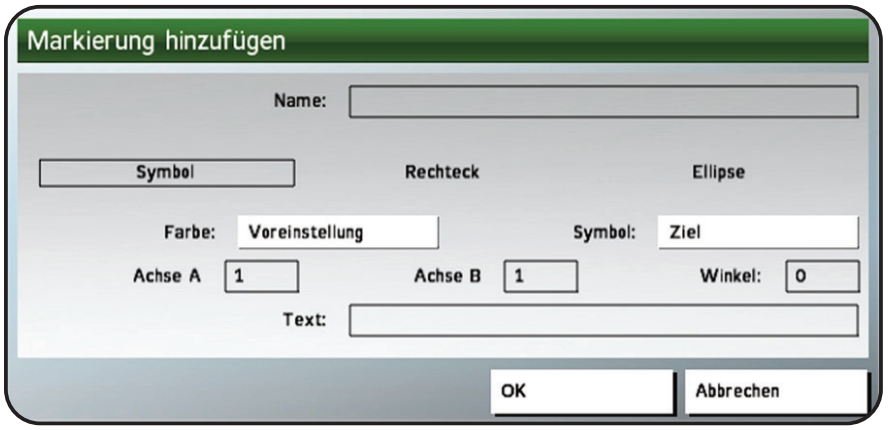
Bei folgendem Bild werden drei Gruppen und ein Funkauslöser aufeinander abgestimmt. Wenn alle drei Gruppen ihre Position erreicht haben, gibt der Spieler per **Funk 0-0-1** den Befehl zum Angriff. Erst dann werden sich die Gruppen zu ihrem nächsten Wegpunkt, also das Ziel, weiter bewegen.



1.7 - Markierungen einfügen (F6)

Mit den **Markern** gestaltet man die zu Anfang noch jungfräuliche Karte zur taktischen Karte um. Sie zeigt dem Spieler den Ablauf der Mission, die Ziele oder ähnliche Sachen an. Dies macht den Missionsablauf noch übersichtlicher. Hierbei empfiehlt es sich jedem **Marker** einen Namen zu geben.

Diese Marker lassen sich später mit dem Briefing verlinken. Klickt man dann später auf einen der Briefing-Links, wandert das Fadenkreuz auf der Karte zu dem zugehörigen Marker.



Name

Hier wird der Name des Markers angegeben, welcher später unter anderem für die Verknüpfung mit dem Briefing gebraucht, damit die Ziele durch das Fadenkreuz auf der Karte gezeigt werden können oder welcher es ermöglicht den Marker zu versetzen oder ihm ein anderes Symbol zuzuweisen.

Art

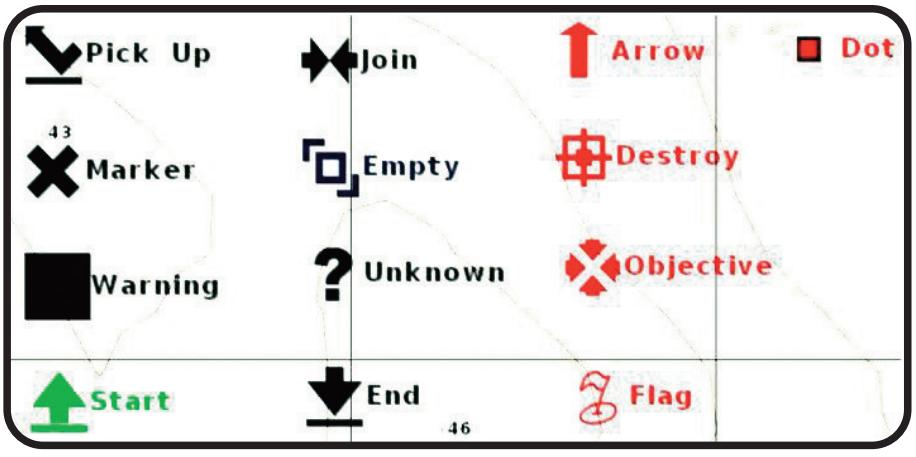
Die Auswahl der Art des Markers. Dies kann statt einem Symbol auch ein Kreis oder ein Rechteck sein, mit dem man eine größere Fläche markieren kann. Zum Beispiel ein Feindgebiet oder ähnliches.

Farbe

Farbe des Markers. Die Farben beschränken sich hierbei auf **rot, schwarz, grün, blau, gelb** und **weiß**

Symbol

Hier eine Übersicht der verschiedenen taktischen Zeichen mit Bezeichnung:



Mit diesen taktischen Zeichen in Verbindung mit großflächigen Markern (Ellipse/Rectangle) hat man schnell eine übersichtliche Missionskarte geschaffen. Folgend die Bezeichnungen mit Bedeutung, wobei die englischen Bezeichnungen zugleich auch die Klassennamen der Marker sind. Außer bei Objective! Objective hat seitens BIS den Klassennamen **Flag** bekommen.

Objective	- Ziel (**Flag**)	**Join**	- Beitreten
Flag1	- Fahne	**PickUp**	- Auflesen
Dot	- Punkt	**Unknown**	- Unbekannt
Destroy	- Zerstören	**Marker**	- Markierung
Start	- Start	**Arrow**	- Pfeil
End	- Ende	**Empty**	- Leer
Warning	- Achtung		

Achse A/Achse B
Hier wird die Größe des Markers eingestellt.

Winkel
Hier hat man die Möglichkeit den Winkel des Markers zu bestimmen.

Text
Hier wird der Text angegeben, der später auf der Karte lesbar sein soll. Zum Beispiel: Ziel Alpha. Mit folgender Syntax ist es sogar möglich dem Marker den Spielernamen oder Einheitennamen zuzuweisen:

"S1M" setMarkerText Name S1

Dabei liest das Spiel den Spielernamen der Einheit mit Namen **S1** automatisch aus.

Marker versetzen, ändern oder löschen
Marker lassen sich während des Spielverlaufs vom Symbol her ändern, versetzen oder auch löschen. Als Beispiel dient hier mal die Erfüllung eines Missionsziels, bei welchem man den Marker von rot in grün einfärben oder ihn einfach löschen könnte.

Dazu muss ein Marker erstmal benannt werden. Dieser heißt jetzt für die folgend aufgeführten Syntaxformbeispiele mal **Marker1**. Man hat jetzt eine Menge von Möglichkeiten diesen anzusprechen. Dies kann wieder von einem Wegpunkt, Auslöser oder auch Skript erfolgen. Hierbei gibt es unter anderem folgende Syntaxformen:

Marker wird zur Position [x,y] versetzt:

"Marker1" setmarkerPos [x,y]

Marker zur Position von Objekt Ziel versetzt:

"Marker1" setmarkerPos getpos Name

Marker1 wird zur Position von Marker2 versetzt:

"Marker1" setMarkerPos getMarkerPos "Marker2"

Setzt Art und Aussehen von Marker1:

"Marker1" setMarkerType "Start"

Ändert die Farbe des Markers:

"Marker1" setMarkerColor "ColorBlue"

Ändert die Markergröße in [Höhe, Breite]

"Marker1" setMarkerSize [2,4]

Löscht den Marker:

deleteMarker "Marker1"

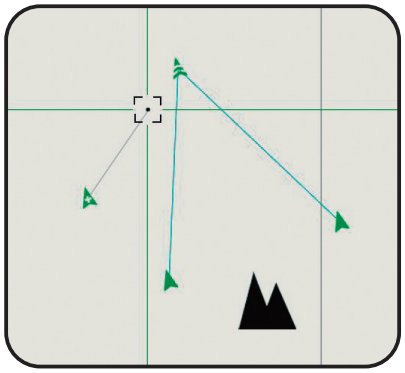
1.8 - Einheiten und Objekte drehen

Zum Drehen einer Einheit oder ganzen Gruppen und Objekten, markiert man diese zunächst, drückt dann die **Shift-Taste**, klickt dann mit der **linken Maustaste** auf eine der Einheiten oder ein Objekt, hält diese gedrückt und bewegt das Fadenkreuz in die jeweilige Richtung. So ist es auch möglich mehrere Einheiten oder Gruppen gleichzeitig zu drehen. Dazu markiert man diese und macht alles so, wie zuvor beschrieben.

1.9 - Einheiten verbinden

Einzelne Einheiten lassen sich im Editor je nach Bedarf zu Gruppen verbinden oder auch wieder trennen. Dazu wählt man zunächst die Taste F2, klickt mit der linken Maustaste auf die Einheit, hält die Taste gedrückt und zieht den Verbindungsstrich zu der Einheit mit der diese verbunden werden soll oder einfach ins Leere, wenn man diese eine Einheit von einer Gruppe trennen möchte. Auf diese Weise kann man auch eine Einheit oder ein Objekt mit einem Auslöser verbinden, wenn dieser beispielsweise nur von dieser einen Einheit bzw. diesem einen Objekt ausgelöst werden soll.

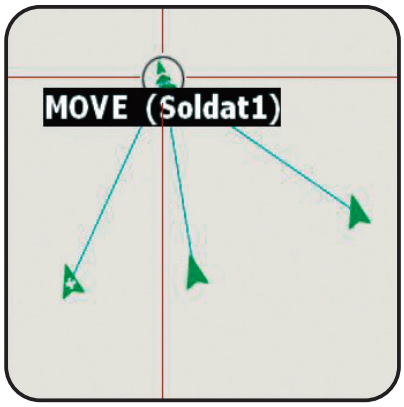
Auf dem linken Bild sieht man, wie eine Einheit gerade mit einer Gruppe verbunden wird und auf dem Rechten, wie ein Auslöser objektbezogen angelegt wurde.

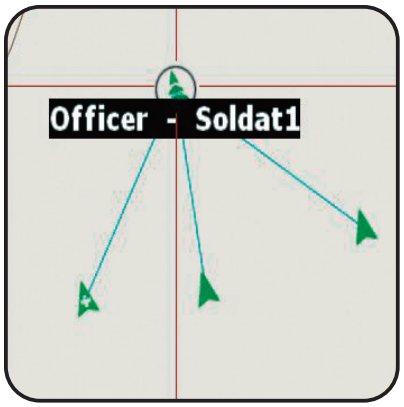


1.10 - Einheiten mit Wegpunkten bearbeiten

Wenn man eine Einheit bereits mit einem Wegpunkt versehen hat und möchte diese im Nachhinein bearbeiten, wird man schnell feststellen, dass sich stets das Wegpunktmenü öffnet, aber nicht das Einheitsmenü.

Dazu fährt man einfach mit dem Fadenkreuz auf die Einheit, drückt die **Shift-Taste** und hält diese gedrückt. Jetzt sieht man, dass sich die Bezeichnung ändert. Klickt man nun doppelt auf die Einheit, wird sich das Einheitsmenü öffnen.





Kapitel 2
- Die Dateien -

Nachdem du im Kapitel 1 in die Oberfläche eingeführt wurdest, wird dich dieses Kapitel eine Ebene tiefer, nämlich zum Standgerüst einer Mission führen. Hier werden dir die einzelnen Dateien erklärt, die für eine Mission wichtig sind. In ihnen werden unter anderem wichtige Informationen gespeichert und konfiguriert.

Legislator (Daniel Bötticher)

2.1 - Der Missionsordner

In einem Missionsordner werden alle Dateien abgelegt, die für eine Mission benötigt werden. Es bietet sich hierfür an viele Ordner anzulegen, um dadurch eine bessere Übersicht zu gewährleisten. Um dies zu erreichen, sollte man für jeden der Dateitypen, also Musik, Sounds, Skripte, Szenen, Bilder usw., einen extra Ordner anlegen. Wichtig ist auch, dass man alle Dateinamen klein schreibt. Ein fertiger Missionsordner schaut dann etwa so aus:

Die hier zu lesenden Dateinamen sind vorgegeben und nicht variabel! Lediglich das Bild **Title.paa** und die Ordner könnten anders benannt werden. Hier mal die näher erläuterten Dateibedeutungen:

Mission.sqm	Missionskoordinaten
Description.ext	Missionskonfiguration (Musik, Sound, Waffen, Identitäten, Ressourcen usw.)
Stringtable.csv	Textausgabe für mehrsprachige Missionen
Briefing.html	Briefing beim Missionsstart
Overview.html	Missionsinfo in Missionsauswahlmenü
Titel.paa	Overview Bild

Die verschiedenen Briefingdateien stellen das Briefing in verschiedenen Sprachen dar. Wenn man seine Mission natürlich nur mit einer Sprache ausstatten will, genügt die **Briefing.html**, der Rest der HTML-Dateien kann dann gelöscht werden. Bei nur einer Sprache ist es egal, ob man in der Briefing.html deutsch, englisch oder sonstiges verwendet. Mehr aber im Kapitel 2.13 im Unterbereich „Das Briefing“.

Music	Für Musikdateien
Sound	Für Soundateien (z.B. Sprache oder Effekte)
Scripts	Für die Skripte (.sqs)
Scenes	Für die Zwischensequenzen (z.B. Intro, Outro usw.)
Function	Für die Funktionen (.sqf)
Pictures	Für die Bilder (z.B. Titel.paa)

Möchte man eine Datei aus einem Unterordner aufrufen, gibt man in der Syntax den jeweiligen Ordner, dann Backslash und die jeweilige Datei an. Zum Beispiel das Aufrufen eines Skriptes:

[] exec “scripts\script.sqs“

2.2 - Die Mission.sqm

Die Mission.sqm ist die wichtigste von allen Dateien, denn darin sind alle wichtigen Informationen und Koordinaten einer Mission gespeichert. Darunter sind unter anderem Informationen der im Editor gesetzten Einheiten, Objekte, Marker, Wegpunkte, Auslöser usw. enthalten. Des Weiteren findet man am Anfangsbereich noch diverse andere Informationen, wie verwendete Addons oder unter Info den Missionsnamen, Wetter und Uhrzeit.

Die Erläuterungen in diesem Unterkapitel könnten ein wenig schwieriger sein, aber sollen keinesfalls vom Editieren abschrecken. Man muss diese Datei nicht unbedingt auswendig beherrschen, aber ein paar Kenntnisse erleichtern das Editieren in einigen Bereichen ungemein.

Der Anfangsbereich

Auf der Folgeseite sieht man den Anfangsbereich einer Mission.sqm im Bereich Class Mission. Zunächst werden die verwendeten Addons gelistet. Hier sind nur original ArmA-Addons zu sehen.

Achtung! Lädt man ein externes Addon, das eine schlecht gemachte Config.ccp hat, beim Missionsstart mit, kann es vorkommen, dass sich dieses Addon einfach mit in die Liste einträgt, obwohl es in der Mission nicht verwendet wird. Man kann es dann einfach markieren und herauslöschen.

Das Problem kommt dann, wenn ein anderer Spieler diese Mission spielen möchte und das Addon nicht hat. Die Mission kann dann nicht geladen werden und der Download war umsonst. Dieses Problem gab es gerade in Operation Flashpoint sehr oft. Spieler, die keine Ahnung vom Editing haben, werden so eine Mission schnell beiseite legen und sich eine andere runterladen.

Nach den Addons folgend kommt man zur **Class Intel**, in welcher unter anderem

- **Briefingname**
- **Widerstandseinstellung**
- **Startwetter**
- **Späteres Wetter**
- **Späterer Nebel**
- **Sichtweite**
- **Datum**
- **Uhrzeit**

definiert sind.

```
addOns[]=
        {
                "cacharacters",
                "sara",
        };
        addOnsAuto[]=
        {
                "cacharacters",
                "sara"
        };
        randomSeed=8635907;
        class Intel
        {
                briefingName="@STR_M11_Name";
                resistanceWest=0.000000;
                startWeather=0.000000;
                forecastWeather=0.000000;
                forecastFog=0.375187;
                viewDistance=1000.000000;
                month=6;
                day=2;
                hour=3;
                minute=50;
        };
```

Neben der **Class Mission** gibt es noch die **Class Intro**, **Class OutroWin** und **OutroLoose**, die vom Aufbau her genauso sind, wie die **Class Mission**. Hier sind jedoch die Einheiten, Wegpunkte usw. für die jeweilige Sequenz definiert.

Es ist möglich Änderungen direkt in der Mission.sqm vorzunehmen. Dabei ist es wichtig, dass man die Änderungen richtig vornimmt. Wenn man das Ergebnis danach im Editor sehen will, muss man die Mission dort erst neu laden.

Sollte das Spiel beim Laden komplett abstürzen, ist wohl ein Fehler unterlaufen. Deshalb bietet es sich an vorher immer eine Kopie der Mission.sqm zu machen, um später eine funktionierende Datei zu haben.

Einheits- und Objektklassen

Nach dem Anfangsbereich kommt man in der Class Mission in den Unterbereich **Class Groups** und damit in den Unterbereich **Class Vehicles** in welcher die Einheiten, Objekte und die dazugehörigen Wegpunkte definiert sind.

```
class Groups
        {
        items=22;
        class Item0
        {
                side="WEST";
                class Vehicles
                {
                        items=1;
                        class Item0
                        {
                                position[]={7973.895020,4.460081,9351.659180};
                                id=0;
                                side="WEST";
                                vehicle="SoldierWB";
                                player="PLAYER COMMANDER";
                                leader=1;
                                rank="CORPORAL";
                                skill=0.200000;
                                text="aP";
                                init="this addWeapon ""binocular"";
                        };
                };
        };
```

Wie man sieht, sind hier sämtliche Informationen für diese Einheit namens **S1** definiert. Hier die Erläuterung der Begriffe:

Items=1	Zeigt die Anzahl der Items der **Class Groups** an. Also die Anzahl der Gesamtgruppen aller Seiten auf einer Karte
Class Item0	Ist die Gruppe **0**. Die nächste Gruppe hieße **Class Item1**
Side	Die Seite der jeweiligen Gruppe. Auch eine einzelne Einheit wird als Gruppe definiert und angegeben!
Class vehicles	Sagt aus, dass es sich um ein Vehikel handelt
Items=1	Die Anzahl der Items(Einheiten) der **Gruppe Class Item0** an

Class Item0	**Class Item0** ist Leader der **Gruppe Class Item0**. Der dem Leader unterstellte Soldat ist **Class Item1**, der nächste **Class Item2** usw.
Presence	Wahrscheinlichkeit der Anwesenheit (Nicht bei Spieler!)
Position	YXZ-Koordinaten des Spielers in der Anordnung X Z Y
Azimut	Blickrichtung der Einheit (Wert definierbar von **0** bis **360**)
ID	ID der Einheit
Side	Spielerseite
Vehicle	Art der Einheit
Player	Spieler
Leader	Gibt an, ob die Einheit ein Leader ist.
Skill	Fähigkeit der Einheit (Wert definierbar von **0** bis **1**)
Health	Gesundheitsstatus (Wert definierbar von **0** bis **1**)
Ammo	Munitionsstatus (Wert definierbar von **0** bis **1**)
Text	Name der Einheit (Variable)
Init	Die Initzeile der Einheit (Angabe einer Syntax etc.)

Wegpunkt-Klassen

Unter der jeweiligen Gruppe sind die zu der Gruppe gehörigen Wegpunkte angeordnet. Diese sind von der Anordnung her den Einheiten gleich, aber eben von der Zusammenstellung etwas anders.

```
class Waypoints
        {
        items=1;
        class Item0
        {
                position[]={7970.289551,4.731988,9346.483398};
                placement=50.000000;
                CombatMode="RED";
                Speed="FULL";
                combat="COMBAT";
                description="Hold this position!";
                expActiv="[] exec""scripts\script.sqs";
                class Effects
                {
                timeoutMin=10.000000;
                timeoutMid=3.000000;
                timeoutMax=30.000000:
                };
                showWP="NEVER";
        };
        };
```

Wie man sieht, sind hier sämtliche Informationen für diesen Wegpunkt der Gruppe definiert. Der nächste Wegpunkt schaut, je nach Einstellung wieder ganz anders aus. Hier die dazugehörigen Erläuterungen:

Items=1	Zeigt die Anzahl der Items der **Class Waypoints** an. Also die Anzahl der gesamten Wegpunkte der Gruppe.

Class Item0	**Class Item0** ist der erste Wegpunkt. Der zweite würde dann **Class Item1**, der übernächste **Class Item2** usw. heißen.
Position	YXZ-Koordinaten des Wegpunktes, in der Anordnung XZY.
Placement	Ist der zufallsbedingte Platzierungsradius des Wegpunktes.
CombatMode	Der jeweilige Kampfmodus der Gruppe ab diesem Wegpunkt.
Formation	Die jeweilige Formation der Gruppe ab diesem Wegpunkt.
Speed	Die Geschwindigkeit der Gruppe ab diesem Wegpunkt.
Combat	Das jeweilige Verhalten der Gruppe ab diesem Wegpunkt.
Description	Die Beschreibung des Wegpunktes, welche man, wenn der Wegpunkt im Spiel angezeigt wird, lesen kann.
ExpActiv	Die Aktivierungszeile des Wegpunktes, die bei Auslösung des Wegpunktes ausgelöst wird. Hier wird ein Skript mit Namen **Script.sqs** aus dem Ordner **Scripts** aktiviert.
TimeOutMin	Die Mindestauslösezeit des Wegpunktes
TimeOutMid	Der Mittelwert der Auslösezeit des Wegpunktes
TimeOutMax	Die maximalste Auslösezeit des Wegpunktes
ShowWP	Gibt an, ob der der Wegpunkt im Spiel angezeigt wird oder nicht.

Bei diesem Wegpunkt wurden keine Effekte definiert. Diese sind dann bei den Auslöserklassen mit erklärt.

Marker-Klassen

Nach den Gruppen und den dazugehörigen Wegpunkten folgt die Klasse der Marker. Ab hier werden sämtliche Marker aufgelistet, die in der Mission gesetzt werden. Das Ganze sieht etwa so aus:

```
class Markers
	{
		items=28;
		class Item0
		{
			position[]={2452.061035,0.760200,3673.595703};
			name="TargetOne";
			text="Objective Alpha";
			type="Flag";
			a=2.000000
			b=2.000000
			angle=0.100000
		};
	};
```

Hier die Beschreibung der einzelnen Punkte:

Items=1	Zeigt die Anzahl der Items der **Class Markers** an. Also die Anzahl der gesamten Marker in der Mission
Class Item0	**Class Item0** ist der erste Marker. Der zweite würde dann **Class Item1**, der übernächste **Class Item2** usw. heißen.
Position	YXZ-Koordinaten des Markers, in der Anordnung XZY.

Name	Der Name des Markers.
Text	Die Beschreibung des Markers, die man später auf der Karte liest
Type	Die Art des Markers. Hier ein Fadenkreuz.
a	Die Größe des Markers in X-Richtung
b	Die Größe des Markers in Y-Richtung
Angle	Der Winkel des Markers.

Auslöser-Klassen

Diese sind von der Sache her ähnlich wie ein Wegpunkt zu betrachten und auch die Anordnung ist, wie man sehen kann, fast gleich. Hier sind auch mal Effekte mit aktiviert worden, um diese mal in der Mission.sqm sehen zu können.

```
class Sensors
        {
                items=53;
                class Item0
                {
                        position[]={8012.703613,6.300000,9301.049805};
                        a=100.000000;
                        b=100.000000;
                        activationBy="WEST";
                        timeoutMin=10.000000;
                        timeoutMid=3.000000;
                        timeoutMax=30.000000:
                        age="UNKNOWN";
                        name="DetectorOne";
                        expCond="Var1";
                        expActiv="[] exec ""scripts\script.sqs""";
                        expDesactiv="[] exec ""scripts\animation-end.sqs""";
                };
        };
```

Items=1	Zeigt die Anzahl der Items der **Class Sensors** an. Also die Anzahl der gesamten Auslöser auf der Karte.
Class Item0	**Class Item0** ist der erste Auslöser. Der zweite würde dann **Class Item1**, der übernächste **Class Item2** heißen usw.
a	Die Größe des Auslösers in X-Richtung
b	Die Größe des Auslösers in Y-Richtung
ActivationBy	Aktivierung durch „WEST“
TimeOutMin	Die mindest Auslösezeit des Auslösers
TimeOutMid	Der Mittelwert der Auslösezeit des Auslösers
TimeOutMax	Die maximalste Auslösezeit des Auslösers
Age	Unknown
Name	Der Name des Auslösers
ExpCond	Die Bedingung der Auslösung. Hier die Variable **Var1**.
ExpActiv	Die Aktivierungszeile des Auslösers, die bei der Auslösung des Auslösers ausgelöst wird. Hier wird ein Skript mit Namen **Skript.sqs** aus dem Ordner **Scripts** aktiviert.
ExpDesactiv	Die Deaktivierungszeile des Auslösers. Hier kann man den Auslöser wieder deaktivieren.

2.3 - Die Description.ext

Die Description.ext ist für unsere Mission genauso wichtig wie die Mission.sqm. In ihr werden zwar nicht die Einheiten gesetzt, aber eine Menge wichtiger Dinge definiert. Diese ist zuständig für das Vordefinieren von Musik, Sounds, Respawn, Ressourcen, das Auswählen von Waffen im Briefing, den Accessoires (Kompass usw.) und diversen anderen Dingen, die man zum Spielen benötigt.

Die Description.ext wird in dem Missionsordner der jeweiligen Mission angelegt. Dazu erstellt man eine neue Textdatei und benennt diese in Description.ext um. Ganz wichtig hierbei ist, dass man diese mit einem **Texteditor (z.B. Notepad)** bearbeitet und nicht mit Word oder Excel!

Hier wird die Description.ext aus Übersichtsgründen nur grob erklärt. Zu den Einzelheiten siehe in den jeweiligen Kapiteln nach, darin sind die Unterbereiche sehr ausführlich erläutert.

Missionsstarttext	**Kapitel 4.2**
Missionswertung	**Kapitel 4.4**
Identitäten	**Kapitel 5.53**
Musik	**Kapitel 5.52**
Sound	**Kapitel 5.52**
Respawn	**Kapitel 7.2**
Waffenauswahl im Briefing	**Kapitel 3.6**

Es ist, je nach Mission, nicht notwendig alle Bereiche in die Datei einzubauen. Man nimmt nur das, was für die Mission benötigt wird. Damit spart man sich zum einen eine Menge Arbeit und zum anderen auch gleich mögliche Fehlerquellen, die dabei auftauchen können. Für eine Einzelspielermission ist es schließlich völlig überflüssig, wenn man den Respawn mit angibt.

Es ist auf jeden Fall darauf zu achten, dass alle Klammern **{** die geöffnet werden, auch wieder geschlossen **};** werden. Andernfalls stürzt ArmA sofort ab! Auch andere Fehler machen sich teilweise durch einen Absturz bemerkbar! Deshalb ist hier ein gewissenhaftes Arbeiten angesagt.

Möchte man Missionszubehör, wie den Kompass oder die Uhr in der Mission verbergen oder verfügbar machen, macht man das mit **1** oder **true** für **aktiv**/**sichtbar** oder mit **0** oder **false** für **inaktiv**/**unsichtbar**.

Hinter die beiden **//** oder einem Semikolon **;** hat man die Möglichkeit etwas zu schreiben, was ArmA ignoriert. Dies lediglich dient der Übersicht in Form einer Überschrift, die man bei einer großen Description schnell verlieren kann.

Hat man nun Änderungen in der Datei vorgenommen und wieder in den Editor gewechselt, muss man die Mission erst nochmal speichern oder neu laden, damit die

Änderungen wirksam werden. Sollte das Spiel dabei abstürzen, nicht gleich die Geduld verlieren und einfach kurz nach dem Fehler suchen. Deshalb ist es sinnvoll einen Abschnitt nach dem anderen zu definieren. Somit weiß man sofort, dass der Abschnitt betroffen ist, den man gerade bearbeitet hat.

Hier sieht man, wie eine **Description.ext** in etwa aussehen kann.

```
// ======================== Description.ext ======================>
Debriefing = 1;
OnloadIntro = 1;
OnLoadIntroTime = 1;
OnLoadMissionTime = 1;
Saving = 0;
// === Titlecut ================================================>
OnloadIntro= M o r p h i c o n  p r o u d l y  p r e s e n t s
onLoadMission= A R M E D  A S S A U L T
// === Punktvergabe ============================================>
minScore=200
avgScore=2500
maxScore=6000
// === Missionszubehör =========================================>
ShowCompass = 1;
ShowMap = 1;
ShowGPS = 1;
ShowWatch = 1;
// === Respawn =================================================>
respawn=3;
respawn_delay=10;
// === Waffen ==================================================>
class Weapons
{
   class M4
   {
      count = 4;
   };
   class Javelin
   {
      count = 2;
   };
};
// === Magazine ================================================>
class Magazines
{
   class 20Rnd_556x45_Stanag
   {
      count = 10;
   };
   class Javelin
   {
      count = 6;
   };
};
// === Musik ===================================================>
class CfgMusic
                {tracks[]= { Track1,Track2};
                class Track1
                {
                                name = "Track1";
                                sound[] = {\music\track1.ogg, db+0, 1.0};
                };
```

```
		class Track2
		{
			name = "Track2";
			sound[] = {\music\track2.ogg, db+0, 1.0};
		};
};
// ===  Sounds ==============================================>
class CfgSounds
{
		sounds[]= {Sound1};
		class Sound1
		{
			name = "Sound1";
			sound[] = {\sounds\sound1.ogg, db+0, 1.0};
		};
};
// ===  Radio ===============================================>
class CfgRadio
{
		sounds[] = { };
};
// ===  Umgebung ============================================>
class CfgSFX
{
		sounds[] = {};
};
class CfgEnvSounds
{
		sounds[] = {};
};
// ===  Identitäten =========================================>
class CfgIdentities
{
		class MrMurray
		{
			name = "MrMurray";
			face = "Face33";
			glasses = "none";
			speaker = "Dan";
			pitch = 1.00;
		};

		class Memphisbelle
		{
			name = "Memphisbelle";
			face = "Face10";
			glasses = "none";
			speaker = "Howard";
			pitch = 1.00;
		};

		class Dan
		{
			name = "Dan";
			face = "Face22";
			glasses = "none";
			speaker = "Russell";
			pitch = 1.00;
		};

};
// End Of File
```

2.4 - Die Stringtable.csv

Die Stringtable.csv dient zum Hinterlegen von Textvariablen. Sie ermöglicht es, verschiedene Sprachen in eine Mission einzubinden oder einfach nur Text zu hinterlegen. Diese Datei sollte dabei generell mit dem Texteditor bearbeitet werden! Windows schlägt aber standardmäßig Excel als Bearbeitungsprogramm vor.

Wenn man diese Datei nun bearbeiten will, muss man auf diverse Kleinigkeiten achten. Zunächst definiert man den Kopf mit den jeweiligen Sprachen. Dabei achte man darauf, dass diese mit Kommas getrennt werden.

LANGUAGE,English,German,Czech,Notes

Auch in der jeweiligen Zeile werden die Sprachen nun durch ein Komma getrennt und in " " gesetzt. In der Praxis sieht das dann in der jeweiligen Zeile so aus:

STR_Titel,"Night Patrol","Nacht Patrouille","...",Missionsname

Hier sieht man nun ganz gut, wie das ausschaut. Am Anfang die Adresse, mit STR_Titel, dann folgend die Sprachen und zum Schluss noch eine Beschreibung, die zur Orientierung beiträgt. STR_ wird dabei generell zuerst angegeben. Das Wort Titel dahinter ist Variable und kann daher frei definiert werden. Eine Textzeile für einen Text in einer Mission könnte dann so aussehen:

STR_Mission_1,"Hold Position!","Position halten!","...",Missionstext1

Möchte man den Text zum Beispiel beim Laden der Mission aufrufen, gibt man folgende Syntax an:

onLoadMission=$STR_Titel;

STR_Titel kann also als individuelle Adresse gesehen werden, welche man angibt, wenn man den Text in die Mission einbinden möchte. Dieser würde dann in der jeweilig ausgewählten und vordefinierten Sprache ausgegeben. Das **@** vor STR_ verwendet man im Editorbereich, also in einem Auslöser oder Wegpunkt, während man in einer Config oder der Description.ext das **$** Zeichen verwendet.

Aus dem Editor Text aufrufen:	**@STR_Titel**
Aus der Description bzw. einer Config:	**$STR_Titel**

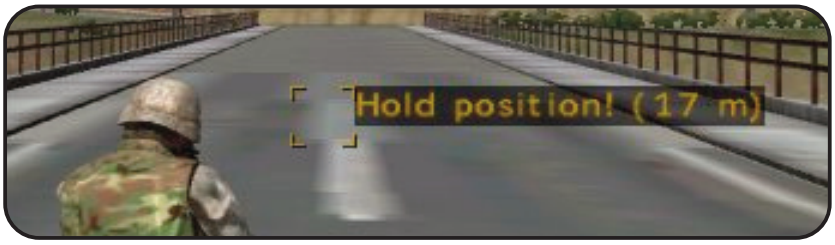
Bei dem folgenden Bild, wurde ein Wegpunkt mit Text versehen, welcher direkt aus der Stringtable bezogen wird, welche dafür wie folgt definiert wurde:

STR_Mission_1,"Hold Position!","Position halten!","...",Missionstext1

Bei dem Wegpunkt wurde dabei folgendes in die Description-Zeile geschrieben:

@STR_ Mission_1

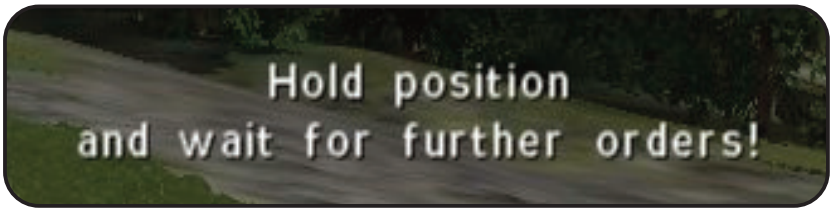
Wie man hier sehen kann, wird der Text dann direkt so in die Mission übergeben.



Natürlich ist es hierbei auch möglich in einem Satz oder Text einen Zeilenumbruch anzugeben, was gleich etwas besser aussieht. Dazu verwendet man lediglich ein **\n**

STR_Mission_1,"Hold Position\nand wait for orders!","","...",Missionstext1

Hier wurde eine Texteinblendung verwendet, was in der Mission dann so aussieht:



Hier nochmal ein kurzes Dateibeispiel einer Stringtable aus einer originalen Armed Assault Mission, wobei man hier sehr gut erkennen kann, dass darin nur eine Sprache definiert wurde.

```
LANGUAGE,English,Notes

STR_M11_Name,"Night Patrol",Mission Name
STR_M11_OnLoad,"You're on duty tonight",Onload
STR_M12_OnLoad,"Don´tsleep and keep your eyes open!",Onload

COMMENT, ---------- Main Mission ----------

STRCAMP_OBJSTART,Guard the military installation,
STRM_07an01,"Southern sector, Sahrani",prebriefing
STRM_07an02,"NATO base in La Riviere, Sahrani",prebriefing
STRM_07an03,"Near Paraiso - One hour later, Sahrani",prebriefing
```

2.5 - Die Init.sqs

Die Init.sqs kann sozusagen als Initialisierungsdatei angesehen werden, welche beim Missionsstart automatisch abgerufen wird und nicht gesondert gestartet werden muss. Diese Datei bringt eine Menge Übersicht mit sich, da man nun nicht alle Befehle in die Initzeile einer Einheit oder des Spielers schreiben muss, welche dann, je nach Menge, sehr unübersichtlicht werden würde. Denn gerade beim Spieler sammeln sich oftmals eine Menge Befehle an. Aber was schreibt man denn nun alles dieses Skript? Eigentlich alles, was beim Missionsstart sofort ausgeführt, also initialisiert werden soll.

Wie man sieht, sind im folgenden Beispiel nun ein GPS-System, die Spielbeschleunigung und die Verdeckung der Missionsziele 1-3 vordefiniert. Zudem wird hier noch zum Editieren die Teleport.sqs gestartet, die uns das Editieren und Testen erleichtern soll. Diese sollte logischerweise dann später deaktiviert oder entfernt werden.

```
;Einblendung
titleCut [" ", "BLACK IN"]; titleFadeOut 4

;Funktion vorladen
SearchLight = compile preprocessFile "Searchlight.sqf";

;Identität zuweisen
Player setIdentity "Mr-Murray";

;Missionsziele werden verdeckt
"MZiel1" ObjStatus "Hidden";
"MZiel2" ObjStatus "Hidden";
"MZiel3" ObjStatus "Hidden";

;GPS-System
[] exec "marker.sqs";

;Spielbeschleunigung unterdrücken
[] exec "time.sqs";

;Editierskript. Wichtig! Nur für die Editierzeit, später entfernen bzw. deaktivieren!!!
;Teleport
[] exec "teleport.sqs";
;Ende
```

Natürlich kann man hier noch weit mehr vordefinieren, als oben angegeben. Zum Beispiel das Verhalten von Einheiten, die Bewaffnung, Variablen, Einheitsstatus löschen, Funktionen laden und so weiter. Diese Datei sollte nur mal als Beispiel dienen. Alle hier angegebenen Skript oder ähnliches müssen natürlich extra geschrieben werden bzw. die Missionsziele auch so vordefiniert und benannt werden.

Wie man außerdem sieht, ist hier alles sehr übersichtlich gehalten. Dabei wird alles, was hinter einem Semikolon steht, von ArmA ignoriert und dient hier als Beschreibung um die Übersicht zu behalten.

2.6 - Das Skript

Ein Skript ist lediglich eine Datei im Missionsordner, welches für bestimmte Zwecke geschrieben ist. Dieser Abschnitt erklärt nicht das Skripten selbst, sondern beschreibt lediglich die Datei und wie man diese aufruft. Mehr zum Thema Scripting findet sich im **Kapitel 9**.

Die Skripte in Armed Assault enden, wie auch schon bei Operation Flashpoint, mit der Endung **.sqs**. Zum Erstellen eines Skriptes erstellt man zunächst eine Textdatei, die man dann einfach umbenennt. Windows wird die Datei dann als unbekanntes Format deklarieren. Dem entgegnet man dann mit der Windowsmöglichkeit Rechtsklick und anschließend „**Öffnen mit**“ und wählt dort als Standardprogramm den **Texteditor** aus.

Im Laufe der nächsten Kapitel wird man noch mit dem einen oder anderen Skript vertraut gemacht und wird verstehen um was es dabei geht. In einem Skript kann man sich dann relativ frei austoben, obwohl es natürlich auch hier Grenzen gibt.

Die **Init.sqs** als solches ist ja schon ein Skript, welches ohne Verwendung einer Syntax automatisch beim Missionsstart von ArmA aufgerufen wird. Auch die Mission.sqm oder die Description.ext sind ja sozusagen auch irgendwo Skripte, nur halt mit einer anderen Endung und Funktion.

Möchte man nun ein Skript per Auslöser, Wegpunkt oder aus einem anderen Skript aufrufen, benötigt man diese Syntax:

[] exec "scripte\meinskript.sqs"

oder

this exec "scripte\meinskript.sqs"

Nachdem das Skript aufgerufen wurde, wird es seine Befehle abarbeiten und dann je nach Funktion und Ablauf dann am Skriptende, wo **exit** stehen sollte, beendet.

2.7 - Die Funktion

Die Funktion kann man von der Sache her mit einem Skript vergleichen. In beiden sind Befehle hinterlegt, aber es gibt doch kleine aber feine Unterschiede. Genauso wie ein Porsche und ein Käfer. Beides sind zunächst Autos, aber wenn man ins Detail geht, so wird man schnell feststellen, dass der Porsche dem Fortschritt nach weiter ist.

Der Unterschied besteht im Wesentlichen darin, dass SQS-Skripte jedes Mal gelesen werden, wenn sie aufgerufen werden und die SQF-Funktionen nur einmal in den Speicher geladen werden. Während bei Operation Flashpoint noch größtenteils auf SQS-Basis gearbeitet wurde, wird bei Armed Assault empfohlen mehr auf SQF-Basis zu arbeiten.

Anwendungsmäßig sind Funktionen so zu benutzen, dass sie eine bestimmte Aufgabe lösen und dies mit einem so kurzen und bündigen Schreibaufwand wie möglich. Hinzu kommt, dass man diese Funktion dann auch in anderen Missionen jederzeit verwenden kann, ohne sie umschreiben zu müssen. Man schreibt sie lediglich ein einziges Mal zum Erfüllen eines ganz bestimmten Zweckes. Beispielsweise ist es möglich mit Hilfe einer Funktion einen Vektor zu berechnen. Oder auch viele andere Bereiche lassen sich mit Funktionen wesentlich einfacher realisieren. Es ist besser viele kleine Funktionen zu haben, als nur ein langes Skript. Hierbei geht es weniger um die Performance. Wichtiger ist die Wiederverwertbarkeit dieser Funktion.

2.8 - Das Paa-Format

Das Paa-Format ist lediglich eine Bilddatei, wie auch das Jpg-Format. In Armed Assault sind größtenteils alle Grafiken im Paa- oder Pac-Format vorzufinden. Hierzu gehören selbstverständlich auch die Texturen sämtliche Objekte in Armed Assault.

Im Unterkapitel „Der Missionsordner" dieses Kapitels ist eine Grafik zu sehen, auf der eine Datei namens **Title.paa** zu sehen ist. Diese Grafik ist für das Overview vorgesehen und ist in der Overview.html so definiert. Dieses Bild sieht man, wenn man im Einzelmissionsmenü eine Mission zum Spielen auswählt.

Der Name für dieses Bild ist aber variabel und müsste dann dementsprechend auch so in der Overview.html angegeben werden. Mehr dazu aber in diesem **Kapitel 2.12** im Unterbereich „Der Overview". Arma unterstützt, wie auch schon Operation Flashpoint, das Jpg-Format. Man kann also auch Bilder in diesem Format einbinden. Ganz wichtig ist es darauf zu achten, dass die angegebene Bildergröße passt. ArmA nimmt teilweise nur Grafiken an, die einen Zweierpotenzwert haben. Aber in manchen Bereichen gibt es auch Ausnahmen, wie z.B. im Briefing oder Overview. Zweierpotenzen sind: 2,4,8,16,32,64,128,256,... usw. Als Beispiel mal diese Formate:

64x64 **128x128** **128x64** **256x256**

Zum Betrachten und zum Bearbeiten von Paa- bzw. Pac-Dateien verweise ich auf die Armed Assault Fanseiten, auf denen man die entsprechenden Tools finden wird.

2.9 - Die PBO

Die PBO ist eine gepackte Datei in der alle Dateien, die sich vorher beim Editieren einer Mission im Missionsordner angesammelt haben, verpackt sind. Man könnte sie also mit einer Rar- oder Zip-Datei vergleichen, nur dass die gepackten Dateien von der Größe her nicht verkleinert werden.

Die PBO entsteht, wenn man seine fertige Mission im Editor als Single- oder Multiplayermission abspeichert. Während **Singleplayermissionen** im Verzeichnis **ArmA/Missions** hinterlegt werden, sind die **Multiplayermissionen** im Verzeichnis **ArmA/MPMissions** zu finden.

PBO´s beschränken sich natürlich nicht nur auf die Missionen. Auch sämtliche Addons von Armed Assault sind, wie auch schon in Operation Flashpoint, in PBO´s gepackt. Diese Dateien lassen sich mit Hilfe von diversen Tools packen und auch wieder entpacken. So dass man beispielsweise schon vorhandene Missionen oder Addons entpacken und aus ihnen lernen bzw. Skripte oder ähnliches verwenden kann. Für die Tools verweise ich wieder auf die Armed Assault Fanseiten.

2.10 - Die Sounddateien

Der überwiegende Teil der Sounddateien in Armed Assault ist im Wss-Format oder Ogg-Format konvertiert. Es ist aber auch möglich Wavedateien zu verwenden. Jedoch sollte man hier auf die Größe der Datei achten, da man ja die Mission auch irgendwann zum Download anbieten möchte.

Das Wss- oder Ogg-Format ist hierfür genau das richtige, weil die Dateien je nach Soundlänge eine minimale Größe haben. Dies ermöglicht die Verwendung von vielen Sounds in einer Mission, ohne dass diese am Ende eine überdimensionale Dateigröße hat. Eine nähere Erklärung zum Einbinden von Sounds findet man im **Kapitel 5.52** im Bereich -Eigenen Sound einbinden-.

Sounddateien, bis auf Musik, sollten Mono mit einer Frequenz von 44.100 kHz konvertiert sein. Mono daher, damit man den Entfernungseffekt nutzen kann. Möchte man nämlich ein Objekt oder eine Einheit mit

Name say "Soundname"

etwas sagen bzw. erklingen lassen, würde man es auf der ganzen Karte hören, was ja unlogisch wäre. Deshalb Mono! Sind die Dateien also Stereo, erklingen die Sounds global.

Die Konvertierungsprogramme für diese Dateien findet man auf den Armed Assault Fanseiten oder auch sonst bei anderen Anbietern im Internet.

2.11 - Die LIP-Dateien

Die Lip-Datei ist für die Bewegung der Lippen zuständig. Jede Sounddatei, die für die Sprachausgabe ausgelegt ist, kann mit einer Lip-Datei versehen werden, damit sich die Lippen der jeweiligen Einheit beim Sprechen auch bewegen.

Hierbei gibt es lediglich 3 Werte(1,2,3), die man dafür benötigt. Zuerst legt man die Framerate der einzelnen Bewegungsbilder, hier mit dem Wert **0.040**, fest. Man kann diesen Wert quasi als Zeitabschnitt bezeichnen. Alle 0.040 Sekunden bzw. Frames ändert sich nun die Mundbewegung. Nach der Festlegung der Framezeit beginnt der eigentliche Bewegungsablauf der Lippen, welche die Öffnungsgrade **0** bis **3** haben. Wobei der Wert **0** für geschlossenen und **3** für weit geöffneten Mund steht.

Hier wurde eine Sounddatei verwendet, die genau eine Sekunde lang ist. Teilt man 1 Sekunde durch 0.040 so erhält man 25. Hier sind aber gar keine 25 Zeilen enthalten, sondern nur 20. Das liegt daran, dass man es auch so schreiben kann:

Zum Beispiel von Zeit **0.560** setzt man den Lippenwert **1** und pausiert bis Zeit **0.720** und setzt dann den Lippenwert **2**. Hier wurde also eine **4** Frame lange Bewegungspause der Lippen einsetzt.

```
frame = 0.040
0.000, 0
0.040, 1
0.080, 2
0.120, 3
0.160, 2
0.200, 1
0.240, 0
0.280, 1
0.320, 2
0.360, 1

          weiter -->
```

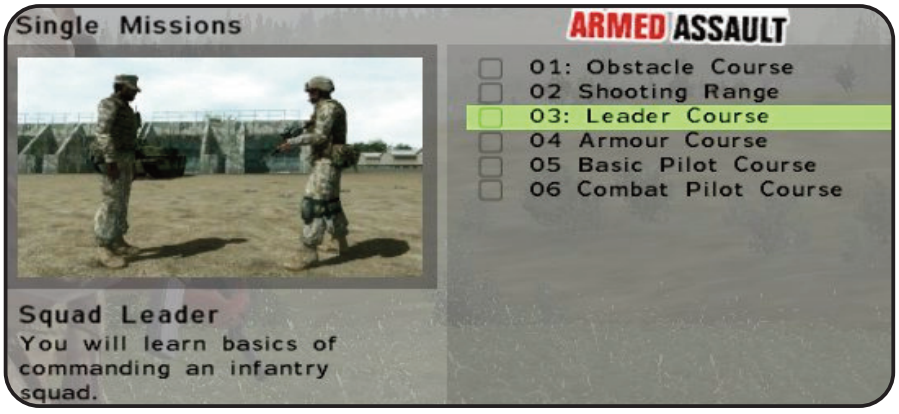
```
0.400, 0
0.440, 1
0.480, 3
0.520, 0
0.560, 1
          4 Frame-Pausen
0.720, 2
          2 Frame-Pausen
0.800, 1
0.840, 2
0.920, 1
1.000, 2
```

Dieses Beispiel ist ja nur für eine Sekunde. Man kann sich dann schon vorstellen, wie lang eine 10 oder 15 Sekunden lange Lip-Datei aussehen kann.

Damit man das aber alles nicht per Hand machen muss, gibt es Tools, die für die Erstellung solcher Dateien konzipiert sind. Hier verweise ich mal wieder auf die Armed Assault Fanseiten, auf denen man die entsprechenden Tools finden und runterladen kann.

2.12 - Das Overview

Das Overview ist der Abschnitt, den man bei der Einzelspielermissionsauswahl als kurzen Infoteil zur Mission sehen kann. Dieser wird in der Overview.html zusammen mit einem Bild erstellt. Im Spiel sieht das dann etwa wie auf dem Bild aus.



Rechts ist die Missionsauswahl und links die Info mit Bild zu der Mission zu sehen.

Zum Erstellen eines Overviews wären gewisse Html-Kenntnisse von Vorteil, aber auch wer sich damit nicht auskennt, bekommt das ohne weiteres hin. Am besten ist es immer, wenn man sich die Overview aus einer anderen Mission herauskopiert, einfach umschreibt und mit einen eigenem Bild versieht.

Zu Beginn erstellt man zunächst einmal eine Textdatei, welche man später, wenn man alles fertig konfiguriert hat, dann in Overview.html umbenennt. Diese Textdatei öffnet man dann und trägt dort etwa folgenden Quelltext ein, welcher aus einer original Armed Assault Mission stammt.

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<meta name="GENERATOR" content="VB">
<title>Overview</title>
</head>
<body bgcolor="#FFFFFF">
<br>
 <!--Nachtwache-->
<br>
<p align="center"><img src="Title.paa" width="230" height="115"></p>
<BR>
<p>
<! ---Missionsinfo>
Wieder steht eine langweilige Nachtwache an. Es ist eine warme Nacht…
<! ---Ende Missionsinfo>
</p></body>
</html>
```

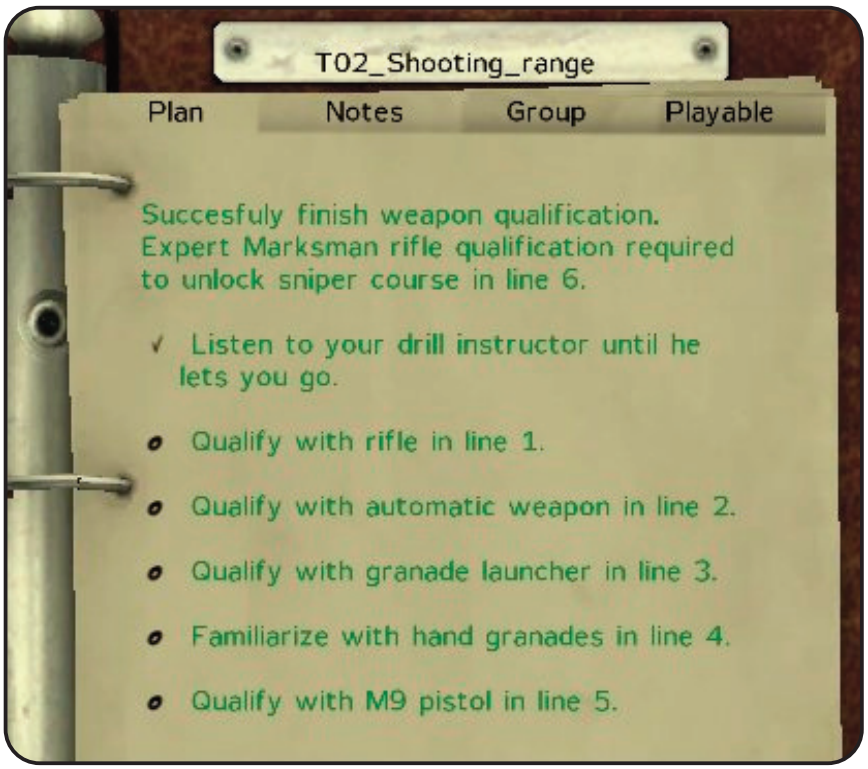
2.13 - Das Briefing

Die Briefing.html, die das Briefing direkt zur jeweiligen Mission darstellt, ist schon etwas umfangreicher als die Overview.html zu schreiben. Sie umfasst wesentlich mehr Text in Form von Informationen zur Mission und die dazugehörigen Missionsziele.

Im folgend zu sehenden Briefingausschnitt sieht man ganz gut den Missionsinfotext und darunter die Missionsziele, von welchen das Erste bereits abgehakt ist und die anderen noch zu erfüllen sind. Mehr Informationen zu den Missionszielen gibt es im **Kapitel 4.5**. Hier geht es zunächst erstmal darum eine Briefing.html zu erstellen und diese zu verstehen.

Der einfachste Weg hierzu ist, wenn man sich eine Briefing.html aus einer schon bestehenden Mission herauskopiert und diese an seine eigenen Bedürfnisse bzw. Mission anpasst. Ein weiter einfacher Weg wäre, eines der vielen Briefingtools, die es auf den Armed Assault Fanseiten zum Download gibt, zu verwenden.

Möchte man ein vorhandenes Briefing ändern, klickt man mit Rechtsklick auf die Datei und wählt "Öffnen mit" und öffnet die HTML-Datei dann ganz einfach mit dem Texteditor.



Trotz allem folgt hier jetzt der trockene Weg zum Erstellen eines Briefings. Im Briefing selbst gibt es ja, wie man oben sieht, den Abschnitt Plan und den Abschnitt Notes. Beide stellen zwei verschiedene Seiten dar, die aber in einer Datei verarbeitet werden.

Zum Erstellen eines Briefings wird zunächst wieder eine ganz normale Textdatei benötigt. Diese wird zunächst wieder normal als Textdatei geöffnet und in ihr der Quelltext nach den eigenen Bedürfnissen verfasst. Nach Abschluss der Arbeit, wird die Datei geschlossen und in Briefing.html umbenannt.

Wie man im **Kapitel 2.1** -Der Missionsordner- sehen kann, gibt es mehrere Briefingdateien, welche in den jeweiligen Sprachen verfasst sind. Normalerweise ist in der normal benannten Briefing.html alles in Englisch geschrieben. Aber wenn man nur eine Sprache für seine Mission vordefiniert, ist es egal welche Sprache darin vorkommt. Möchte man beispielsweise mehrere Sprachen einbinden, so sollte man sich schon an die Regel halten und die Dateien dementsprechend benennen.

Briefing.German.html
Overview.German.html
Briefing.France.html
Overview.France.html

Das englische Briefing und Overview würde in diesem Fall einfach nur wie folgt benannt:

Briefing.html
Overview.html

Auf den folgenden zwei Seiten ist nun das passende Briefing zum Briefingbild auf der Vorseite zu sehen. Es wurden lediglich die Notizen hinzugefügt, die im Bild als Notes benannt sind. Mit ein wenig Geduld, Ideenreichtum und Kreativität kann jeder mit ein bisschen Übung ein gut ausgefeiltes Briefing erstellen.

So kann ein Briefing einer Mission später im Quelltext etwa aussehen:

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<meta name="GENERATOR" content="vb">
<title>Briefing</title>
</head>
<body bgcolor="#FFFFFF">
<h2>
<a name="Main"></a>
</h2>
<! --- Die Notizen – ab hier werden die Notizen(Notes) eingetragen.>
<h6>
Damn that's my first shooting lesson.
<br>
</h6>
<! --- Ende der Notizen>
<hr>
<! --- Der Missionsplan – Ab hier wird die Missionsbeschreibung angegeben.>
<p><a name="Plan"></a>
Successfully finish weapon qualification.<br>
```

```
Expert Marksman rifle qualification required to unlock sniper course in line 6.
</p>
<hr>
<! --- Die Missionsziele – Ab hier erfolgt die Eintragung der Missionsziele>
<p><a name="OBJ_1"></a> Listen to your drill instructor until he lets you go.
</p><hr>
<p><a name="OBJ_2"></a> Qualify with rifle in line 1.
</p><hr>
<p><a name="OBJ_3"></a> Qualify with automatic weapon in line 2.
</p><hr>
<p><a name="OBJ_4"></a> Qualify with granade launcher in line 3.
</p><hr>
<p><a name="OBJ_5"></a> Familiarize with hand granades in line 4.
</p><hr>
<p><a name="OBJ_6"></a> Qualify with M9 pistol in line 5.
</p><hr>

<! --- Ende des Missionsplans>
<! --- Abschlussbriefing – Festlegung des Abschlussbriefings>
<hr><br>
<h2><p><a name="Debriefing:End1">Qualified</a></p></h2>
<br><p>
Now I'm a qualified infantryman.
</p><br>
<hr><br>
<h2><p><a name="Debriefing:End2">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End3">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End4">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End5">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End6">Give UP</a></p></h2>
<br><p>
I gave it up. The infantry training is boring.
</p><br>
<! --- END debriefing --->
</body>
</html>
```

Mit **<html>** und **<head>** wird die Html-Datei zunächst eröffnet. Danach folgen einige hier jetzt nicht so wichtige Standartzeilen. Die Hintergrundfarbe wird mit **<body bgcolor="#FFFFFF">** definiert, obwohl die bei Armed Assault ohnehin fest ist. Viel wichtiger ist zum Beispiel **
, was für einen Zeilenumbruch steht. Hinzu kommen noch Befehle wie **<hr> für eine horizontale Linie, welche man aber nicht sehen kann, dann das **<p>**, welches für einen Absatz steht und zu guter letzt das **<a>**, welches einen Link hervorhebt. Mann hat ja die Möglichkeit Wörter mit Marker zu verbinden, wo das Fadenkreuz dann hinwandert, wenn man diesen Link im Briefing anklickt.

Folgend ein kurzes Beispiel:
Hat man im Briefing beispielsweise einen Marker mit dem Namen **Ziel** gesetzt, verlinkt man diesen im Briefing wie folgt. Der Satz im Briefing lautet: **Erobern Sie das Ziel**. Von diesem Satz soll das Wort Ziel mit dem Marker namens **Ziel** verlinkt werden. Im Quelltext sieht das dann so aus:

Erobern Sie das Ziel

Wie man sieht, ist der Marker namens Ziel angegeben und zwischen dem **<a>** und **** das Wort Ziel. Wenn man jetzt im Briefing auf das Wort **Ziel** klickt, wandert das Fadenkreuz beim Anklicken auf den Marker namens **Ziel**, wie auf dem unteren Bild sehr gut zu sehen ist. Befehle, die mit einem **Backslash** versehen sind, beenden die jeweilige Aktion.

Kapitel 3
– Waffen – Fahrzeuge – Einheiten – Objekte –

Nachdem du in den ersten beiden Kapiteln mit der Oberfläche und den Dateien vertraut gemacht wurdest, kommen wir nun zu den etwas spezielleren Bereichen. Du bist nun fähig Einheiten zu setzen, diese mit Wegpunkten zu verbinden und weißt sogar, in welchen Dateien du welchen Bereich finden und konfigurieren kannst. Hier werden dir nun alle Bereiche erläutert, die mit Waffen, Fahrzeugen, Einheiten und Objekten verbunden sind.

Laggingape (Till Breuer)

3.1 - Die Handwaffen und statische Waffen

Hier eine Übersicht sämtlicher Hand- und Statikwaffen, mit den jeweiligen Bezeichnungen, den Magazinen und dem Zubehör.

WESTEN / WIDERSTAND – Leichte Handwaffen



Waffe:	**M16A2**	**M16A2GL**	**M4GL - M4A1GL**
Magazin:	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag
	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag
Granate:		1Rnd_HE_M203	1Rnd_HE_M203
Flares:		FlareWhite_M203	FlareWhite_M203
		FlareGreen_M203	FlareGreen_M203
		FlareRed_M203	FlareRed_M203
		FlareYellow_M203	FlareYellow_M203



Waffe:	**M4**	**M4A1SD**	**M4AIM**
Magazin:	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag
	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag
	30Rnd_556x45_StanagSD	30Rnd_556x45_StanagSD	30Rnd_556x45_StanagSD



Waffe:	**M16A4 - M4A1**	**M16A4_GL**	**M16A4_ACG_GL**
Magazin:	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag	20Rnd_556x45_Stanag
	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag	30Rnd_556x45_Stanag
Granate:		1Rnd_HE_M203	1Rnd_HE_M203
Flares:		FlareWhite_M203	FlareWhite_M203
		FlareGreen_M203	FlareGreen_M203
		FlareRed_M203	FlareRed_M203
		FlareYellow_M203	FlareYellow_M203



Waffe:	**M16A4_ACG**	**MP5A5**	**MP5SD**
Magazin:	20Rnd_556x45_Stanag	30Rnd_9x19_MP5	30Rnd_9x19_MP5
	30Rnd_556x45_Stanag	30Rnd_9x19_MP5SD	30Rnd_9x19_MP5SD

Waffe: **M4SPR**
Magazin: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD

M249
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD
200Rnd_556x45_M249

M240
100Rnd_762x51_M240

Waffe: **G36a**
Magazin: 30Rnd_556x45_ G36

G36C
30Rnd_556x45_G36

G36K
30Rnd_556x45_ G36

Waffe: **M24**
Magazin: 5Rnd_762x51_M24

M107
10Rnd_127x99_M107

Waffe: **M9**
Magazin: 15Rnd_9x19_M9
15Rnd_9x19_M9SD

M9SD
15Rnd_9x19_M9
15Rnd_9x19_M9SD

WESTEN / WIDERSTAND – Schwere Handwaffen

Waffe: **Stinger**
Magazin: Stinger

M136
M136

Javelin
Javelin

WESTEN / WIDERSTAND – Statische Waffen

Waffe: **M119**
Magazin: 30Rnd_105mmHE_M119

M2StaticMG
100Rnd_127x99_M2

SearchLight

OSTEN – Leichte Handwaffen

Waffe: **AK74**
Magazin: 30Rnd_545x39_AK
Granate:
Flares:

AK74GL
30Rnd_545x39_AK
1Rnd_HE_GP25
FlareWhite_GP25
FlareGreen_GP25
FlareRed_GP25
FlareYellow_GP25

AKS74U
30Rnd_545x39_AK

Waffe: **AKS74UN**
Magazin: 30Rnd_545x39_AK
30Rnd_545x39_AKSD

PK
100Rnd_762x54_PK

SVD
10Rnd_762x54_SVD

Waffe: **AKS74PSO**
Magazin: 30Rnd_545x39_AK

KSVK
5Rnd_127x108_KSVK

Waffe: **Makarov**
Magazin: 8Rnd_9x18_Makarov
8Rnd_9x18_MakarovSD

MakarovSD
8Rnd_9x18_Makarov
8Rnd_9x18_MakarovSD

OSTEN – Schwere Handwaffen

Waffe: **6G30**
Magazin: 6Rnd_HE_6G30

RPG7V
PG7V

Strela
Strela

OSTEN – Statische Waffen

Waffe: **D30**
Magazin: 30Rnd_122mmHE_D30

DSHKM
50Rnd_127x107_DSHKM

AGS
29Rnd_30mm_AGS30

Ausrüstung allgemein

Waffe: **Handgrenade**
Magazin: Handgrenade

HandGrenadeTimed
HandGrenadeTimed

Pipebomb
Pipebomb

Waffe: **Mine**
Magazin: Mine

MineE
MineE

Binocular

Waffe: **NVGoggles**
Magazin:

SmokeShell
SmokeShell
SmokeShellRed
SmokeShellGreen

3.2 - Die Waffenbezeichnungsliste

Hier nochmal die Handwaffen und Magazine in Listenform mit kurzer Bemerkung dazu. Natürlich macht es bei einer Waffe ohne Schalldämpfer keinen Sinn ein Magazin mit dem Anhang SD zu vergeben.

Westen / Widerstand

Waffenklasse	Bezeichnung	Munition	
M16A2 M16A4 M16A4_ACG M4 M4A1 M4A1SD M4AIM M4SPR	M16A2 M16A4 M16A4 - Zielfernrohr M 4 - Standard M 4 A1 - Standard M 4 - Schalldämpfer M4 - Aimpoint M 4 - Zielfernrohr	Magazin:	20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 30Rnd_556x45_StanagSD
M16A2GL M16A4_GL M16A4_ACG_GL M4GL M4A1GL	Sturmgewehre mit Granatwerfer	Magazin: Granate: Flares:	20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 30Rnd_556x45_StanagSD 1Rnd_HE_M203 FlareWhite_M203 FlareGreen_M203 FlareRed_M203 FlareYellow_M203
M249	M249 SAW	Magazin:	20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 200Rnd_556x45_M249 30Rnd_556x45_StanagSD
M240	M240	Magazin:	100Rnd_762x51_M240
G36a G36C G36K	G 36 – Standard G 36 - Kommando G 36 – Kommando II	Magazin:	30Rnd_556x45_G36
M24	Scharfschützengewehr	Magazin:	5Rnd_762x51_M24
M107	Scharfschützengewehr	Magazin:	10Rnd_127x99_M107
MP5A5 MP5SD	MP5 - StandardMP5 - Schalldämpfer	Magazin:	30Rnd_9x19_MP5 30Rnd_9x19_MP5SD
M9 M9SD	Pistole Pistole - Schalldämpfer	Magazin:	15Rnd_9x19_M9 15Rnd_9x19_M9SD
M136	M136 - Panzerfaust	Magazin:	M136
Javelin	Javelin - Panzerfaust	Magazin:	Javelin
Stinger	Stinger - Fliegerfaust	Magazin:	Stinger

Osten

Waffenklasse	Bezeichnung	Munition	
AK74	AK 74	Magazin:	30Rnd_545x39_AK
AK74GL	AK 74 mit Granatwerfer	Magazin: Granate: Flares:	30Rnd_545x39_AK 1Rnd_HE_GP25 FlareWhite_GP25 FlareGreen_GP25 FlareRed_GP25 FlareYellow_GP25
AKS74U AKS74UN	AKS 74 U - Standard AKS74UN - Schalldämpfer	Magazin:	30Rnd_545x39_AK 30Rnd_545x39_AKSD
AKS74PSO	AKS74 - Zielfernrohr	Magazin:	30Rnd_545x39_AK AK74PSO
PK	MG	Magazin:	100Rnd_762x54_PK
KSVK	Scharfschützengewehr	Magazin:	5Rnd_127x108_KSVK
SVD	Scharfschützengewehr	Magazin:	10Rnd_762x54_SVD
Makarov MakarovSD	Pistole Pistole - Schalldämpfer	Magazin:	8Rnd_9x18_Makarov 8Rnd _9x18_MakarovSD
6G30	Granatwerfer	Magazin:	6Rnd_HE_6G30
RPG7V	Panzerfaust	Magazin:	RPG7V
Strela	Fliegerabwehr	Magazin:	Strela

Ausrüstung

Waffenklasse	Bezeichnung	Munition
Handgrenade	Handgranate	Handgrenade
HandGrenadeTimed	Handgranate (zeitverzögert)	HandGrenadeTimed
Grenade	Granate	Grenade
TimeBomb	Zeitbombe	TimeBomb
PipeBomb	Sprengsatz	PipeBomb
SmokeShell	Weiße Rauchgranate	SmokeShell
SmokeShellRed	Rote Rauchgranate	SmokeShellRed
SmokeShellGreen	Grüne Rauchgranate	SmokeShellGreen
Mine	Panzermine	Mine
MineE	Schützenmine	MineE
Binocular	Fernglas	Binocular
NVgoggles	Nachsichtgerät	NVgoggles
LaserDesignator	Lasermarkiergerät	LaserBatteries
CarHorn	Autohupe	CarHorn
SportCarHorn	Sportautohupe	SportCarHorn
TruckHorn	Lkw-Hupe	TruckHorn
BikeHorn	Fahrradklingel	BikeHorn

3.3 - Einheiten bewaffnen und ausrüsten

Sämtliche Einheiten aus Armed Assault lassen sich be- bzw. entwaffnen. Dazu muss man wissen, dass eine Einheit immer nur ein Gewehr tragen kann. Als zweite große Waffe kann maximal eine Panzer- oder Flugabwehrwaffe mitgeführt werden. Man kann einer Einheit also erst eine Waffe zuteilen, wenn man zuvor eine andere entfernt hat. Diese kann man dabei einzeln oder auch komplett entfernen.

Mit folgender Syntax entfernt man eine einzelne Waffe:

this removeWeapon "M4" oder **Name removeWeapon "M4"**

Die Magazine, Handgranaten usw. bleiben dabei weiterhin vorhanden. Würde man dieser Einheit nun eine Waffe zuweisen, die die gleichen Magazine benötigt, wäre diese gleich zu Anfang geladen, was nicht immer so ist. Wenn man nämlich mit dem Befehl

removeAllWeapons Name

arbeitet, werden der Einheit alle Waffen und alle Magazine abgenommen. Wenn man diese Einheit nun bewaffnen möchte, muss man eine gewisse Reihenfolge beachten, da die Waffe sonst zu Missionsbeginn nicht geladen sein wird. Man gibt zuerst die Magazine und erst zuletzt die Waffe an!

Nachdem man bei der Einheit nun das M4 entfernt hat, weist man ihr mit dieser Syntax eine neue zu:

this addWeapon "M4A1SD" oder **Name addWeapon "M4A1SD";**

Die Eintragungen kann man in der Initzeile der jeweiligen Einheit oder auch woanders vornehmen. Als Beispiel in Skripten, Auslösern oder Wegpunkten.

Zum Entfernen eines Magazins gilt diese:

this removeMagazine "30Rnd_556x45_Stanag"

und zum zuweisen eines neuen Magazins diese Syntax:

this addMagazine "30Rnd_556x45_StanagSD"

Natürlich treffen diese Befehle nicht nur auf die Waffen, sondern auch auf die nötige Ausrüstung zu. Der normale Soldat trägt in der Regel kein Fernglas oder Nachtsichtgerät bei sich. Da aber ein Fernglas in den Weiten von Sahrani sehr von Vorteil wäre und auch ein Nachtsichtgerät in dunkler Nacht den notwendigen Durchblick verleiht, gibt man in der Initzeile der Einheit die jeweils aufgeführte Syntax an.

für das Fernglas	**this addWeapon "Binocular";**
und für das Nachsichtgerät	**this addWeapon "NVGoggles";**

3.4 - Die Waffen- und Munitionskiste

Alle Waffen- und Munitionskisten lassen sich individuell, also je nach Bedarf, ausstatten. Dabei ist es vollkommen egal, ob man Waffen und Munition in eine oder verschiedene Kisten packt. Um eine Waffen- bzw. Munitionskiste selbst zu definieren, muss man diese zunächst entleeren. Das macht man mit:

clearWeaponCargo this oder **clearWeaponCargo Name**
clearMagazineCargo this oder **clearMagazineCargo Name**

Diese Einträge nimmt man in der Initzeile der jeweiligen Kiste vor, ist aber auch in Bereichen wie Skripten, Auslösern usw. möglich. Nachdem die Kiste nun entleert wurde, fügt man die Waffen und Ausrüstung hinzu, welche die Kiste später enthalten soll. Für **this** kann auch jeweils der **Name** der Kiste angegeben werden.

Im folgenden Beispiel wird eine Kiste mit 2 schallgedämpften M4A1, den dazu passenden 10 Magazinen und 6 Handgranaten beladen.

```
this addWeaponCargo ["M4A1SD",2];
this addMagazineCargo ["30Rnd_556x45_StanagSD",10];
this addMagazineCargo ["Handgrenade",6];
```

Tipp! Auch Fässer lassen sich mit Waffen und Munition befüllen. Dort wird das gleiche Verfahren, wie bei den Munitionskisten verwendet, nur dass man die Fässer eben vorher nicht mit den Clearbefehlen leeren muss.

3.5 - Fahrzeuge be- und entladen

Viele Fahrzeuge in Armed Assault sind bereits mit Magazinen, Handgranaten und ähnlichem beladen. Man hat dort also auch dort als Soldat jederzeit die Möglichkeit sich neu zu bewaffnen. Natürlich kann man die Fahrzeuge im Editor auch selbst mit Magazinen und Waffen bestücken. Hierzu wird das gleiche Verfahren, wie bei den Munitionskisten verwendet. Das Entladen erfolgt wieder mit:

```
clearWeaponCargo this
clearMagazineCargo this
```

und das Beladen mit der jeweiligen Syntax:

```
this addWeaponCargo ["M4A1SD",2];
this addMagazineCargo ["30Rnd_556x45_StanagSD",10];
```

Hier wurden nun 2 schallgedämpfte M4A1 Sturmgewehre mit den dazugehörigen Magazintypen, hier 10 Magazine, in das Fahrzeug geladen.

3.6 - Waffenauswahl im Briefing

Als Leader einer Mehrspielergruppe hat der Spieler später über das Briefing in der Mission die Möglichkeit sich und seine Gruppe selbst auszustatten, vorausgesetzt man hat dies vorher so in der Description.ext, siehe **Kapitel 2.3**, vordefiniert.

Dazu muss eine Klasse für die jeweilige Waffe und eine Klasse für die dazugehörige Magazinsorte vordefiniert werden. Im folgenden Beispiel wurden 6 schallgedämpfte M4A1 mit 20 dazugehörigen Magazinen, 2 M136 Panzerfäuste mit 6 Schuss und dazu noch 20 Handgranaten festgelegt.

```
  // Hier beginnt der Part Class Weapons
class Weapons
{
        class M4A1SD
        {
        count = 6;
        };
        class M136
        {
        count = 2;
        };
 };
// Hier endet der Part Class Weapons

// Hier beginnt der Part Class Magazines
class Magazines
{
        class 30Rnd_556x45_StanagSD
        {
        count = 20;
        };
        class M136
        {
        count = 6;
        };
        class HandGrenade
        {
        count = 20;
        };
};
// Hier endet der Part Class Magazines
```

Die Art der Waffe

Die Anzahl der Waffen

Die Art der Waffe

Die Anzahl der Waffen

Die Art der Magazine

Die Anzahl der Magazine

Die Art der Magazine

Die Anzahl der Magazine

Die Art der Waffe

Die Anzahl der Magazine

Möchte man die Waffenauswahl erweitern, fügt man lediglich die jeweilige Klasse der Waffe und die Klasse der Magazine selbst in diesen Abschnitt der Description.ext ein.

3.7 - Die Fahrzeugklassen

Hier die Fahrzeugübersicht, mit der Editorbezeichnung, einer Beschreibung und dem Klassennamen, welcher benötigt wird, wenn man Fahrzeuge mit einer Syntax irgendwo auf der Karte erstellen oder anderweitig ansprechen möchte.

WESTEN

Fahrzeug	Beschreibung	Klassenname
Land		
M1Abrams	Kampfpanzer	M1Abrams
M113	Schützenpanzer	M113
M113Ambul	Sanitätsschützenpanzer	M113Ambul
Vulcan	Flugabwehrpanzer	Vulcan
Stryker ICV M2	Schützenpanzer mit M2-Maschinengewehr	Stryker_ICV_M2
Stryker ICV MK19	Schützenpanzer mit Granatwerfer	Stryker_ICV_MK19
Stryker TOW	Schützenpanzer mit Panzerabwehrwaffe	Stryker_TOW
HMMWV	Hummer	HMMWV
HMMWV M2	Hummer mit M2	HMMWV50
HMMWV TOW	Hummer mit Panzerabwehrwaffe	HMMWVTOW
HMMWV MK 19	Hummer mit Granatwerfer	HMMWVMK
Truck 5 t	Lkw 5 Tonner	Truck5t
Truck 5 t Open	Lkw 5 Tonner - offen	Truck5tOpen
Truck 5 t MG	Lkw 5 Tonner mit Maschinengewehr	Truck5tMG
Truck 5 t Repair	Lkw 5 Tonner - Reparaturfahrzeug	Truck5tRepair
Truck 5 t Reammo	Lkw 5 Tonner - Munitionsfahrzeug	Truck5tReammo
Truck 5 t Refuel	Lkw 5 Tonner - Tankfahrzeug	Truck5tRefuel
Motorcycle	Motorrad	M1030
Luft		
AH 1 Z	Cobra-Kampfhubschrauber	AH1W
AH 6	Little Bird Helikopter bewaffnet	AH6
AV 8 B	Harrier mit Raketen	AV8B2
AV 8 B (GBU)	Harrier mit Bomben	AV8B
A10	A10 mit Raketen und GAU12	A10
MH 6	Little Bird Helikopter unbewaffnet	MH6
UH 60	Blackhawk - Helikopter mit MG	UH60MG
UH 60 (FFAR)	Blackhawk - Helikopter mit Raketenwerfer	UH60
Camel	Doppeldecker	Camel
Parachute	Fallschirm	ParachuteWest
Wasser		
CRRC	Schlauchboot	Zodiac
RHIB	Patrouillenboot mit Maschinengewehr	RHIB
RHIB 2 Turret	Patrouillenboot mit MG und Granatwerfer	RHIB2Turret

OSTEN

Fahrzeug	Beschreibung	Klassenname
Land		
T72	Kampfpanzer	T72
BMP2	Schützenpanzer	BMP2
BMP2Ambul	Sanitätsschützenpanzer	BMP2Ambul
ZSU	Shilka Flugabwehrpanzer	ZSU
BRDM2	Schützenpanzer	BRDM2
BRDM2_ATGM	Schützenpanzer mit Panzerabwehrwaffe	BRDM2_ATGM
UAZ	Jeep	UAZ
UAZMG	Jeep mit Maschinengewehr	UAZMG
Ural	Lkw	Ural
UralOpen	Lkw – offen	UralOpen
UralRepair	Lkw - Reparaturfahrzeug	UralRepair
UralReammo	Lkw – Munitionsfahrzeug	UralReammo
UralRefuel	Lkw - Tankfahrzeug	UralRefuel
Motorcycle	Motorrad	TT650G
Luft		
SU 34	SU 34 mit Raketen, FFAR, Bordkanone	SU34
SU 34B	SU 34 mit Raketen und Bordkanone	SU34B
Mi 17	Helikopter mit Maschinengewehr	Mi17_MG
Mi 17	Helikopter mit Raketenwerfer	Mi17
KA-50	Kampfhubschrauber	KA50
Camel E	Doppeldecker	Camel2
Parachute	Fallschirm	ParachuteEast
Wasser		
PBX Boat	Schlauchboot	PBX

WIDERSTAND

Fahrzeug	Beschreibung	Klassenname
Land		
M113 RACS	Schützenpanzer	M113_RACS
Vulkan RACS	Flugabwehrpanzer	Vulkan_RACS
4x4	Jeep geschlossen	Landrover
4x4 MG	Jeep mit Maschinengewehr M2	LandroverMG
4x4 Open	Jeep offen	Landrover_Closed
Luft		
AH6 RACS	Kampfhubschrauber	AH6_RACS
MH6 RACS	Little Bird	MH6_RACS
Parachute	Fallschirm	ParachuteG
Wasser		
CRRC	Schlauchboot	Zodiac2

ZIVILISTEN

Fahrzeug	Bemerkung	Klassenname
Land		
Pick-Up	Pick-Up blau	Datsun1_civil_1_open
Pick-Up 2	Pick-Up rot geschlossen	Datsun1_civil_2_covered
Pick-Up 3	Pick-Up grün	Datsun1_civil_3_open
Offroad	Geländewagen grau offen	Hilux1_civil_1_open
Offroad2	Geländewagen rotes Verdeck	Hilux1_civil_3_open
Offroad3	Geländewagen weiß offen	Hilux1_civil_2_covered
Sedan	Auto weiß	Car_sedan
Hatchback	Auto rot	Car_hatchback
Skoda	Skoda weiß	Skoda
Skoda (Blue)	Skoda blau	SkodaBlue
Skoda (Red)	Skoda rot	SkodaRed
Skoda (Green)	Skoda grün	SkodaGreen
Policecar	Polizeijeep	Landrover_Police
Bus	Stadtbus	Bus_city
UralCivil	Lkw gelb geschlossen	UralCivil
UralCivil 2	Lkw blau geöffnet	UralCivil2
Tractor	Trakor	Tractor
Motorcycle	Motorrad	TT650C
Luft		
Parachute	Fallschirm	ParachuteC

3.8 - Die Einheitsklassen

WESTEN

Einheit	Beschreibung	Klassenname
AA Specialist	Fliegerabwehrsoldat mit Stinger	SoldierWAA
AT Specialist	Panzerabwehrsoldat mit M 136	SoldierWAT
Automatic Rifleman	Soldat mit Maschinengewehr M249	SoldierWAR
Camel Pilot	Doppeldecker Pilot	BISCamelPilot
Crewman	Fahrzeugbesatzung	SoldierWCrew
Engineer	Pionier	SoldierWMiner
Grenadier	Grenadier	SoldierWG
Machinegunner	Soldat mit Maschinengewehr M240	SoldierWMG
Medic	Sanitätssoldat	SoldierWMedic
Officer	Offizier	OfficerW
Pilot	Pilot	SoldierWPilot
Rifleman	Soldat mit M4AIM	SoldierWB
SF Assault	Special Forces mit M4A1GL	SoldierWSaboteurAssault
SF Marksman	Special Forces mit M4 SPR	SoldierWSaboteurMarksman
SF Recon	Special Forces mit M4 A1 SD	SoldierWSaboteurRecon
SF Saboteur	Special Forces mit M4 A1 SD	SoldierWSaboteurPipe
SF Saboteur 2	Special Forces mit MP 5 SD	SoldierWSaboteurPipe2
Sniper	Scharfschütze mit M24	SoldierWSniper
Squad Leader	Zugführer mit M4 AIM	SquadLeaderW
Team Leader	Gruppenführer mit M4 AIM	TeamLeaderW

OSTEN

Einheit	Beschreibung	Klassenname
AA Specialist	Fliegerabwehrsoldat mit Strela	SoldierEAA
AT Specialist	Panzerabwehrsoldat mit RPG 7 V	SoldierEAT
Camel Pilot	Doppeldecker Pilot	BISCamelPilot2
Crewman	Fahrzeugbesatzung	SoldierECrew
Engineer	Pionier	SoldierEMiner
Especas	Speznaz mit AKS 74 U	SoldierESaboteurPipe
Especas Marksman	Speznaz mit AKS74PSO	SoldierESaboteurMarksman
Especas Saboteur	Speznaz mit AKS 74 UN	SoldierESaboteurBizon
Grenadier	Grenadier	SoldierEG
Machinegunner	Soldat mit Maschinengewehr PK	SoldierEMG
Medic	Sanitätssoldat	SoldierEMedic
Officer	Offizier	OfficerE
Pilot	Pilot	SoldierEPilot
Rifleman	Soldat	SoldierEB
Sniper	Scharfschütze mit Dragunov (SVD)	SoldierESniper
Squad Leader	Zugführer	SquadLeaderE
Team Leader	Gruppenführer	TeamLeaderE

WIDERSTAND

Einheit	Beschreibung	Klassenname
AA Specialist	Fliegerabwehrsoldat mit Stinger	SoldierGAA
AT Specialist	Panzerabwehrsoldat mit M136	SoldierGAT
Crewman	Fahrzeugbesatzung	SoldierGCrew
Engineer	Pionier	SoldierGMiner
Grenadier	Grenadier	SoldierGG
Machinegunner	Soldat mit Maschinengewehr M240	SoldierGMG
Medic	Sanitätssoldat	SoldierGMedic
Officer	Offizier	SoldierG
Pilot	Pilot	SoldierGPilot
Rifleman	Soldat	SoldierGB
Royal Commando	Königliches Kommando mit MP 5 SD	SoldierGCommando
Royal Guard	Königliche Garde mit G36c	SoldierGGuard
Royal Marksman	Königlicher Scharfschütze mit G36a	SoldierGMarksman
Sniper	Scharfschütze mit M24	SoldierGSniper
Squad Leader	Zugführer	SquadLeader
Team Leader	Gruppenführer	TeamLeader

ZIVILISTEN

Einheit	Beschreibung	Klassenname
Civilian bis Civilian21	Nähere Beschreibung nicht erforderlich. Durchnummeriert von Civilian bis Civilian21.	Civilian
N/A	König	King
N/A	Kriegsberichterstatter	FieldReporter
N/A	Leibwächter	Anchorman
N/A	Premierminister	NorthPrimeMinister
N/A	Reporterin	MarianQuandt
N/A	Reporterin	MarianQuandt02
N/A	Reporterin	MarianQuandt03
N/A	Reporterin	MarianQuandt04
N/A	Zombie	Civil_Undead_1
N/A	Zombie	Civil_Undead_2
N/A	Zombie	Civil_Undead_3
N/A	Zombie	Civil_Undead_4

Einheiten, die hier mit N/A angegeben sind, sind im Editor selbst nicht verfügbar. Diese kann man, wie in **Kapitel 5.45** näher erläutert, mit dem CreateVehicle-Befehl erzeugen.

INSEKTEN

Typ	Beschreibung	Klassenname
N/A	Möwe	Seagull
N/A	Libelle	Dragonfly
N/A	Hausfliege	HouseFly
N/A	Honigbiene	Honeybee
N/A	Mosquito	Mosquito
N/A	Schmetterling	Butterfly

3.9 - Waffen- und Magazintypen ausgeben lassen

Mit folgend erläuterter Syntax hat man die Möglichkeit sich die Waffen- und Magazintypen einer Einheit in Form einer Texteinblendung ausgeben zu lassen.
Dabei verwendet man für das Ausgeben der Waffentypen:

```
hint format ["%1", weapons this];
hint format ["%1", weapons Name];
```

und für das Ausgeben der Magazintypen:

```
hint format ["%1", magazines this];
```

3.10 - Abgefeuerten Typ ausgeben lassen

Neben dem Ausgeben von Waffen- und Magazintypen, hat man natürlich noch die Möglichkeit sich ausgeben zu lassen, welche Einheit gerade mit was geschossen hat.
Dabei werden folgende Daten anhand einer Texteinblendung ausgegeben:

Name der Einheit
Der Waffentyp
Der Geschosstyp
Die Schussart (Single/Burst)

Nachdem die Einheit **Name** im Spiel die Waffe abfeuert, blendet dieser Text ein:

Dazu macht man sich einen Eventhandler zu Nutze, mit welchem die Syntax in der Initzeile einer Einheit, wie im folgenden Beispiel, angelegt wird:

```
this addEventhandler ["fired", {hint format ["%1", _this]}]
```

Das Ganze geht natürlich auch namenbezogen für externe Aktionen:

```
Name addEventhandler ["fired", {hint format ["%1", _this]}]
```

Kapitel 4
- Die Mission -

Nachdem du in den ersten drei Kapiteln die Oberfläche, die Dateien und die Waffen kennen gelernt hast, kommen wir nun zu den etwas spezielleren Bereichen des Missionsdesigns. Hier wirst du lernen, wie man eine Mission startet, die Ziele festlegt, die Erfüllung angemessen bewertet und am Ende die Mission ordentlich beendet.

Switcher83 (Matthias Schulz)

4.1 - Der Missionsname

Wenn man eine neue Karte erstellt, gibt man unter **Info** zunächst den Namen der Mission an. Dies ist zwar nicht unbedingt erforderlich, aber wenn man dass nicht macht, wird die Mission später im Einzelmissionsauswahlmenü mit dem Inselnamen angezeigt. Das sieht dann etwa so aus:

Bei **Beispielmission** wurde bei **Info** unter **Name** auch der Text **Beispielmission** eingetragen. Bei **Beispielmission2** war das nicht der Fall und deshalb steht dort **Beispielmission2.Intro**.

4.2 - Der Missionsstart

Beim Intro- und Missionsstart hat man die Möglichkeit eine Texteinblendung und Uhrzeit in individueller Form anzuzeigen zu lassen. Der Text dafür ist variabel, sollte aber auch nicht zu lang sein. Zum Anzeigen dieses Textes und der Uhrzeit muss man die Description.ext bearbeiten und falls diese im eigenen Missionsordner noch nicht vorhanden ist, muss man eine erstellen. Nähere Informationen zum Thema Description.ext findet man im **Kapitel 2.3**.

Zum Vordefinieren des Textes und der Uhrzeit trägt man im Kopf der Description.ext etwa folgendes ein:

```
onLoadIntro = Morphicon proudly presents
onLoadMission = Convoy Attack

onLoadIntroTime = true bzw. false oder 1 bzw. 0
onLoadMissionTime = false
```

Möchte man keines von Beiden anzeigen lassen, setzt man den Wert dazu lediglich auf **0** oder schreibt stattdessen die Wortform **false** hinter das Gleichheitszeichen und schon werden der Text und die Uhrzeit nicht angezeigt.

Als Text kann man hier natürlich auch die Adresse zur Stringtable.csv definieren. Das sieht dann etwa so aus:

onLoadIntro = $STR_Missionsstart

Mehr Informationen zum Thema Stringtable.csv sind im **Kapitel 2.4** zu finden.

4.3 - Das Missionszubehör

Der Missionsersteller hat die Möglichkeit zu bestimmen, ob man das jeweilige Missionszubehör in einer Mission zur Verfügung stehen soll oder eben nicht. Dazu müssen wieder diverse Zeilen in der Description.ext definiert werden, um die Zubehörteile anzeigen oder verbergen zu lassen.

Zum Anzeigen wird wieder der Wert **1** oder **true** und zum Verbergen der Wert **0** oder **false** verwendet. Folgend die Übersicht der dazu benötigen Befehle:

ShowGPS = 1; - GPS

ShowCompass = 1; - Kompass

ShowRadio = 1; - Funkgerät

ShowMap = 1; - Karte

ShowNotePad = 1; - Briefing

ShowWatch = 1; - Uhr

ShowDebriefing = 1; - Abschlussbriefing

4.4 - Die Missionswertung

Wie in jedem Spiel kann der Spieler in jeder Mission Punkte erreichen. Diese müssen dazu vorher in der Description.ext definiert werden. Die Anzahl der zu erreichenden Punkte kann dabei frei bestimmt werden. In der Description.ext wird das wie folgt angegeben:

minScore=200	Die Mindestpunkte
avgScore=3000	Die Mittelpunkte
maxScore=6000	Die Höchstpunkte

Der Spieler bekommt bereits im Spiel automatisch für jede getötete Einheit Punkte zugewiesen. Wenn man aber nun machen möchte, dass der Spieler für das Erfüllen eines gewissen Missionziels eine gewisse Wertung erhält, macht man das mit folgender Syntax:

Player addRating Wert

Natürlich kann man dem Spieler auch Punkte abziehen. Dann, wenn zum Beispiel etwas zerstört oder getötet wurde, was nicht hätte passieren dürfen. Dazu müsste dann bei obiger Syntax ein Minuswert verwendet werden.

Möchte man eine Abfrage starten dass, wenn der Spieler eine gewisse Punktzahl erreicht hat, die Mission zum Beispiel beendet wird, erstellt man einen Auslöser mit der **Größe** (Achse a/b) **0** und schreibt in die Bedingungszeile:

Rating Player > Wert oder **Rating Player >= Wert**

Bei **Typ** wählt man nun einfach **Ende1** aus. Wenn der Spieler nun den Wert erreicht oder überschritten hat, wird die Mission beendet und das Briefing wird mit der jeweiligen Wertung angezeigt.

4.5 - Die Missionsziele

Die Missionsziele sind wohl das wichtigste an einer Mission. Denn ohne Missionsziel gibt es ja bekanntlich auch keine Mission! Diese müssen natürlich wieder auf irgendeine Art vordefiniert werden.

Die Missionsziele werden, wie man im **Kapitel 2.13** nachlesen kann, angelegt und eventuell auch in der **Init.sqs**, siehe **Kapitel 2.5,** vordefiniert, wenn man diese zunächst verdecken möchte. Verdecken heißt, dass der Spieler das verdeckte Ziel zuerst nicht im Briefing sehen kann. In der Mission müssen jetzt natürlich noch die jeweiligen Ziele definiert und konfiguriert werden.

Beispielmission

Der Spieler sieht im Briefing den Auftrag „Erobern Sie diese Ortschaft". Der zweite Auftrag „Zerstören Sie den Munitionstruck" wird zunächst durch den Eintrag in der Init.sqs verdeckt. Wenn der Ort von Feinden befreit wurde, soll das erste Ziel abgehakt und das zweite Ziel sichtbar gemacht werden.

Dazu setzt man über die Ortschaft einen Auslöser mit der Beispielgröße **300** (bei **Achse a/b**), wählt bei **Aktivierung** die Seite aus, die den Ort bewachen soll und wählt bei Art der Aktivierung **Nicht vorhanden** aus. Der Auslöser würde also ausgelöst werden, wenn die jeweilig ausgewählte Seite nicht mehr im Auslöserbereich vorhanden ist.

In der **Aktivierungszeile** gibt man nun die Befehle an, die ausgeführt werden sollen, wenn der Auslöserbereich von Feinden befreit wurde. Dazu gehörten **Ziel 1 abhaken** und **Ziel 2 sichtbar** machen. Als kleinen Infotext fügt man ggf. noch einen **Hint** hinzu. Das Ganze schaut dann so aus:

"1" ObjStatus "Done"; "2" ObjStatus "Visible"; hint "Missionsplan aktualisiert"

Auf der Karte setzt man über der Ortschaft, wie in der folgenden Briefing.html schon definiert, einen Marker namens **ZielX**, welchen das Briefingfadenkreuz ansteuert, wenn der Spieler im Briefing auf den Link „**diese Ortschaft**" klickt.

Beispielabschnitt der dazugehörigen Briefing.html:

```
  <p>
<a name="OBJ_1"></a>
Erobern Sie <a href="marker:ZielX">diese Ortschaft </a>
</p>
<hr>
<p><a name="OBJ_2"></a>
Zerstören Sie den Munitionstruck!
</p>
<hr>
```

Der dazugehörige Eintrag in der Init.sqs, der das zweite Missionsziel zunächst verdeckt, muss hierbei **"2" ObjStatus "Hidden"** lauten.
Nun folgend die unterschiedlichen Befehle, die für den jeweiligen Missionszielstatus gebraucht werden:

Hidden	- Missionsziel wird verdeckt
Visible	- Missionsziel wieder sichtbar
Active	- Missionsziel wird wieder aktiviert
Done	- Missionsziel wird abgehakt
Failed	- Missionsziel fehlgeschlagen

4.6 - Mission beenden

Nach dem Erfüllen der Missionsziele sollte eine Mission normalerweise beendet werden. Natürlich gibt es da viele Möglichkeiten diese zu beenden. Diese hängen aber einzig und allein von der Story, den Missionszielen oder dem Verlauf der Mission ab. Hier mal ein paar mögliche Beispiele für die Beendung einer Mission.

Einheitsbezogen aus lokaler Sicht

Auf dem rechten Bild sind mehrere Einheiten zu sehen. Ziel soll es sein den Schützenpanzer zu beseitigen. Dieser wurde mit der Taste F2 mit dem Auslöser verbunden und der Auslöser wie folgt definiert:

Achse a/b: 50
Typ: Ende #1
Aktivierung: Nicht vorhanden

Wird der Panzer nun zerstört oder verlässt er den Auslöserbereich, wird die Mission beendet. Da der Panzer aber auf jeden Fall zerstört werden soll und die Mission nicht beendet werden soll, wenn dieser den Bereich verlässt kommen wir zur globalen Sichtweise.

Einheitsbezogen aus globaler Sicht

Dazu setzt man ebenfalls einen Auslöser, aber legt keinen Auslöserbereich fest, und stellt diesen wie folgt ein:

Achse a/b: 0
Typ: Ende #1
Bedingung: ! (alive Panzer1)

Der Panzer wurde nun **Panzer1** benannt und der Auslöserwirkungsbereich mit dem Wert **0** auf globales Prüfen geschaltet. Man kann den Auslöser nun frei auf der Map platzieren, da er global prüft, ob **Panzer1** noch lebt. Als Bedingung der Auslösung wurde ja bei Bedingung die Syntax **! (alive Panzer1)** angegeben.

Der Panzer kann sich nun frei auf der Karte bewegen und die Mission wird erst beendet, wenn dieser zerstört ist. Natürlich könnte man bei der lokalen Sichtweise die Prüfweite auf 5000 oder so stellen , dann würde der Panzer den Bereich auch nicht verlassen können, aber der Übersicht halber ist die globale Sichtweise zu bevorzugen. Je weniger Auslöserkreise auf einer Karte, desto besser die Übersicht beim Editieren.

Einheitsbezogen aus globaler Sicht mit mehreren Einheiten

Hierzu gelten die gleichen Vorgaben wie bei **Einheitsbezogen aus globaler Sicht**, nur werden in der Bedingungszeile, noch weitere Bedingungen angegeben.

! (alive Panzer1) AND ! (alive MG1) AND ! (alive Soldat1)

Das **AND** steht für **UND** und verbindet somit diese Bedingungen zu einer. Die Mission wird nun erst beendet, wenn die Objekte namens **Panzer1**, **MG1** und **Soldat1** tot sind.

Mission wird beendet, wenn der Auslöserbereich feindleer ist

Auf dem unteren Bild sind mehrere Einheiten zu sehen. Ziel soll es sein die Ortschaft (Auslöserbereich) vom Feind zu befreien. Dabei ist es egal, ob diese eliminiert oder aus dem Auslöserbereich vertrieben werden.

Der Auslöser wird hierbei wie folgt definiert:

Achse a/b:	50
Typ:	Ende #1
Aktivierung:	OSTEN
	Nicht vorhanden

Ab der Version 1.05 gibt es auch die Möglichkeit einen Auslöser so zu konfigurieren, dass der Auslöser erst auslöst, wenn alle Feindeinheiten eliminiert bzw. nicht präsent sind und mindestens eine befreundete Einheit im Auslöserbereich ist. Dazu definiert man beim Auslöser unter **Aktivierung** einfach, von welcher Seite (**Erobert durch...**) dieser Bereich erobert werden soll. Die verfeindete Gegenseite ist dabei egal.

Variablenbezogene Beendung der Mission

Die variablenbezogene Auslösung einer Mission ist nicht unbedingt schwerer, aber je nach Ausmaß ein klein wenig aufwendiger. Als Beispiel könnte nun folgendes dienen. Drei unterschiedliche Auslöserbereiche sollen vom Feind befreit werden. Erst wenn diese 3 Bereiche, als Beispiel 3 Ortschaften, befreit sind, soll die Mission beendet werden. Das Ganze kann man nun wie folgt lösen:

Auslöser 1 (Bereich 1):

Achse a/b:	100
Aktivierung:	OSTEN (nicht vorhanden)
bei Aktivierung:	Ziel1=true

Auslöser 2 (Bereich 2):

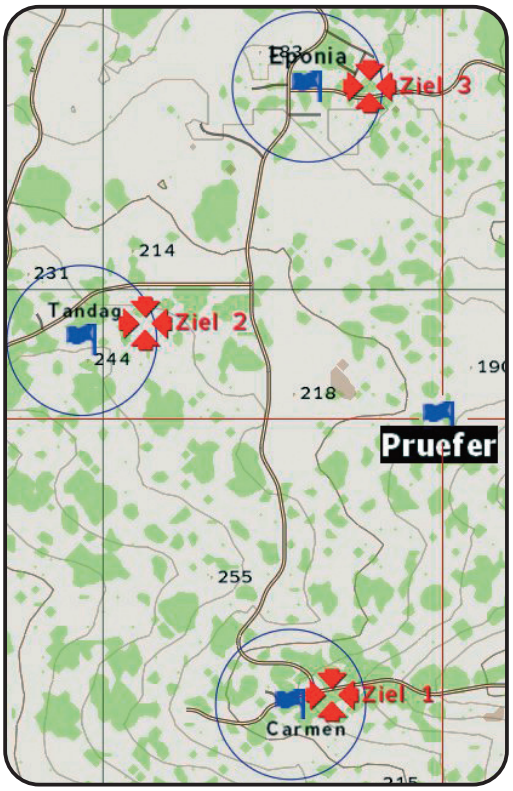
Achse a/b:	150
Aktivierung:	OSTEN (nicht vorhanden)
bei Aktivierung:	Ziel2=true

Auslöser 3 (Bereich 3):

Achse a/b:	100
Aktivierung:	OSTEN (nicht vorhanden)
bei Aktivierung:	Ziel3=true

Auslöser 4 (Prüfer):

Achse a/b:	0
Typ:	Ende #1
Bedingung:	Ziel1 AND Ziel2 AND Ziel3



Wie man nun oben sehen kann, wird bei jedem der drei Auslöser bei der Auslösung eine Variable (**Ziel1**, **Ziel2**, **Ziel3**) auf **true** geschaltet. Beim Auslöser 4 wurden diese 3 Variablen als Bedingung angegeben, um diesen auszulösen. Sind diese nun erfüllt, wird die Mission beendet. Es bietet sich an bei **Timeout** noch ein paar Werte anzugeben, um die Beendungszeit etwas flexibel zu gestalten.

4.7 - Mission speichern

Dieser Befehl ermöglicht das Speichern während des Spielens. Der Missionsersteller kann so auf der Karte also Speicherpunkte setzen. Zum Beispiel, wenn der Spieler einen bestimmten Punkt auf der Karte erreicht oder ein Missionsziel erfüllt hat, wird das Spiel so automatisch gespeichert. Die Syntax hierfür lautet lediglich:

Savegame

Wählt man später im Menü **Neuer Versuch**, beginnt der Spieler wieder an der Stelle, an welcher das Spiel gespeichert wurde. Möchte man Speichern komplett verbieten, schreibt man in die Description.ext lediglich:

saving = 0;

Xsive (Pierre Wirsik)

Kapitel 5
- Missionszubehör -

In diesem Kapitel ist alles erläutert, was in einer Mission zu gebrauchen ist. Es stellt auch gleichzeitig, Dank seiner vielen Unterpunkte, das umfangreichste Kapitel dieses Buches dar. Hier solltest du viele Antworten auf deine offenen Fragen erhalten und zudem noch eine Menge Anregungen, die zu deiner Mission beitragen, bekommen.

5.1 - Leeres oder verschlossenes Fahrzeug

Zum Erstellen eines leeren Fahrzeuges wählt man zunächst die Taste F1 und klickt doppelt auf die Karte. Nach dem Öffnen des Menüs schaltet man bei Auswahl der Seite auf **Leer**, wählt bei **Klasse** und **Einheit** den Vehikeltyp aus und klickt dann auf **OK**.

Wenn das Fahrzeug verschlossen bzw. nicht nutzbar sein soll, wählt man bei **Fahrzeugstatus** einfach **Abgeschlossen** für verschlossen aus. Möchte man das Fahrzeug später nutzbar machen geht das wie folgt:

Name1 Lock true	- Verschließt das Fahrzeug
Name1 Lock false	- Öffnet das Fahrzeug

5.2 - Fahrer/Beifahrer eines Fahrzeugs

Mit den folgenden Befehlen lassen sich Einheiten an beliebige Positionen eines Fahrzeuges beamen. Dies lässt sich unter anderem in der Initzeile der jeweiligen Einheit oder auch in einem Skript etc. definieren. Gibt man das in der Initzeile der jeweiligen Einheit an, kann für **Name** auch **this** stehen!

Name moveinDriver Fhz1	- Fahrer des Fahrzeugs
Name moveinCargo Fhz1	- Beifahrer des Fahrzeugs
Name moveinCommander Fhz1	- Kommandant des Fahrzeugs
Name moveinGunner Fhz1	- Schütze des Fahrzeugs
Name moveinTurret Fhz1	- Schütze des Fahrzeugs (i.d.R. MG)
Name moveInCargo [Fhz1,3]	- Mitfahrer auf beliebiger Cargoposition

5.3 - Einheit hat Fahrzeugverbot

Diesen Befehl verwendet man, wenn man möchte, dass eine bestimmte Einheit ein Fahrzeug nicht nutzen kann oder soll. Dazu trägt man in der Initzeile des Fahrzeugs folgende Syntax ein.

[Name1, Name2, Name3] allowGetIn false

In dieses Fahrzeug dürfen **Name1**, **Name2**, **Name3** nun nicht einsteigen. Möchte man das Fahrzeug später wieder für diese Einheiten nutzbar machen, schaltet man wieder auf **true**.

[Name1, Name2, Name3] allowGetIn true

5.4 - Einheit in Fahrzeug?

Manche Abläufe erfordern das Prüfen, ob sich eine Einheit noch in einem Fahrzeug befindet oder auch nicht. Dieses kann man z.B. als Bedingung nehmen, um damit einen Auslöser oder ein Skript zu starten.

Um zu prüfen, ob **Name1** in dem Fahrzeug sitzt, nutzt man diese Syntax:

Name1 in Fahrzeugname

und gibt sie in die Bedingungszeile eines prüfenden Auslösers ein. Steigt die Einheit nun in das definierte Fahrzeug, wird der Auslöser aktiviert.

In einem Skript muss das Ganze so aussehen:

? Name1 in Fahrzeugname

Um zu prüfen ob eine Einheit nicht mehr in einem Fahrzeug sitzt, benötigt man folgende Syntax:

Not (Name1 in Fahrzeugname)

Für das **NOT** kann auch ein **!** verwendet werden. Gleiche Eigenschaft!

5.5 - Fahrzeug fährt erst, wenn Einheit eingestiegen ist

Angenommen man hat ein Fahrzeug auf der Karte, welches auch schon mit Wegpunkten versehen ist, welches erst losfahren soll, wenn eine bestimmte Einheit, als Beispiel der Spieler, eingestiegen ist. Dies kann ein normales Auto, Boot oder auch ein Helikopter sein.

Dazu gibt man in der Bedingungszeile als Bedingung folgende Syntax an:

Name in Fahrzeugname oder **Vehicle Name == Fahrzeugname**

Gruppenbezogen

Natürlich geht das Ganze auch gruppenbezogen. Das Fahrzeug soll warten, bis die Gruppe eingestiegen ist. Dazu wird bei der folgenden Syntax auch der Fahrer mitgezählt, der schon im Truck sitzt. Man muss also die Größe der Gruppe plus den Fahrer rechnen. Dazu gibt man in der Bedingungszeile folgendes an:

count crew Truck1 >= 10 oder **count crew Truck1 == 10**

5.6 - Gruppe zu Missionsbeginn in Fahrzeug

Wenn eine Gruppe gleich zu Missionsbeginn in einem Fahrzeug, als Beispiel einem Helikopter sitzen soll, muss in der Initialisierungszeile des Leaders der Gruppe folgende Syntax angegeben werden:

{_x moveInCargo Heli1} forEach Units Group This

oder

{_x moveInCargo Heli1} forEach Units Grp1

5.7 - Ein- und aussteigen lassen

Dafür setzt man zunächst ein leeres Fahrzeug und einen Soldat auf die Karte. Dann weist man dem Soldaten einen Wegpunkt zu, welchen man genau auf das Fahrzeug setzt und wählt bei **Typ Einsteigen** aus. Dann setzt man den nächsten Wegpunkt an der Zielposition und wählt dort **Aussteigen** aus.

Möchte man Einheiten per Syntax aussteigen lassen, geht das wie folgt:

unassignVehicle Fahrzeug1 - Einheit verlässt Fahrzeug

{unassignVehicle _x} forEach units Grp1 - Einheiten verlassen Fahrzeug

Leader befehlen dabei ihren Einheiten auszusteigen

5.8 - Geschwindigkeit einer Einheit

Die Geschwindigkeit einer Einheit kann bei einem Wegpunkt unter **Geschwindigkeit** oder auch per Syntax bestimmt werden. Diese lautet wie folgt:

Name1 SetSpeedMode "Limited" - langsam

Name1 SetSpeedMode "Normal" - mittel

Name1 SetSpeedMode "Full" - schnell

5.9 - Einheiten starten bzw. stoppen

Als Beispiel dient hier eine Einheit oder Gruppe, die verschiedene Wegpunkte zugewiesen bekommen hat. Aus irgendwelchen Missionsgründen soll die Einheit oder die Gruppe nun zwischen zwei Wegpunkten verweilen. Das macht man mit folgender Syntax:

Name1 stop true

Die Einheit wird nun dort verweilen, bis die Syntax wieder auf false gesetzt wird.

Name1 stop false

5.10 - Einheit bleibt stehen

Der Befehl **dostop this** in der Init-Zeile einer Einheit bewirkt, dass diese an ihrer gesetzten Position stehen bleibt. Dies ermöglicht eine Gruppe auf die Karte zu setzen und die Mitglieder frei zu verteilen, ohne dass diese wieder zu ihrem Leader zurücklaufen und sich somit wieder in Formation begeben.

Teambezogen
Sehr nützlich, wenn man Anführer einer Gruppe ist und nicht möchte, dass seine Untergebenen bei Missionsbeginn gleich hinter einem her laufen und sie selbst in Stellungen unterbringen möchte. Sie bleiben somit an der zuvor im Editor gesetzten Stelle stehen, wenn man bei der jeweiligen Einheit **dostop this** in die Initialisierungszeile einträgt. Zusätzlich ist es wichtig bei der jeweiligen Einheit unter **Speziell** den Wert **Keine** zu definieren!

Feindbezogen
Auch hier ist dieser Befehl sehr nützlich, denn nun kann man ein feindliches Team sehr gut in der Landschaft verteilen. Hierbei ist es ebenfalls wichtig bei der Einheit bei **Speziell Keine** auszuwählen. Wird der Feind nun angegriffen, werden die Einheiten auf ihrer festgelegten Position in Stellung gehen.

5.11 - Einheit starten lassen

Als Beispiel dient hier eine Gruppe, die ihre zugewiesenen Wegpunkte anlaufen soll, wenn der Spieler einen Wegpunkt oder einen Auslöser ausgelöst hat. Dem Leader der Gruppe schreibt man dabei

this stop true

in die Initzeile, während man bei dem Wegpunkt des Spielers oder dem Auslöser, den der Spieler auslösen soll, dies angibt:

Name1 stop false

Die Gruppe wird dann ihre Wegpunkte anlaufen. Oben wurde der Wert **this** verwendet, weil die Syntax in der Initzeile des Leaders der Gruppe angegeben wurde, während beim zweiten Befehl der Name des Leaders benötigt wird, weil der Befehl von dem Wegpunkt bzw. Auslöser kommt. Der Leader sollte also auch mit einem Namen versehen werden.

5.12 - Einheit bewegt sich zu Bestimmungsort

Man kann Einheiten auch ohne Wegpunkte zu bestimmten Punkten auf der Karte schicken, ohne sie zuvor mit Wegpunkten versehen zu haben. Hierbei gibt es verschiedene Möglichkeiten.

Objektbezogen:	**Name domove getpos Name**
ID-bezogen:	**Name domove getpos (object ID)**
Koordinatenbezogen:	**Name domove [X,Y,Z]**
Markerbezogen:	**Name domove getMarkerPos "Markername"**

Möchte man eine Gruppe zu einer dort oben aufgeführten Position schicken, lässt man das **do** von **domove** weg, da sich sonst zunächst nur der Leader zu der angegebenen Position bewegen wird und die Gruppe erst folgt, wenn er sein Ziel erreicht hat. Hier ein Syntaxbeispiel für Objektbezogen:

Name move getpos Name oder **Leader Name move getpos Name**

Sehr interessante Beispiele findet man im **Kapitel 6.6** unter -Der Mapclick- und **Kapitel 6.2** -Das GPS-System-.

5.13 - Streife laufen, fahren oder fliegen

Eine Streife soll ständig um ein Lager Streife laufen bzw. fahren. Dazu weist man der Streife die Wegpunkte zu und schließt den Kreis, indem man den letzten Wegpunkt in den Bereich des ersten setzt und im Wegpunktmenü bei **Typ** den Befehl **Wiederholen** auswählt.

5.14 - Fluchtverhalten einer Einheit oder Gruppe

Hiermit legt man das Fluchtverhalten von Einheiten fest. Diese werden, wenn die Situation aussichtslos erscheint, je nach Einstellung früher oder später die Flucht ergreifen und irgendwo in den Weiten Sahranis untertauchen. Doch Achtung: Manchmal kommen sie auch zurück und greifen nochmal an.

Hierbei gelten alle Werte zwischen **0** und **1**, also auch die Dezimalwerte, wobei die **0** für kein und **1** für maximales Fluchtverhalten steht. Schreibt man in die Initzeile eines Gruppenführers diese Syntax:

this allowFleeing 0.8

bezieht sich der Befehl auf die ganze Gruppe, welche je nach Einstellung die Flucht ergreifen wird. Viel abwechslungsreicher ist die Verwendung eines Zufallwertes:

this allowFleeing (random 0.8)

5.15 - Einheiten, Objekte, Auslöser u. Marker versetzen

Einheiten, Objekte und Auslöser lassen sich auch während des Spielverlaufs beliebig versetzen oder besser ausgedrückt, an eine beliebige Positionen beamen. Dazu muss man das jeweilige zum Versetzen bestimmte Objekt zunächst mit einem Namen versehen. Danach lässt sich ein Objekt wie folgt versetzen:

Objektbezogen:	**Name setpos getpos Name**
ID-bezogen:	**Name setpos getpos (object ID)**
Koordinatenbezogen:	**Name setpos [X,Y,Z]**
Markerbezogen:	**Name setpos getMarkerPos "Marker1"**
Marker zu Marker:	**"M1" setmarkerpos getmarkerpos "M2"**
Marker zu Objekt:	**"Marker1" setmarkerpos getpos Name**
Vehikelbezogen:	**Name setpos getpos vehicle Player**
Vehikelbezogen II:	**"Marker1" setmarkerpos getpos vehicle Player**

Ein weiterer Befehl in welcher die Höhe, hier mit dem Wert **10** definiert, mit beinhaltet wäre:

Name1 setpos [(getpos Name2 select 0),(getpos Name2 select 1),10]

Name1 würde nun an die Position von **Name2** in **10** Meter Höhe versetzt werden.

Eine Gruppe versetzt man hierbei mit dem jeweiligen Befehl von oben zum Ziel:

{_x setpos getpos Name} foreach units Gruppe1

5.16 - Objekte versetzen oder höher setzen

Sämtliche in irgendeiner Form setzbaren Objekte und Einheiten lassen sich versenken bzw. höher setzen. Eine Ausnahme sind Fahrzeuge und Einheiten, welche sich nicht im Boden versenken, aber dennoch höher setzen lassen. Hiermit ist es beispielweise möglich, Soldaten in Häusern oder auf Dächern höhengenau zu platzieren. Setzt man einen Soldaten oder ein Fahrzeug in eine Höhe X, wird dieser schwerkraftbedingt wieder zu Boden fallen. Statische Objekte, wie beispielsweise ein Sandsack, würden in der Luft schweben. Hierbei gibt es verschiedene Varianten. Die in der Initzeile eines Objektes eingetragene Syntax:

this setpos [(getpos this select 0),(getpos this select 1),10];

lässt das Objekt an der im Editor gesetzten Position in einer Höhe von 10 Metern schweben. Im Multiplayer funktioniert dieser Befehl erst ab ArmA-Version 1.08!

Der Inhalt des Arrays **[ ]** wird wie folgt definiert. Die erste **( )** stellt den Erhalt der **X-Position**, die zweite **( )** den Erhalt des **Y-Position** und die Zahl nach den Klammern die **Z-Position** des Objektes dar. Ersetzt man die beiden **this**, also das **this** vor der Klammer, welches für das Objekt welches zu **this**, also dem **this** in der **[]** versetzt werden soll, durch einen Namen, wird es auch dieser Satz verständlicher.

Name1 soll zu **Name2** versetzt werden:

Name1 setpos [(getpos Name2 select 0),(getpos Name2 select 1),10];

Übersetzt heißt das: **Name1** setze zu erhaltener **X-Position** und zu erhaltener **Y-Position** von **Name2** auf Höhe **10** Meter.

Neben dem setPos- und getPos-Befehlen gibt es noch setPosASL und getPosASL Befehle, welche die Höhe des Objektes über dem Meeresspiegel wiedergeben.

5.17 - Flughöhe einer Einheit

Die Flughöhe einer Einheit lässt sich unterschiedlich bestimmen. Dies kann man an dem jeweiligen Wegpunkt der Einheit oder auch per Auslöser oder Skript bestimmen. Hierzu gilt folgende Syntax:

Name flyInHeight 120

5.18 - Punktgenaue Helikopterlandung

Setzt man für einen Helikopter einen Wegpunkt an der Position an der er landen soll, weicht die KI oftmals weit vom Landepunkt ab und landet, wo es ihr gerade möglich ist. Dies kann man umgehen, wenn man zunächst ein Heli-H setzt. Hierbei ist es vollkommen egal, ob es ein sichtbares oder unsichtbares Heli-H ist. Danach erst setzt man den Landewegpunkt direkt auf das Heli-H und wählt im Wegpunktmenü bei **Typ Transport Entladen** oder **Aussteigen** aus. Der Helikopter wird nun an der festgelegten Position landen. Eine weitere Möglichkeit bietet diese Syntaxform:

Heliname land "Positionsname"

5.19 - Einheit begibt sich in ein Gebäude

Dazu weist man der Einheit einen Wegpunkt zu, wobei man direkt auf das begehbare Gebäude, bei dem die Bezeichnung am Fadenkreuz erscheint, wenn man mit der Maus über das Gebäude fährt, klickt und im Wegpunktmenü bei **Position im Haus** die jeweilige **Position** auswählt. Die Einheit wird sich dann sofort zu der jeweiligen Position im Haus begeben.

5.20 - Einheit verlässt Gruppe oder tritt anderer bei

Dies lässt sich zum Einen mit Wegpunkten, wie im **Kapitel 1.5** –Wegpunkte einfügen– im Unterbereich **Anschließen und führen (Join and Lead)** erklärt, oder zum Anderen mit einer Syntax realisieren. Mit folgender Syntax würde **Name1** eine Gruppe verlassen:

[Name1] join GrpNull

und mit dieser einer anderen Gruppe beitreten:

[Name1] join Name2

Name1 wurde also zunächst einer **Nullgruppe**, die nicht existent ist, und danach der Gruppe **Name2** unterstellt. Für mehrere Einheiten sieht das Ganze so aus:

[Name1, Name2, Name3] join GrpNull

und danach:

[Name1, Name2, Name3] join Name4

Hiermit ist es sogar möglich seiner Gruppe einen Feindsoldaten zu unterstellen, der dann auf der eigenen Seite kämpft, nur eben wie ein Feindsoldat gekleidet ist.

5.21 - Einheit ein Ziel zuweisen

Auch hier gibt es verschiedene Definitionsmöglichkeiten, wobei die Einheiten keinen Unterschied zwischen Freund oder Feind machen. Die Syntaxes hierfür lauten wie folgt:

Name1 doTarget Name2	**Name1** wendet sich **Name2** zu
Name1 CommandTarget Name2	**Name1** befiehlt einer Einheit auf **Name2** zu zielen

Feuern

Name1 doFire Name2	**Name1** schießt auf **Name2**
Name1 doFire ObjNull	**Name1** schießt auf nichts mehr
Name1 fire "Waffentyp"	**Name1** feuert die Waffe blind ab
Name1 CommandFire Name2	**Name1** befiehlt einer Einheit auf **Name2** zu schießen

5.22 - Einheit wendet sich anderer zu

Wenn man möchte, dass eine Einheit auf ein Objekt oder eine andere Einheit schaut, regelt man das mit folgenden Syntaxes, wobei es da Unterschiede gibt. Bei der einen Befehlsform dreht sich die Einheit mit eigener Bewegung und bei der anderen Befehlsform wird sie in Blickrichtung gebeamt.

Name1 setDir 160	- Name wird in Richtung 160 gedreht
Name1 setFormDir 160	- Name dreht sich in Richtung 160
Name1 setDir getDir Name2	- Name bekommt Azimut von Name2
Name1 doWatch Name2	- Schaut zu Name2 (dreht sich bei Bedarf um)
Name1 lookAt Name2	- Schaut zu Name2 (dreht sich bei Bedarf um)
Name1 glaceAt Name2	- Schaut kurz zu Name2 (nur Kopfbewegung)

Textausgabe:

Titletext [format["Blickrichtung %1", getDir Name1], "plain down"]

5.23 - Einheit wählt Waffe

Mit folgender Syntax bringt man eine Einheit dazu die Zweitwaffe zu wählen. Natürlich muss hierbei die Bezeichnung der Waffenklasse stimmen und die Einheit die entsprechende Waffe auch im Inventar haben.

Name selectWeapon "Stinger"

5.24 - Einer Einheit Schaden zufügen bzw. heilen

Man kann Einheiten per Syntax einen direkten Schadenswert in Form eines Wertes oder auch den Schadenswert einer anderen Einheit zuweisen. Hierbei steht die **0** für keinen Schaden und die **1** für maximalen Schaden. Also tot. Die Dezimalwerte zwischen **0** und **1** stellen die Zwischenwerte dar. Bitte beachten, dass Setdamage mit einem oder auch zwei **m** geschrieben werden kann. Wobei es getDammage nur mit doppeltem **m** gibt.

Name1 wird ein Schadenswert zugewiesen:

Name1 setDamage 1 oder **Name setDammage 1**

Name1 wird der Schadenswert von **Name2** zugewiesen:

Name1 setDamage getDammage Name2

Name1 setDammage getDammage Name2

Möchte man den Schadenswert als Bedingung verwenden, sieht das Ganze so aus:

? damage Name1 >= 0.5 oder **? getDammage Name1 >= 0.5**

5.25 - Einrichten einer Todeszone

Das Einrichten einer Todeszone kann verschiedene Gründe haben. Wenn man zum Beispiel auf die Schnelle ein Schlachtfeld mit vielen Gefallenen erstellen möchte oder wenn man einen Bereich absperren möchte den eine oder alle Seiten nicht betreten dürfen. Dazu schreibt man in die Initzeile eines Auslösers, bei dem zuvor die gewünschte Größe eingestellt wurde, folgendes:

{_x setDamage 1} foreach thislist

Bei Art der Auslösung muss man nun nur noch die Seite definieren, durch die der Auslöser ausgelöst werden soll. Wenn der Auslöser mehrfach auslösen soll, muss man ihn natürlich noch auf **mehrfach** stellen.

5.26 - Das Prüfen eines Bereiches

Mit diesem Befehl kann man sich die Einheiten eines Auslöserbereiches anzeigen lassen. Je nach Auslösereinstellung ist es möglich, sich nur die jeweilige Seite oder alle Einheiten dieses Bereiches anzeigen zu lassen. Dazu verwendet man zwei Auslöser, welche wie folgt definiert werden.

Bei dem **prüfenden Auslöser** setzt man zunächst die **Achse a/b** auf die Größe des Bereiches, der geprüft werden soll. Bei **Art der Auslösung** wird die jeweilige Seite, die gelistet werden soll ausgewählt und benennt ihn bei **Name** mit **Bereich1**.

Den **Funkauslöser** definiert man bei **Achse a/b** mit **0**, stellt ihn auf **mehrfach** und gibt in die Initzeile folgendes ein:

Player sidechat format["%1",list Bereich1]

Wenn man nun den Funk benutzt, werden die Einheiten im **Bereich1** angezeigt.

5.27 - Einheiten in einem Bereich ansprechen

Mit der folgenden Befehlsform lassen sich Einheiten eines festgelegten Bereiches ansprechen. Dabei kann man dies wieder, wie im **Kapitel 5.26** schon erläutert, seitenbezogen oder komplettbezogen definieren. Hierzu muss wieder ein Bereich mit Namen festgelegt werden und die folgende Syntax irgendwo in einem Wegpunkt, Skript oder wie in **Kapitel 5.26** einem anderen Auslöser angegeben werden.

{_x setBehaviour "Stealth"} forEach list Bereich1

Alle Einheiten dieses Bereiches würden nun in Deckung gehen. Statt dem Befehl **setBehaviour "Stealth"** kann man natürlich auch einen anderen Befehl eingeben.

5.28 - Einheitsstatus speichern oder laden

Dieser Befehl lässt sich hervorragend in Kampagnen verwenden, bei denen der Spieler oder auch die anderen Einheiten bei Beginn der Folgemission den Endstatus von der vorhergehenden Mission haben sollen.

Möchte man diesen Befehl in seiner Kampagne nutzen, sollte man darauf achten, dass die Einheit, von der der Status gespeichert wird, auch noch am Leben ist. Zumindest dann, wenn dieser Status später auf den Spieler übertragen werden soll.

Mit Savestatus kann man, wie das Wort schon sagt, den Status einer Einheit speichern. Dies geht allerdings nur in Verbindung mit einer Kampagne , da der Wert direkt in der **Objects.sav** der Kampagne gespeichert wird. Im Single- sowie Multiplayerbereich ist dieser Befehl von daher nicht funktionsfähig.

Savestatus

Status1 ist eine Variable, die daher auch anders heißen kann. Mit der folgenden Syntax wird nun der Status von **Name1** unter der Variable **Status1** gespeichert.

xy=Name1 saveStatus "Status1"

Dieser Speicherwert beinhaltet nun einige Informationen dieser Einheit. Hierzu gehören unter anderem:

- die Identität
- der Gesundheitsstatus
- der Waffenstatus
- der Munitionsstatus

Wenn man jetzt einem anderen Soldaten genau diesen gespeicherten Status zuweisen will, macht man dies wie nun folgend erklärt.

Loadstatus

xy=Name2 loadStatus "Status1"

Der Einheit mit **Name2** wird dann der gespeicherte **Status1** von **Name1** zugewiesen. Das heißt, dass **Name2** genauso wie **Name1** aussieht, genau die gleiche Waffe trägt, den gleichen Munitionsstatus und den gleichen Gesundheitsstatus hat. **Name2** ist dann also ein Klon von **Name1**.

DeleteStatus

Natürlich lässt sich jeder Satus auch löschen. Dies erfolgt mit:

deleteStatus "Status1"

5.29 - Bekanntheitsgrad einer Einheit

Diese Syntax lässt sich für viele nützliche Bereiche verwenden. Zum Beispiel kann man so einer Einheit die Information über eine andere Einheit geben oder den Bekanntheitsgrad auch als Bedingung für eine dann auszuführende Aktion verwenden. Hierbei gelten wieder alle Werte zwischen **0** und **1**.

0	**Name1** hat keine Kenntnisse von **Name2**
2	**Name1** hat geringe Kenntnisse von **Name2**
3	**Name1** hat ausreichend Kenntnisse von **Name2**
4	**Name1** hat volle Kenntnisse über **Name2**

Möchte man eine Einheit über eine andere in Kenntnis setzen, weist man ihr einen den Kenntniswert wie folgt zu:

Name1 reveal Name2

Name1 hat nun Kenntnis von **Name2**.

Möchte man den Kenntniswert als Bedingung für die Auslösung eines Auslösers verwenden, schreibt man in die Bedingungszeile folgendes:

Name1 knowsAbout Name2 > 1

Wenn **Name1** nun mehr als **1** von **Name2** weiß, wird der Auslöser ausgelöst und damit die Aktionen die darin definiert wurden durchgeführt.

Die Kenntnis wird im Laufe der Zeit weniger und landet irgendwann wieder beim Wert 0. Demnach lässt sich die obige Syntax auch andersrum, also mit kleiner als, verwenden.

Name1 knowsAbout Name2 < 0.8

Dies ist ganz nützlich, wenn die Einheit immer in der Nähe der anderen bleiben soll. Ist sie das nicht oder wird sie getötet, sinkt ja der Wert irgendwann unter, wie oben definiert, **0.8** und der Auslöser wird ausgelöst.

5.30 - Freundlicher Feind

Beendet man die folgend aufgeführte Syntax für eine Einheit an, so wird sie nicht mehr vom Feind beschossen und auch auf andere Weise ignoriert werden. Hiermit kann man zum Beispiel eine feindliche Einheit als Gefangenen definieren, die ja andernfalls sofort erschossen werden würde. Das Ganze macht man dann mit:

this setCaptive true oder **Name setCaptive true**

Zurücksetzen kann man das dann wieder, indem man **true** wieder auf **false** setzt.

5.31 - Befreundete Parteien

Die feindlichen Seiten in Armed Assault lassen sich frei bestimmen. Hierbei ist es nun möglich West und Ost auf einer Seite gegen den Rest kämpfen zu lassen oder die Zivilisten als böse zu definieren. Man hat dabei einen so großen Spielraum, dass man dies sogar wieder mit einem Wahrscheinlichkeitswert(Random) belegen kann. Somit weiß man nie genau, wie der Feind am nächsten Tag so drauf ist.

Hierbei sind folgende Seiten frei gegeneinander definierbar:

West - Ost - Guerrila - Civilian - Enemy - Logic

Es kann auch festgelegt werden, dass die eine Seite friedlich zur anderen ist, aber nicht umgekehrt. Das heißt, dass die eine Seite sofort das Feuer eröffnet, aber die andere Seite schießt nicht zurück. Statt **Guer** kann übrigens auch **Resistance** verwendet werden.

Der SetFriend-Wert bewegt sich auch hierbei wieder zwischen **0** und **1**, wobei alle Werte über **0.6** für befreundet, und alle unter **0.6** für feindlich stehen.

Folgend mal ein paar Syntaxbeispiele:

WEST setFriend [EAST,1]

Jetzt wäre West freundlich zu Ost, aber nicht umgekehrt, dazu braucht man eine zweite Syntax dieser Form.

WEST setFriend [EAST,0.7]; EAST setFriend [WEST,0.7]

Nun sind beide Seiten freundlich zueinander, was sich ja im Missionsverlauf dank dieser Syntax noch ändern kann oder auch nicht. Setzt man den Wert wieder runter, verfeinden sich beide Parteien wieder.

WEST setFriend [EAST,0]; EAST setFriend [WEST,0]

Zufallsbezogen

Natürlich ist auch hier wieder alles per Zufall möglich, was eben dieses Spiel ausmacht. Man kann alles auch so definieren, dass man beim Missionstart gar nicht weiß, ob sein gegenüber Freund oder Feind ist. Dazu verwendet man dann diese Syntaxform:

GUER setFriend [EAST,(Random 0.9)]

Hier wurde nun **Random** mit eingetragen, welcher einen Zufallswert für **0.9** ermittelt. Dieser kann dann mit **0.9** (freundlich), aber genauso gut mit **0.3** (feindlich) ausgegeben werden.

Deathmatchbezogen

Dies ist im **Kapitel 7.6** ausführlich erläutert. Kurz, man kann eine Seite auch mit sich selbst verfeinden.

EAST setFriend [EAST,0]

5.32 - Der Alarm

Mit den in **Kapitel 5.27** erläuterten Syntaxes lassen sich nun einige verschiedene Alarmformen realisieren. Natürlich liegt es am Ende immer an der Mission, wann ein Alarm ausgelöst werden soll. Hier mal ein paar Beispiele.

Beispiel 1 – Alarmierung bei Entdeckung in einem Auslöserbereich
Hierzu setzt man einen Auslöser mit folgenden Einstellungen auf die Karte:

Aktivierung	BLUFOR
	Von Osten entdeckt
Effekte	Alarm

Wird in diesem Auslöserbereich nun eine westliche Einheit von östlichen Einheiten entdeckt, wird der Auslöser aktiviert und der Alarm ausgelöst.

Beispiel 2 – Einheitsbezogene Alarmierung
Hierbei soll nur Alarm ausgelöst werden, wenn eine bestimmte Einheit entdeckt wird.

Aktivierung	Den Auslöser mit der Einheit oder dem Spieler verbinden
	Von Osten entdeckt
Effekte	Alarm

Beispiel 3 – Alarm, wenn Einheit oder Objekt nicht mehr present (tot) ist

Aktivierung	Den Auslöser mit der Einheit oder dem Objekt verbinden
	Nicht vorhanden
Effekte	Alarm

Beispiel 4 – Alarm, wenn die Gruppe namens Grp1 kleiner als Wert ist

Aktivierung	Keine
Achse a/b	0
Bedingung	Count Units Grp1 < Wert
Effekte	Alarm

Zusätze
Bei allen Alarmvarianten kann man diverse Aktionen über die **Aktivierungszeile** starten lassen. Zum Beispiel, dass alle Einheiten des Alarmbereiches namens **Bereich1** über die Einheit, hier der Spieler(Player), informiert werden.

{_x reveal Player} foreach list Bereich1

5.33 - Tod als Bedingung

Um zu prüfen ob eine Einheit noch am Leben ist, benötigt man einen Auslöser mit den folgenden aufgeführten Einstellungen. Natürlich gibt es auch hier wieder verschiedene Varianten. Hier mal eine sehr oft verwendete Methode.

Aktivierung	Keine
Achse a/b	0
Bedingung	! (alive Name) oder not alive Name

Der Auslöser prüft nun durch **Achse 0/0** global, ob Name noch lebt und löst bei Tod die bei **Aktivierung** oder **Effekte** gesetzten Aktionen aus.

5.34 - Distanz zweier Einheiten oder Objekte

Manche Situationen machen es notwendig, dass man die Distanz zweier Einheiten oder Objekte als eine Bedingung für die Auslösung eines Auslösers oder ähnliches benötigt wird. Die Syntaxes hierfür sehen unter anderem wie folgt aus:

Player distance Jeep1<= 50 oder **Player distance Jeep1 == 50**

Lokaler Variable Wert zuweisen: **_distance = Player distance Jeep1**

Textausgabe:

Titletext [format["%1 Meter", Player distance Jeep1], "plain down"]

5.35 - Einem Fahnenmast eine Flagge zuweisen

Alle Fahnenmasten lassen sich individuell mit Fahnen bestücken. Das geht auch, wenn sie bereits eine Fahne haben, wie es ja in ArmA der Fall ist. Dazu packt man die jeweilige Fahne in den Missionsordner und gibt in der Initzeile der Fahne einfach den Namen der Datei mit der Dateierweiterung an.

this setflagtexture "Name.paa"

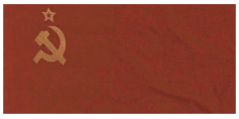
Möglich sind die Dateiformate **PAC**, **PAA** und **JPG** in der Größe **512 x 256** Pixel.
Spielintern sind folgende Flaggen mit dieser Syntax und diesem Pfand setzbar:

this setFlagTexture "\ca\misc\data\usa_vlajka.paa"

USA
usa_vlajka_co.paa

Russland
rus_vlajka_co.paa

Sowjetunion
rus_vlajka_co.pac



SLA
sever_vlajka.paa

Racs
jih_vlajka.paa

5.36 - Brennende Feuerstelle

Möchte man ein Feuer erst zu einem späteren Zeitpunkt anzünden oder erlöschen, gibt man dem Feuer zunächst einen Namen und nutzt folgende Syntax:

this inflame true bzw. **Name inflame false**

5.37 - Switchbare Einheit hinzufügen oder entfernen

Mit den folgenden Befehlen ist es möglich dem Spieler während einer Mission switchbare Einheiten zuzuweisen oder auch zu entfernen. Folgend wird eine Einheit spielbar

AddSwitchableUnit Name

und wieder unspielbar gemacht:

RemoveSwitchableUnit Name

Möchte man zu einem anderen Spieler wechseln, nutzt man folgende Syntax:

selectPlayer Name

5.38 - Spielerseite , -namen, -typ auslesen bzw. ausgeben

Die Seite eines Spielers im Mehspielerbereich lässt sich wie folgt auslesen:

? Side Player == West

Name Player == "Mr-Murray"

Textausgabe:

Titletext [format["%1 -Soldat", Side Player],"plain down"]

Titletext [format["Hey %1", Name Player],"plain down"]

Titletext [format["Typ: %1", typeOf Player],"plain down"]

5.39 - Spielereingabe unterdrücken

Für einige Dinge, wie zum Beispiel Sequenzen, ist es manchmal von Vorteil die Spielereingabe zu unterdrücken. Man sollte aber darauf achten, dass die Sperre nicht zu lange aktiv bleibt, da es den Spieler evtl. zum Spielabbruch bewegen könnte. Zum Aktivieren setzt man disableUserInput auf **true** und zum Deaktivieren wieder aus **false**.

disableUserinput true

5.40 - Karte auf den Monitor erzwingen

Das Erzwingen der Karte auf den Monitor ist ganz gut für Sequenzen oder ähnliches zu gebrauchen. Hier hat man dann die Möglichkeit die verschiedensten Dinge zu simulieren. Zum Beispiel Truppenbewegungen und vieles mehr.

ForceMap true

Jetzt hat man u.a. die Möglichkeit Marker auf der Karte wandern zu lassen, um den Spieler in eine Situation einzuweisen. Es bietet sich dabei an die Spielereingabe zu unterdrücken.

5.41 - Sichtweite ändern

Für manche Bereiche oder Aktionen bietet es sich an die Sichtweite zu erhöhen oder herabzusetzen. Zum Beispiel bei einer Zwischensequenz, auf einem Berg, wo eine gute Aussicht dargestellt werden soll. So könnte man für diese kurze Szene die Sichtweite kurzzeitig erhöhen und später wieder herabsetzen um Performance einzusparen. Als Sichtweite kann hierbei zwischen **500** bis **10.000** gewählt werden.

SetViewDistance 500

5.42 - Wetter einstellen

Das Wetter wird ja zunächst ganz normal im Editor definiert. Möchte man es aber im Missionsverlauf oder einer Sequenz ändern benötigt man folgend aufgeführte Syntax mit Werten zwischen **0** und **1**. Je nach Mission bietet es sich immer an ein Zufallwetter zu bestimmen. Dies verbunden mit Zufallszeit und Zufallsnebel bringt gleich ein wenig mehr Abwechslung in die Mission.

120 SetOvercast 0.8

Der erste Wert stellt die Zeit in Sekunden dar, in der das Wetter umschwenken soll. Der zweite Wert steht für die Wetterart. Ist dieser **0**, scheint die Sonne und der Himmel ist Wolkenleer. Mit jedem Dezimalwert, der nun dazu kommt, tauchen mehr Wolken am Himmel auf. Bei einem Wert von **0.5** ist der Himmel komplett bewölkt und bei einem Wert von **1** ist mit starken Gewittern zu rechnen.

Zufallswetter:
Dieses lässt mit einer Zufallssyntax bestimmen, welches den Wetterwert per Zufall bestimmt und festlegt. Lässt man diese Syntax gleich zu Missionsbeginn starten, ist bei jedem Missionsstart mit einem anderen Wetter zu rechnen.

0 setOvercast (Random 0.8)

Regen:
Der Regen lässt sich auch unabhängig von SetOvercast einstellen.

0 setRain 0.8

10 setRain (Random 0.8)

Nebel:
Die Nebelwerte werden nach gleichem Verfahren wie beim Wetter oder Regen bestimmt. Jedoch lautet die Syntax hierfür:

10 setFog 0.6

60 setFog (Random 0.8)

5.43 - Datum und Uhrzeit einstellen

Auch dies ist während eines Missions- oder Kampagnenverlaufs möglich und kann dabei entweder mit einem festen Wert oder einer Zufallszeit bestimmt werden.

Jahr, Datum, Uhrzeit:
Mit folgend aufgeführter Syntax bestimmt man gleich das Jahr, den Tag, die Stunde und die Minute auf einmal. Dies erfolgt auf in dieser Reihenfolge.

setDate [2006, 11, 30, 9, 0]

Hier wurde jetzt der 30.11.2006 mit der Uhrzeit 09:00 bestimmt. Natürlich lassen sich hier auch wieder einige Werte per Zufall generieren. Hier mit Uhrzeitbeispiel:

setDate [2006, 11, 30, (ceil Random 8), (Random 60)]

Uhrzeit:
Die Uhrzeit kann man mit dieser Syntax auch einzeln bestimmen.

SkipTime 1	Stellt die Zeit um 1 Stunde vor
SkipTime –1	Stellt die Zeit um 1 Stunde zurück

5.44 - Zeitlupe oder Zeitsprint

Mit der nun folgenden Syntax ist es möglich das Spiel zu verlangsamen, also in Zeitlupe zu versetzen oder aber auch zu beschleunigen. Gerade bei Zwischensequenzen kann man hiermit geniale Zeitlupenszenen gestalten. Des Weiteren ist es aber auch möglich einen Bullet Mode, wie in **Kapitel 6.10** erläutert, in die Mission einzubauen, was zwar nicht sonderlich realistisch ist, aber dennoch als Feature verwendet werden kann.

Zeitlupe:
Alle Werte unter **1** stehen hierbei für Zeitlupe, wobei gerade die Dezimalwerte den Effekt ausmachen. Je kleiner der Wert, desto langsamer der Ablauf.

SetAccTime 1.0	- Normale Zeit
SetAccTime 0.0500	- Zeitlupe

Zeitsprint:
So, wie man die Zeit verlangsamen kann, kann man sie auch beschleunigen, wofür es aber eigentlich im Missionsbereich selbst keine richtige Verwendung geben sollte. Hierbei gelten alle Werte zwischen **1** und **8**, welche der KI und dem Spieler schnelle Füße verleihen.

5.45 - Einheiten und Objekte erzeugen

Es ist während eines Missionsablaufs möglich Einheiten, ganze Gruppen oder auch Objekte frei auf der Karte zu erzeugen und auch wieder zu löschen. Damit kann man eine Menge Performance einsparen und die Einheiten erst erzeugen, wenn sie benötigt werden. Dabei gibt es verschiedene Varianten. Hier ein paar Beispiele.

Objektbezogen:
In folgendem Beispiel wird ein Geschütz mit Typ **D30** an Position XYZ erzeugt:

Ari1="D30" createVehicle [x,y,z]

Mit dem setDir-Befehl richtet man dieses dann zusätzlich aus.

Ari1 setDir 190

Es kommt vor, dass ein Vehikel nach dem Erzeugen nicht nutzbar ist. Dafür gibt man dann zusätzlich beim Erstellen den folgenden Befehl mit an:

Ari1 lock false

Somit ist es nun auf jeden Fall nutzbar. Möchte man ein Objekt direkt an der Position eines anderen Objektes erzeugen, dann sieht das so aus:

Bombe="SH_120_HE" createVehicle position Player

Folgend wird eine Bombe über der Einheit **S1** in der Höhe von **50** Metern erzeugt.

Bombe="SH_120_HE" createVehicle [(getpos S1 select 0),(getpos S1 select 1),50]

Einheitsbezogen:
Zum Erzeugen einer Einheit wird die Zeile schon etwas länger. Hier wird ein Westsniper namens **Name1** auf der Position seines Leaders namens **S1**, der vorher schon auf der Karte sein muss, erstellt. Der Leader könnte dabei auch eine Logik sein. Der nun erzeugte Soldat hat einen Skillwert von **0.4** und ist vom Dienstgrad her **Corporal**.

"SoldierESniper" createUnit [position player, S1, "Name1=this",0.4,"Corporal"]

Man kann das natürlich auch anders definieren, indem man als Position einen Marker angibt, welchen man zuvor auf die Karte an die Position des Leaders gesetzt hat. Diesem würde man bei **Achse a/b** einfach den Wert **0** geben, damit der Marker nicht sichtbar ist

"SoldierESniper" createUnit [getMarkerPos "Marker1",EGrp1,0.8,"Corporal"]

Dieser hat jetzt mal keinen Namen zugewiesen bekommen, weil man das nur machen muss, wenn man diese Einheit später auf irgendeine Art ansprechen möchte. Der Leader hieß in diesem Fall **EGrp1**.

Gruppenbezogen:
Neben einzelnen Einheiten lassen sich auch Gruppen erstellen. Dazu kann man die Gruppen gleich individuell mit Namen versehen und sie so dann später ganz einfach ansprechen. Als Beispiel soll hier folgendes Skript dienen, bei welchem eine Gruppe an der Position des Markers "GrpOneM" erstellt wird, den man zuvor auf die Karte gesetzt hat.

```
GrpOne = Creategroup EAST;
_Leader="SquadLeaderE" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Sergant"];
_Unit2="SoldierEB" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit3="SoldierEB" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit4="SoldierEG" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit5="SoldierEMG" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit6="SoldierEAT" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit7="SoldierESniper" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
 exit
```

Wie man sieht, wurde hier der ersten Einheit, die lokal **Leader** benannt wurde, ein höherer Dienstgrad zugewiesen, was sie automatisch zum Leader der Gruppe macht. Der Name der Gruppe ist nun, wie am Anfang des Skriptes definiert, **GrpOne**. Damit diese Gruppe nun auch in der Mission richtig agiert, muss man noch einen weiteren, sehr wichtigen Punkt beachten.

Merke:
Erstellt man feindliche Einheiten erst im Laufe der Mission, ohne dass zuvor irgendwelche Einheiten dieser Seite auf der Karte platziert waren, muss man der Seite ein Center zuweisen, damit diese Einheiten miteinander kommunizieren können. Danach verwendet man den SetFriend-Befehl und verfeindet die Seiten miteinander. Macht man das nicht, wird die KI nicht auf gegnerische Einheiten feuern. Es bietet sich hierbei an, das Center und den SetFriend-Befehl in der Init.sqs zu initialisieren. Hat man jedoch von Anfang an Einheiten aller Kriegsparteien auf der Karte, so werden diese Center beim Missionsstart automatisch von der Engine erstellt. Folgend die Einträge in einer Beispiel-Init.sqs:

Init.sqs

```
Createcenter EAST
Createcenter WEST

WEST setFriend [EAST,0]
EAST setFriend [WEST,0]
```

Center
So wie man ein Center erstellen kann, kann man es mit **deleteCenter SEITE** auch wieder löschen, was aber eher unsinnig und unwirksam ist.

5.46 - Flares, Rauch und Explosionen erzeugen

Zum Erzeugen einer Flare oder einer Rauchgranate über einem Objekt oder einer XYZ-Position gilt zwar der gleiche Befehl, wie im Kapitel 5.45, aber die Klassennamen weichen von den Magazinnamen ab. Deshalb hier nochmal explizit.

Flare1="F_40mm_Green" createVehicle [x,y,z]

Oder über oder an der Position eines anderen Objektes:

Rauch="Smokeshell"
createVehicle [(getPos Name1 select 0),(getPos Name1 select 1), 10]

Wobei dies in einer Zeile stehen muss, was hier nicht möglich ist. Oder in kurz:

Rauch=" Smokeshell" createVehicle position Name

Merke!
Flares sollten generell ab einer Höhe von 100 Metern erzeugt werden!
Dabei stehen folgende Flares

F_40mm_White	**F_40mm_Red**
F_40mm_Green	**F_40mm_Yellow**

und folgende Rauch- und Gewehrgranaten zur Auswahl:

Smokeshell	**SmokeshellRed**
SmokeshellGreen	**B_40mm_HE**

Natürlich kann man statt einer Flare oder einer Rauchgranate auch ganz andere Arten erzeugen. Zum Beispiel eine **SH_125_HE** oder ähnliches für eine Explosion. Dazu bitte im **Kapitel 3.10** nachschlagen und die Shellklassen mit Hilfe eines Hints ausgeben lassen.

In der Vorschau sieht das dann später etwa so aus:

5.47 - Einheiten und Objekte löschen

Während man zum Löschen von Objekten nur eine Syntaxform verwendet, benötigt man zum Löschen von Einheiten verschiedene Syntaxformen. Wenn man beispielsweise nur den Bordschützen eines Helikopters löschen will, der ja nicht mit einem Namen versehen wurde. Die Syntaxes schauen wie folgt aus:

deleteVehicle Name - Löscht alles, was einen festen Namen besitzt.

Möchte man nun den Bordschützen des Helikopters löschen, muss dieser das Fahrzeug zunächst verlassen. Vorher gibt man ihm aber noch einen Namen, indem man folgende Syntax in die Initzeile des Helis schreibt:

Name=(Gunner Heli1)

Jetzt lässt man ihn aussteigen, wobei man hier auch **(Gunner Heli1)** schreiben kann.

Name action ["Eject", Heli1] oder einfach **doGetOut Name**

Der Name wird erst beim Löschen wichtig, da **(Gunner Heli1)** nicht funktionieren würde.

deleteVehicle Name

Außer dem **Gunner** gibt es noch den **Commander**, den **Driver** und die **Crew**.

Einheiten in Bereich löschen

Dazu setzt man einen Auslöser beliebiger Größe und benennt ihn beliebig(hier: Zone1). Dann wählt man die Seite aus, die gelöscht werden soll oder stellt ihn mit **Jeder** auf alle Seiten und Objekte. Die Syntax für das Löschen schreibt sich dann:

{deleteVehicle _x} forEach list Zone1

5.48 - Funkmenü verändern

Hiermit hat man die Möglichkeit während einer Mission die Einträge im Funkmenmü umzubenennen. Diese sind von der Engine her von **1** bis **10** durchnummeriert.

1=Alpha, 2=Bravo, 3=Charlie, ...

Umbenennen kann man Diese mit folgender Syntax:

1 SetRadioMsg "Alpha-Team"

Der erste Eintrag von Radio Alpha wurde nun in **Alpha Team** umbenannt. Möchte man einen leeren Funkmenüeintrag haben, setzt man statt einem Text einfach ein Leerzeichen zwischen die Anführungsstriche.

Die Funkmöglichkeit ist zwar eigentlich noch da, aber der Spieler sieht sie nicht und denkt, dass sie nicht verfügbar ist. Statt dem Text wäre dann nichts zu sehen.

Dies kann man nutzen, wenn zum Beispiel ein Team ausfällt oder ähnliches.

5.49 - Einer Gruppe ein Rufzeichen zuweisen

Mit setGroupId ist es möglich verschiedenen Gruppen verschiedene Rufzeichen zuzuweisen. Somit weiß man immer genau, wer gerade gesprochen hat oder einen Sidechat abgegeben hat. Mit Armed Assault hat man im Gegensatz zu Operation Flashpoint nun auch die Möglichkeit seinen Gruppen uneingeschränkt eigene Namen zu geben. und eine freie Farbe zu definieren. Hier die Syntaxbeispiele:

Standard:

```
Name setGroupid ["Alpha","GroupColor0"]
```

Freier Text:

```
Name setGroupid ["Assault-Team-Alpha","GroupColor2"]
```

Standardmäßig stehen folgende Werte für Gruppenname und -farbe zur Auswahl:

Gruppennamen:

"Alpha"	"Golf"	"Convoy"
"Bravo"	"Hotel"	"Guardian"
"Charlie"	"Kilo"	"November"
"Delta"	"Yankee"	"Two"
"Echo"	"Zulu"	"Three"
"Foxtrot"	"Buffalo"	

Gruppenfarben:

0 – keine Farbe	4 – Blau
1 – Schwarz	5 – Gelb
2 – Rot	6 – Orange
3 – Grün	7 – Rosa

Tipp:

Seitens des Spiels gibt es ab und dann Probleme, dass die Farbe nicht ausgegeben wird, was aber schon seit Operation Flashpoint bekannt ist. Dies umgehen wir einfach, indem wir beides in eine Zeile schreiben. Die Farbe gibt man dabei manuell an und lässt den hinteren Part, wo man diese normalerweise angibt, einfach offen. Bei folgendem Beispiel heißt das Team Assault-Team-Alpha und hat die Farbe Red bekommen.

```
Name setGroupid ["Assault-Team-Alpha - Red",""]
```

5.50 - Funkspruch abgeben

Hierbei gibt es verschiedene Möglichkeiten diese auszugeben. Dabei können diese global, seiten-, gruppen- oder vehikelbezogen angezeigt werden. Hier die Syntaxform mit den verschiedenen Varianten:

Name Sidechat "Test 1-2"	**Name** spricht zu seiner Truppenseite
Name Groupchat "Test 1-2"	**Name** spricht nur zu seiner Gruppe
Name Globalchat "Test 1-2"	**Name** spricht zu allen Seiten
Name Vehiclechat "Test 1-2"	**Name** spricht zu Leuten innerhalb seines Fahrzeuges

Möchte man einen Funkspruch durch das Hauptquartier abgeben lassen:

[Seite,"HQ"] sidechat "Move to your position and wait for further orders!"

5.51 - Sound erstellen

Was in Operation Flashpoint nicht möglich war, wurde in Armed Assault berücksichtigt. Das trifft neben den vielen anderen neuen Möglichkeiten auch auf diesen Bereich zu. Man hat jetzt die Möglichkeit, die spielinternen Sounds an einer bestimmten Position zu erstellen. Im folgenden Beispiel wird der Sound namens LittleDog an der Position des Spielers erstellt.

Hund1=createSoundSource ["LittleDog",position Player,[],10]

Der Wert **10** steht dabei für die Entfernung vom Spieler. Natürlich kann man diesen jetzt beliebig versetzen, da er beim Erstellen den Namen **Hund1** bekommen hat und somit ansprechbar ist. Das geht dann, wie schon bekannt, mit:

Hund1 setpos getpos Name

So kann man sich nun beliebig die Sounds aus ArmA erstellen lassen, ohne irgendwas auf der Karte setzen zu müssen. Die Sounds findet man hierbei unter **Leer/Sounds** oder in einem Auslöser bzw. Wegpunkt unter Effekte. Unter anderem gibt es folgende Sounds:

Stream, Alarm, BadDog, BirdSinging, Chicken, Cock, Cow, Crow, Crickets1, Crickets2, Crickets3, Crickets4, Dog, Frog, Frogs, LittleDog, Music, Owl, Wolf

Möchte man eigene Sounds mit diesem Befehl irgendwo erstellen, so müssen diese erst in der Description.ext vordefiniert werden und können dann ganz normal mit der Angabe des in der Description.ext festgelegten Namens aufgerufen werden.

MeinSound=createSoundSource ["Soundname",position Player,[],10]

5.52 - Eigenen Sound einbinden

Wenn man eigenen Sound oder Musik in eine Mission einbinden möchte, so geht das nur über die Description.ext, die bereits im **Kapitel 2.3** erklärt wurde. Hier nun die explizite Erklärung zum Thema Musik und Sounds einfügen.

Zunächst legt man im Missionsordner einen Ordner namens Sounds und einen Ordner namens Music, natürlich klein geschrieben, an und packt dort seine Sounddateien rein. Die Description bietet dafür verschiedene Unterbereiche, so dass man diese einzeln steuern kann, wenn man zum Beispiel die Melodie von der Lautstärke her runterfadet, soll der Funkspruch, der gleichzeitig läuft seine Lautstärke behalten. Deshalb gibt es dafür verschiedene Unterbereiche(Soundklassen). Die Sound- oder Musikdefinierung sieht in der Description.ext etwa wie folgt aus:

Description.ext

```
// ===  Musik ==========================================>
class CfgMusic
{
	tracks[]= { Track1,Track2 };

	class Track1
	{
		name = "Track1";
		sound[] = {\music\track1.ogg, db+0, 1.0};
	};

	class Track2
	{
		name = "Track2";
		sound[] = {\musik\track2.ogg, db+0, 1.0};
	};
};

// ===  Sounds ========================================>
class CfgSounds
{
	tracks[]= { Artillerie };
	class Artillerie
	{
		name = " Artillerie ";
		sound[] = {\sounds\ artillerie.ogg, db+0, 1.0};
	};
};
```

Weiter auf nächster Seite.

```
// ===  Radio/Funk ==========================================>
class CfgRadio
        {
        sounds[] = { };
        };
// ===  Umgebung ============================================>
class CfgSFX
        {
        sounds[] = {};
        };
class CfgEnvSounds
        {
        sounds[] = {};
        };
```

Wie man oben sehen kann, sind dann noch Bereiche wie Funk (Radio) und zwei verschiedene Umgebungssoundklassen definiert.

```
class CfgMusic
{
    tracks[]= { Track1,Track2 };

    class Track1
    {
    name = "Track1";
    sound[] = {\music\track1.ogg, db+0, 1.0};


    titles[] = { }; };
};
```

- Klasse CfgMusic

- Trackliste

- Klasse Track1

- Name Track1
- Quelle,
  db = Laustärke,
  1.0 = Geschwindigigkeit
- Texteinblendung zum Sound

Mit **db** hat man die Möglichkeit die Lautstärke der Sounddatei festzulegen. Hierbei kann man kann gut zu laute oder zu leise Dateien dementsprechend anpassen.

Die Geschwindigkeit, welche oben mit **1.0** festgelegt ist, bestimmt die Geschwindigkeit mit der die Sounddatei abgespielt wird. Somit kann man Sprachsamples höher oder tiefer klingen lassen. Doch sollte man den Wert nicht zu hoch setzen, sonst hat man schnell den Micky Mouse Effekt.

Möchte man dann eine Sounddatei im Editor abspielen, so wählt man die im Wegpunkt oder einem Auslöser im Untermenü Effekte aus. Dazu muss man die Mission erst neu geladen oder gespeichert haben, damit ArmA die Description neu einlesen kann. Die nun definierten Sounds lassen sich natürlich auch per Syntax starten.

PlayMusic "Track1"

PlayMusic ["Track1", 30] (Spielt Track1 ab Sekunde 30)

PlaySound "Artillerie"

Name say "Soundname"

Die lassen sich natürlich auch, wie schon erwähnt in der Laustärke runterfaden.

10 Fadesound 0.5 **10 Fademusic 1** **0 FadeRadio 0.1**

Der erste Wert steht für die Zeit in der das geschehen soll und der zweite Wert für den jeweiligen Lautstärkenwert, den der Sound erhalten soll.

Sounds mit Text verknüpfen

Sounds lassen sich ohne Weiteres mit Text verknüpfen. Dabei kann man auf einen Stringtablewert zurückgreifen oder einen eigenen Text direkt in der Description.ext mit angeben. Der Text wird dann beim Abspielen des Sounds mit angezeigt. Im Normalfall nutzt man das bei der Wiedergabe von Sprach- oder Funksounds.

In der Description.ext sieht das dann etwa wie folgt aus:

```
// ===  Radio ================================================>
class CfgRadio
{
 sounds[] = {RadioMsg1, RadioMsg2};

 class RadioMsg1
                    {
                     name = "";
                     sound[] = {"\sound\ radiosound1.ogg", db+10, 1.0};
                     title = "It´s done.  I am ready for further orders.";
                    };

class RadioMsg2
                    {
                     name = "";
                     sound[] = {"\sound\radiosound2.ogg", db+10, 1.0};
                     title = {$STR_RADIO_2};
                    };
};
```

5.53 - Identität festlegen

Es lässt sich für jede Einheit in einer Mission eine Identität festlegen. Dies ist aber nicht unbedingt ratsam, weil es ein riesiger Aufwand wäre. Deshalb sollte man sich mit dem Vergeben von Identitäten lieber auf die Schlüsselpersonen beschränken.

Eine Identität wird zunächst in der Description.ext vordefiniert, bevor sie eingebunden werden kann. Das Ganze sieht dann ungefähr so aus.

```
class CfgIdentities
{
	class MrMurray
	{
		name = "MrMurray";
		face = "Face33";
		glasses = "none";
		speaker = "Dan";
		pitch = 1.00;
	};

	class Memphisbelle
	{
		name = "Memphisbelle";
		face = "Face10";
		glasses = "none";
		speaker = "Howard";
		pitch = 1.00;
	};

	class Dan
	{
		name = "Dan";
		face = "Face22";
		glasses = "none";
		speaker = "Russell";
		pitch = 1.00;
	};

};
```

Wie man erkennen kann, kann man die Namen frei vergeben. Lediglich die Gesichter und der Sprecher sind hierbei vorgegeben. Bitte auch wieder auf die Einhaltung der Klammern in der Description.ext achten!

Möchte man im Spiel dann die Identität der jeweiligen Einheit zuweisen, schreibt man lediglich den vordefinierten Namen mit der Syntax in die Initzeile:

Name1 SetIdentity "DeinName" oder **this SetIdentity "Mr-Murray"**

Diese Indentität lässt sich in Verbindung mit einer Kampagne auch speichern und später wieder laden. Der Wert wird dabei direkt in der **Objects.sav** der Kampagne gespeichert und kann mit **deleteIdentity "xyz"** ganz leicht wieder gelöscht werden.

Name1 saveIdentity "Name1Save" oder **Name1 loadIdentity "Name1Save"**

Man kann einer Einheit auch nur ein Gesicht zuweisen, das geht dann hiermit:

Name1 SetFace "Face33" oder **this SetFace "Face33"**

In Armed Assault stehen einem **57 Faces**, die mit **Face1** bis **Face53** und **FaceR01** bis **FaceR04** durchnummeriert sind zur Auswahl. Hat man eine eigene Facedatei im Missions- beziehungsweise Userordner, weist man diese wie folgt zu:

Name1 setFace "MeinBild.jpg"

Mögliche Größen: **1024*1024**; **512*512**; **256*256**

Brille (Glasses):

Hier eine Übersicht der Brillenarten, die man in ArmA vergeben kann:

glasses = "sunglasses";	Normale Brille
glasses = "spectacles";	Sonnenbrille
glasses = "none";	Keine Brille

Sprecher:

Folgende Sprecher stehen bis jetzt in ArmA zur Auswahl:

Amy	**Dan**	**Howard**	**Robert**	**Ryan**
Brian	**Dusan**	**Mathew**	**Russell**	

Pitch:

Mit dem Pitchwert (pitch = 1.00) wird hierbei die Sprachhöhe oder -tiefe definiert.

5.54 - Mimiken

Mit folgenden Befehlen kann man den Einheiten Mimiken zuweisen. Das bedeutet, dass man ihnen verschiedene Stimmungen wie fröhlich, ernst oder traurig zuweisen kann.

Name1 setMimic "Smile"

Normal - **Surprise** - **Agresive** - **Hurt** - **Ironic** - **Smile** - **Cynic** - **Angry** - **Sad**

Folgende Animation verändert die Gesichtszüge mit den Werten **0** bis **1**.

Name1 setFaceAnimation 0.5

5.55 - Der Actionbefehl

Die Actionbefehle weisen der jeweiligen Einheiten einen Befehl zu, was diese zu machen hat. Dabei gibt es verschiedene Varianten wie man diese definieren kann. Hier die nähere Erläuterung dazu:

Object:	Die Einheit(Name), die diese Aktion ausführen soll. Nimmt man ein Fahrzeug, wird der Commander gewählt.
Type:	Name der Aktion (siehe Actionbefehlsübersicht)
Target:	Meistens der gleiche Name, wie Name

Syntaxformen:
Hier die verschiedenen Syntaxformen, die dafür in Frage kommen können:

Name action [<type>]

Name action [<type>, <target>]

Name action [<type>, <target>, zusätzliche Parameter]

Für zusätzliche Parameter trägt man beispielsweise die Waffe oder die Magazine ein, wie es in der Liste beschrieben ist. Neben **Action** gibt es noch **Fire**, was sich aber auf die wenigsten Befehle bezieht. Folgend mal ein paar Beispiele:

Name fire

["PipebombMuzzle","PipebombMuzzle", Pipebomb]
- legt Sprengsatz

["M203Muzzle", "M203Muzzle", "1Rnd_HE_M203"]
- schießt Granate

["M203Muzzle", "M203Muzzle", "FlareGreen_M203"]
- schießt Flare

["HandGrenadeMuzzle","HandGrenadeMuzzle","HandGrenade"]
- wirft Granate

["SmokeShellRedMuzzle","SmokeShellRedMuzzle","SmokeShellRed"]
- Wirft Rauch

["bombLauncher","bombLauncher","6Rnd_GBU12_AV8B"]
- Harrier wirft Bombe ab

Name action

["TouchOff",Name]	- löst Sprengsatz aus
["Eject",Heli1];	- steigt aus
["Hidebody", Name2]	- versteckt Leiche
["CancelAction", Name]	- bricht Aktion ab

Actionbefehlsübersicht

Folgend nun eine Befehlsübersicht über die wichtigsten Aktionbefehle. Einige funktionieren von Haus aus nicht bzw. sind von Haus aus noch nicht aktiviert worden.

["None", <target>]
["GetInCommander", <target>]
["GetInDriver", <target>]
["GetInGunner", <target>]
["GetInCargo", <target>]
["Heal", <target>]
["Repair", <target>]
["Refuel", <target>]
["Rearm", <target>]
["GetOut", <target>]
["LightOn", <target>]
["LightOff", <target>]
["EngineOn", <target>]
["EngineOff", <target>]
["SwitchWeapon", <target>, <weapon index>]
["UseWeapon", <target>, <weapon index>]
["TakeWeapon", <target>, <weapon name>]
["TakeMagazine", <target>, <magazine type name>]
["TakeFlag", <target>]
["ReturnFlag", <target>]
["TurnIn", <target>]
["TurnOut", <target>]
["WeaponInHand", <target>, <weapon name>]
["WeaponOnBack", <target>, <weapon name>]
["SitDown", <target>]
["Land", <target>]
["CancelLand", <target>]
["Eject", <target>]
["MoveToDriver", <target>]
["MoveToGunner", <target>]
["MoveToCommander", <target>]
["MoveToCargo", <target>]
["HideBody", <target>]
["TouchOff", <target>]
["SetTimer", <target>]
["Deactivate", <target>]

["NVGoggles", <target>]
["ManualFire", <target>]
["AutoHover", <target>]
["StrokeFist", <target>]
["StrokeGun", <target>]
["LadderUp", <target>, <ladder index>, <ladder position>]
["LadderDown", <target>, <ladder index>, <ladder position>]
["LadderOnDown", <target>, <ladder index>, <ladder position>]
["LadderOnUp", <target>, <ladder index>, <ladder position>]
["LadderOff", <target>, <ladder index>]
["FireInflame", <target>]
["FirePutDown", <target>]
["LandGear", <target>]
["FlapsDown", <target>]
["FlapsUp", <target>]
["Salute", <target>]
["ScudLaunch", <target>]
["ScudStart", <target>]
["ScudCancel", <target>]
["User", <target>, <action index>]
["DropWeapon", <target>, <weapon name>]
["DropMagazine", <target>, <magazine type name>]
["UserType", <target>, <action index>]
["HandGunOn", <target>, <weapon name>]
["HandGunOff", <target>, <weapon name>]
["TakeMine", <target>]
["DeactivateMine", <target>]
["UseMagazine", <target>, <magazine creator>, <magazine id>]
["IngameMenu", <target>]
["CancelTakeFlag", <target>]
["CancelAction", <target>]
["MarkEntity", <target>]
["Talk", <target>]
["Diary", <target>]
["LoadMagazine", <target>, <magazine creator>, <magazine id>,
<weapon name>, <muzzle name>]

5.56 - Der Animationsbefehl

Ein Animationsbefehl lässt eine Einheit eine Animation ausführen. Hierbei gibt es aber Unterschiede. Da gibt es einmal den Befehl **SwitchMove**, mit welchem die Einheit in die Animation geswitcht/geschaltet wird und **PlayMove**, mit welchem sie die Animation abspielt. Manche Befehle arbeiten nicht mit dem PlayMove-Befehl, wobei man dann auf den SwitchMove-Befehl zurückgreift. Des Weiteren dauert jede Animation eine gewisse Zeit. Diese sollte man berücksichtigen, wenn die Einheit danach eine weitere Animation ausführen soll. Dazu setzt man einfach einen Delay (z.B.: ~10) zwischen die Befehlszeilen in seinem Skript.

Animationsbefehle sind eine nettes Feature für das Drumherum. Zum Beispiel ein Lager, wo Einheiten Sport treiben, sich Unterhalten oder ein Soldat grüßt, wenn jemand durch das Tor fährt. All das regelt man mit den Animationsbefehlen. Die Syntaxformen für das Ausführen einer Animation sehen dabei so aus:

Name playmove "Animationsbefehl"

Name switchmove "Animationsbefehl"

Folgend eine Liste der wichtigsten Animationsbefehle:

Animation	Bezeichnung
AmovPercMstpSnonWnonDnon_carCheckPush	Fahrzeugkontrolle
AmovPercMstpSnonWnonDnon_carCheckWheel	Fahrzeugkontrolle
AmovPercMstpSnonWnonDnon_carCheckWash	Wäscht Fahrzeug
AmovPercMstpSnonWnonDnon_exerciseKata	Macht Kampfsport
AmovPercMstpSnonWnonDnon_exercisekneeBendA	Kniebeugen lang.
AmovPercMstpSnonWnonDnon_exercisekneeBendB	Kniebeugen schell
AmovPercMstpSnonWnonDnon_exercisePushup	Liegestütze
AmovPercMstpSlowWrflDnon_Salute	Startet Gruß
AmovPercMstpSlowWrflDnon_SaluteIn	Startet Gruß
AmovPercMstpSrasWpstDnon_SaluteIn_end	Beendet Gruß
AmovPercMstpSlowWrflDnon_SaluteOut	Beendet Gruß
AmovPercMstpSrasWpstDnon_SaluteOut_end	Beendet Gruß
AmovPercMstpSnonWnonDnon_seeWatch	Schaut auf die Uhr
AmovPercMstpSnonWnonDnon_talking	Unterhält sich
AmovPercMstpSlowWrflDnon_talking	Unterhält sich
ActsPercMstpSnonWnonDnon_MarianQ_shot1man	Unterhält sich
ActsPercMstpSnonWnonDnon_MarianQ_shot3man	Unterhält sich
ActsPercMstpSnonWnonDnon_MarianQ_shot4man	Sicherung
ActsPercMstpSnonWnonDnon_MarianQ_shot5man	Sicherung 2
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop1	Sitzt
AmovPsitMstpSlowWrflDnon_Smoking	Sitzt und raucht
AmovPsitMstpSlowWrflDnon_WeaponCheck1	Sitzt, checkt Waffe
AmovPsitMstpSlowWrflDnon_WeaponCheck2	Sitzt, checkt Waffe
AmovPsitMstpSnonWnonDnon_ground	Sitzt/Hände hinten
AmovPercMstpSnonWnonDnon_AmovPsitMstpSnonWnonDnon_ground	Setzt sich/steht auf
AmovPsitMstpSnonWnonDnon_ground_AmovPpneMstpSnonWnonDnon	Sitzt/liegt,steht auf

Animation	Bezeichnung
AmovPsitMstpSlowWrflDnon_AmovPercMstpSlowWrflDnon	Steht auf
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop1	Sitzt
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop2	Sitzt und redet
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop3	Sitzt und redet
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop4	Sitzt und redet
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_LoopLong	Sitzt und redet
ActsPercMstpSnonWnonDnon_MarianQ_WarReporter	Steht rum
AwopPpneMstpSgthWnonDnon_start	Wirft liegend Granate
AwopPpneMstpSgthWnonDnon_throw	Wirft liegend Granate
AwopPpneMstpSgthWnonDnon_end	Wirft liegend Granate
AwopPercMstpSgthWrflDnon_Throw1	Wirft Granate
AswmPercMrunSnonWnonDf_AswmPercMstpSnonWnonDnon	Schwimmt
DeadState	Stirbt
SprintBaseDf	Rennt geradeaus
SprintBaseDfl	Rennt linksrum im Kreis
SprintBaseDfr	Rennt rechtsrum im Kreis
AwopPercMstpSoptWbinDnon_rfl	Benutzt Fernglas
AmovPercMstpSnonWnonDnon_turnL	Steht rum
AmovPercMstpSnonWnonDnon_turnR	Steht rum
AmovPercMstpSnonWnonDnon_Ease	Nimmt Haltung an
AmovPercMstpSnonWnonDnon_EaseIn	Nimmt Haltung an
AmovPercMstpSnonWnonDnon_EaseOut	Nimmt Haltung an
AmovPercMstpSnonWnonDnon_AmovPknlMstpSnonWnonDnon	Kniet auf einem Knie
AmovPercMstpSsurWnonDnon	Hände hinter Kopf
AmovPercMstpSnonWnonDnon_AmovPpneMstpSnonWnonDnon	Steht vom Liegen auf
AmovPercMstpSnonWnonDnon_AinvPknlMstpSnonWnonDnon	Legt etwas hin (Pipebomb)
AmovPercMstpSlowWrflDnon_AmovPsitMstpSlowWrflDnon	Setzt sich auf Boden
AinvPknlMstpSnonWnonDnon_AmovPknlMstpSrasWpstDnon	Kniet nieder und keucht
AinvPknlMstpSlayWrflDnon_AmovPknlMstpSrasWrflDnon	Kniet nieder und keucht
AinvPknlMstpSlayWrflDnon_healed	Heilanimation
AinvPknlMstpSlayWrflDnon_healed2	Heilanimation
AinvPknlMstpSnonWnonDnon_healed_1	Heilanimation
AinvPknlMstpSnonWnonDnon_healed_2	Heilanimation
AinvPknlMstpSlayWrflDnon_medic	Verarztet Opfer
AinvPknlMstpSnonWnonDnon_medic_1	Verarztet Opfer
AinvPknlMstpSnonWnonDnon_medic_2	Verarztet Opfer
AidlPknlMstpSnonWnonDnon01	Geht in Deckung
AmovPercMrunSlowWrflDf_AmovPpneMstpSrasWrflDnon	Geht in Stellung
AidlPercMstpSnonWnonDnon08	Schultert Waffe
AinvPknlMstpSnonWnonDnon_1	Kniet nieder (Waffenkiste)
AinvPknlMstpSnonWnonDnon_2	Kniet nieder (Waffenkiste)
AinvPknlMstpSnonWnonDnon_3	Kniet nieder (Waffenkiste)
AinvPknlMstpSnonWnonDnon_4	Kniet nieder (Waffenkiste)
AmovPercMstpSnonWnonDnon_AwopPercMstpSoptWbinDnon	Schultert Waffe/ni.Fernglas
AmovPercMstpSnonWnonDnon_Dancing	N/A
AmovPercMstpSnonWnonDnon_flipflop	N/A

5.57 - KI abschalten

Mit folgenden Befehlen lassen sich KI´s abschalten. Das bedeutet, dass sie dann nicht mehr schießen oder auch sich nicht mehr bewegen. Hierbei gibt es folgende Formen:

Name disableAI "Move"	- Einheit bewegt sich nicht mehr
Name disableAI "Target"	- Einheit lässt von Ziel ab
Name disableAI "Autotarget"	- Einheit verfolgt und beobachtet nichts
Name disableAI "Anim"	- KI kann Animation nicht mehr wechseln

Mit **enableAI** macht man das Ganze am Ende wieder rückgängig und die Einheit wird sich wieder normal verhalten.

5.58 - SetVelocity

Mit dem setVelocity-Befehl hat man die Möglichkeit, ein Objekt in eine Richtung gleiten zu lassen. Dabei fliegt das Objekt in die angegebene Richtung, welche mit XYZ-Werten bestimmt wird. Die Syntax dafür schaut wie folgt aus:

Name setVelocity [0,100,100]

5.59 - Der Informationstext

Mit den folgend erläuterten Befehlen hat man die Möglichkeit kurze Infomationszeilen erscheinen zu lassen. Dabei gibt es Unterschiede welche wie folgt aussehen:

Hint "Text"	- Erscheint nach Aufruf
HintC "Text"	- Erscheint nach Aufruf und Spiel stoppt
HintCadet "Text"	- Erscheint nur im Kadettmodus

5.60 - Einheit bleibt liegen, kniet oder steht

Möchte man machen, dass eine Einheit egal was passiert steht, liegen bleibt oder kniet, nutzt man dafür die folgend aufgeführten Befehle. Kneel und KneelDown sind seitens BI vorgesehen, funktionieren aber noch nicht (Stand Version 1.09). Eventuell funktionieren diese Befehle in Zukunft und daher hier die Auflistung.

Name setUnitPos "Up"	- Einheit bleibt stehen
Name setUnitPos "Middle"	- Einheit kniet
Name setUnitPos "Kneel"	- Einheit kniet
Name setUnitPos "KneelDown"	- Einheit kniet und wählt zwischen Liegen und Knien
Name setUnitPos "Down"	- Einheit bleibt liegen
Name setUnitPos "Auto"	- Einheit entscheidet selbst

Kapitel 6
- Missions Specials -

Nachdem du nun alle Kapitel aufmerksam durchgelesen haben solltest, kommen hier ein paar Besonderheiten für deine Mission, die sich mit ein klein wenig mehr Aufwand schnell realisieren lassen. Deine Mission wird Dank dieser Zusätze um einiges interessanter und spannender werden. All die Features dieses Kapitels sind zwar auf Skriptbasis aufgebaut, sind deshalb aber keineswegs weniger funktionell oder weniger gut als Funktionen.

Da diese Beispiele teilweise sehr umfangreich sind und ein Abtippen oft zu Fehlern führt, werden diese auch in Foren wie **www.forum.german-gamers-club.eu** oder **www.mapfact.net** von mir, mit geeigneter Beispielmission, zum Download angeboten.

Churchill (Anton Voß)

6.1 - Die Fallschirmspringer

Fallschirmspringer sind ein nettes Feature für jede Mission. Hier wird eine von vielen möglichen Varianten erläutert.

Der Helikopter

Zunächst erstellt man einen Helikopter, hier mit dem Namen **Heli1**. Wenn dieser zunächst noch nicht fliegen soll, setzt man den Spritwert des Heli´s zunächst mit dem Befehl **this setFuel 0** auf leer. Wenn der Heli später starten soll, muss der Wert wieder auf **1** gesetzt werden. Zur Zeit steigt die Besatzung noch aus, wenn **Fuel** auf **0** steht. Patchhoffnung.

Die Flughöhe

Die Flughöhe sollte schon mindestens **80** bis **100** Meter betragen, da sich die Einheiten sonst verletzen könnten. Diese legt man mit folgender Syntax fest:

Heli1 FlyInHeight 120

Die Landezone

Die Landezone sollte fernab von Ortschaften und dichten Wäldern sein, um der KI eine gute Landemöglichkeit zu bieten. Andernfalls könnten Verletzungen oder ähnliche Probleme auftreten.

Die Gruppe

Damit die Gruppe am Missionsstart schon im Helikopter sitzt, bekommt der Leader der Gruppe, welche hier mal **Gruppe1** heißt, folgenden Eintrag in seine Initzeile:

{_x moveincargo Heli1} foreach units Gruppe1

Das Skript

Das Skript, welches frei benannt werden kann, sieht dabei wie folgt aus:

```
_aunits = units Gruppe1;
_i = 0;
_j = count _aunits;
#Here
(_aunits select _i) action ["EJECT", Heli1] ;
unassignvehicle (_aunits select _i) ;
_i=_i+1;
~1
?_j>_i : goto "Here"
exit;
```

Das Skript kann man dann beispielsweise am Absprungpunkt per Wegpunkt oder Auslöser mit der Syntax **this exec "skripte\heli.sqs"** starten. Eine weitere Möglichkeit, ohne ein eigenes Skript anlegen zu müssen wäre die Verwendung eines im Spiel integrierten Skriptes, welches wie folgt ausgelöst wird:

[Gruppe,Heliname] exec "para.sqs"

6.2 - Das GPS-System

Dieses System ist sehr nutzvoll, wenn man mehrere Einheiten auf dem Schlachtfeld mit taktischen Zeichen versehen will bzw. einfach nur den eigenen Standort oder den einer anderen Einheit auf der Karte verfolgen möchte.

Hierzu wird wieder ein Skript verwendet, welches in der **Init.sqs** oder **Initzeile** des Spielers gleich zu Missionsbeginn gestartet wird. Dieses nun erläuterte Skript setzt nicht nur die Marker an die Position der angegebenen Einheiten, sondern prüft dazu noch, ob diese noch am Leben sind. Ist dies nicht der Fall, wird der Marker gelöscht.

Beispiel 1:

```
"S1-Symbol" setMarkerText Name Soldat1;
#Start
; Prüfen, ob Soldat1 lebt. Wenn nicht, springt das Skript zum Label Ende.
? (!(alive Soldat1)) : goto "Ende";
; Marker setzen
#Marker
"S1-Symbol" setMarkerPos getpos Soldat1;
~1
;Skript springt wieder zum Start
goto "Start";
#Ende
deleteMarker "S1-Symbol";
exit;
```

Beispiel 2:

Dies lässt sich natürlich auch per **If-Then-Else-Syntax** realisieren, welche in einer Zeile stehen sollte, was hier leider aus Platzgründen nicht möglich ist. Diese Variante ist kürzer, da man alles in wenige Befehlszeile packen kann.

```
#Start
If(alive Soldat1)Then{"S1-Symbol" setMarkerPos getpos Soldat1}
Else{"S1-Symbol" setMarkerType "Empty";exit};
Goto "Start";
```

Erklärung:

Wenn (**If**) **Soldat1** lebt dann (**Then**) setze **S1-Symbol** auf **Soldat1** ansonsten (**Else**) lösche **S1-Symbol** und verlasse Skript (**Exit**).

Anmerkung:

Ein weiteres beliebtes GPS-Verfahren, welches hier nicht erklärt wird ist, die Marker für die jeweilige Einheit erst mit dem Skript zu erstellen und an sie zu heften. Wer aber dieses Buch sogfältig durcharbeitet, ist im Anschluss selbst in der Lage so etwas zu skripten.

6.3 - Der Actionmenüeintrag

Das Actionmenü ist jenes, welches sich rechts unten in der Ecke befindet. Hier ist es möglich Einträge hinzuzufügen und diese später auch wieder zu entfernen. Zum Hinzufügen eines Eintrages benötigt man zunächst diese Syntax:

ID = Player AddAction ["Eigener Eintrag","skript.sqs"]

während man zum Entfernen diese Syntax verwendet:

Player RemoveAction ID

Zum Löschen des Eintrages gibt man lediglich den Namen des Eintrags an, welcher hier kurz mit ID festgelegt wurde. Möchte man einen Eintrag haben, wenn die Einheit in einem Fahrzeug sitzt, gibt man dazu den Fahrzeugnamen an.

ID = Fahrzeug AddAction ["Eigener Eintrag","skript.sqs"]

Auslöserbeispiel:
Zunächst setzt man einen Auslöser auf die Karte, verbindet ihn mit dem Spieler, so dass nur er ihn auslösen kann und nimmt etwa folgende Einstellungen vor:

Aktivierung:	Mehrfach
Achse a/b:	5
bei Aktivierung:	ID = Player AddAction ["Eigener Eintrag","skript.sqs"]
bei Deaktivierung:	Player RemoveAction ID

Zum Testen läuft man nun ein paar Mal in den Bereich und wieder raus. Das Ergebnis wird sein, dass der Eintrag beim Betreten erscheint und beim Verlassen wieder verschwindet.

6.4 - Der Rucksack

Mit Hilfe der Actionmenüeinträge kann man nun hervorragend einen Rucksack, Beintaschen oder ähnliches simulieren. Hier mal als kleines Feature das Rucksackbeispiel, an welchem man gut erkennen kann, was mit den Einträgen so alles möglich ist. Dieses Beispiel ist zunächst nur für Einzelspieler ausgelegt.

Zunächst setzt man dazu den Spieler mit folgendem Eintrag in der Initzeile:

RID = Player AddAction ["Open Backpack","backpack\rucksack.sqs"]

Für den Rucksack legt man nun im Missionsordner einen eigenen Unterordner namens **Backpack** an. Dateien im Missionsordner bitte immer klein schreiben!

In diesen Ordner kommen nun die Skripte für die verschiedenen Aktionen. Für diesen Fall werden jetzt mal die vier folgenden Skripte angelegt.

rucksack.sqs	**firstaid.sqs**
save.sqs	**close.sqs**

Rucksack.sqs

Hier wird zunächst der Eintrag -**Rucksack öffnen**- entfernt und erst dann werden die anderen Einträge hinzugefügt. Die Besonderheit hierbei ist, dass man nur drei Mal die Möglichkeit hat sich zu heilen. Damit nun immer der richtige Eintrag erscheint, wird bei jedem Heilen eine Variable auf **true** gesetzt und beim Wideraufruf des Skriptes berücksichtigt. Angenommen man hätte jetzt schon 3 Mal gespeichert (siehe: **? verband3 : goto "Close“**), wäre die Variable **Verband3** schon **true** und somit springt das Skript zum Label **#Close** über und führt die **Close.sqs** aus. Das Skript wäre somit beendet.

```
; Eintrag wird entfernt
Player RemoveAction RID
Playsound "Rucksackauf"
; Einträge werden hinzugefügt
RIID = Player AddAction ["- Speichern","backpack\save.sqs"];
#Verband
? verband3 : goto "Close";
? verband2 : goto "Verband3";
? verband1 : goto "Verband2";
#Verband1
RIIID = Player AddAction ["- First aid (3)","backpack\firstaid.sqs"];
goto"Close";
#Verband2
RIIID = Player AddAction ["- First aid (2)","backpack\firstaid.sqs"];
goto"Close";
#Verband3
RIIID = Player AddAction ["- First aid (1)","backpack\firstaid.sqs"];
#Close
RIIIID = Player AddAction ["- Close Backpack","backpack\close.sqs"];
 exit;
```

Eine weitere Besonderheit bei diesen Skripten ist, dass hier für die jeweiligen Aktionen Sounds angegeben wurden. Siehe: **Playsound "Rucksackauf"**. Diese, welche natürlich nicht unbedingt wichtig sind und daher auch wegfallen könnten, müssen natürlich zuvor in der **Description.ext** definiert werden.

Ein zweites Feature hierbei ist, dass der Spieler bei der jeweilig durchgeführten Aktion eine Animation ausführen soll, um es etwas realistischer zu gestalten. Als Beispiel hierfür dient die nun folgende **FirstAid.sqs**, bei der der Spieler, wenn er sich heilt, auf dem Boden liegen soll.

Achtung! Diese vier Skripte sind miteinander verschachtelt!

Firstaid.sqs

Hier sieht man nun, dass das Skript zunächst prüft, ob und wie oft schon gespeichert wurde. Würde man zum ersten Mal speichern, wäre noch keine Variable auf **true** geschaltet. Das Skript würde zum Label **#Verband1** durchlaufen, die Variable **Verband1** auf **true** setzen und von dort aus auf das Label **#Heilen** springen, wo alle Untereinträge wieder gelöscht werden, der Spieler wieder den Eintrag **Open Backpack** bekommt und die Aktionen durchgeführt werden, die zur Heilung führen.

Beim zweiten Speichern würde das Skript beim Prüfen zum Label **#Verband2** springen, da ja Verband1 bereits **true** ist. Beim dritten Speichern würde es gleich von der zweiten Zeile aus zu Label **#Verband3** springen. Jetzt wäre auch Verband3 true und das Skript würde beim nochmaligen Ausführen sofort beendet (**? verband3 : exit**).

```
? verband3 : exit;
? verband2 : goto "Verband3";
? verband1 : goto "Verband2";
#Verband1
verband1=true;
goto "Heilen";
#Verband2
verband2=true;
goto "Heilen";
#Verband3
verband3=true;
goto "Heilen";
#Heilen
RID = Player AddAction ["Open Backpack",""backpack\rucksack.sqs"];
Player RemoveAction RIID;
Player RemoveAction RIIID;
Player RemoveAction RIIIID;
~0.2
Player playmove "AinvPknlMstpSlayWrflDnon_healed";
~1
Playsound "Sanipack";
~2
Player switchmove "AinvPknlMstpSlayWrflDnon_healed";
~5
Playsound "Schmerz";
~1
Player setdammage 0;
exit;
```

Save.sqs

Hier kommt nun die **Save.sqs**, mit welcher das Spiel gespeichert wird. Zunächst werden wieder alle Einträge gelöscht, in dem von hier aus die **Close.sqs** gestartet wird und dann das Spiel gespeichert.

```
[] exec " backpack\close.sqs";
Savegame;
 exit;
```

Close.sqs

Und hier natürlich noch die letzte Datei, die gebraucht wird, wenn man den Eintrag **Close Backpack** betätigt.

```
Playsound "Rucksackzu";
Player RemoveAction RIID;
Player RemoveAction RIIID;
Player RemoveAction RIIIID;
RID = Player AddAction ["Open Backpack","backpack\rucksack.sqs"];
 exit;
```

Jetzt nur nochmal eine nähere kurze Beschreibung der vergebenen Eintrags-namen, dann kann der Rucksack erfolgreich gepackt werden.

Name: **RID**

Der Eintrag, der für das Öffnen des Rucksackes zuständig ist.

RID = Player AddAction ["Open Backpack","backpack\rucksack.sqs"]

Name: **RIID**

Der Name, der für das Speichern festgelegt wurde.

RIID = Player AddAction ["- Speichern","backpack\save.sqs"]

Name: **RIIID**

Wurde für alle Firstaid-Einträge definiert, da ja immer nur einer aktiv ist.

RIIID = Player AddAction ["- First aid (1)","backpack\firstaid.sqs"]
RIIID = Player AddAction ["- First aid (2)","backpack\firstaid.sqs"]
RIIID = Player AddAction ["- First aid (3)","backpack\firstaid.sqs"]

Name: **RIIIID**

Ist der Name, der für das Schließen des Rucksacks definiert wurde.

RIIIID = Player AddAction ["- Close Backpack","backpack\close.sqs"]

6.5 - Zufallspositionen

Eine Mission, die immer gleich abläuft, kann mit der Zeit sehr langweilig werden und wird daher schnell zur Seite gelegt oder gelöscht. Hat man aber eine Mission geschaffen, welche eine Menge Überraschungen aufweist und man nie genau weiß woher die Feinde kommen werden, ist doch da schon mal ein wenig mehr Spannung vorhanden. Gerade im Mehrspielerbereich kann es sehr anstrengend sein, wenn man dauernd gesagt bekommt, wo was steht und von wo der Feind kommt. Dazu kann man noch verschiedene Startpunkte für den Spieler oder auch verschiedene Zielpunkte definieren.

Natürlich hat man ja im Editor bei jeder Einheit die Möglichkeit den **Radius der Platzierung** festzulegen, was ja auch eine Menge Dynamik bringt. Diese nun erläuterte Variante ist aber eher für feste Punkte bestimmt, die zuvor definiert wurden. Dieses Skript startet man von der Initzeile des Spielers oder der Init.sqs, um ganz zu Anfang die Positionen festzulegen.

Beispiel: Dynamische Startpunkte

Hierbei macht man sich den Random-Befehl zunutze. Das Ganze kann dann in Skriptform etwa wie folgt aufgeführt aussehen:

```
_start = random 3;
? _start < 1  : goto "P1";
? _start < 2  : goto "P2";
? _start < 3  : goto "P3";

#P1
Player setpos [x,y,z];
exit;

#P2
Player setpos [x,y,z];
exit;

#P3
Player setpos getpos HP1;
exit;
```

Hier wurde ein Randomwert von **3** verwendet. Beim Starten des Skriptes wird ein Wert ausgelost und danach geprüft wie groß dieser Wert ist. Dann folgt der nächste Schritt.

Ist er kleiner als **1** springt das Skript zu **#P1**
Ist er kleiner als **2** springt das Skript zu **#P2**
Ist er kleiner als **3** springt das Skript zu **#P3**

Hinter dem jeweiligen Label, also beispielsweise **#P1**, kann man nun die Position angeben.

Bei **#P1** und **#P2** wurde jeweils eine XYZ-Position definiert, während der Spieler bei **#P3** auf ein unsichtbares Heli-H namens **HP1** gesetzt wird, was auch eine Möglichkeit ist.

Man spart sich dann den Aufwand die XYZ-Positionen zu ermitteln und setzt stattdessen lediglich ein paar unsichtbare Heli-H´s auf die Karte und benennt sie entsprechend. Hierbei bietet sich zusätzlich an, den **Radius der Platzierung** des einzelnen Heli-H´s auszunutzen und ihm einen großen Radius zu geben.

Jetzt hätte man zum Einen die Dynamik, dass man nicht weiß, an welche Position (**P1**,**P2**,**P3**) der Spieler gesetzt wird und zum Anderen, dass das jeweilige Heli-H ja auch noch einen Platzierungsradius hat, so dass der jeweilige Punkt auch nicht genau definiert werden kann.

Beispiel: Dynamische Startpunkte Koordinatenbezogen

In dem folgend erläuterten Beispiel wird ein XYZ-Punkt bestimmt, welcher mit einem zusätzlichen variablen Platzierungsradius (**_radius = 500**) versehen wird.

Zunächst wird die Position **_pos** mit **[0,0,0]** definiert, der Radius bestimmt und für **_start** ein Randomwert von **3** definiert, welchen das Skript beim Start zufällig bestimmt. Danach prüft das Skript, wie groß dieser Wert ist und springt an die jeweilige Position. **_pos** bekommt dann einen der drei XYZ-Werte zugewiesen (**_pos = [X,Y,Z]**), welche man vorher rausgesucht und definiert hat.

Nun hat **_pos** eine feste XYZ-Position erhalten und bekommt jetzt noch einen zusätzlichen Radius (**_radius = 500**) um diese Position zugewiesen, in welchem die Einheit, hier der Spieler, dann per Zufall gesetzt wird.

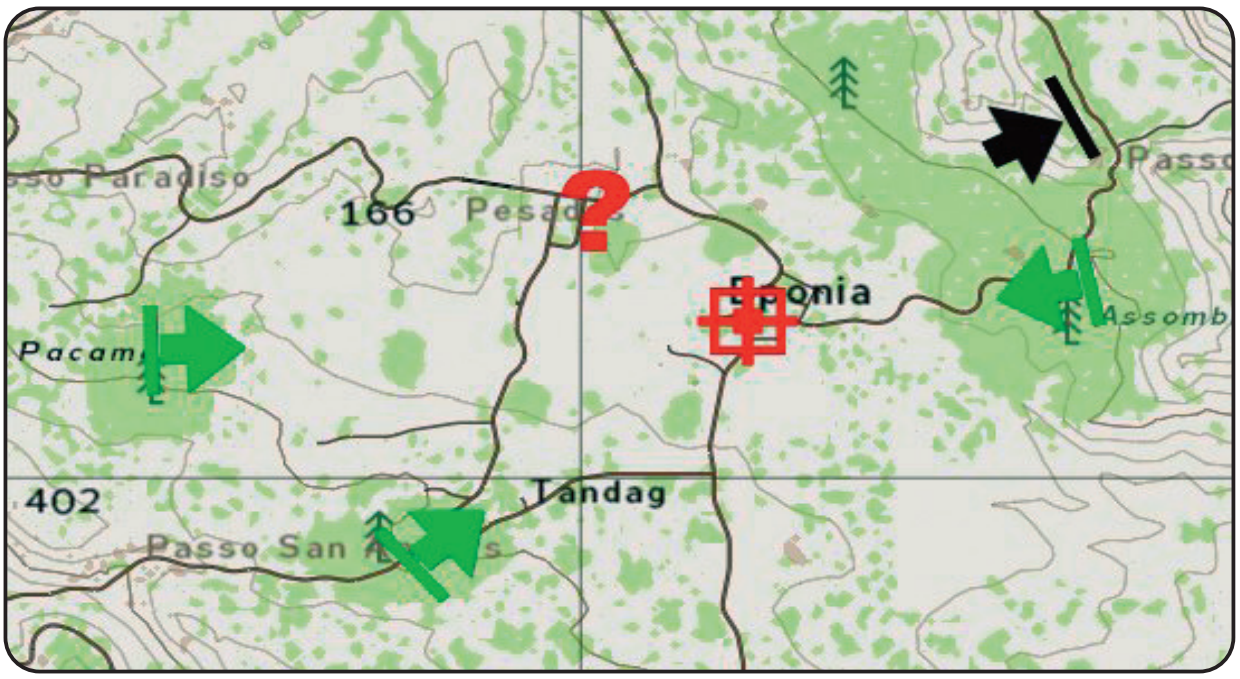
```
_pos = [0,0,0] ;
_radius = 500;
_start = random 3;

? _start < 1 : _pos = [X,Y,Z];
? _start < 2 : _pos = [X,Y,Z];
? _start < 3 : _pos = [X,Y,Z];

; Diese Zeile sollte in einer Zeile definiert werden, was hier nicht möglich ist:
_pos = [(_pos select 0) + _radius/2 - random _radius, (_pos select 1) + _radius/2
          - random _radius, _pos select 2];

Player setPos _pos;
exit;
```

Auf eine Karte kann man sich das etwa so vorstellen, wobei die grünen Marker die Startpunkte des Spielers oder der Einheit darstellen.



6.6 - Der Mapclick

Der Mapclick eröffnet jedem Missionsdesigner eine Menge Möglichkeiten, wie man anhand dieses Kapitels im Unterbereich „Die Artillerie“ sehr gut erkennen kann. Natürlich gibt es noch viel mehr Möglichkeiten diesen einzusetzen. Als Beispiele dienen hier das steuern von anderen Gruppen, Anforderung eines Airstrikes, steuern von Nachschubbewegungen und vieles mehr. Dieses Beispiel ist für Einzelspieler ausgelegt.

Anhand des folgenden Beispiels soll eine KI-Gruppe namens **Alpha1** durch den Spieler per Funkmenü frei auf der Karte gesteuert werden. Dabei wird die Zielposition zunächst mit einem Mapklick festgelegt und dazu ein Marker mit Namen **AMoveP** an die geklickte Stelle gebeamt. Im gleichen Moment bekommt der Leader der Einheit den Befehl sich zum Marker **AmoveP** zu bewegen und betätigt dies mit einem eingespielten Roger-Sound, welchen man dazu selbst in der Description.ext definieren muss.

Die Gruppe selbst wird, wie in diesem Kapitel unter „Das GPS-System“ erklärt, mit einem Marker auf der Karte gekennzeichnet und verfolgt, was hier jetzt aber nicht weiter erklärt wird. Der Marker **AmoveP** soll dieser Marker wieder von der Karte verschwinden und erst beim nächsten Mapclick wieder auftauchen. Dazu wird dieser am Skriptende so lange an die Position [0,0] gebeamt.

Gruppe Alpha 1:

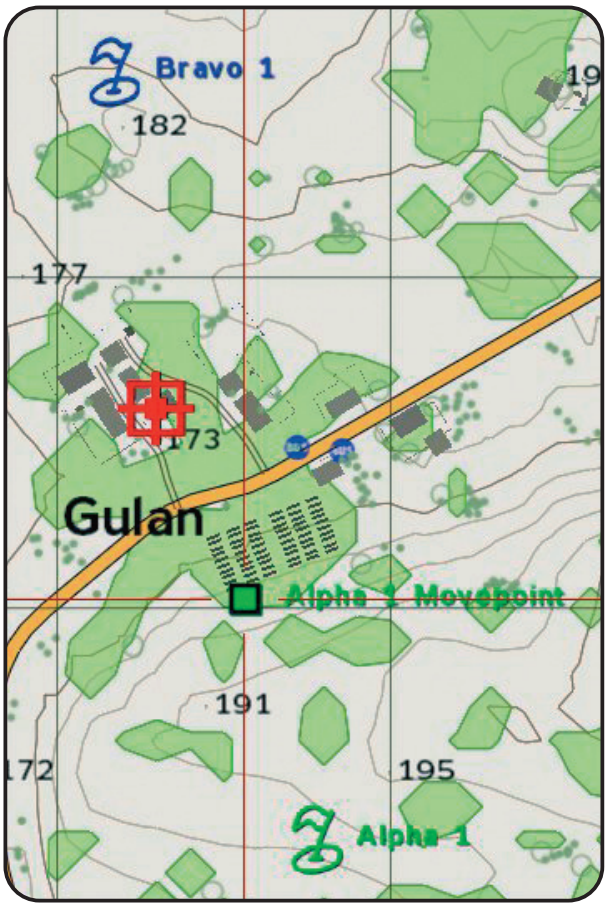
Name:	Alpha1
Initzeile:	Alpha1=group this
Gruppengröße:	Frei festlegbar

Funkauslöser:

Aktivierung:	Radio Alpha Mehrfach
Achse a/b:	0
Text:	0-0-1 Alpha 1
bei Aktivierung:	[] exec “skripte\alpha.sqs"

Marker:

Name:	AmoveP
Text:	Alpha 1 Movepoint
Symbol:	Dot
Achse a/b:	1



Alpha.sqs

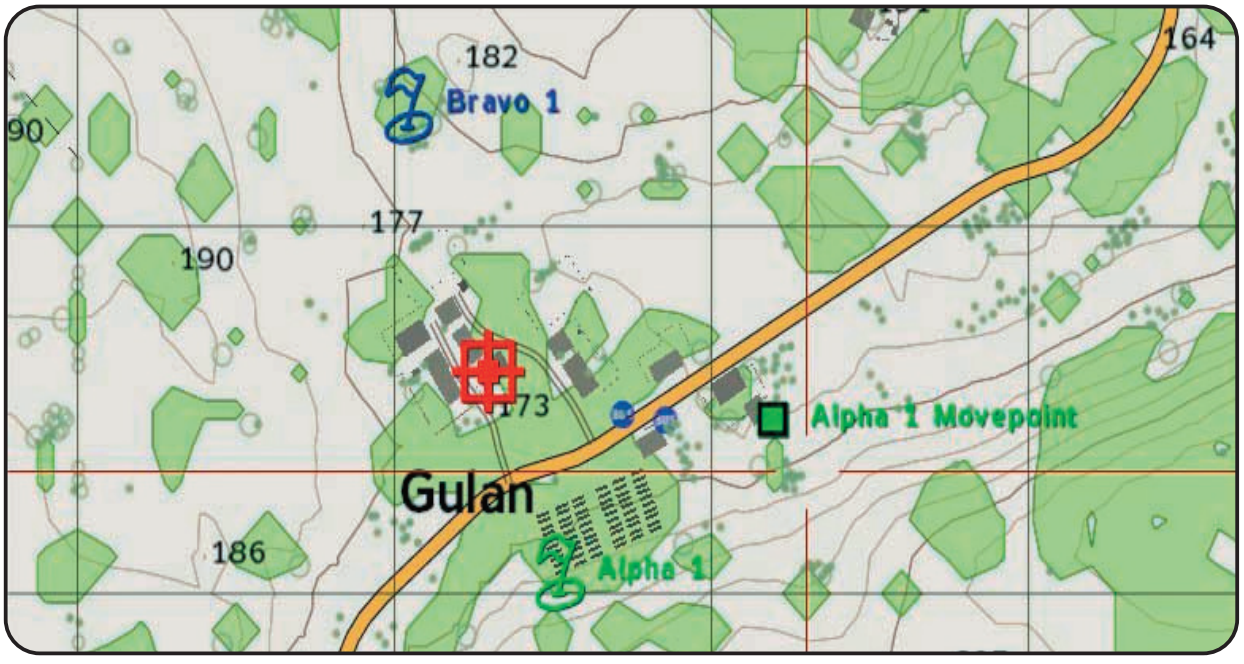
Bei dem Skript geschieht nun folgendes. Zunächst wird die Variable **alphaclick** auf **true** gesetzt und auf dem Monitor erscheint ein Text, der den Spieler aufordert auf die Karte zu klicken. Danach merkt sich ArmA, was bei onMapSingleClick festgelegt wurde und das Skript pausiert bei **@!alphaclick**, bis der Spieler auf die Karte geklickt hat und damit alle Befehle nach **OnMapSingleClick** in den **" "** ausgeführt werden. Unter anderem wird dort die Variable **alphaclick** wieder auf **false** gesetzt.

Jetzt wird der Einblendtext wieder gelöscht und ein Soundsample namens **Roger** abgespielt. Nach einer Pause von **20** Sekunden, wird der Marker **AmoveP** zu einer nicht sichtbaren Position gebeamt und verschwindet somit bis zum nächsten Klick wieder.

```
alphaclick=true;
titletext ["Klicke auf die Karte!","plain down"];

;Die folgende Zeile muss in einer Reihe stehen, was hier nicht möglich ist.
onMapSingleClick "Leader Alpha1 move _pos;
                        alphaclick=false;""AMoveP"" setmarkerpos _pos";

@!alphaclick;
onMapSingleClick "";
titletext ["","plain down"];
~1
Playsound "Roger";
~20
"AMoveP" setmarkerpos [0,0] ;
exit;
```



6.7 - Die Artillerie

Ein ganz besonderes Feature in einer Mission ist die Artillerie. Sei es, dass der Spieler die Möglichkeit hat dieser per Mapclick einen Zielpunkt zu geben oder, dass die KI Artillerieunterstützung zu der Position einer entdeckten Feindeinheit, zum Beispiel dem Spieler, ruft. Die Besonderheit an dieser Artillerie ist, dass diese auch wirklich auf der Karte existent ist, was für ein Artillerieskript eigentlich nicht unbedingt wichtig ist. Hinzu kommt, dass man den Geschützen jederzeit zuschauen kann, wie sie ihre Rohre ausrichten und danach in die für sie festgelegte Richtung feuern.

Dies ist leider nicht ohne ein wenig Skriptaufwand möglich, welcher aber noch recht überschaubar ist. Zunächst nimmt man einige Einstellungen im Editor vor und erstellt im Missionsordner einen Unterordner namens: **Artillerie** (klein geschrieben!)

Spielerbezogen:

Der Spieler soll per Funk die Möglichkeit haben Artillerie zu rufen und dieser per Mapclick auf der Karte einen Zielpunkt zuweisen können. Dabei soll ein Marker namens **Firepoint** diesen Punkt während des Feuervorgangs markieren und danach wieder verschwinden.

Funkauslöser:

Aktivierung: Radio Alpha
Mehrfach
Achse a/b: 0
Text: 0-0-1 Artillerie
bei Aktivierung: [] exec “artillerie\setfire.sqs"

Marker:

Name: Firepoint
Symbol: Destroy
Achse a/b: 1

Unsichtbares Heli H:

Name: ATarget
Position: Irgendwo auf die Karte

Die Geschütze:

Dieses Beispiel ist mit **6** Geschützen der Seite BLUEFOR und dem Geschütztyp **M119** definiert. Diese setzt man auf die Karte und benennt diese wie folgt:

Namen: W1, W2, W3, W4, W5, W6

Ein anderer Geschütztyp funktioniert bei diesem Beispiel nicht! Erklärung folgt.

Setfire.sqs

Beim Benutzen des Funkauslösers, wird das nun folgende Skript gestartet. Dabei wird die Variable **setfire** auf **true** gesetzt und es erscheint ein Text (**titletext**), der den Spieler auffordert auf die Karte zu klicken, um so den Feuerpunkt zu bestimmen. In der darauf folgenden Zeile wird die Definition des Mapclicks gestartet und das Skript pausiert an der Position **@!setfire** und wartet darauf, dass der Spieler auf die Karte klickt.

Beim Klicken wird das unsichtbare Heli-H namens **Atarget** an die angeklickte Position **_pos** gebeamt, die Variable **setfire** wieder auf **false** gesetzt und somit die Bedingung **!setfire** (not setfire) erfüllt. Das Skript kann nun weiterlaufen.

Als nächstes wird der Marker **Firepoint** an die Position des Heli-H´s **ATarget** gesetzt und der onMapSingleClick deaktiviert. Danach wird das Skript **ari.sqs** aufgerufen und der Text, der den Spieler auffordert auf die Karte zu klicken wieder entfernt.

```
setfire=true;
titletext ["Click on the map to set your firedirection","plain down"];
onMapSingleClick "ATarget setpos _pos;setfire=false";

@!setfire;
"Firepoint" setMarkerPos getpos ATarget;
onMapSingleClick "";
[] exec "ari\ari.sqs";
titletext ["","plain down"];
~15
"Firepoint" setMarkerPos [0,0] ;
exit;
```

Ari.sqs

Nachdem dieses Skript durch den Mapklick aktiviert wurde, wird zunächst ein Funksound mit einer Länge von 10 Sekunden abgespielt, den man natürlich erst in der Description.ext definieren muss und erst nach dem Delay von **~10** wird das Skript **fire.sqs** durch das jeweilige Geschütz und dem **ATarget** gestartet. Dies wäre jetzt nur eine Schussfolge!

```
playsound "Firedirection";
~10
;Feuer
[W1,ATarget] exec "artillerie\fire.sqs"
[W2,ATarget] exec "artillerie\fire.sqs"
[W3,ATarget] exec "artillerie\fire.sqs"
[W4,ATarget] exec "artillerie\fire.sqs"
[W5,ATarget] exec "artillerie\fire.sqs"
[W6,ATarget] exec "artillerie\fire.sqs"
 exit
```

Möchte man mehrere Schussfolgen haben, kopiert man den Abschnitt Feuer einschließlich dem Delay und fügt ihn zwischen dem letzten Skriptaufruf (**W6**) und **Exit** ein. Die Geschütze würden dann nach einer Nachladepause nochmal feuern.

Fire.sqs

Nachdem dieses Skript durch die Objekte, also **Geschütz** und **ATarget**, dieses Arrays **[W1,ATarget]** in der **Ari.sqs** aufgerufen wurde, richten sich die Geschütze aus und schießen auf die angeklickte Position.

Dabei wird das erste Objekt des Arrays, also beispielsweise **W1**, mit der lokalen Variable **_K** und das zweite Objekt des Arrays, also das **ATarget** mit der lokalen Variable **_Z** belegt. Danach bekommt die lokale Variable **_X** die X-Position vom **ATarget**(also _Z) und die lokale Variable **_Y** den Y-Wert von **ATarget** zugewiesen.

Mit dem Befehl **_K DoWatch [_X,_Y,5000]** sagt das Skript zu **W1**(also _K), dass dieser zu ATarget und in Höhe 5000 schauen soll. Nach einem kurzen Delay von 5 Sekunden bekommt **W1** mit **_K fire "M119"** den Befehl zu schießen. Kurz darauf werden die Granaten in einem bestimmten Randombereich **_X+((Random 80)-40)** und **_Y = _Y+((Random 80)-40)**, welcher variabel definierbar ist, einschlagen.

Mit dem Befehl **_H say "Ari"** wird ein Artilleriesound, welcher zunächst auch in der Description.ext definiert werden muss, mit dem Geräusch eines anfliegenden Artilleriegeschosses abgespielt. Diesen hört man auch nur, wenn man sich in der Nähe des Einschlagbereiches befindet.

```
_K = _this select 0;
_Z = _this select 1;
_X = Getpos _Z select 0;
_Y = Getpos _Z select 1;
_K DoWatch [_X,_Y,5000] ;
_A =_K Ammo "M119";
~5
_K fire "M119";
@ _A > _K Ammo "M119";
~2
_N = nearestObject [_K,"HeatM119"];
_X = _X+((Random 80)-40) ;
_Y = _Y+((Random 80)-40) ;
_H = "HeliHEmpty" CreateVehicle [_X,_Y] ;
~1
_H say "Ari";
~1
_N setpos [_X,_Y,0];
"SH_125_HE" CreateVehicle [_X,_Y,0] ;
deleteVehicle _H;
exit
```

Dieses Beispiel funktioniert nur mit dem Geschütz **M119**, da es in diesem Skript so definiert wurde. Möchte man ein Geschütz eines anderen Typs verwenden, muss man die Klasse des Geschützes (hier: **M119**) und der Munition (hier: **HeatM119**) dementsprechend angeben.

Die Klassen der Geschütze findet man im **Kapitel 3.2** in der Waffenbezeichnungsliste.

Feindbezogen:
Der Spieler oder befreundete Einheiten sollen in einem definierten Auslöserbereich, wenn sie durch die Gegnerseite entdeckt werden, mit Artillerie beschossen werden. Das Verfahren ist das Gleiche, wie bei der der zuvor erklärten Variante, nur dass die Skripte nun im Unterordner **Feindartillerie** liegen und eine **Setfire.sqs** nicht benötigt wird. Auch einen Marker, der die Beschusszone zeigt, und ein **Heli-H** wird es nicht geben.

Variante 1:
Man müsste jetzt noch nicht mal östliche Artillerie verwenden, sondern setzt lediglich einen Auslöser mit folgenden Einstellungen auf die Karte:

Auslöser:

Aktivierung:	Westen Mehrfach Von Osten entdeckt
Achse a/b:	2000 (den Bereich festlegen!)
bei Aktivierung:	[thislist] exec "feindartillerie\ari.sqs"

und ändert die **Ari.sqs** im Unterordner, wie auf der nächsten Seite erläutert wird, dementsprechend ab. Lediglich die **Ari.sqs** würde hierbei abgeändert werden. Die **Fire.sqs** bleibt dabei auf den Geschütztyp **M119** eingestellt!

Alle Westeinheiten, die nun in diesem Bereich durch Osteinheiten entdeckt werden, werden mit Artillerie beschossen. Dass dies die eigene Artillerie ist, merkt man dabei nicht.

Variante 2:
Möchte man aber für die Ostseite extra Artillerie anlegen, definiert man das wie folgt:

Auslöser:

Aktivierung:	Westen Mehrfach Von Osten entdeckt
Achse a/b:	2000 (den Bereich festlegen!)
bei Aktivierung:	[thislist] exec "feindartillerie\ari.sqs"

Die Geschütze:
Dieses Beispiel ist mit **4** Geschützen der Seite Ost und dem Geschütztyp **D30** definiert. Diese setzt man auf die Karte und benennt sie wie folgt:
Namen: **E1, E2, E3, E4**

Ari.sqs
Die Besonderheit dieser **Ari.sqs** ergibt sich aus der festgelegten Auslösesyntax im Auslöser, welche ja wie folgt definiert wurde: **[thislist] exec " "**. Das bedeutet, dass jede Westeinheit, die durch die Ostseite entdeckt wird, im Skript die lokale Variable **_Ziel** zugewiesen bekommt und somit die Zielkoordinate darstellt. Das jeweilige **Geschütz** und **_Ziel** würden dann die **Fire.sqs** im Unterordner Feindartillerie aufrufen.

```
_Ziel = _this select 0;
[E1,_Ziel] exec "feindartillerie\fire.sqs";
[E2,_Ziel] exec "feindartillerie\fire.sqs";
[E3,_Ziel] exec "feindartillerie\fire.sqs";
[E4,_Ziel] exec "feindartillerie\fire.sqs";
 exit
```

Fire.sqs
Dieses Skript wird nun nur noch auf den Ost-Geschütztyp **D30** abgestimmt. Dabei muss der Klassenname des Geschützes, hier D30, und der Munitionsname, hier **HeatD30,** vergeben werden. Man könnte hier als Geschütz also auch einen Panzer o.ä. verwenden. Im **Kapitel 3.9** wird erklärt, wie man sich den Waffen- und Munitionstyp ausgeben lässt.

```
_K = _this select 0;
_Z = _this select 1;
_X = Getpos _Z select 0;
_Y = Getpos _Z select 1;
_K DoWatch [_X,_Y,5000];
_A =_K Ammo "D30";
~5
_K fire "D30";
@ _A > _K Ammo "D30";
~3
_N = nearestObject [_K,"HeatD30"];
_X = _X+((Random 80)-40);
_Y = _Y+((Random 80)-40);
_H = "HeliHEmpty" CreateVehicle [_X,_Y] ;
_H say "Ari";
~1
_N setpos [_X,_Y,0];
"SH_125_HE" CreateVehicle [_X,_Y,0];
deleteVehicle _H;
 exit;
```

6.8 - Tote Einheiten bzw. Fahrzeuge löschen

Gerade im Mehrspielerbereich ist es sehr wichtig, eine gute Performance zu haben, was aber auch im Einzelspielerbereich nicht unbeachtet bleiben sollte. Je mehr Einheiten sich auf der Karte bewegen bzw. dargestellt werden müssen, desto langsamer oder auch manchmal unfunktioneller die Mission. Dieses Skript löscht in einem zuvor festgelegten Auslöserbereich und in variabel definierbarer Zeit tote Einheiten von der Karte. In dieser Auslösesyntax **[2] exec "bodydelete.sqs"** kann man die Anzahl der toten Einheiten, die maximal auf der Karte liegen bleiben sollen, frei definieren. Statt der **2** gibt man dazu einfach einen höheren Wert an und somit bleiben mehr Leichen liegen.

Dieses Skript ist nicht seitenbezogen. Hier kommt es darauf an, wie man den Auslöser einstellt. Dies ist mal ein Einstellungsbeispiel für Osten:

Auslöser:

Aktivierung:	OSTEN
	Einfach
	Vorhanden
Achse a/b:	2000 (den Bereich festlegen!)
bei Aktivierung:	[2] exec "skripte\bodydelete.sqs"

Möchte man, dass alle Einheiten egal welcher Seite gelöscht werden sollen, wählt man bei **Aktivierung** einfach nur **Jeder** aus. Möchte man, das nur Westeinheiten gelöscht werden sollen, eben **Westen** und so weiter. Für dieses Skript wurde im Missionsordner ein Unterordner namens **Skripte** angelegt.

Die Besonderheit an dem Skript ist, dass Soldaten bevor sie gelöscht werden, dank dieser Zeile: **(Gravedigger) action ["hidebody",_P]** erst im Boden versinken.

Dazu braucht man einen Dummy, der im Skript **Gravedigger** benannt wurde. Dieser macht es dann erst möglich, dass die toten Soldaten zunächst im Boden versinken, bevor sie gelöscht werden. Diesen setzt man einfach irgendwo weit weg vom Geschehen auf eine Insel, damit ihm nichts passiert. Hierbei ist es egal, was es für eine Einheit ist. Er kann jeder Seite angehören. Lediglich der Name muss mit dem im Skript übereinstimmen.
Fahrzeuge werden dabei nicht im Boden versinken, sondern direkt gelöscht!

Soldatenbezogen:
Dieses Skript ist auf die Typenklasse **Man (_T="Man")** festgelegt. Es löscht also nur Vehikel, die auch diese Typenklasse haben. Gibt man hier **Land** an, löscht es alle Typenklassen, die **Land** unterstellt sind. Da sich außer Flugzeuge und Boote alles andere auf Land bewegt, wird somit alles was diese Typenklasse besitzt und nicht mehr lebt, von der Karte gelöscht.

Das Skript **Bodydelete.sqs** sieht dabei wie folgt aus:

```
? !(local server):exit;

_W=_this select 0;
_L=[]+thislist;
_A=[];
_G=[];
_T="Man";

{ if (_T counttype [_x] == 1) then {_G=_G+[_x]} } foreach _L;

#Again
{ if (not alive _x) then {_A=_A+[_x]} } foreach _G;
_G=_G-_A;
? count _A > _W :_P=_A select 0;_A=_A-[_P] ;
(Gravedigger) action ["hidebody",_P] ;
~10
deletevehicle _P;
? count _A == _W and count _G == 0 :exit;
goto "Again"
```

6.9 - Spielbeschleunigung dauerhaft unterdrücken

In manchen Missionen ist es vorteilhaft die Spielbeschleunigung zu unterdrücken. Zum Beispiel, wenn man nicht möchte, dass der Spieler den Ablauf beschleunigt. Denn dies könnte je nach Situation negative Auswirkungen auf den Missions- und Spielverlauf haben. Wenn man zum Beispiel eine hohe Skriplast oder ähnliches hat, könnte die Spielbeschleunigung gegebenenfalls zu Fehler führen.

Da man dies im Editor nicht mit nur einem Befehl lösen kann, regelt man das mit einem kleinen Skript, welches etwa wie folgt aussehen kann:

```
; Spielbeschleunigung unterdrücken

#Prüfen
? not alive Player : exit;
SetAccTime 1.0;
~0.1
goto "Prüfen"
```

6.10 - Der Bullet Mode

Hier ein kleines Feature, welches zwar nicht sonderlich realistisch ist, aber trotzdem ein wenig die Möglichkeiten vorn Armed Assault aufzeigen soll. Mit diesem Bullet Mode ist es möglich, während des Spielens auf Zeitlupe umzuschalten, so dass man Kampfszenen in Zeitlupe sehen und mit einem Screenshot festhalten kann. Dies wurde hierbei mit einem Actionmenüeintrag und zwei Skripten realisiert. Natürlich ist das Ganze nur im Singleplayerbereich möglich und für Multiplayer eher unbrauchbar.

Zunächst versieht man den Spieler mit einem Actionmenüeintrag:

ID=Player AddAction ["Bullet Mode ON","Bulleton.sqs"];

Das ist der Eintrag, mit dem man den Bullet Mode zunächst startet. Das Skript sieht dabei wie folgt aus:

Bulleton.sqs

```
;Eintrag wird entfernt
Player RemoveAction ID;

;Eintrag wird hinzugefügt
IID = Player AddAction ["Bullet-Mode OFF ","bulletoff.sqs"];

;Zeitlupe wird gesetzt
SetAccTime 0.0900;
 exit;
```

Dem Spieler nach dem Auslösen des Bulletmodes der Eintrag **Bullet-Mode ON** entfernt und er bekommt den Eintrag **Bullet-Mode OFF** hinzugefügt. Alles läuft jetzt, bis der Spieler den Bulletmode wieder ausschaltet, in Zeitlupe ab. Bei folgendem Skript wird das gleiche Verfahren wie oben verwendet, nur eben umgekehrt.

Bulletoff.sqs

```
;Eintrag wird entfernt
Player RemoveAction IID;

;Eintrag wird hinzugefügt
ID = Player AddAction ["Bullet-Mode ON ","bulleton.sqs"];

;Zeitlupe wird zurückgenommen
SetAccTime 1.0;
 exit;
```

Ein nettes Zusatzfeature wäre es, wenn man einen Musiktitel mit angibt, der abgespielt wird, wenn der Modus aktiviert und gestoppt wird, wenn der Modus wieder ausgeschaltet wird.

6.11 - Das Feindmeldescript

Möchte man in einer Mission haben, dass eine befreundete KI irgendwelche Feindeinheiten meldet und diese dann auf der Karte für einen Moment mit einem blinkenden Marker markiert werden sollen, damit man beispielsweise seine Artillerie darauf lenken oder andere Einheiten dorthin schicken kann, kann man das wie im nun folgenden Beispiel realisieren. Im Multiplayer würde man das Skript nur Serverseitig ausführen. Dazu in der ersten Skriptzeile den Zusatz **?(!(local server)):exit** mit angeben.

Man setzt irgendwo einen Marker auf die Karte und dazu einen Auslöser, der den Bereich festlegt, welche wie folgt eingestellt:

Auslöser:

Aktivierung:	Ost Von Westen entdeckt Mehrfach
Achse a/b:	5000
bei Aktivierung:	thislist exec "skripte\signal.sqs"

Marker:

Name:	Target1
Farbe:	Red
Achse a/b:	1
Symbol:	Destroy

Das Skript dazu sieht dann etwa so aus:

```
_Target = _this select 0;
signalcounter = 0;
"Target1" setMarkerPos getpos _ Target;
"Target1" SetMarkerType "Destroy";
#Start
? (signalcounter>=10) OR not alive _Target : goto "Ende";
signalcounter = signalcounter+1;
~0.8
"Target1" SetMarkerColor "ColorRed";
~0.8
"Target1" SetMarkerColor "ColorBlack";
goto "Start"
#Ende
~1
signalcounter = 0;
"Target1" SetMarkerType "Empty";
"Target1" SetMarkerColor "ColorBlack";
exit;
```

6.12 - Der Airstrike

Luftschläge sind auf jedem Schlachtfeld ein taktischer Vorteil. Deshalb wird hier eine Möglichkeit vorgestellt, wie man einen Airstrike umsetzen kann. Diese Version muss gegebenenfalls an die jeweilige Mission angepasst werden. Das heißt, man testet den Airstrike vorher im vorgesehenen Missionsgebiet und verändert gegebenfalls die Flughöhe oder die Zeit bis zum Abwurf. Dabei werden die Treffer zum Teil genauer.

Dieses Beispiel ist leider nicht multiplayertauglich, da die Bombe des Harriers das Spiel abstürzen lässt. Um diesen Airstrike in die Mission einzubauen setzt man zunächst folgende Objekte auf die Karte.

Unsichtbares Heli-H:

Leer/Objekte: H (invisible)
Name: ASTarget

Funkauslöser:

Aktivierung: Radio Alpha
Text: 0-0-1 AIRSTRIKE
bei Aktivierung: [] exec airstrike.sqs

Marker:

Name: Firedirection
Farbe: rot
Symbol: Destroy
Achse a/b: 1

Nach dem Aktivieren von Radio Alpha klickt der Spieler auf die Karte und dabei werden das Atarget und der Marker Firedirection auf diese Position gesetzt. Danach wird ein Flugzeug mit Pilot generiert, der dieses Ziel anfliegt. Wenn es in einer gewissen Distanz zum Ziel ist, bekommt es den Bombenabwurfbefehl und wirft diese ab. Danach fliegt es weg und wird am Ende wieder gelöscht.

Airstrike.sqs

Die Logik und den Marker setzt man hierbei irgendwo an den Kartenrand, damit diese für den Spieler nicht zu sehen sind und der Marker und das Ziel später wieder an eine nicht sichtbare Position auf der Karte gebeamt werden können. Jetzt muss man nur noch folgendes Skript anfertigen und es kann losgehen.

```
setfire=true;
titletext ["Click on the map to set your firedirection","plain down"];
onMapSingleClick "ASTarget setpos _pos;setfire=false";

@!setfire;
"Firedirection" setmarkerpos getpos ASTarget;
Playsound "Firedirection";
onMapSingleClick "";
titletext ["","plain down"];

;=========DEFINE=========================
_dropPosition  = getpos ASTarget;
~0.5
_dropPosX = _dropPosition select 0;
_dropPosY = _dropPosition select 1;
_dropPosZ = _dropPosition select 2;
~0.1
_planespawnpos = [_dropPosX + 3000, _dropPosY, _dropPosZ + 1000];
_pilotspawnpos = [_dropPosX + 3000, _dropPosY, _dropPosZ + 1000];

;=========CREATE=========================
_PlaneG = creategroup WEST;
_plane = createVehicle  ["AV8B",_planespawnpos,[], 0, "FLY"];
_plane setpos [(getpos _plane select 0),(getpos _plane select 1),900] ;
;Die beiden folgenden Zeilen müssen in einer Zeile stehen!
_pilot = "SoldierWPilot" createUnit [GetMarkerPos "Firedirection",  _PlaneG, "P1=this"];

_Plane setVelocity [100,0,0] ;
~0.4
P1 moveinDriver _plane;
P1 setDamage 0;
P1 action ["gear_up", vehicle P1] ;
_plane flyinHeight 100;
_plane setspeedmode "full";

#Check
P1 doMove getpos ASTarget;
P1 doTarget ASTarget;
P1 doWatch ASTarget;
? (_plane distance ASTarget) < 1500 : goto "Drop"
goto "Check"
```

```
;=========FIRE=======================

#Drop
_i = 0
_plane FlyInHeight 100;
_plane setpos [(getpos _plane select 0),(getpos _plane select 1),100] ;
~13

#FIRE
_i=_i+1
_plane fire "BombLauncher";
~0.2
? _i <= 6 : goto "FIRE"

;=========FLY AWAY=======================

ASTarget setpos [0,0,0];
"Firedirection" setmarkerpos [0,0];
_plane setSpeedMode “Full”
~4
_plane FlyInHeight 300;
P1 doMove getpos ASTarget;

#Check2
_plane Setdamage 0;
P1 Setdamage 0;
? (_plane distance Player) > 2500 : goto "Ende";
goto "Check2"

;=========DELETE=======================

#Ende
deletevehicle _plane;
deleteGroup _PlaneG
deletevehicle P1;
exit
```

Wenn man die grün markierten Werte bei dem Label **#Drop** und **#Check** verändert, wirkt sich das auf die Treffergenauigkeit aus. Je nach geographischer Gegebenheit ist diese besser oder eher schlechter, was ja in der Realität auch so ist.

Nachdem der Bomber das Ziel erreicht und seine Bomben abgeworfen hat, bekommt der den Befehl wegzufliegen um, wenn er ausreichend weit vom Spieler entfernt ist, gelöscht zu werden. Würde man ihn sofort löschen, würde der Sound sofort verschwinden, was ja nicht so realistisch ist. Am Anfang des Skriptes wurde ein Sound (**Playsound "Firedirection"**) gestartet, der natürlich auch vorhanden sein und zuvor in der Description.ext definiert werden muss.

Kapitel 7
- Multiplayer -

In diesem Kapitel werden dir einige grundlegende Bereiche aus dem Bereich Multiplayer erläutert und näher gebracht. Du wirst Dank dieses Kapitels in der Lage sein deine eigene Multiplayermission erfolgreich zu erstellen.

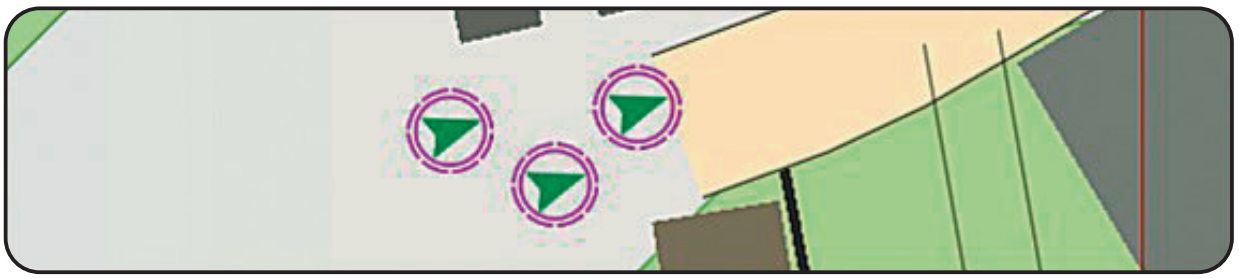
AlexXx (Alexander Zanft)

7.1 - Die Multiplayermission

Das Erstellen einer Multiplayermission erfordert schon etwas mehr Wissen, was ein weiteres Buch dieses Umfangs füllen würde. Deshalb wird dieser Themenbereich hier nur zum Ansatz behandelt und ein paar grundlegende Bereiche erklärt, die dir das Erstellen einer einfachen Multiplayermission ermöglichen.

Einheiten

Zunächst setzt man Einheiten auf die Karte, welche **Playable** definiert sind. Wird in der Description.ext **disabledAI=1** definiert, so werden nur die tatsächlichen Spieler angezeigt. Spielbare Einheiten, die nicht durch einen Spieler belegt sind, werden somit gelöscht und nicht durch KI ersetzt.

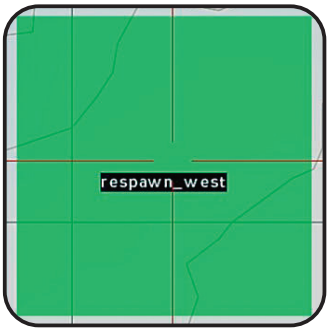


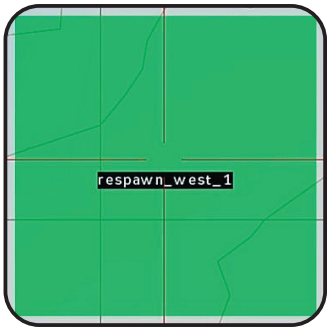
7.2 - Die Respawnpunkte

Als feste Respawnpunkte legt man idealerweise Marker mit folgenden Bezeichnungen an:

West:	**Respawn_west**	Widerstand:	**Respawn_guerrila**
East:	**Respawn_east**	Zivilisten:	**Respawn_civilian**

Ein weiterer R-Punkt pro Seite hieße dann **Respawn_west_1**, **Respawn_west_2**,...





Statt eines Marker hat man natürlich auch die Möglichkeit ein Objekt oder eine Spiellogik als Respawnpunkt zu verwenden. Der Nachteil hierbei ist, dass man dann direkt an dem Punkt gespawnt wird, während man bei einem Marker flexibel im Bereich der Markerfläche gespawnt wird.

7.3- Flexible Respawnpunkte

Möchte man seine Mission mit flexiblen Respawnpunkten versehen, kann man dies auf unterschiedliche Art und Weise machen. Hier soll als Beispiel mal die Erfüllung eines Missionsziels dienen.

Bei jeder Mission fängt man irgendwo auf der Map an und hat ein oder mehrere Ziele. Der Respawnpunkt sitzt zunächst am allgemeinen Startpunkt und soll erst dann versetzt werden, wenn Missionsziel 1 erfüllt ist. Logischerweise sollte er dann auch in den Bereich des ersten Missionsziels versetzt werden, um von dort aus dann, bei Erfüllung des zweiten Missionsziels, auf die Position des zweiten Missionsziels versetzt zu werden.

Dazu hat man dann wieder verschiedene Möglichkeiten. Hier ein Beispiel:

Respawnmarker

Name: Respawn_West
Achse a/b: 50/50

Auslöser

Typ: Einfach
Name: AreaOne
Achse a/b: 50
Art der Auslösung: OPFOR
Nicht vorhanden
bei Aktivierung: "Respawn_West" setMarkerPos getPos AreaOne
hint "Gratulation - Ziel 1 ist erfüllt!"

In diesem Beispiel wurde nun ein Auslöser mit Namen **AreaOne** gesetzt, der prüft, ob der Bereich feindfrei ist. Ist dies erfolgt, wird der Marker **Respawn_West** an die Position des Auslösers mit Namen **AreaOne** versetzt und es erscheint eine Texteinblendung "**Gratulation - Ziel 1 ist erfüllt!**". Natürlich kann man den Marker auch an eine andere Position des ersten Ziels versetzen. Dies sollte hier lediglich als kleines Beispiel dienen.

7.4 - Die MP-Description.ext

In der Description.ext definiert man die Grundsätze für die MP-Mission. Also zum Beispiel die Art oder Zeit bis zum Respawn. Hierzu gehören unter anderem folgende Komponenten:

respawn=3;	- Die Art des Respawn
respawndelay=6;	- Die Zeit bis zum Respawn
respawnvehicle=3;	- Die Art des Vehiclerespawn
respawnVehicleDelay=10;	- Die Zeit bis zum Vehiclerespawn
disabledAI=1;	- Einheiten, die als spielbar definiert wurden werden als KI nicht dargestellt.
AIkills=1;	- Die Punkte der KI werden auch gewertet

Folgend ein Auszug einer Multiplayer-Description.ext in welcher alle wichtigen, für eine Multiplayermission benötigten Komponenten, definiert sind.

Description.ext

```
respawn=3;
respawnDelay=6;
respawnVehicle=3;
respawnVehicleDelay=10;

disabledAI=0;
AIkills=1;
respawnDialog = false;

class Header
{
	gameType = CTF;
	minPlayers = 2;
	maxPlayers = 10;
};

titleParam1 = "Time limit:";
valuesParam1[] = {10000, 300, 600, 900, 1200, 1500, 1800, 2100, 3600, 7200};
defValueParam1 = 1800;
textsParam1[] = {"Unlimited", "5 min", "10 min", "15 min", "20 min", "25 min",
		"30 min", "35 min", "60 min", "120 min", };
titleParam2 = "Score to win:";
valuesParam2[] = {10000, 5, 7, 10, 15, 20, 25, 30};
defValueParam2 = 5;
textsParam2[] = {"Unlimited", 5, 7, 10, 15, 20, 25, 30};
```

7.5 - Die Respawnarten

In einer Multiplayermission gibt es verschiedene Arten des Respawns. Diese legt man gleich beim Editieren in der Description.ext fest. Dies gilt für Fahrzeuge genauso, wie für Soldaten. Allerdings macht es nicht Sinn für ein Fahrzeug einen Respawn als Möwe festzulegen. Für den Vehiclerespawn verweise ich auf **Kapitel 7.10**.

Respawnarten:

0 oder **"None"**	- Kein Respawn
1 oder **"Bird"**	- Respawn als Möwe
2 oder **"Instant"**	- Respawn am Todesplatz
3 oder **"Base"**	- Markerrespawn (respawn_west,…)
4 oder **"Group"**	- Gruppenrespawn (Ist keine weitere KI vorhanden, dann Respawn als Möwe!)
5 oder **"Side"**	- Seitenrespawn (Ist keine weitere KI vorhanden, dann Respawn als Möwe!)

Bei Vehikeln gelten die Werte **0**, **2** und **3** mit den gleichen Respawnarten.

7.7 - Das Deathmatch

Möchte man eine Deathmatch Mission erstellen, die ein oder mehr Spieler auch alleine gegeneinander und die KI spielen können, ist es ja notwendig die eigene Seite zu verfeinden, damit sich auch die KI´s Einheiten ihrer Seite beschießen.

Variante 1
Der Setfriend-Befehl wäre eine Möglichkeit:

East setFriend [East,0.1]

Hat man mehrere Seiten, muss man eben alle gegeneinander verfeinden.

Variante 2
Diese Variante verfeindet die Einheiten egal welcher Seite gegeneinander:

this addRating ((- rating this) - 100000)

Einheiten die diesen Eintrag nicht haben, werden nicht von Einheiten der eigenen Seite beschossen.

Merke
Bei Deathmatch-Missionen mit KI ist es notwendig diese mit mindestens 2 Wegpunkten und großem Radius zu versehen, damit diese sich auch bewegt.

7.6 - Multiplayerbereich festlegen

Da das Areal auf dem gespielt werden kann, mit seinen über 400 Quadratkilometern doch etwas groß ist, gibt es die Möglichkeit einen Spielbereich einzugrenzen. Dies muss man dann auf der Karte, sowie auch direkt in der Landschaft sehen können.

Dazu gibt es eine integrierte Funktion, mit welcher man einen solchen Bereich festlegen kann. Dazu bedarf es lediglich eines Objektes auf der Karte, welches den Mittelpunkt dieser Zone darstellt und in dem die Syntax eingetragen wird. Dieses Objekt ist im Idealfall ein unsichtbares Helipad, welches man unter **Leer/Objekte** finden kann.

Dieses setzt man nun in die Mitte des spielbaren Bereiches und gibt in der Initzeile folgende Syntax zum Aufruf der Funktion an:

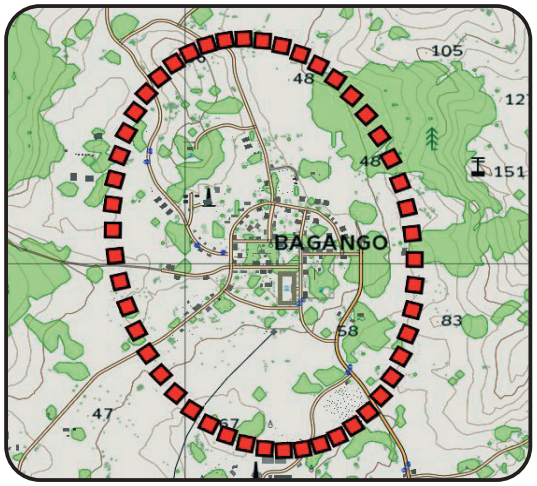
```
Bereich1 = [this,400,400,100,10] execVM "area.sqf"
```

Beim Start der Mission wird das Objekt, also das Helipad, automatisch **Bereich1** genannt und je nach definiertem Bereich erscheinen um diesen Mittelpunkt herum Warnschilder und auf der Karte Marker, die diesen Bereich eingrenzen.

Der Array erklärt sich hierbei wie folgt:

```
Name = [Mittelpunkt,X-Wert,Y-Wert,Objektanzahl,Winkel]
```

Alle Werte sind hierbei variabel und können somit frei bestimmt werden. Auf der Karte und in der Mission könnte das später etwa so aussehen:



Mit Hilfe dieser Funktion erspart man sich somit eine Menge Arbeit, Zeit und Nerven, die Objekte alle selbst setzen zu müssen. Um das abgesperrte Gelände kann man nun Todesauslöser oder ähnliches setzen, um ein Verlassen des Bereiches durch einen Mitspieler zu vermeiden.

7.8 - Der Class Header

Der Class Header ist eine Definition in der Description.ext und sollte bei jeder MP-Mission dringenst mit angegeben werden. In ihm wird die min- und maximale Spieleranzahl, sowie der Missionstyp definiert, was einem Spieler später bei der Auswahl eines Spielservers (siehe Grafik) mit angezeigt wird. Auf dem Bild wurde beim oberen Server ein Class Header definiert, aber beim unteren nicht.

GameType: Hier wird der Missionstyp mit angegeben. Zum Beispiel:

SC	- Sector Control
DM	- Deatmatch
CTF	- Capture The Flag
COOP	- Kooperation
TEAM	- Team

MinPlayers: Angabe der minimalen Spieleranzahl

MaxPlayers: Angabe der maximalen Spieleranzahl

In der Description.ext sieht das dann wie folgt aus:

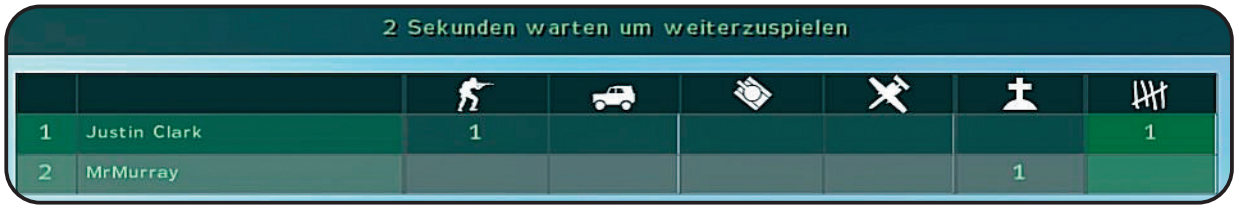
```
class Header
{
        gameType = CTF;
        minPlayers = 2;
        maxPlayers = 8;
};
```

7.9 - Der Respawndialog

Der Respawndialog ist jener, welcher angezeigt wird, wenn man das Zeitliche gesegnet hat und auf seinen Respawn wartet. Diesen kann man in der Description.ext ein- bzw ausschalten. Das macht man mit folgendem Eintrag:

```
respawnDialog = false;
```

Zum Einschalten des Dialogs vergibt man den Wert **true**.



7.10 - Der Fahrzeug Respawn

Für den Vehiclerespawn gibt es verschiedene Möglichkeiten. Zum Einen die Standardvariante und zum Anderen die selbstentwickelte Variante, welche in dieser kleinen Buchversion nicht enthalten ist. Deshalb hier erstmal die Standardvariante:

Mit folgender Syntax in der Initzeile des jeweiligen Fahrzeugs, legt man für dieses die individuelle Konfiguration fest. Die Syntax hierfür lautet:

Fahrzeug1 respawnVehicle [Zeit,Anzahl]

Hat man für dieses Fahrzeug nun einen individuellen Respawnzeitwert vergeben, so wird das Spiel die Zeitvorgabe aus der **Description.ext** übersehen und die für das Fahrzeug definierte **Zeit** nutzen. Setzt man die **Anzahl** der Respawns in der Initzeile auf **0**, wird das Fahrzeug unendlich gespawnt.

In der Description.ext legt man Standardmäßig folgende Zeilen fest:

respawnvehicle=3;	- Die Art des Vehiclerespawn
respawnVehicleDelay=10;	- Die Zeit bis zum Vehiclerespawn

Respawnarten

Bei Fahrzeugen hat man, was die Respawnarten angeht, nur zwei Möglichkeiten. Hierbei stehen der **Respawn am Todesplatz (2)** oder der **Respawn am Seitenrespawnplatz (3)** zur Auswahl. Diesen definiert man ganz normal in der Description.ext:

respawnvehicle=3;	- Die Art des Vehiclerespawn
0 oder **"None"**	- Kein Respawn
2 oder **"Instant"**	- Respawn am Todesplatz
3 oder **"Base"**	- Markerrespawn (respawn_west,…)

Kapitel 8
- Das Camscripting -

Hier kommen wir zu einem Kapitel, was zwar teilweise sehr arbeitsaufwendig ist, aber die Ergebnisse locker mit Hollywoodfilmen zu vergleichen sind. Kleine Intro´s, Outro´s oder Zwischensequenzen heben eine Mission immer ein wenig hervor und man wird besser in die Rolle des Charakters versetzt. Da man über diese Thematik auch ein eigenes Buch dieses Umfangs schreiben könnte, kommen hier lediglich ein paar Einführungen zum Erstellen eigener Szenen.

Blacktiger (Simon Eichenberger)

8.1 - Die Steuerung

Die Steuerung einer Kamera hängt von der Tastaturbelegung ab und ist ansonsten ziemlich einfach. Folgend erläuterte Tastaturbelegung ist von der Installation her standardmäßig so vorhanden. Doch bevor man überhaupt eine Kamera steuern kann, so sollte man sie erstmal starten.

Name1 exec "camera.sqs" oder **this exec "camera.sqs"**

Dies sind zunächst die wichtigsten Tasten um Positionen festzulegen, Objekte anzuvisieren und so weiter. Im Tastatureinstellungsmenü hat man natürlich noch eine etwas größere Auswahl an Tasten, die aber jetzt erstmal weniger interessant sind. Diese reichen aus um großartige Sequenzen zu realisieren.

Linke Maustaste	- Position speichern
Maus vor zurück	- Kamera vor und zurück bewegen
Pfeiltaste auf	- Kamera vorwärts bewegen
Pfeiltaste ab	- Kamera rückwärts bewegen
Pfeiltaste links	- Kamera nach links bewegen
Pfeiltaste rechts	- Kamera nach rechts bewegen
Numpad 4	- Nach links schwenken
Numpad 6	- Nach rechts schwenken
Numpad 8	- Kamera schaut auf
Numpad 2	- Kamera schaut ab
Numpad +	- Ranzoomen
Numpad -	- Wegzommen
Bild auf	- Kamera anheben
Bild ab	- Kamera senken
L	- Fadenkreuz an/aus
V	- Kamera aus
Shift links	- Kamera beschleunigen
Strg	- Objekt selektieren (siehe Bild)

8.2 - Die Kamerakoordinaten

Nachdem man seine Kamera nun ausgerichtet hat, klickt man die linke Maustaste. Die Kamerakoordinaten werden nun im Zwischenspeicher gespeichert. Bei Operation Flashpoint drückte man damals die Strg-Taste und es wurde eine Clipport.txt im Hauptverzeichnis erstellt, in der dann alle Kameradaten erfasst wurden. Nachdem man also seine Position nun im Zwischenspeicher gespeichert hat, muss man sie ja irgendwo einfügen.

Dazu erstellt man in seinem Missionsordner eine neue Textdatei, welche man dann frei benennen kann. Diese heißt jetzt mal **Intro.sqs**. In diese fügt man nun die Koordinaten mit **Strg-V** oder Rechtsklick mit der Maustaste und **Einfügen** wählen, ein. Das Ganze schaut dann etwa wie folgt aus:

```
;=== 0:06:18
_camera camPrepareTarget [101880.56,-28486.36,1887.85]
_camera camPreparePos [9626.16,10062.31,2.00]
_camera camPrepareFOV 0.691
_camera camCommitPrepared 0
@camCommitted _camera
```

Das ist eine gerade festgelegte Kameraposition als Datenblock. Hier sind unter anderem die nun folgend erläuterten Daten enthalten.

;=== 0:06:18
Die Uhrzeit der Festlegung. Diese ist eher unwichtig und kann wegfallen.

_camera camPrepareTarget [101880.56,-28486.36,1887.85]
Blickrichtung der Kamera, für die auch ein Objekt angegeben werden kann.

_camera camPreparePos [9626.16,10062.31,2.00]
Kameraposition (X,Y,Z)

_camera camPrepareFOV 0.700
Kamerazoom. Je kleiner der Wert, desto größer der Zoom.

_camera camCommitPrepared 0
Zeit, wie lange die Kamera zu dieser Position braucht. Bei dem Wert 0 ist sie sofort da, aber schreibt man einen Wert dahin, braucht die Kamera von der alten bis zu dieser neuen Sekunde die jeweils definierte Zeit.

@camCommitted _camera
Hier pausiert das Skript und warten, bis die Kamera ihre Position erreicht hat.

8.3 - Kamera erstellen

Wenn man nun einen solchen Koordinatenblock gespeichert hat, muss man das Skript erstmal so definieren, dass die Kamera jetzt auch an dieser Position startet. Dabei kommen folgende Zeilen hinzu:

_camera = "camera" camCreate [9626.16,10062.31,2.00]

Für die XYZ-Werte kann man nun den _camera camPreparePos-Wert angeben oder auch auf [0,0,0] lassen, wenn die Kamera ohne Zeitverzögerung gleich weitergesetzt wird.

_camera camPrepareTarget [101880.56,-28486.36,1887.85]
_camera camPrepareFOV 0.691
_camera camCommitPrepared 0
@camCommitted _camera

Und der jeweilige Effekt muss jetzt natürlich hiermit noch angegeben werden.

_camera cameraEffect ["internal","back"]

Der folgende Befehl schaltet den Cinemarahmen ab und man hat Vollbild.

showcinemaborder false

Nachdem die ganzen Daten nun verarbeitet wurden, sieht das Skript so aus:

```
_camera = "camera" camCreate [9626.16,10062.31,6.00]
_camera camPrepareTarget [101880.56,-28486.36,1887.85]
_camera camPrepareFOV 0.700
_camera camCommitPrepared 0
@camCommitted _camera
_camera cameraEffect ["internal","back"]
showcinemaborder false
```

Dies ist jetzt eher unspektakulär, da die Kamera erzeugt wird und starr in der Luft steht, ohne dass irgendwas passiert (siehe Bild). Dazu sind nun noch mehr Zeilen notwendig.

8.4 - Die erste Szene

Dazu wird die erste Kameraposition gleich übernommen, aber für den Targetwert bei **_camera camPrepareTarget [101880.56,-28486.36,1887.85]** ein Objekt angegeben. Die Kamera wird dann erstellt und schaut genau auf dieses Object, was hier mal **Flieger1** heißt. Zudem wurde auch der Zoomwert auf **600**, also ranzoomen, verändert.

Jetzt braucht man aber einen weiteren Koordinatenblock um die Kamera dorthin wandern zu lassen. Nachdem dem der nun festgelegt wurde, wird er gleich in das Skript integriert und angepasst.

Intro.sqs

```
;Introsequenz

titleCut [" ", "BLACK IN"]; titleFadeOut 4
Playmusic "Track1"

;Position 1 Flugzeug
_camera = "camera" camCreate [9626.16,10062.31,6.00]
_camera camPrepareTarget Flieger1
_camera camPrepareFOV 0.600
_camera camCommitPrepared 0
@camCommitted _camera
_camera cameraEffect ["internal","back"]

showcinemaborder false

;Position 2 Flugzeug
_camera camPrepareTarget Flieger1
_camera camPreparePos [9657.99,10121.22,1.04]
_camera camPrepareFOV 0.500
_camera camCommitPrepared 30
@camCommitted _camera
```

Wie man sehen kann wurden hier gleich zwei weitere Zeilen eingefügt:

titleCut [" ","BLACK IN"]; titleFadeOut 4

Blendet schwarz in die Sequenz mit einer Dauer von 4 Sekunden ein.

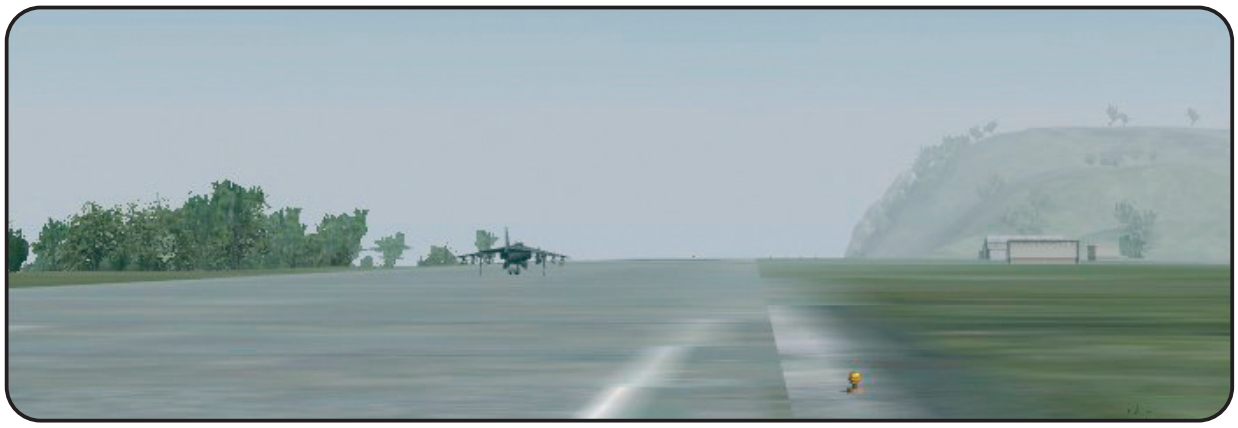
Playmusic "Track1"

Zeitgleich wird ein eigenes Musikstück gestartet, um die Sequenz soundtechnisch ein wenig zu untermalen.

Die Kamera ist nun auf **Flieger1** fixiert und braucht **30** Sekunden bis zur nächsten Kameraposition, wobei sie während der Fahrt langsam ranzoomt, aber der Flieger einfach schneller ist und irgendwann am Horizont verschwindet.

Position 1 Flugzeug:

Position 2 Flugzeug:



Nun kann man das natürlich noch ausweiten, aber die Funktion sollte somit verstanden sein. Jetzt muss das Skript natürlich auch beendet werden. Dazu kommen am Ende des Skriptes noch folgende Zeilen hinzu:

```
6 Fademusic 0

titleCut ["", "BLACK  OUT"]; titleFadeOut 4
~6

player cameraEffect ["terminate","back"]
camDestroy _camera
~1

Playmusic " "
0 Fademusic 1
 exit
```

Die Szene blendet nun langsam aus und auch die Musik wird dabei leiser. Nach 6 Sekunden wird die Kamera gelöscht und der Spieler kann seine Mission starten. Nachdem man die Musik mal, wie oben zu sehen, runtergefadet hat, fadet man sie am Ende des Skriptes wieder hoch, damit man später wieder Musiksound hat.

8.5 - Kamera an ein Fahrzeug/Einheit heften

Man hat auch die Möglichkeit die Kamera an ein Fahrzeug zu hängen, damit diese dann das Fahrzeug verfolgt. Die Kamera lässt sich dabei so gut ausrichten, dass man sie frei an einer Position des Fahrzeugs positionieren kann. Dazu wird zunächst erstmal die Einheit oder das Fahrzeug erstellt, welches hier mal den Namen **Auto** hat.

```
;Kamera erzeugen
_camera = "camera" camCreate [0,0,0]
_camera camSetTarget Auto
_camera camSetPos [0,0,0]
_camera camSetFOV 0.700
_camera camCommit 0
@camCommitted _camera
_camera cameraEffect ["internal","back"]

; Position der Kamera im/am/um das Fahrzeug
_car = Auto

;Position der Kamera längs des Fahrzeugs (vorne/hinten/in)
_dx = -6

;Position der Kamera neben dem Fahrzeug (links/rechts/in)
_dy = 0

;Höhe der Kamera (unter/über/in)
_dz = 2

#Loop
;Die folgenden zwei Blöcke sind jeweils eine Zeile, was hier nicht möglich ist.
_camera camSetTarget [(10 * sin (getDir _car))+(getpos _ car select 0), 10*cos
(getDir _car)+(getpos _car select 1), (getpos _car select 2)]

_camera camSetPos [(getpos _ car select 0) + _dx * sin (getDir _car) - _dy * cos
(getDir _car), (getpos _car select 1) + _dx * cos (getDir _car) + _dy * sin
(getDir _car), (getpos _car select 2)+_dz]

_camera camSetFOV 0.900
_camera camCommit 0
@camCommitted _camera

;Um das Skript zu beenden, setzen wir eine Bedingung. Hier: Wenn unser Auto
;näher als 50 Meter an die Einheit (P1) kommt, soll die Szene beendet werden.
?P1 distance Auto < 50 : goto "Ende"
goto "Loop"

#Ende
P1 cameraEffect ["terminate","back"]
camDestroy _camera
exit
```

8.6 - Text- und Einblendeffekte

Man hat die Möglichkeit verschiedene Arten von Texteinblendungen zu definieren. Dabei verwendet man unter anderem folgende beiden Syntaxformen:

titleCut ["Hallo", "Black Out"]; titleFadeOut 4

oder

TitleText ["Test", "White In"]; titleFadeOut 4

Hier die Übersicht über die einzelnen Varianten:

Plain	- Text erscheint mitten auf dem Bild
Plain Down	- Text erscheint am unteren Bildrand
Black	- blendet vom Bild ins Schwarze
Black Faded	- blendet vom Bild ins Schwarze
Black In	- blendet von schwarz ins Bild
Black Out	- blendet Bild ins Schwarze aus
White In	- blendet von weiß ins Bild
White Out	- blendet Bild ins Weiße aus

Zeilenumbruch

Um einen Zeilenumbruch mit einzusetzen setzt man lediglich ein **\n** an die jeweilige Position im Text. Setzt man zwei **\n\n** hintereinander hat man eine Leerzeile und so weiter.

TitleText ["Paraiso\nOne day later...", "Black In"]; titleFadeOut 6

So sieht das dann später ingame aus:

Kapitel 9
- Scripting -

In diesem Kapitel werden dir einige allgemeine Abschnitte aus dem Bereich Scripting näher gebracht. Du wirst Dank dieses Kapitels einige Skripte und Befehle in diesem Buch besser verstehen und bist am Ende sogar in der Lage eigene kleine oder auch größere Skripte zu schreiben und anzuwenden.

Woody (Andreas Holzwarth)

9.1 - Die Variable

Eine Variable ist ein veränderbarer Wert. Dieser kann dabei ein Wort oder auch eine Zahl sein, dass ist völlig egal. Hierbei gibt es die lokale und die globale Variable. Während eine globale Variable überall gültig ist, ist eine lokale Variable nur für einen Bereich definiert. Hier mal ein Beispiel mit Variablen:

Heli1 FlyInHeight 120

Der Name **Heli1** stellt eine globale Variable dar, während **120** nur ein variabler Wert aber keine Variable in Form eines Namens ist. Man kann eine Zahl nur als Wert verwenden und nicht als Namen vergeben. Verbinden man die Zahl mit einem Buchstaben, wie oben mit **Heli1** dargestellt, sieht das schon wieder ganz anders aus.

Lokale Variable

Eine lokale Variable erkennt man an dem Unterstrich vor der Variable. Diese Variable ist nur in diesem Skript bzw. Bereich gültig. So ist es möglich, die Variable für mehrere Einheiten zu verwenden, ohne das man noch weitere Variablen vergeben muss.

Als Beispiel dient hier mal ein Skript, welches so definiert ist, dass zunächst drei Einheiten **Name1**, **Name2** und **Name3** eine Animation ausführen sollen. Da es aber irgendwo auf der Karte vielleicht auch noch weitere Einheiten gibt, die auf das Skript zugreifen sollen, verwendet man darin eine lokale Variable. Man muss dafür also nur ein Skript schreiben und vergibt darin eine lokale Variable für eine Vielzahl von Einheiten.

Die Namen, die das Skript auslösen sollen, gibt man debei in einen Array mit an:

[Name1,Name2,Name3] exec "skript.sqs"

Wird das Skript nun gestartet, wird jedem der drei Beispielsoldaten die lokale Variable _man zugewiesen. Jede der drei Einheiten wird also in dem Skript lokal behandelt und angesprochen. Im Skript sieht das dann etwa so aus:

```
;Animationsskript

;Einheit bekommt lokalen Wert zugewiesen
_man = _this select 0

;Einheit führt Animation aus
_man playmove "Animation";

;Skript wird verlassen
exit
```

Diese Einheit mit dem globalen Namen z.B. **Name1** hat nun die lokale Variable **_man** zugewiesen bekommen und führt den angegebenen Befehl aus.

Globale Variable

Neben lokalen Variablen gibt es natürlich noch die globalen Variablen. Während eine lokale Variable nur in einem ihr vorbestimmten Bereich gültig ist, wird eine globale Variable, wie der Name schon sagt, für den globalen Bereich festgelegt. Wenn man einem Soldaten einen Namen gibt, so ist das eine globale Variable und diese kann nur einmal vergeben werden. Möchte man einem zweiten Soldaten im Editor den gleichen Namen geben, wird sich das Programm mit einer Fehlermeldung melden. Diese Variable kann man nun von überall, also Skript, Funktion, Auslöser und Wegpunkt ansprechen.

Fest vergebene Variablen

Einige Werte sind vom Spiel aus schon fest vergeben. Diese lauten wie folgt:

Player	- Der Spieler
This	- Einheit oder Objekt
Time	- Uhrzeit im Spiel
_time	- Lokalzeit
_x	- Ein Element eines Arrays
_this	- Lokale Einheit
Pi	- 3,14…

Variablenzustände

Man kann einer Variable Zustände oder Werte zuweisen. Zum Beispiel kann man sie in ihrem Zustand auf **true** schalten oder ihr einen Textstring zuweisen.

Name1= true	- Variable bekommt den Wert WAHR
Name1= 44	- Variable bekommt einen Wert
Name1= "MeinText"	- Variable bekommt einen Textstring
Name1= [Wert1,Wert2]	- Variable bekommt einen Arraywert

Variable speichern

Variablen lassen sich auch jederzeit speichern und sind im weiteren Verlauf aufrufbar.

saveVar "Variablenname"

9.2 - Wahrheitswerte

Ein Wahrheitswert ist ein Zustand eines Wertes. Man kann ihn mit einem **Ein-** oder **Ausschalter** vergleichen. Setzt man eine Variable auf **true**, wird die jeweilige Aktion gestartet und setzt man sie auf **false** wird die Aktion wieder beendet. Für **true** kann man auch **1** und für **false** auch **0** schreiben.

true	Wird zurückgegeben wenn Bedingung erfüllt ist
false	Wird zurückgegeben wenn Bedingung nicht erfüllt ist

9.3 - Logische Operatoren

AND	- Logisches UND zum Verknüpfen von zwei oder mehreren Variablen
OR	- Logisches ODER zur kontrollierten Auswahl zwischen zwei oder mehreren Variablen
NOT	- Logisches NICHT zur kontrollierten Bestimmung von zwei oder mehreren Variablen
!	- Steht ebenfalls für NOT, also NICHT
?	- WENN
:	- DANN
If	- WENN
Then	- DANN
Else	- SONST
Exit	- Stoppt die Ausführung eines Skriptes
Do	- Mach (siehe While Do)
#	- Überschrift (Label) Merke: Setze nie ein Semikolon hinter ein Label!
Goto	- Gehe zu
>	- Größer als
<	- Kleiner
<=	- Kleiner oder gleich
>=	- Größer oder gleich
==	- Gleich
~	- Zeitverzögerung in Sekunden (~3)
;	- Das Semikolon trennt Befehle oder zeigt das Ende einer Zeile an
@	- Pausiert und wartet bis die Bedingung dahinter wahr ist
ForEach	- Für jede Einheit {_x reveal Player} foreach List Bereich1
ThisList	- Für jede Einheit (Seite) in einem Auslöserbereich
Count	- Gibt die Anzahl der vorhandenen Elemente eines Arrays zurück
Random	- Bestimmt einen Zufallswert
Case	- Falls (Bsp: **case 1 : exit** (Übersetzt: Ist **Fall** gleich Wert **1 dann exit**)
Ceil	- Rundet Wert auf. (Bsp: **ceil 5.25** wäre **6** / **ceil -5.25** wäre **5**)
Floor	- Rundet Wert ab. (Bsp: **round 5.25** wäre **5** / **round** -**5.55** wäre -**6**)
Round	- Rundet Wert auf/ab. (Bsp: **round 5.25** wäre **5** / **round 5.55** wäre **6**)

9.4 - Die While-Do-Schleife

Diese Schleife läuft so lange, bis **a** größer als **b** ist. Zu **a** wird dabei so lange der Wert **1** zu hinzugezählt, bis **a** größer ist und somit die Schleife beendet wird. Der maximale Wert für Armed Assault liegt derzeit bei 100.000.

```
While {a<b} do {a=a+1}
```

Übersetzt: Solange **a** kleiner ist als **b**, zähle zu **a** den Wert **1** dazu.

9.5 - Der Zähler

Möchte man einen Zähler in ein Skript einsetzen, muss man zunächst den Ursprungswert am Skript- oder Missionsstart auf **0** setzen. Der Variable **Counter** wird somit zunächst der Wert **0** zugewiesen. Danach startet der eigentliche Zähler und zählt zur lokalen Variable **_Counter** bei jedem Durchlauf den Wert **1** hinzu. Dies passiert aber nur so lange, bis die Variable **_Counter** >=, also größer gleich, **10** ist und beendet dann das Skript.

```
_counter = 0;
#Start
? (_counter>=10) : exit
_counter = _counter+1;
goto "Start";
```

9.6 - If-Then-Else

Diese Syntax heißt übersetzt so viel wie: Wenn – Dann – Sonst. Oder eben so viel wie: WENN Bedingung erfüllt, DANN mach dies, ANSONSTEN mach das. Hier nochmal zwei Syntaxformen dazu:

```
IF (a>b) THEN {c=1} ELSE {c=2}
```

Als Beispiel mal folgendes. Ein Marker soll, solange die Einheit lebt, immer an die aktuelle Position dieser Einheit gesetzt werden. Ist diese Einheit nicht mehr vorhanden, soll das Skript beendet werden.

```
#Start
~0.5
If(alive Soldat1)Then{"S1-Symbol" setMarkerPos getpos Soldat1}
                  Else{"S1-Symbol" setMarkerType "Empty";exit};
goto "Start";
```

Wenn (**If**) **Soldat1** lebt dann (**Then**) setze **S1-Symbol** auf **Soldat1** ansonsten (**Else**) lösche **S1-Symbol** und verlasse Skript (**Exit**).

9.7 - Der Delay

Ein Delay ist eine Verzögerung und wird in Skripten (SQS) verwendet. In Funktionen nimmt man stattdessen den Befehl **Sleep**. Diese bekommen dann einen Wert der als Sekunden gewertet wird. Das Skript oder die Funktion zählt dann den gesetzten Wert runter und folgt erst dann dem weiteren Verlauf.

~300	- Skript pausiert 300 Sekunden bis zur Weiterführung
~random 300	- Generiert einen Zufallswert von 0-300 und pausiert
Sleep 300	- Funktion „schläft“ 300 Sekunden

9.8 - Random

Mit dem Befehl **Random** hat man die Möglichkeit einen Wert per Zufall generieren zu lassen. So könnte man zum Beispiel eine Variable mit einem Zufallswertwert versehen. Das Ganze schaut dann etwa wie folgt aus:

```
_start = random 4
? _start < 1 : goto "Start1";
? _start < 2 : goto "Start2";
? _start < 3 : goto "Start3";
? _start < 4 : goto "Ende";
```

Hier wurde ein Wert bis maximal 4 generiert und das Skript prüft danach wie groß dieser Wert ist und springt zum jeweiligen Label Start oder Ende.

Natürlich kann man diesen Operator auch anderweitig benutzen. Zum Beispiel um eine Einheit an eine Zufallsposition in einem Gebäude setzen oder einen Delay mit einem Zufallswert versehen. Das schaut dann etwa so aus:

```
Name1 setpos (nearestBuilding this buildingPos random 10)
```

oder der Delay:

```
~random Wert
```

9.9 - Waituntil

WaitUntil heißt übersetzt **warte bis** und kann somit als Bedingung für etwas genutzt werden. Es ist also wie ein @ nur eben für Funktionen. Die Funktion wartet dann, bis diese Bedingung erfüllt ist.

```
_Wert = 0;
waitUntil {_Wert = _Wert +1; _Wert >= 100};
```

Der Inhalt dieser Editieranleitung wird dir das Leben im Armed Assault Editor wesentlich angenehmer gestalten. Du hast hiermit die Möglichkeit auch ohne Programmierkenntnisse schnell und einfach anspruchsvolle Missionen für Armed Assault zu erstellen.

Ganz richtig, ohne Programmierkenntnisse! Gewisse Vorkenntnisse aus dem Vorgänger Operation Flashpoint wären natürlich von Vorteil, sind aber nicht zwingend notwendig. In dieser Anleitung werden dir die Teilbereiche des Editors erläutert und anhand von Beispielen näher erklärt. Dazu werden dir die nahezu unbegrenzten Möglichkeiten aufgezeigt, die du in diesem Spiel und dem Editor hast.

Du wirst mit ein bisschen Geschick, Ideenreichtum und Kreativität deine eigenen Szenarien umsetzen und dank der Kameramöglichkeiten und dem Einbinden eigener Sounddateien Missionen erstellen, die schon fast mit einem Hollywoodfilm zu vergleichen sind und den Spieler in seinen Bann ziehen werden.

Dynamische Missionen erstellen, bei denen bei jedem Start das Wetter oder die Uhrzeit anders ist, dass Erfüllen von Missionszielen bei jedem Spielablauf verschieden abläuft und dass bestimmte Einheiten oder der Spieler bei jedem Neustart an einer anderen Position der Insel startet, sollte hiermit für dich kein Problem mehr sein.

Jetzt liegt es nur noch an deinem Ideenreichtum, deiner Kreativität und natürlich dir, gute Missionen erstellen zu können. Ein Drehbuch, Szenario oder eine Story für deine Mission ist hier nicht enthalten, dass musst du dir schon selbst ausdenken. Ansonsten hast du mit dem Editor und dieser Anleitung alles, was du brauchst um deine Ideen umzusetzen. Und wenn mal was nicht klappt, Editor aus, Spiel an und einfach mal entspannt in den Krieg ziehen.

Dieses Spiel baut als Operation-Flashpoint-Nachfolger mit eigens entwickelter Programmiersprache auf seinen Vorgänger auf. Es sind zwar hier und da Änderungen erfolgt und viele Neuheiten eingeflossen, aber vom Grundkonzept ist alles nahezu gleich geblieben. Der Editor ist, wie auch bei seinem Vorgänger, mit gleicher übersichtlicher und benutzerfreundlicher Oberfläche zu bewundern. Und auch die Missionsordner und der Inhalt dieser, sind vom Grundsatz her gleich geblieben. Doch lies, probier und editier dich selbst mit Hilfe dieser Anleitung durch die Welt von Armed-Assault.

Viel Erfolg und Spaß mit dem Editor wünschen BI, Morphicon und Mr-Murray

Demnächst im Handel erhältlich!

Armed Assault Editing Guide Deluxe Edition

Die in Kürze erscheinende Printversion des allseits beliebten Editing Guides von Sascha "Mr-Murray" Hoffmann wird ein über 320 Seiten starkes Hardcoverwerk mit über 200 Unterpunkten, welche in 11 Kapiteln zusammengefasst sind.

Damit verdoppelt sich der Umfang dieser Anleitung und beinhaltet wichtige Zusätze zu denen unter anderem sämtliche Neuerungen von ArmA: Queen´s Gambit, das Einbinden von Dialogen uvm. zählen.

Ein absolutes MUSS für jeden Moddingfan von Armed Assault, egal ob Anfänger oder Fortgeschritten!

Mehr Informationen dazu auf: www.mr-murray.de.vu

Kontakt: mr-murray@bossmail.de