Über diesen Guide

Dieser Editing Guide entsteht in Anlehnung an den berühmten Guide von Mr. Murray. Dieser ist jedoch leider ziemlich in die Jahre gekommen und alles andere als aktuell.

Außerdem sind mit ArmA 2 und besonders mit ArmA 3 viele neue Sachen hinzugekommen die es damals bei Ersterscheinen des Ur-Editing Guide (ca.2008) nicht gab.

Daher soll diese komplette Neuschrift an diesem Punkt anknüpfen und besonders auf die Neuerungen hinweisen. Trotzdem bleibt die Zielgruppe des Guides diejenigen die das Missions-Editing gerade erst erlernen wollen. Nebenher dient dieser Guide natürlich auch als Nachschlagewerk für solche die schon die ersten Erfahrungen gesammelt haben. Ein paar Profi-Tipps von mir runden die Sache ab sodass hoffentlich auch die Erfahrenen unter euch noch den einen oder anderen nützlichen Tipp finden werden. ☺

Was möchte ich in diesem Guide anders machen? Ich werde versuchen stets meine eigene Meinung oder meine Ansätze zu bestimmten Themen, als solche gekennzeichnet, mitunter zu bringen. Außerdem werde ich versuchen neben dem technischen Handwerk des Missionsbaus auch den zwischenmenschlichen Bereich mit einzubeziehen. Damit sind bestimmte Gedanken- und Auftrittsmuster gemeint die man während des Schaffens seines eigenen Werkes beachten sollte um die Mission stets spielbar und interessant für andere zu gestalten. Denn in den meisten Fällen wünscht sich der Schöpfer doch dass andere sein Werk spielen und auch an diesem Freude haben. Dabei versuche ich behilflich zu sein.

Im Folgenden erläutere ich kurz ein paar Symbole die Ihr immer wieder an verschiedenen Stellen in diesem Nachschlagewerk finden werdet. Diese Symbole dienen der Übersicht und markieren wichtige oder besondere Stellen.

Wichtig: markiert Stellen die einer besonderen Aufmerksamkeit bedürfen

Neu: markiert etwas was mit ArmA 3 hinzugekommen ist

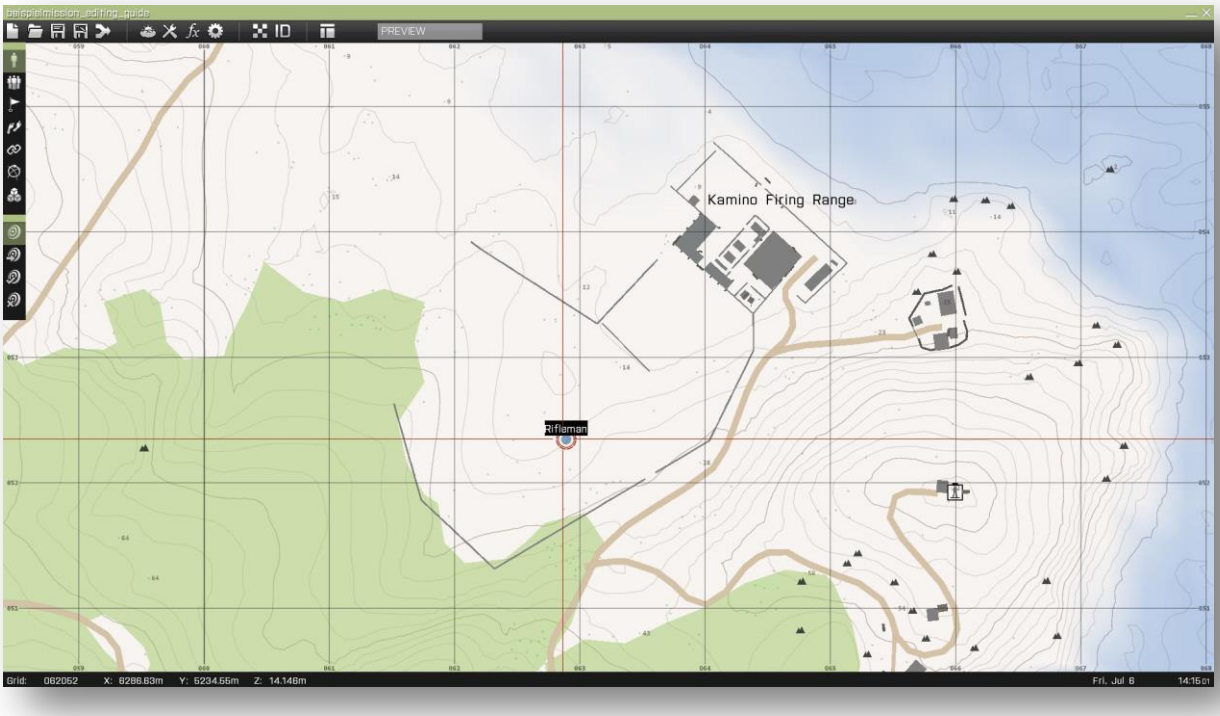
Stilmittel: besonders entscheidend für das Erscheinungsbild einer Mission

1. Die Grundlagen des Editors und Editing

Dieses Kapitel wird euch die Grundlagen des Editors näher bringen. Nachdem Ihr dieses Kapitel abgearbeitet habt, wisst ihr wie Ihr den Editor bedient und was die Symbole im Editor zu bedeuten haben. Außerdem werden die Funktionen hinter den diversen Schaltflächen im Editor kurz erläutert.

1.1 Der Editor und seine Oberfläche

Der Editor kommt mit ArmA 3 auch mit einer komplett neuen Oberfläche daher und ist für die Veteranen unter euch daher im ersten Moment vielleicht etwas ungewöhnlich gestaltet. Trotzdem ist der Aufbau des Editors unter ArmA 3 wirklich übersichtlich und benutzerfreundlich.



Ganz oben in der grünen Spalte findet ihr den Dateinamen unter dem Ihr euer Projekt gespeichert habt. Wurde das Projekt noch nicht gespeichert wird dies durch ein simples Sternchen (*) in der linken oberen Ecke symbolisiert.

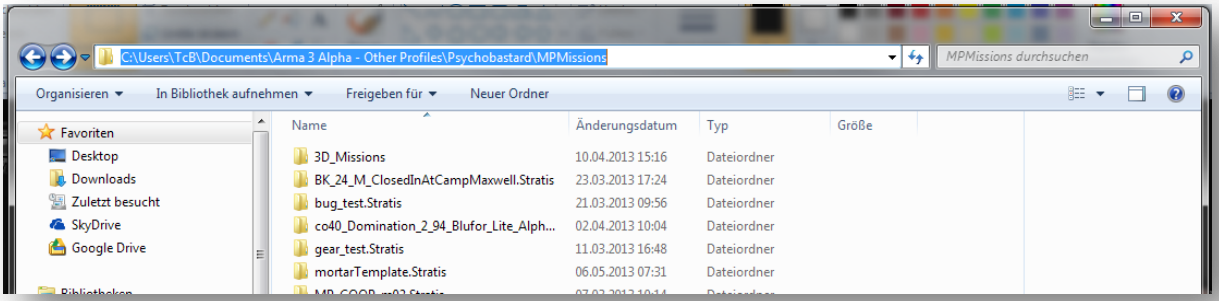
Gleich darunter befinden sich in dem schwarzen Balken eine ganze Reihe kleiner Buttons welche die Grundfunktionen des Editors beherbergen. Laden, Speichern, Funktionsmanager und auch die Missionsvorschau (Preview) sind hier beinhaltet.

In dem schwarzen Balken der am linken Bildschirmrand zu sehen ist sind schließlich die eigentlichen Editing Funktionen untergebracht mit denen Ihr letztendlich eure Mission gestalten könnt. Auf jede einzelne dieser Editing Funktionen wird im Folgenden noch genauer eingegangen.

Neu, Laden, Speichern, Speichern unter, Verbinden

Die Funktionen dieser Buttons sollten klar sein. Denkt immer mal wieder daran den Speicher Button zu verwenden um Datenverlust und verlorener Kreativität vorzubeugen. Mit „Speichern unter“ könnt ihr euch einen neuen Speicherzielpfad für eure Mission suchen. Der Standard Speicherpfad unter der Verwendung von Windows 7 und auch Windows 8 ist unter folgendem Pfad zu finden:

C:\Users*name PC Konto***\Documents\Arma 3 Alpha - Other Profiles***euerSpielerProfilName***\MPMissions**



Beachtet, dass wenn Ihr während Ihr euch im Multiplayer Editor befindet und auf „Vorschau“ (engl. Preview) klickt die Mission automatisch gespeichert wird!

Das Verbinden (engl. Mergen) von Missionen kann sehr hilfreich und zeitsparend sein wenn ihr auf die Verwendung von Vorlagen (engl. Templates) die Ihr euch selbst erstellt habt oder auf die aus der Community zurückgreift. Ich persönlich mache oft Gebrauch von dieser Funktion da ich viel mit selbst erstellten Templates arbeite.

Intel, Debug Console, Function & Config Viewer

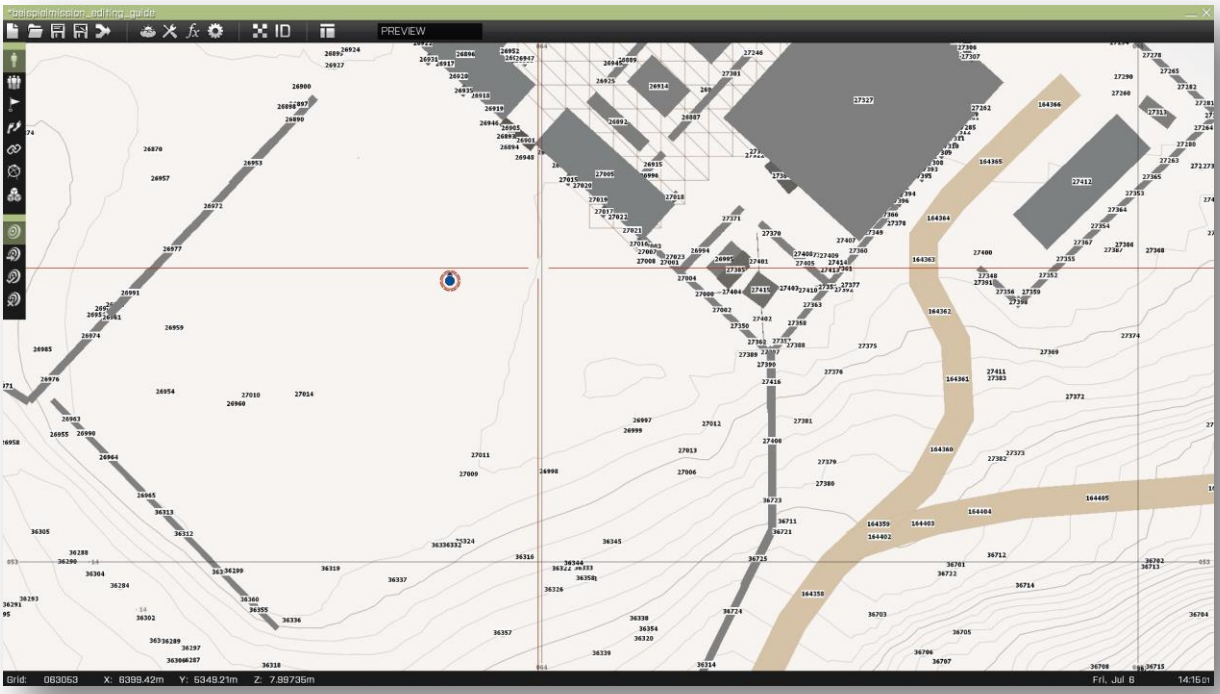
Hinter dem Button „**Intel**“ verbirgt sich alles was an Umwelteinflüssen auf eure Mission einwirken kann. Daher ist dieser Bereich ein sehr wichtiges Stilmittel für eure Mission und kann deren Erscheinungsbild und Stimmung grundlegend beeinflussen.

Die anderen drei Buttons in diesem Bereich werden zu einem späteren Zeitpunkt genauer behandelt und sind eher für den fortgeschrittenen Missionsbauer nützlich. Das Thema „**Intel**“ ist so umfangreich, dass es in einem eigenen Abschnitt auf Seite xx behandelt wird.

Textur anzeigen, ID’s anzeigen

Sich die Geländetextur im Editor anzeigen zu lassen kann dann besonders nützlich sein, wenn es darum geht Objekte im Editor auf einer bestimmten Fläche zu positionieren ohne ständig in die Vorschau gehen zu müssen um zu prüfen wie das Objekt platziert wurde.

Lässt man sich die Gelände-ID's anzeigen werden diese erst sichtbar wenn man im Editor tief genug heran zoomt. Welchen Sinn haben diese ganzen Zahlen kann man sich jetzt fragen.



Jede Objekt-ID ist ein Objekt in der Welt von ArmA mit dem interagiert werden kann. Diese Objekte können Häuser, Straßen, Bäume, Felsen oder Zäune sein. Die Engine hinter ArmA benutzt diese ID's um die KI (engl. AI) in der Welt agieren zu lassen. Wir können diese ID's nutzen um feste Gegenstände anzusprechen oder abzufragen.

Zum besseren Verständnis ein kleines Beispiel:

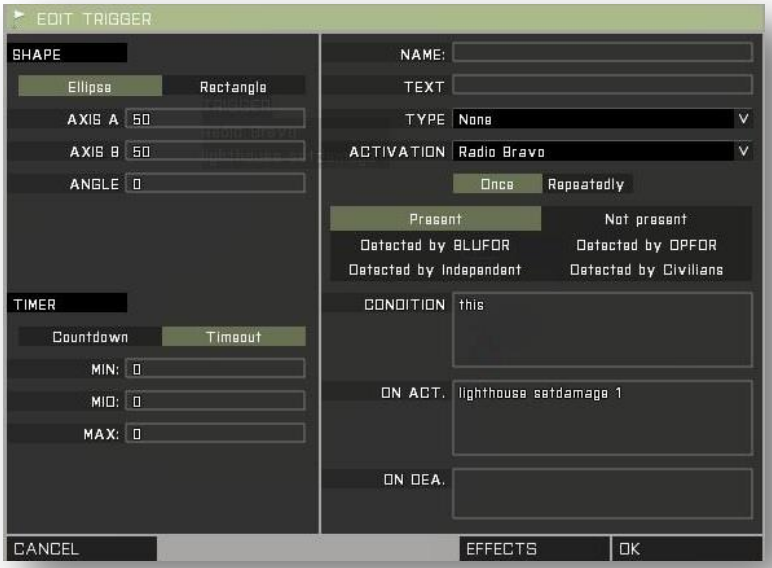
Ich möchte gerne ein Missionsziel festlegen. Ziel soll es sein einen Leuchtturm auf Stratis zu zerstören. Wenn die Spieler dieses Ziel erreicht haben soll die Mission beendet werden. Dazu muss ich jetzt den Leuchtturm der fest auf der Insel steht „abfragen" können. Dazu schalte ich die ID's auf sichtbar und suche mir die ID des Leuchtturmes heraus. In meinem Fall ist das die ID **103049**. Nun platziere ich eine **Game-Logic** in der Nähe des Leuchtturmes und schreibe folgendes in die Intialisierungszeile der Logik:

```
Lighthouse = position this nearestObject 103049;
```

Lighthouse stellt jetzt eine Variable dar dessen Namen ich mir selbst ausgedacht habe und man frei bestimmen kann. Hinter der Variable **Lighthouse** steht jetzt der Leuchtturm als Objekt welchen ich mit einem weiteren Auslöser (engl. Trigger) abfragen kann.

Zur Kontrolle könnt ihr einen Trigger platzieren der ungefähr so aussehen kann:

Ein Trigger mit Radioauslöser BRAVO und einem Kommando in der Aktivierung welches den Schaden des genannten Objektes, in dem Fall von **Lighthouse**, auf 1 setzt.




```
Lighthouse setDamage 1;
```

Klickt auf Vorschau (engl. Preview) um in das Spiel zu gelangen. Drückt die Tasten in folgender Reihenfolge für Radio-Code BRAVO: 0 – 0 – 2. Wenn alles funktioniert hat sieht das Ergebnis folgendermaßen aus. ☺

Editor Layout

Mein liebster Button im Editor unter ArmA 3! Denn die alt Eingesessenen unter euch finden das neue Design des Editors, auch wenn es mindestens genauso funktional sein mag wie jeher, vielleicht etwas befremdlich. Daher ist es super das man mit einem einfachen Mausklick auf diesen Button das Design wieder so herstellen kann wie man es von den Vorgängern seit OFP gewohnt ist. Eine klasse Sache – jeder kann wählen und selbst bestimmen wie er mit dem Editor weiter arbeiten möchte.

1.2 Der Intel – Dialog

In dem Eingabefeld **„Name“** gebt Ihr eurer Mission einen Namen. Mit diesem Namen wird eure Mission später z.B. in der Missionsauswahl angezeigt oder als Name der laufenden Mission auf einem Server im Multiplayer. Die Beschreibung sollte ein paar kurze und wesentliche Informationen zu der Mission enthalten. Was Ihr hier angebt bleibt natürlich völlig eurem freien Willen überlassen. Ich persönlich gebe hier gerne den Namen des Erstellers (also meinen Namen) und eine Versionsnummer der Mission an. Das kann später äußerst nützlich sein um verschiedene Missionsentwicklungsstadien auch noch unterscheiden zu können.

Die Angabe des Datums für eine Mission wird von vielen unterschätzt oder als unwichtig erachtet. Das ist aber ein Fehler. Unter allen ArmA Teilen (auch OFP) wurden die Jahreszeiten simuliert. Das hat sich auch jetzt nicht geändert. Was bedeutet das für euch?

Ganz einfach das ein Wintermonat in der Regel wesentlich kürzere Tage hat als ein Sommermonat. Beachtet das also wenn ihr ein Datum auswählt. Beträgt die geplante Spielzeit einer Mission z.B. eine Stunde und ihr wollt das die Mission während eines Sonnenunterganges stattfindet, schaut nicht nur auf die Uhrzeit sondern auch auf das Datum damit der Spieler nicht plötzlich ohne Nachtsichtgerät im dunklen stehen muss. Das ist ein Gamebreaker (Spielstopper)!

Neben der Länge eines Tages wird auch die Temperatur der Oberflächen simuliert. Das ist wichtig wenn in der Mission Wärmesichtgeräte eine Rolle spielen sollen. Wird in der Wüste ein Wintermonat gewählt und ihr schaut durch ein Wärmesichtgerät erscheint die gesamte Umgebung erst einmal schwarz und ohne Kontur. Das ist so weil die Umgebung kalt ist und wird von sehr vielen fälschlicherweise als Bug abgetan!

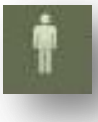
„**Overcast**“ beschreibt die Bewölkung des Himmels mit einem Wert zwischen 0 und 100, wobei 0 für strahlenden Sonnenschein bei klarem Himmel steht. „**Fog**“, also Nebel ist jetzt selbsterklärend.

Seit ArmA 3 lässt sich nun auch der „**Wind**“ direkt im Intel Dialog beeinflussen. (Stärke und Windböen) Leider wird bisher die Ballistik davon noch nicht beeinflusst und es bleibt bei einem optischen und akustischen Effekt. Außerdem lässt sich die Höhe der Wellen steuern was zu einem elementaren Stilmittel werden kann. Stellt euch vor ihr möchtet einen amphibischen Angriff in euer Mission unterbringen. Ein Boot (z.B. Speedboat mit Minigun) soll vom Wasser kommend die Küste unter Sperrfeuer nehmen. Für den Spieler ist jetzt der Wellengang bei der Zielerfassung ein elementarer Unterschied im Schwierigkeitsgrad der Mission.

Wird „**Lightning**“ (Blitzen) selbst bestimmt, überschreibt dieser Parameter den „**Overcast**“ Wert und setzt diesen auf 85. Der Sinn dahinter ist, dass es natürlich nur Blitzen kann wenn das entsprechende Wetter dafür vorhanden ist. Den Regen kann man zum jetzigen Zeitpunkt leider noch nicht direkt in diesem Dialog beeinflussen.

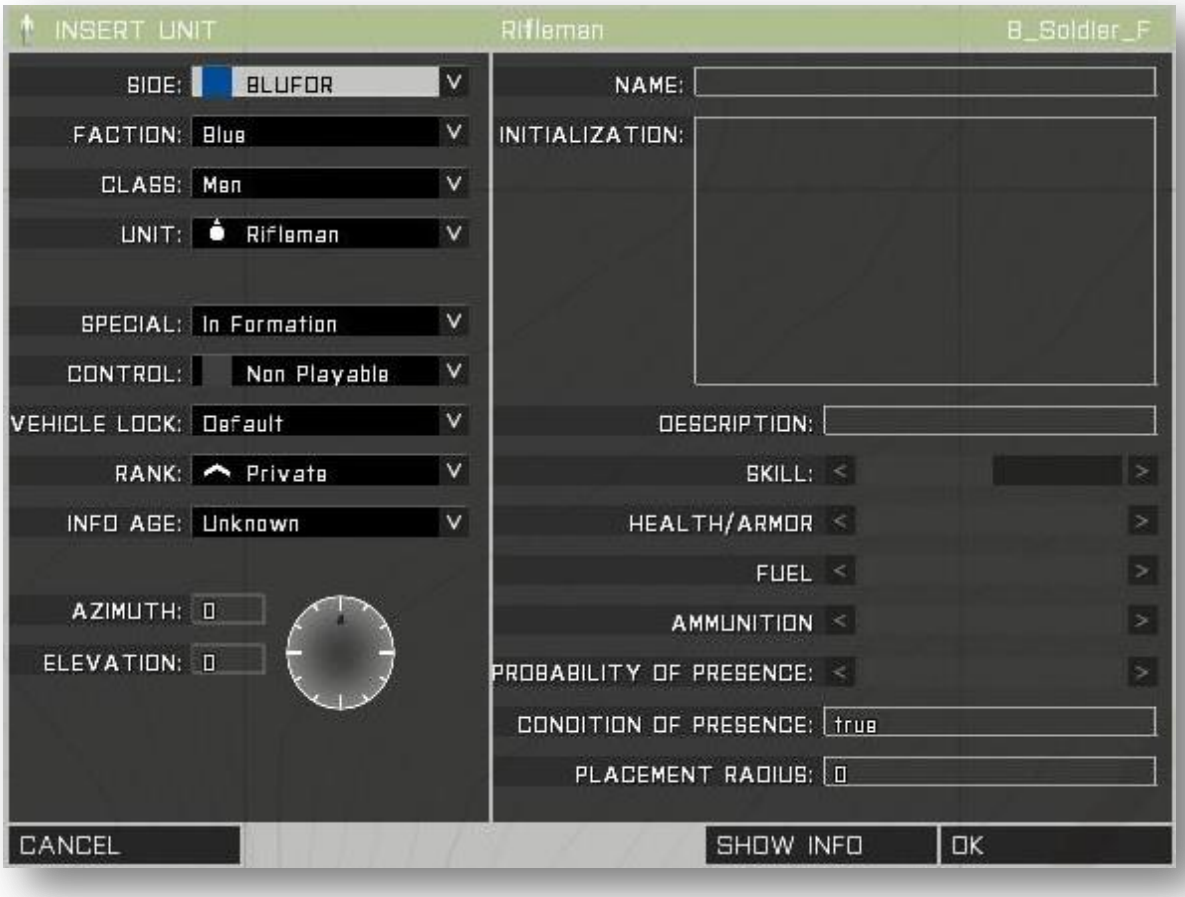
Im rechten Bereich des Dialoges befinden sich sämtliche Vorhersagewerte (engl. Forecasted) für die diversen Wetterparameter die individuell und unabhängig von den Startwerten im linken Bereich definiert werden können. Im unteren Bereich lässt sich jetzt auch noch die Zeit bis zum Erreichen des Vorhersagewertes definieren. Diese neue Funktion finde ich persönlich besonders praktisch. Im Ganzen finde ich das neue Wettersystem und die neuen Einstellmöglichkeiten die mit ArmA 3 gekommen sind sehr gelungen und hilfreich.

Die letzte Option die euch in dem Intel Dialog gegeben wird ist die Einstellung wie sich eine unabhängige Partei zu euch oder euren Feinden verhalten wird. Die unabhängige Fraktion unter ArmA 3 ist die griechische Armee. Und hier könnt ihr nun festlegen wie diese den anderen Fraktionen gegenüber gesinnt sein wird.

1.3 Einheit platzieren 

So fängt jede neue Mission an – zuerst wird eine erste Einheit platziert, nämlich der Spieler!

Wählt dieses Symbol (Einheit, engl. Unit) und macht einen Doppelklick mit der linken Maustaste auf der Karte. Folgender Dialog öffnet sich der im ersten Moment erst mal recht unübersichtlich erscheint da er sehr viele Einstellmöglichkeiten bietet. Mit einem einfachen Klick auf OK habt ihr jedoch erst mal eure erste Einheit platziert.



Erst nachdem Ihr diesen ersten Schritt getan habt könnt ihr in die Vorschau des Editors wechseln. Und erst wenn eine erste Einheit als Spieler erstellt wurde kann man „leere Einheiten“ platzieren. Leere Einheiten können leere Fahrzeuge oder andere statische Objekte sein. Erst nachdem ein Spieler platziert wurde wird die im folgenden Bild rot markierte Option sichtbar!

Im Folgenden wird auf jeden der diversen Einstellmöglichkeiten im Einheiten Dialog kurz eingegangen.

Seite

BLUFOR, OPFOR, INDIPENDENT, CIVILIAN	-	stellt die spielbaren Seiten dar
Ambient Life	-	Tierleben
Game Logic	-	eine Spiellogik
Empty	-	leere Fahrzeuge / Objekte aller Seiten

Fraktion

Hier befinden sich verschiedene Fraktionen / Armeetypen die sich aber innerhalb einer Seite befinden (z.B. BLUFOR)

Klasse

Hier werden alle unter der vorher bestimmten Seite und Fraktion verfügbaren Klassen aufgelistet. Mit Klassen sind hier die verschiedenen Einheiten- und Waffengattungen gemeint. (z.B. Men, Car, Air, …)

Einheit

Alle Einheiten die sich innerhalb der vorher gewählten Klasse befinden sind hier zu finden.

Spezial

Keine – (engl. None) Wird diese Option gewählt kann eine platzierte Einheit wirklich frei platziert werden ohne besondere Einflüsse.

Formation – (engl. Formation) Hier hingegen ist es etwas anders: Ist die Einheit die diese Option erhält Mitglied einer Gruppe, wird diese egal wo sie platziert wurde, zu Spielbeginn bzw. Vorschau direkt in die Formation des Gruppenführers gesetzt.

Fliegend – (engl. Fly) wird diese Option auf ein flugfähiges Objekt angewendet, befindet sich dieses zu Missionsstart direkt fliegend in der Luft. Das funktioniert jedoch nur wenn das Objekt auch einen Piloten hat. Auf Infanterie oder Fahrzeuge hat diese Option keinen Einfluss.

Im Laderaum – (engl. Cargo) Wird diese Option auf eine Infanterieeinheit vergeben befindet diese sich direkt im Laderaum eines Fahrzeuges. Bedingung hierfür ist jedoch, dass sich eines der Gruppenmitglieder bereits in dem Fahrzeug befindet und das Fahrzeug ausreichend Laderaum zur Verfügung hat.

Kontrolle

Nicht spielbar – (engl. Non playable) ist sofort gewählt und steht für eine Einheit die direkt als KI gesteuert wird bzw. sich selbst steuert

Spielbar – (engl. Playable) steht für eine Einheit die Optional gespielt werden kann und einen Spielerslot in der Multiplayer-Lobby bietet oder die Möglichkeit auf einen Teamswitch im Singleplayer oder Multiplayer bietet.

Spieler – (engl. Player) stellt den Standard Spieler-Slot dar, welcher immer benötigt wird und besetzt sein sollte.

Fahrzeug abgeschlossen

Standard – (engl. Default) es wird keine besondere Einstellung getroffen, das Fahrzeug ist offen. Diese Option hat ausschließlich auf Fahrzeuge einen Effekt.

Abgeschlossen – (engl. Locked) Das Fahrzeug ist für alle verschlossen und kann nicht verwendet werden.

Offen – (engl. Unlocked) Das Fahrzeug ist offen und kann von jedem verwendet werden

Abgeschlossen für Spieler – (engl. Locked (Player)) das Fahrzeug ist nur für die Spieler verschlossen und KI Soldaten können weiterhin das Fahrzeug verwenden.

Mit folgender Syntax kann ebenfalls der Fahrzeugstatus festgelegt oder im Laufe des Spieles verändert werden. Diese Syntax kann in einem Script oder in der Initzeile des Fahrzeuges stehen.

```
Name lock true;        - Fahrzeug verschlossen
Name lock false;       - Fahrzeug offen
```

Wobei „**Name“** das Fahrzeug als Variable darstellt. (alternativ wenn es in der Initzeile des Fahrzeuges steht: „**this“**)

Rang vergeben

Der Dienstgrad einer Einheit ist dahingehend wichtig, dass die Einheit einer Gruppe mit dem höchsten Rang auch automatisch der Gruppenführer wird. Nach dem eventuellen Ableben dieser Einheit wird wiederum automatisch die Einheit mit dem nächsthöheren Rang zum Gruppenführer ernannt.

Der Rang kann während des Spielverlaufes auch mit folgender Syntax geändert werden:

```
Name setRank „Captain“;
```

Info über Alter

Damit ist nicht das Alter der Einheit in Lebensjahren gemeint, sondern der Bekanntheitsgrad der Einheit zu Spielbeginn. Umso „jünger“ die Einheit ist, desto mehr weiß der Gegner über dessen aktuellen Status und Position.

Der Bekanntheitsgrad kann mit folgender Syntax geändert werden:

```
Name setTargetAge „5 MIN“;
```

Ausrichtung

Mit dem **Azimut** kann eine Einheit oder ein Objekt gedreht werden. Dazu wird ein Winkelwert (einfache Zahl zwischen 0 und 359) eingegeben.

Neu und sehr praktisch seit ArmA 3: Jetzt kann auch bequem die **Höhe** einer Einheit oder eines Objektes bequem im Editor angegeben werden. Dazu wird ein simpler Zahlenwert als **Höhe** (engl. Elevation), Angabe in Meter, in das Eingabefeld getippt. 

Eine weitere und bequemere Möglichkeit ein Objekt zu drehen ist es, das Objekt mit der Maus im Editor zu markieren. Anschließend die Linke Shift-Taste gedrückt halten und erneut mit gedrückter linker Maustaste und einer Mausbewegung das Objekt zu verdrehen.

Außerdem können Objekte und Einheiten mit folgender Syntax gedreht werden:

```
Name setDir 90;          - Einheit blickt nach Osten
```

Name, Initialisierung und Beschreibung

Name – der Name einer Einheit oder eines Objektes ist sein Name als Variable. Dieser Name muss also nur vergeben werden wenn ihr diese Einheit oder dieses Objekt später „ansprechen“ oder „abfragen“ möchtet. Dazu ein kleiner Exkurs in die Welt der Skripte:

In allen bisher gebrachten Syntaxbeispielen steht vor dem eigentlichen Befehl immer der Ersatz oder Platzhalter „**Name**“. Dieser „Name“ stellt immer die Variable der Einheit oder des Objektes dar. Wollt ihr einer Einheit also z.B. einen bestimmten Rang per Syntax zuweisen könnte das so bei euch aussehen:

Der **Name**, also die **Variable** der Einheit ist hier: **Alpha_Leader**

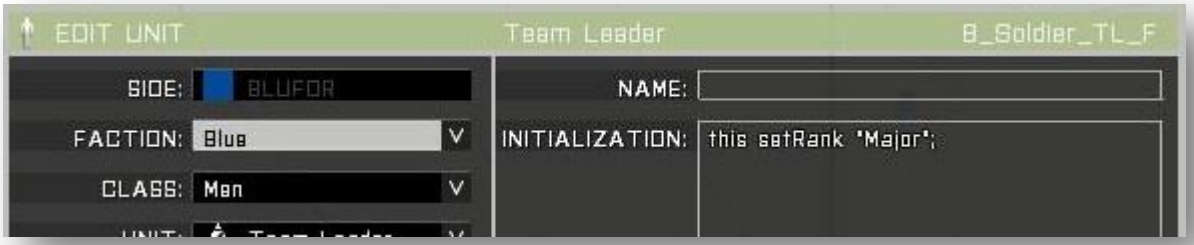
Also ist die konkrete Syntax zum vergeben des Dienstgrades:

```
Alpha_Leader setRank „Major“;
```

Eine Alternative die gleiche Aufgabe, jedoch ohne die Vergabe einer eigenen Variable mit Hilfe einer sogenannten „***Magic Variable***“ zeige ich euch im nächsten Beispiel…

Initialisierung- das Initialisierungsfeld eines Objektes oder einer Einheit dient zum Ausführen bestimmter Befehle welche direkten Einfluss auf die Einheit oder das Objekt haben sollen. Einträge die sich in der Initialisierung befinden werden direkt und unmittelbar nach dem Start einer Mission ausgeführt. Außerdem werden hier ausgeführte Einträge global auf allen „Maschinen“ ausgeführt. Mit Maschinen sind die beteiligten Lokalitäten in einem Spiel beschrieben. Im Falle einer Single Player Mission seid das Ihr, also euer Rechner. Im Falle einer Multiplayer Mission die auf einem echten Server läuft, seit das mindestens Ihr, der Server selbst und alle anderen Spieler auf dem Server. Mehr dazu findet ihr im Kapitel Multiplayer/Lokalitäten. Auch ganze Skripte können hier initialisiert, also gestartet werden.

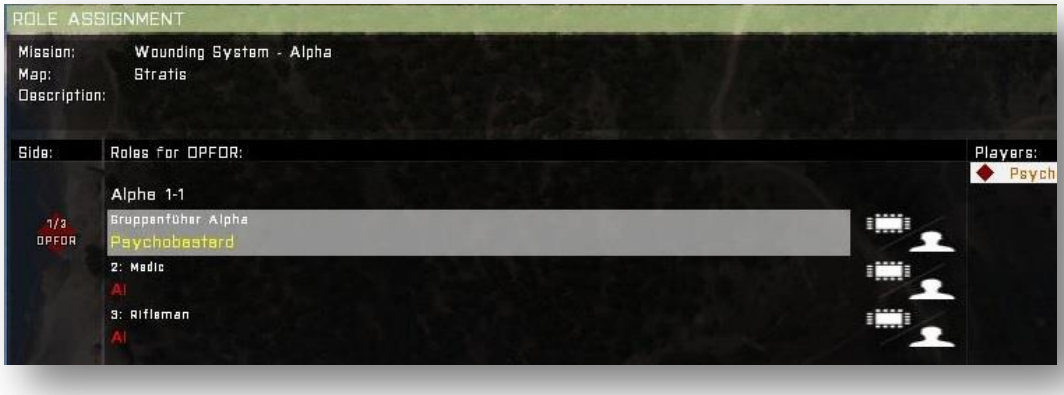
Im Folgenden wieder ein Beispiel wie Ihr die Initialisierung nutzen könnt um einen Rang zu vergeben. Jetzt jedoch ohne die Verwendung einer eigenen Variable ganz allein mit einer „***Magic Variable***“:



Was ist passiert? Es wird **keine Variable** vergeben. Außerdem wurde der Variablenname in dem Initialisierungsfeld durch „**this**“ ausgetauscht. Dieses „**this**“ ist die magische Variable und beschreibt „**diese**“ **Einheit** oder „**dieses**“ **Objekt** auf welche der Befehl „**setRank**“ eben angewendet werden soll.

Bitte beachtet – „**this**“ kann nur im Kontext mit dieser Einheit in dieser Initialisierung verwendet werden und darf nicht mit einem „**_this**“ in Skripten verwechselt werden!!!

Beschreibung – die Beschreibung einer Einheit ist wichtig wenn die Einheit einen eigenen Namen haben soll. Dieser Name ist jetzt wirklich nur ein ganz normaler Name und keine Variable. Das ist dann hilfreich wenn Ihr diese Einheit als Spielbare Einheit definiert und diese in der Multiplayer Lobby mit einem bestimmten Namen angezeigt werden soll.



Wie im Bild zu sehen wurde die Einheit als „**Gruppenführer Alpha**“ benannt. Dieser Eintrag wurde in der Beschreibung der Einheit eingefügt.

Physischer Status

Fähigkeiten – (engl. Skill) beschreibt die Stärke einer Einheit und hat natürlich nur Einfluss auf von der KI kontrollierte Einheiten. Die Fähigkeiten werden mit einer Zahl in einem Bereich zwischen 0 und 1 angegeben. Wobei 1 gleich 100% der Fähigkeiten entspricht oder ein nach ganz rechts geschobenen Balken. Der „**Skill**“ kann auch mit folgender Syntax verändert werden:

```
Name setSkill 0.6;

Name setUnitAbility 0.6;
```

Eine weitere und professionellere Art und Weise die Fähigkeiten einer Einheit zu beeinflussen ist der **setSkill-Array** Befehl.

```
Name setSkill [„aimingAccuracy“, 0.6];
```

- aimingAccuracy - Zielgenauigkeit
- aimingShake - Ziel verreißen
- aimingSpeed - Zielgeschwindigkeit zum Erfassen
- endurance - Ausdauer
- spotDistance - Erfassungsreichweite von Zielen
- spotTime - Erfassungszeit / Reaktionszeit
- courage - Mut (Fluchtverhalten)
- reloadSpeed - Nachladezeit
- commanding - Fähigkeiten als Kommandant
- general - generell, äquivalent mit normalen setSkill

Wie Ihr sehen könnt ist es damit möglich einzelne bestimmte Fähigkeiten einer Einheit zu fördern oder zu schwächen. So könnt Ihr einem Scharfschützen wesentlich bessere Zielfähigkeiten verleihen als einem gewöhnlichen Soldaten.

Anbei noch ein grober Richtwert welcher Skill in etwa welchem Schwierigkeitsgrad entspricht. Es gibt nichts frustrierenderes als gegen eine große Horde von „Super-KI-Soldaten“ zu kämpfen weil der Missionsbauer der Meinung war mit extremen Schwierigkeitsgrad die Spieltiefe steigern zu können. Gebt diesem Punkt also eine besondere Aufmerksamkeit wenn ihr eine Mission entwickeln wollt.

- Novice < 0.25
- Rookie >= 0.25 und <= 0.45
- Recruit > 0.45 und <= 0.65
- Veteran > 0.65 und <= 0.85
- Expert > 0.85

Gesundheit / Panzerung – (engl. Health/Armor) abhängig davon ob es sich um einen Infanteristen oder ein Fahrzeug/Objekt handelt wird hier die Gesundheit bzw. der Status der Hülle beeinflusst. Der Wert wird dabei wieder zwischen 0 und 1 gebildet, wobei 1 gleich 100% entspricht.

```
Name setDamage 1;        -  100% Schaden
```

Auch hier gibt es noch einen Befehl für die fortgeschrittene Beeinflussung des Schadens:

```
Name setHit [„motor“, 1];      - Motor des Fahrzeuges defekt
                 ↙          ↘
              Part          Schadenswert von 0 bis 1
```

Es existiert eine riesige Palette an möglichen Parts die gewählt werden können. Einige können nur an bestimmten Fahrzeugen, andere wieder nur an Infanteristen angewandt werden. Leider sind viele der Parts nicht auf Englisch sondern auf Tschechisch benannt. Das vereinfacht die Suche nach dem passenden Part nicht gerade. Im BI-Wiki existiert eine Translation-Table in der unter anderem auch alle möglichen Hit-Parts gelistet sind. Probieren und Experimentieren ist hier der richtige Weg.

Translation Table: https://community.bistudio.com/wiki/Translations

Treibstoff – (engl. Fuel) setzt die Tankfüllung eines Fahrzeuges zum Missionsstart fest und hat natürlich keinen Einfluss auf die Alkoholreserven des Spielers. Der Wert ist natürlich wieder zwischen 0 und 1 einstellbar. Auch hier kann ein Skriptbefehl das gleiche erledigen:

```
Name setFuel 0.2;      - der Fahrzeugtank ist zu 20% gefüllt
```

Neben dem Treibstofftank für den Eigenverbrauch kann ein Fahrzeug auch einen zweiten Tank zur Versorgung anderer Fahrzeuge besitzen. Dies ist z.B. bei einem Supportfahrzeug wie dem Treibstoff-LKW der Fall. Dieser Tank kann nur mit folgenden Skriptbefehlen und nicht direkt im Editor angesprochen werden:

```
Name setFuelCargo 0.7;      - setzen der Füllmenge
getFuelCargo Name;          - Abfrage des Füllstatus   NEU
```

Munition – (engl. Ammunition) hier wird natürlich die Menge der mitgeführten Munition beeinflusst. Es gibt jedoch diesmal keinen direkten Befehl der Infanteristen beeinflussen könnte. Wird der Schieberegler nach links geschoben verringert sich daher nur die Anzahl der Magazine die die Einheit mit sich führt. Bei Fahrzeugen verringert sich die Munitionsanteil über alle Waffensysteme prozentual. Ein Support-LKW mit Munition bzw. dessen Munitionsvorrat kann folgender Maßen bestimmt werden:

```
Name setAmmoCargo 0.7;   - setzen der Füllmenge
getAmmoCargo Name;       - Abfrage des Füllstatus
```

Anwesenheitswahrscheinlichkeit – (engl. Propability of Presence) gibt die Wahrscheinlichkeit der Anwesenheit einer Einheit zwischen 0 und 100% an und ist ein gutes Stilmittel um für Abwechslungsreiche Missionen zu sorgen.

Anwesenheitsbedingung und Platzierungsradius – Der Platzierungsradius wird als Zahl in Meter angegeben. Die Einheit oder das Objekt werden entsprechend zufällig innerhalb dieses Radius platziert dessen Zentrum der aktuelle Standort der Einheit ist.

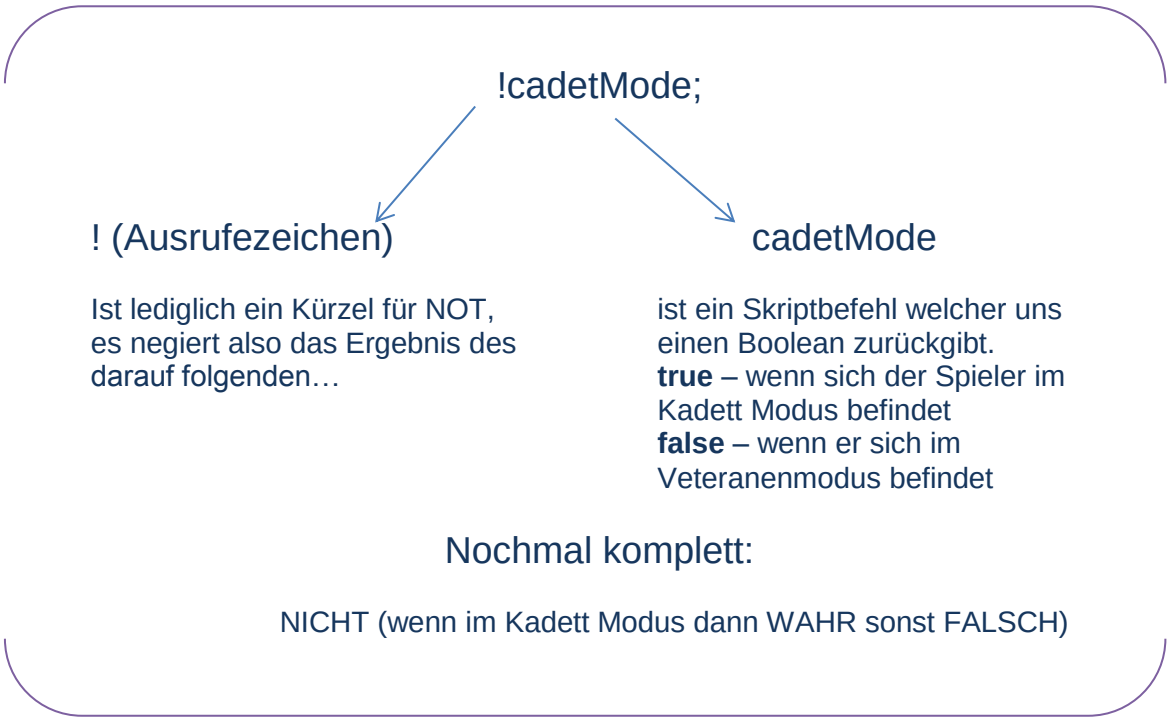
Die Anwesenheitsbedingung macht das was der Name schon sagt. Wenn die Bedingung in diesem Feld „**Wahr**“ ist wird die Einheit erst platziert. Daher steht in diesem Feld auch immer Schlicht und einfach „**true**“. Dieses „**true**“ bzw. die nötige Erfüllte Wahrheitsbedingung wird in der Skriptsprache als sogenannter **Boolean** (Wahrheitswert) bezeichnet.

Was bedeutet das für uns? Es bedeutet in diesem Feld haben Zahlen nichts verloren. Es darf sich ausschließlich ein Code der einen **Bool** zurück liefert (also Wahr o. Falsch / true or false) in diesem Feld befinden!

Ein kleines Anwendungsbeispiel:

In aktuellen Missionen eigentlich nicht mehr anzutreffen war diese Art der Missionsgestaltung zu OFP Zeiten noch Gang und Gebe. Es handelt sich also um einen Old-School Tipp. ☺

Was macht denn nun dieser in dem Bild gezeigte Code in der Anwesenheitswahrscheinlichkeit? Nehmen wir den Code einmal genau unter die Lupe:



Was?! Nicht wenn im Kadett Modus dann wahr sonst falsch?!

„Der trinkt doch“ denkt Ihr euch jetzt bestimmt. Das ist aber die einfachste Logik die es gibt. Denn eine Maschine versteht das. Daher jetzt nochmal für die menschlichen Gedankengänge was da passiert:

Die Abfrage macht nichts anderes außer zu prüfen in welchem **Schwierigkeitsgrad** der Spieler eben das Spiel beginnt. Wurde die Schwierigkeitsstufe **Kadett** gewählt, so wird die Einheit in diesem Beispiel sich nicht auf der Karte befinden. Warum? Die Abfrage „**cadetMode“** liefert uns in dem Fall ein „**true**“, jedoch wird dieses „**true**“ durch unser „**!**“ (Ausrufezeichen) was das gleiche bedeutet wie „**NOT**“ ist negiert, also NICHT TRUE – nicht Wahr – also FALSCH. Wir erinnern uns – die Einheit wird nur platziert wenn die Prüfung dieses Feldes ein „**true**“ ergibt.

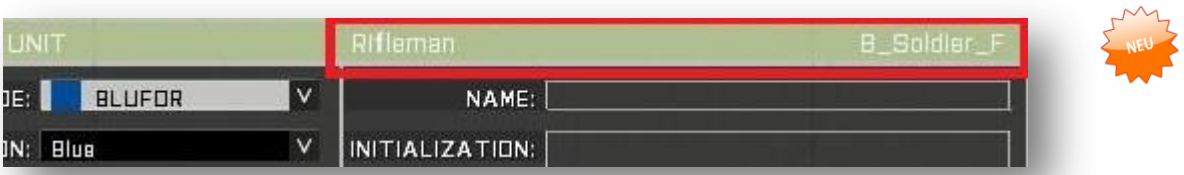
Befindet sich der Spieler jedoch im Veteranen Modus oder höher wird die Einheit erstellt. Warum könnt Ihr jetzt selber erklären. ☺

Die Bedingung der Anwesenheit kann mit der Wahrscheinlichkeit kombiniert werden. Dazu noch der Platzierungsradius und ihr habt ein sehr simples aber extrem mächtiges Werkzeug um eine Mission abwechslungsreich und damit interessant zu gestalten.

Classname im Editor auslesen

Da wären wir auch schon am Ende was das erstellen einer Einheit angeht. Ihr seht schon das man allein mit dieser kleinen Aktion im Editor eine enorm große Auswahl an Optionen und Einstellmöglichkeiten hat die einem dabei helfen eine Mission zu gestalten.

Eine kleine und sehr nützliche Sache, gerade für die Fortgeschrittenen unter euch, wäre da aber noch zu nennen. Im oberen Bereich des Einheiten Dialogs findet ihr jetzt neben dem sogenannten „**Display Name**“ auch den „**Classname**“ der eben selektierten Einheit. Das ist eine äußerst praktische Funktion die einem das schnelle finden von passenden Classnames innerhalb des Editors ermöglicht. Bisher, also in den Vorgängern von ArmA 3, war das nur mithilfe von Listen möglich die von der Community online bereit gestellt wurden.



„Und wo ist der Nutzen?“ - fragen sich die Neulinge.

Der Classname einer Einheit oder eines Objektes ist der „Pfad“ bzw. der Objektname der benötigt wird um die Einheit in Skripten zu nutzen. Man kann diesen Classname dann für bestimmte Abfragen heran ziehen. Zum Beispiel um danach zu Fragen ob eine Einheit ein Sanitäter ist oder nicht. Häufigstes Anwendungsbeispiel ist das Erstellen von Einheiten mit Hilfe von Skripten. Denn um eine Einheit zu erstellen muss man natürlich angeben was für eine Einheit erstellt werden soll.

1.4 Gruppen erstellen 

Diese Funktion bietet euch die Möglichkeit nicht nur einzelne Einheiten sondern gleich ganze Gruppen von Einheiten zu platzieren. Das spart natürlich Zeit. Außerdem sind die hier vordefinierten Gruppenverbände bereits militärisch logisch zusammen gestellt. Ihr müsst euch also nicht so sehr darum kümmern ob die Gruppe die ihr haben wollt auch „militärisch korrekt“ ist.

Die Schaltflächen die Ihr hier findet sind euch mittlerweile bereits aus dem Einheiten Dialog vertraut. Wählt eine Seite, eine Fraktion und eine Klasse. Mit Klasse wird hier die Waffengattung bzw. der Heerestyp gemeint. Zur Auswahl steht z.B. Infanterie, Special Forces oder motorisierte Infanterie. Anschließend könnt ihr euch für eine Gruppe unter „Einheiten“ entscheiden und platzieren.

Neben dem recht übersichtlichen Dialog bietet die Schaltfläche „Gruppe“ jedoch noch eine weitere Steuerungsmöglichkeit im Editor. Denn es wird euch natürlich auch die Möglichkeit gegeben eine Gruppe manuell zusammen zu stellen.

Platziert hierfür einfach mehrere einzelne Einheiten auf der Karte welche Ihr zu einer Gruppe zusammenfassen möchtet. Stellt sicher dass die Schaltfläche „Gruppe“ aktiv ist. Jetzt könnt Ihr per Klick mit der linken Maustaste auf eine der Einheiten und gedrückter Maustaste per Mausbewegung die Einheiten miteinander verbinden. Ob eine Verbindung zustande gekommen ist erkennt Ihr an der blauen Linie die jetzt zwischen den Einheiten existiert.

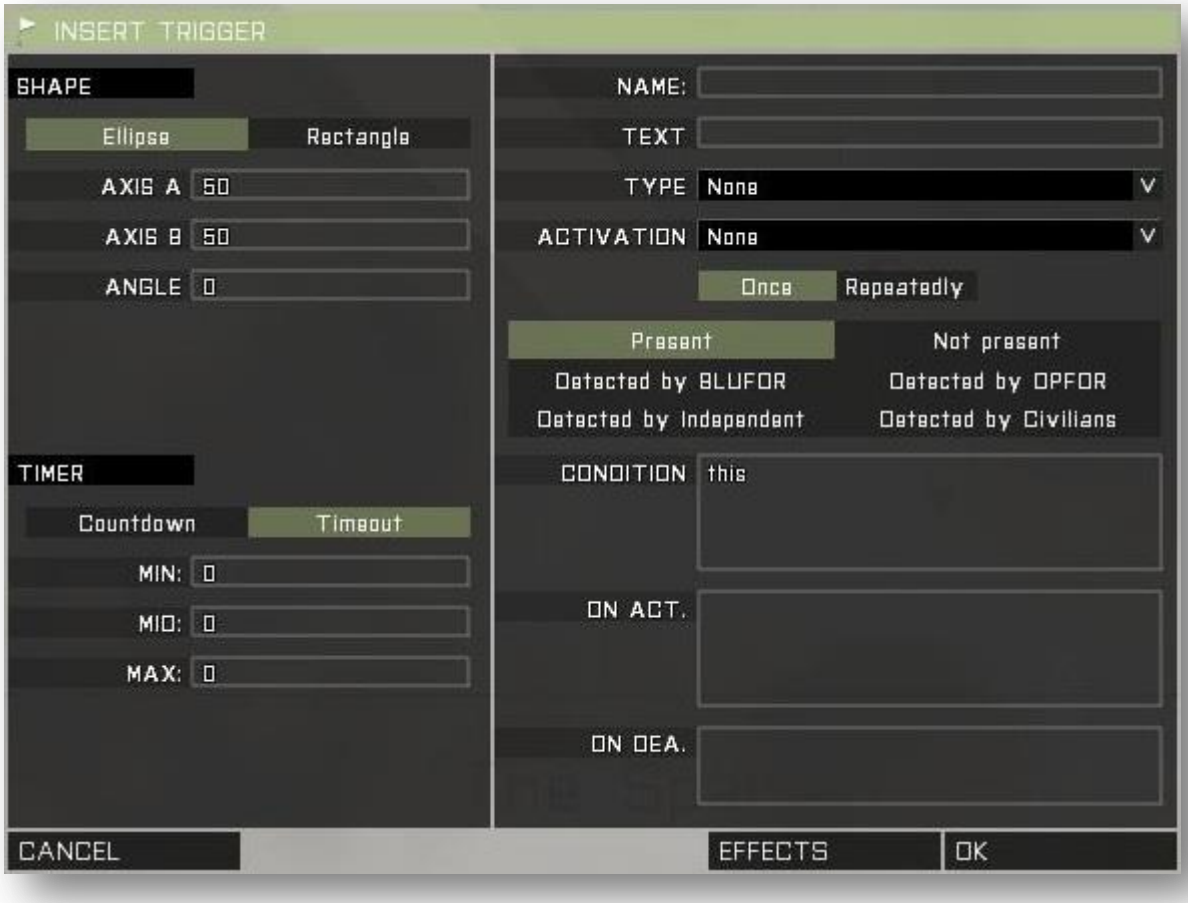
Wie auf dem Bild zu sehen habe ich vier Einheiten miteinander verbunden. Wenn alle 4 Einheiten denselben Rang (Dienstgrad) verwenden, wird derjenige der Anführer auf den Ihr die Maus gezogen habt. Wenn die Ränge unterschiedlich sind, wird derjenige der vier Soldaten mit dem höchsten Rang zum Gruppenführer.

Genauso wie Ihr eine Gruppe manuell erstellt, könnt Ihr diese auch wieder manuell trennen. Zieht dazu bei gedrückter Linker Maustaste die blaue Linie einfach ins Leere und der Soldat wird wieder aus der Gruppe ausgegliedert.

1.5 Auslöser

Mit den sogenannten **Auslösern** (engl. Trigger) lässt sich nahezu alles realisieren was Ihr möchtet. Diese Funktion ist daher die umfangreichste und wichtigste unter allen verfügbaren Schaltflächen im Editor.

Mit dem Trigger lassen sich territoriale Bereiche auf Anwesenheit oder Abwesenheit bestimmter Einheiten, Objekte oder ganzer Fraktionen prüfen. Genauso gut kann man damit aber auch alles Erdenkliche in einer Trigger Area (Bereich) ansprechen und beeinflussen. Ein Trigger aktiviert oder deaktiviert Skripte. Er kann Variablen verändern, ein Spiel beenden oder über die Schaltfläche „Effekte“ Musik- Sound- und Texteinblendungen realisieren. Außerdem lassen sich mit dem Trigger unkompliziert Aktionen über einen Funkspruch realisieren.



Timer – hiermit kann die Auslösung des Triggers verzögert werden. Der **MIN / MID / MAX** Wert ist eine Zahl in Sekunden. Mit diesem Werkzeug könnt Ihr Dynamik in eure Mission bringen bzw. einen Ablauf verzögern wenn dies benötigt wird.

Der Unterschied zwischen Countdown und Timer:

- **Der Countdown verzögert die Aktivierung** des Triggers, also dessen Effekt oder Wirkung
- Das **Timeout** hingegen ist die Zeit, in welcher permanent die **Auslösebedingung** gegeben sein muss, erst dann wird der Trigger aktiviert

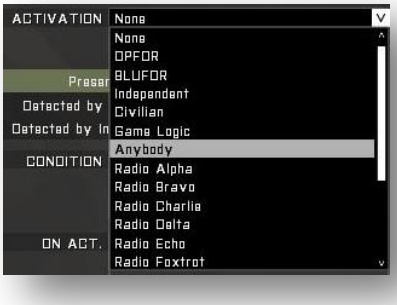
Name – steht hier wie auch bei den Einheiten für einen möglichen Variablennamen des Triggers.

Text – ist die Beschreibung oder unter der Verwendung des Funkauslösers der angezeigte Funkspruch oder Funkbefehl.

Typ des Auslösers –

None (keine)	Standard
Guarded by OPFOR	Geschützt durch Osten
Guarded by BLUFOR	Geschützt durch Westen
Guarded by Independent	Geschützt durch Widerstand
Switch	Schalten
End #1 bis #6	beendet die Mission (6 mögliche Enden)
Lose	Ein weiterer End-Typ, Verlieren

Aktivierung durch – Wer oder Was kann den Trigger aktivieren?



OPFOR
BLUFOR
Independent (Widerstand)
Civilian (Zivilisten)
Game Logic (Spiellogik)
Anybody (Jeder)
Funk Alpha bis Juliet
Erobert durch BLUFOR
Erobert durch OPFOR
Erobert durch Independent

Art der Aktivierung – Wie wird der Trigger aktiviert, bezogen auf „**Aktivierung durch**“

Einfach / Mehrfach (engl. Once / Repeatedly): Mit der Mehrfachen Auslösung wird der Trigger immer wenn die Bedingung erneut erfüllt wurde aktiviert. Um die Bedingung erneut zu erfüllen, muss die Bedingung zwischenzeitlich unterbrochen worden sein!

Beachtet! Eine einfache Auslösung ist im Multiplayer nicht immer eine einfache – beachtet hierzu die gesonderten Informationen im Multiplayer Kapitel auf Seite xx.

Die weiteren Schaltflächen wie **Anwesend / nicht Anwesend** (engl. Present / Not present) sind selbsterklärend.

Die Bedingung – (engl. Condition) ein **Wahrheitswert** der den Trigger aktiviert

In diesem Fenster wird die Auslösebedingung definiert. Hierbei ist entscheidend, dass die Bedingung „**true**“ oder „**false**“ liefert, also den Datentyp **Boolean**. (wurde bereits im Thema Anwesenheitsbedingung, Seite 16, behandelt) Standardmäßig steht hier wenn Ihr einen Trigger anlegt „**this**“ als Bedingung. Das „**this**“ Verweist auf die bereits bestehenden Parameter unter „Typ des Auslösers“, „Aktivierung durch“ und „Art der Aktivierung“.

Die Bedingung lässt sich aber natürlich auch beliebig mit den vorherigen Parameter kombinieren. Mit Hilfe der zur Verfügung stehenden Skriptbefehle die uns ArmA bietet kann mit einem Trigger so ziemlich alles abgefragt werden was man sich vorstellen kann.

Wird hier eine Syntax eingegeben die nicht korrekt ist oder nicht dem Datentyp „**Bool**“ entspricht, wird euch das mittels einer Warnmeldung mitgeteilt und Ihr könnt den Trigger nicht per Mausklick auf OK vollenden bis der Fehler behoben wurde.

Aktivierung – (engl. Activation) Aktion die nach erfüllter Bedingung eintritt

In diesem Eingabefeld kann eine Aktion nach erfüllter Bedingung ausgeführt werden. Das ist in den meisten Fällen das setzen bzw. verändern eines Variablenwertes. Dabei ist es diesmal unerheblich um was für einen Datentyp es sich dabei handelt. Es muss aber nicht immer eine Variable sein. Es kann z.B. auch ein Code ausgeführt werden. Oder gleich ein ganzen Skript oder eine Funktion gestartet werden die wiederum alles Erdenkliche beinhalten kann.

Wird hier eine Syntax eingegeben die nicht korrekt ist wird euch das mittels einer Warnmeldung mitgeteilt und Ihr könnt den Trigger nicht per Mausklick auf OK vollenden bis der Fehler behoben wurde.

Im Folgenden sind Beispielhaft ein paar typische Syntaxbeispiele die in einer Aktivierung vorkommen könnten dargestellt.

- Eine Variable setzen bzw. Variablenwert ändern:

```
Geisel_befreit = true;
```

- Einen Code ausführen:

```
this addAction [“Generator abschalten“,“scripts\generator.sqf“];
```

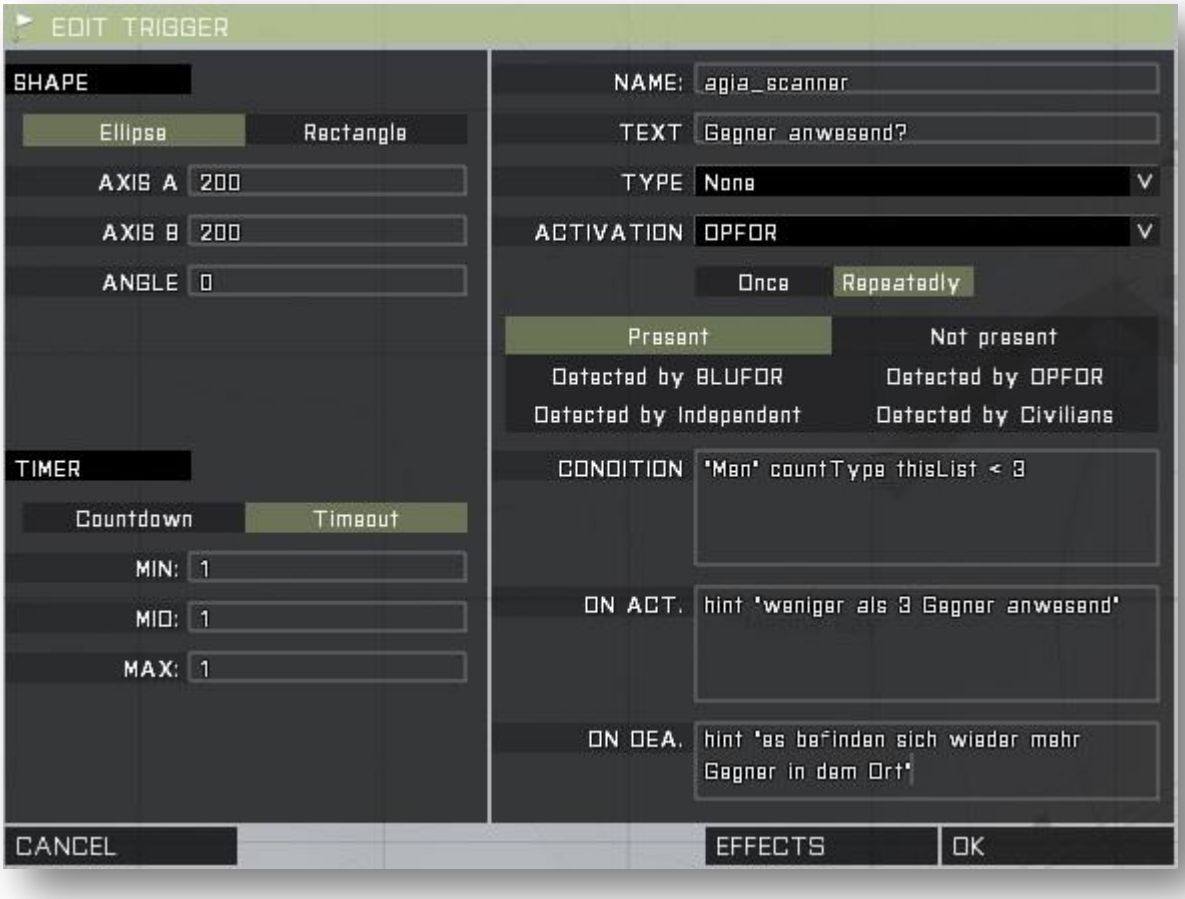
- Ein Skript starten:

```
data = [this, geisel] execVM “scripts\hostage.sqf“;
```

Deaktivierung – (engl. Deactivation) Aktion die Ausgeführt wird wenn eine bereits erfüllte Bedingung plötzlich nicht mehr erfüllt ist.

Dieses Feld bietet denselben Funktionsumfang wie die Aktivierung selbst, jedoch invertiert.

Das folgende Beispiel soll die Funktionen des Triggers noch einmal kurz anhand eines praktischen Beispiels erläutern.



Was macht dieser Trigger denn nun?

Es handelt sich um einen „**Area-Trigger**“, er befindet sich also über einem definierten Gebiet und soll über dieses Gebiet Informationen sammeln die ich auswerten möchte. In diesem Fall will ich wissen ob sich noch Gegner in dem Gebiet befinden. Daher steht die „Aktivierung durch“ auf OPFOR. Außerdem habe ich die Mehrfachaktivierung eingeschaltet und verwende neben dem Aktivierungsbereich auch den Deaktivierungsbereich um mir eine Textmeldung auf dem Bildschirm anzeigen zu lassen. (Befehl „**hint**“)

Die Bedingung lautet:

```
“Men“ countType thisList < 3
```

„Übersetzt“ bedeutet das: Zähle die definierten Einheiten (also OPFOR) des Typus „Men“ in diesem Bereich.

Diese Anweisung alleine stellt jedoch keinen Rückgabewert des Datentyps „Bool“ dar sondern liefert mir eine Zahl. Genauer gesagt die Anzahl der Infanterieeinheiten von OPFOR in diesem Gebiet. Daher wird ein Vergleicher und eine weitere Zahl hinzugefügt. Jetzt kann der gezählte Wert mit einer anderen Zahl, also der 3, verglichen werden. Das Gesamtergebnis der Bedingung liefert jetzt folglich einen Bool, also „**true**“ oder „**false**“.

Im Falle das die Bedingung „true“ wird, sich also weniger als drei OPFOR Infanterieeinheiten in dem Gebiet aufhalten, erscheint auf dem Bildschirm eine Textmeldung mit dem Inhalt: „weniger als 3 Gegner anwesend“, initialisiert durch den Code im Aktivierungsbereich. Laufen jetzt plötzlich wieder mehr gegnerische Soldaten in das Gebiet wird die Deaktivierung initialisiert. Es erscheint also die Textmeldung: „es befinden sich wieder mehr Gegner in dem Ort“.

Da der Trigger auf mehrfache Auslösung eingestellt wurde kann sich dieser Vorgang natürlich wiederholen. Und wie lauten die Bedingungen dafür?
Das wisst Ihr bereits da dies bereits im Abschnitt „Art der Aktivierung“ behandelt wurde.

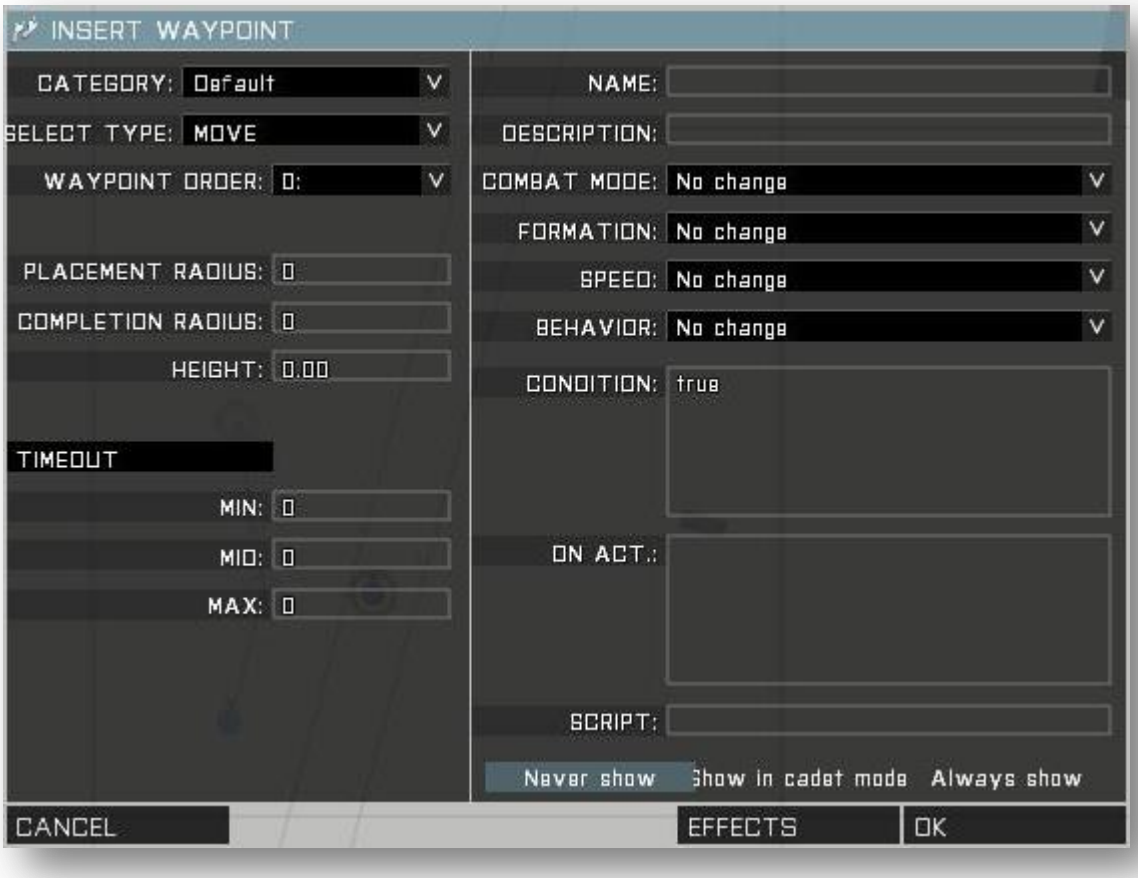
Effekte mit einem Trigger – (engl. Effects)

Hinter dieser Schaltfläche verbergen sich diverse Möglichkeiten **Musik, Sounds, Texte** oder andere grafische Ressourcen einfach einfügen und abspielen zu lassen. Die Effekte werden dann abgespielt wenn auch der Trigger aktiviert wurde.

Die euch hier gebotenen Möglichkeiten werden meiner Meinung nach viel zu selten benutzt. Musik oder Texteinblendungen können Stimmung aufbauen oder bestimmte Momente untermalen und stellen ein essentielles Stilmittel dar.

1.6 Wegpunkte

Die Vergabe von Wegpunkten (engl. Waypoints) haucht den KI-Soldaten Leben ein oder zeigt den Spielern wohin diese gehen sollen. Wegpunkte lassen sich einfach mit der Maus erstellen. Klickt auf die Schaltfläche „Wegpunkte“, anschließend auf einen Soldaten oder ein Fahrzeug um dieses zu markieren. Jetzt genügt ein Doppelklick mit der linken Maustaste auf einen Punkt auf der Karte um an dieser Position einen Wegpunkt zu setzten. Es öffnet sich der **Wegpunktdialog**.



Und wie sollte es anders sein, wieder wird euch eine große Auswahl an Einstellungsmöglichkeiten geboten mit denen Ihr das Verhalten des Wegpunktes bzw. der zugehörigen Einheit beeinflussen könnt.

Mit den meisten der hier dargestellten Schaltflächen seid Ihr bereits aus vorherergehenden Kapiteln vertraut. Daher möchte ich an dieser Stelle nur kurz auf die für den Wegpunktdialog spezifischen Einstellungen näher eingehen.

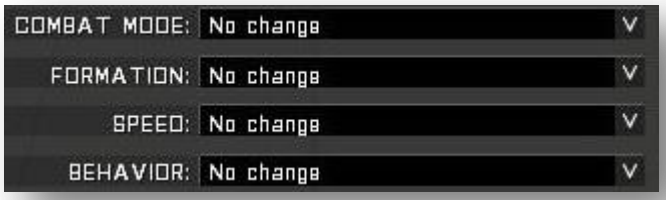
Platzierungs- und Abschlussradius

Beide Angaben werden als Zahl in Metern ausgedrückt. Der **Platzierungsradius** stellt eine Fläche um das Zentrum des Wegpunktes dar, in dem sich bei Missionsstart jetzt zufällig der Zielpunkt des Wegpunktes befinden wird. Gibt man einer Einheit also einen Wegpunkt mit einem Platzierungsradius von 100 Metern, wird sich diese **irgendwo im Radius** von 100 Metern um den eigentlichen Standort des Wegpunktes hinbewegen. Der Bereich wird euch durch eine gestrichelte Linie um den Wegpunkt herum im Editor visualisiert.

Der **Abschlussradius** legt fest wie dicht die Einheit an dem Zielpunkt sein muss bis der Wegpunkt als erreicht gilt. Diese Option ist besonders dann interessant wenn man Flugzeugen oder Helikoptern einen Wegpunkt vergibt. Durch die Trägheit in der Bewegung kann es sein, dass der Spieler oder auch die KI große Probleme dabei bekommt einen Wegpunkt bei großer Geschwindigkeit in der Luft zu „**treffen**“. Daher bietet es sich immer an eine gewisse **Tolleranz** in Form des Abschlussradius für diese Einheitentypen zu definieren.

Die **Höhe** eines Wegpunktes ist neu und erst seit ArmA 3 in den Einstellungen zu finden. Was vorher nur mit Hilfe von Skripten realisiert werden konnte lässt sich nun ganz einfach im Editor erledigen. Darüber hinaus erkennt der Wegpunkt bereits beim platzieren in welcher Höhe er sich befinden muss. Wird der Wegpunkt z.B. über ein hohes begehbares Gebäude (z.B. Tower auf dem Flugfeld) gelegt, befindet sich der Wegpunkt **automatisch auf dem Niveau** des höher gelegenen Raumes. Wird jetzt zusätzlich eine Höhe definiert, addiert sich diese Höhe zu der automatisch vergebenen Höhe. Mit negativen Einträgen lässt sich der Wegpunkt natürlich auch nach unten verschieben. Wird jetzt noch ein Aktivierungsradius mit der Höhe kombiniert, kann man sich den Aktivierungsbereich wie eine **Kugel** im Raum vorstellen durch welche die Einheit schreiten muss um den Wegpunkt zu erreichen. Das ist besonders praktisch um z.B. Helikopter nicht nur Zweidimensional sondern **Dreidimensional** zu führen. Achtung, dabei aber nicht denken das der Helikopter jetzt automatisch auf diese Höhe fliegen wird!

Verhalten bei Wegpunkt



Diese vier Einstellungen beeinflussen das **Verhalten, die Aufmerksamkeit und die Geschwindigkeit** von KI-Soldaten bei ihren Bewegungen. Jedoch erst nachdem der Wegpunkt in dem die Einstellungen getroffen wurden auch abgearbeitet wurde! Besonders das **Verhalten** (engl. Behavior) beeinflusst die Art und Weise wie sich die Einheit durch das Gelände bewegen wird und wieviel Zeit sie dafür benötigt. Beispielsweise führt die Einstellung „Sicher“ (engl. Save) zu einer gehenden Bewegung mit gesenkter Waffe, geradlinig von einem Punkt zum anderen. Das Verhaltensmuster „Tarnung“ (engl. Stealth) führt hingegendazu das sich der Soldat geduckt von Deckung zu Deckung bewegen wird um möglichst unerkannt zu bleiben. Dafür benötigt er dann natürlich auch mehr Zeit.

Der **Kampfmodus** (engl. Combat Mode) kann auch mit folgender Syntax beeinflusst werden:

```
Name setCombatMode “RED“;
```

Die Formation (engl. Formation) kann auch mit folgender Syntax beeinflusst werden:

```
Name setFormation “COLUMN“;
```

Die Geschwindigkeit (engl. Speed) kann auch mit folgender Syntax beeinflusst werden:

```
Name setSpeed “FULL“;
```

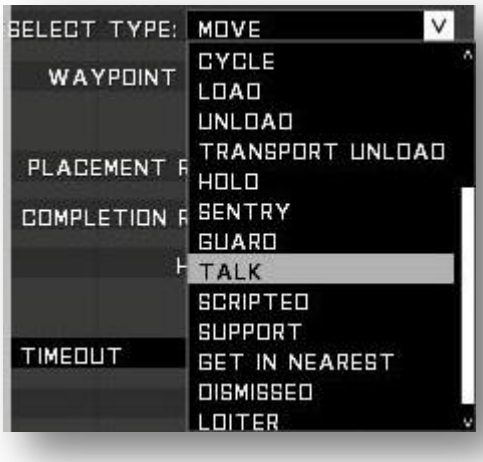
Das Verhalten (engl. Behavior) kann auch mit folgender Syntax beeinflusst werden:

```
Name setBehaviour “STEALTH“;
```

Wegpunkte anzeigen

Diese Option ist dann interessant wenn Ihr einem Spieler einen Wegpunkt zuweisen wollt. Wählt Ihr hier aus das der Wegpunkt angezeigt wird bedeutet dies, dass der Spieler mittels einer **HUD-Marke** den Wegpunkt im Gelände angezeigt bekommt. Ein kleiner Pfeil markiert dann die Position. Außerdem wird die Distanz in Metern angezeigt. Wird ein Text unter „Beschreibung“ eingegeben wird auch dieser zusätzlich zum Wegpunkt auf dem HUD angezeigt. „**Anzeigen im Kadettmodus**“ führt dazu das die Anzeige vom eingestellten **Schwierigkeitsgrad** abhängig gemacht wird.

Der Wegpunkt Typ



Der Typ des Wegpunktes stellt die wichtigste Einstellmöglichkeit bei einem Wegpunkt dar. Der **Typ** legt fest ob es sich um eine einfache Bewegung handelt oder ob bei der Aktivierung eine bestimmte **Aktion** ausgeführt werden soll. Das kann z.B. das besteigen eines Trucks sein. Einige **Typen** benötigen um die gewünschte Aktion ausführen zu können zusätzliche Objekte mit denen der Wegpunkt synchronisiert werden muss. Das Thema „**Synchronisieren**“ wird im Abschnitt 1.7 auf Seite 32 noch näher erläutert.

Im folgenden möchte ich euch die wichtigsten Wegpunkttypen benennen und kurz ansprechen was man mit diesem Typ anstellen kann. Genaue Anwendungsbeispiele werden euch dann im Kapitel xx gezeigt.

Bewegen – (engl. Move) eine einfache Bewegung von A nach B

Einsteigen – (engl. Get In) Wegpunkt muss auf ein Fahrzeug zeigen, der Soldat steigt in dieses ein sofern Plätze frei sind.

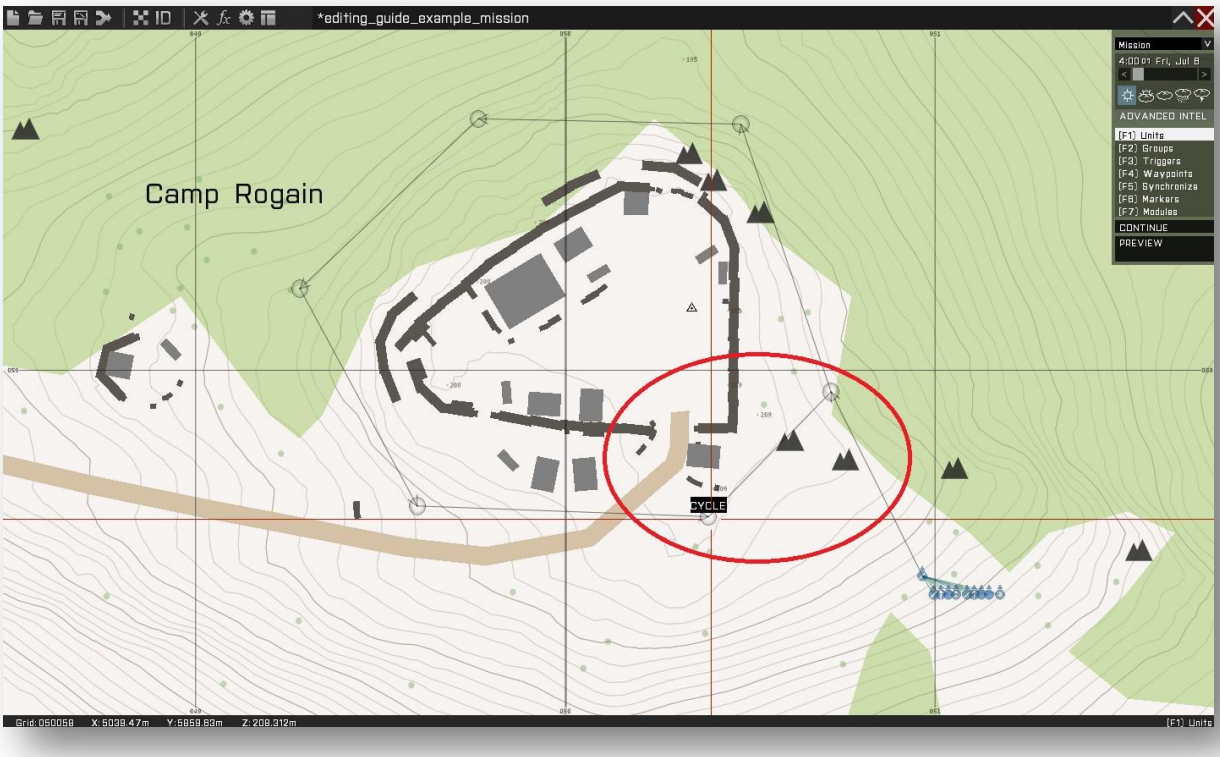
Suchen und Zerstören – (engl. Seek and Destroy) die Einheit bewegt sich zu dem Punkt und fängt an für eine Weile die Umgebung abzusuchen. Dafür bewegt die Einheit sich auch frei. Hat sich die Einheit davon überzeugt das keine Kontakte in der Umgebung sind gilt der Wegpunkt als abgearbeitet.

Anschließen – (engl. Join) Wegpunkt muss auf ein Mitglied einer anderen Gruppe zeigen. Bei erreichen des Mitgliedes tritt die Einheit dieser Gruppe bei.

Wiederholen – (engl. Cycle) erstellt eine Schleife um bestimmte Wegpunkte zu wiederholen. Siehe hierzu auch „**Patrouille erstellen**“.

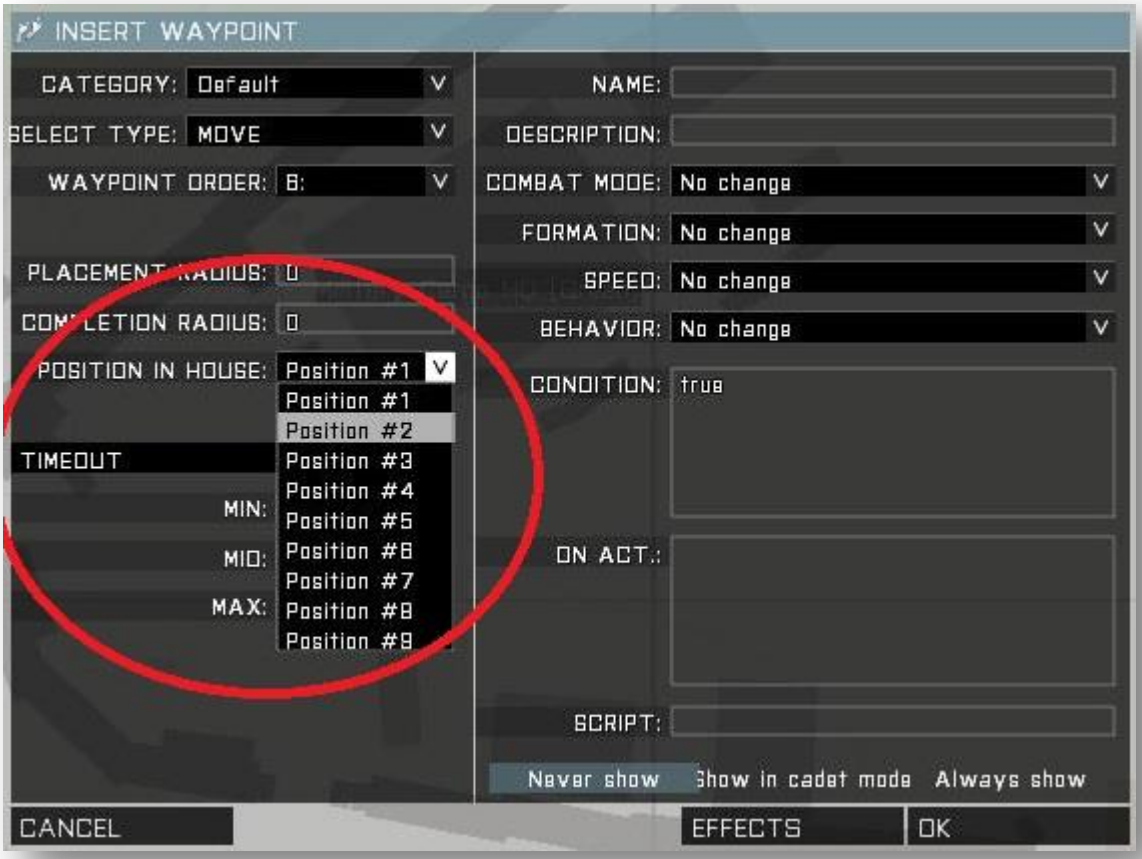
Patrouille erstellen

Das folgende Beispiel soll zeigen wie man eine **Patroullie** erstellt die in einer Endlosschleife fortgeführt wird. Dazu wurden der Gruppe eine Reihe ganz normaler „**Bewegen**“ Wegpunkte zugeteilt. Der letzte Wegpunkt in dieser Abfolge ist jedoch ein Wegpunkt vom Typ „**Wiederholen**“. Eine schwarze Linie, hier rot eingekreist, zeigt auf den Wegpunkt der wiederholt werden soll. Die Verbindung wird immer automatisch mit dem jeweils am **nähesten** gelegenen Wegpunkt der Gruppe oder dem Gruppenführer aufgebaut. Achtet also darauf wohin die Verbindung zeigt wenn Ihr diesen Wegpunkttyp verwendet.



Wegpunkt auf ein Gebäude

Ein weiterer Spezialfall den ich euch gerne näher bringen möchte. Wenn Ihr einen Wegpunkt genau auf ein **Gebäude** setzt werden euch weitere Optionen in dem Wegpunktedialog angeboten. Achtet beim erstellen des Wegpunktes darauf, das sich die Maus genau über einem Gebäude befindet. Das ist der Fall wenn der Tooltipp des Mauszeigers den Namen des Gebäudes anzeigt.



Statt der Höhe des Wegpunktes erscheint im Dialog jetzt ein Dropdown welches euch die zur Verfügung stehenden **Positionen** des begehbaren Gebäudes anzeigt. Die Positionen stellen in der Regel taktische **Schlüsselpositionen** in einem Raum dar. Das kann z.B. eine Tür oder ein Fenster oder auf dem Dach sein.

Weist einer Einheit KI-Soldaten einen solchen Wegpunkt zu und die Einheiten werden sich in das Gebäude begeben und die jeweiligen Positionen besetzten. So ist es euch möglich **ohne Skripte** und großen Aufwand ein Gebäude durch Soldaten besetzen zu lassen.

Wendet Ihr diesen Befehl auf eine ganze Gruppe an und stellt den Wegpunkttyp zusätzlich auf „**Bewachen**“ werden die Einheiten das Gebäude teilweise besetzen und zusätzlich einen Verteidigungsring mit Hilfe der Rundumsicherung um selbiges errichten.

1.7 Synchronisieren 

Dieses Werkzeug wird oft unterschätzt zu wenig angewandt. Dabei hilft es schnell und übersichtlich bestimmte Funktionen zu realisieren. Es können die verschiedensten Sachen miteinander synchronisiert werden. So zum Beispiel Wegpunkten mit Wegpunkten oder Wegpunkte mit Triggern oder Einheiten mit Modulen.

Im Folgenden wird die Synchronisation von verschiedenen Dingen anhand von Praxisbeispielen aufgezeigt.

Wegpunkt mit Wegpunkt

Das folgende Beispiel isst ein unter Missionsbauern oft benötigter Ablauf: Ein Helikopter soll zu einer bestimmten Landezone fliegen, dort landen, warten bis eine Gruppe eingestiegen ist und weiter fliegen, z.B. um die Mission erfolgreich zu beenden.

Hier erhält der Helikopter einen Wegpunkt der genau auf einem in diesem Fall unsichtbaren Heli-H (gelbes Fragezeichen) platziert wird. Der Typ steht auf „**Einladen**“ (engl. Load).

Die Einheit die den Helikopter besteigen soll erhält einen Wegpunkt neben der Landezone vom Typ „**Einsteigen**“ (engl. Get In).

Jetzt werden die beiden Wegpunkte **synchronisiert**. Was macht diese Synchronisierung der beiden Wegpunkte jetzt? Sie sorgt dafür, dass die Wegpunkte aufeinander zeitlich abgestimmt werden und der Soldat dort wartet bis der Helikopter gelandet ist. Andersrum würde der Helikopter auf der Landezone warten bis der Soldat eingestiegen ist. Erst wenn beide Einheiten ihren Wegpunkt abgearbeitet haben geht es weiter. Der Helikopter steigt wieder auf und fliegt zum nächsten Wegpunkt welcher dann z.B. wieder vom Typ „**Transport entladen**“ sein könnte.

Synchronisiert wird ganz einfach mit der Maus. Wählt „Synchronisieren“ (F5) aus und klickt auf den Wegpunkt des Soldaten. Mit **gedrückter Maustaste** zieht ihr eine jetzt erscheinende blaue Linie auf den Wegpunkt des Helikopters - fertig.

In diesem Beispiel wurde der Soldatenwegpunkt mit dem des Helikopters synchronisiert. Es macht in einigen bestimmten Fällen einen enormen Unterschied **von** Wem **nach** Was die Synchronisierung vorgenommen wird!

Wegpunkt mit Trigger

Schauen wir uns folgende Ausgangslage an:

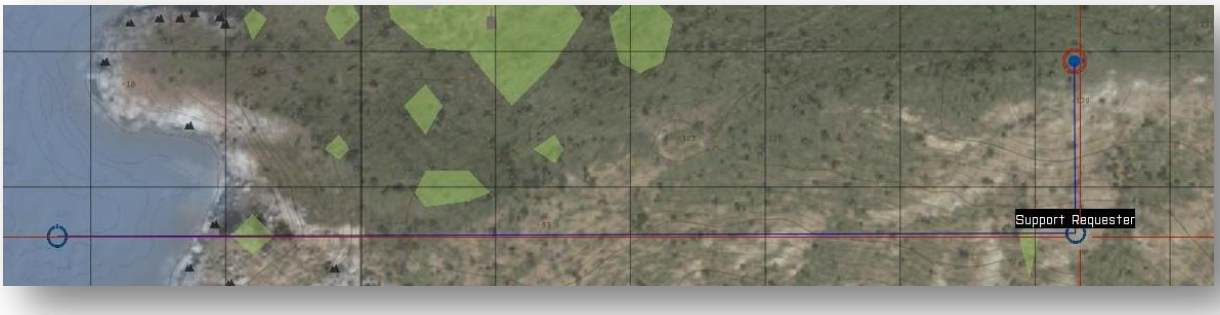
Hier soll der Alpha-Squad einen Punkt mittels zweier Wegpunkte angreifen. Aber er soll am ersten Wegpunkt darauf warten bis der Scharfschütze die Overwatch-Position erreicht hat um diesen dabei zu unterstützen.

Der **Trigger** prüft dabei die Anwesenheit von BLUFOR in dem Gebiet. Wird der **Trigger aktiviert** wird auch der Wegpunkt für die **Abarbeitung** von dem Alpha-Squad frei gegeben. Es ist also auch hier möglich ohne weitere Variablen und Skripte einen koordinierten Angriff vorzubereiten.

Der Trigger welcher dem Alpha-Squad das Vorrücken erlaubt könnte genauso gut ein **Funkauslöser** sein. Der Spieler könnte in Ruhe die Lage observieren und erst wenn er bereit ist mittels Radiosignal (z.B. **0-0-1** Funk Alpha) dem Alpha-Squad den Befehl zum Angriff übermitteln.

Einheit mit Modul

Im folgenden Beispiel soll mittels der Support Module dem Spieler die Möglichkeit gegeben werden CAS (Close Air Support) zur Unterstützung zu ordern. Dafür sieht der Aufbau dann wie im folgenden Bild aus.



Der Spieler wird mit dem sogenannten „**Support Requester Modul**“ (Unterstützung anfordern) synchronisiert. Anschließend wird ein beliebiges Support-Modul auf der Karte platziert. In meinem Fall handelt es sich um das „**Support Provider CAS (virtual)-Modul**“. Dieses wird ebenfalls mit dem „**Support Requester Modul**“ synchronisiert.

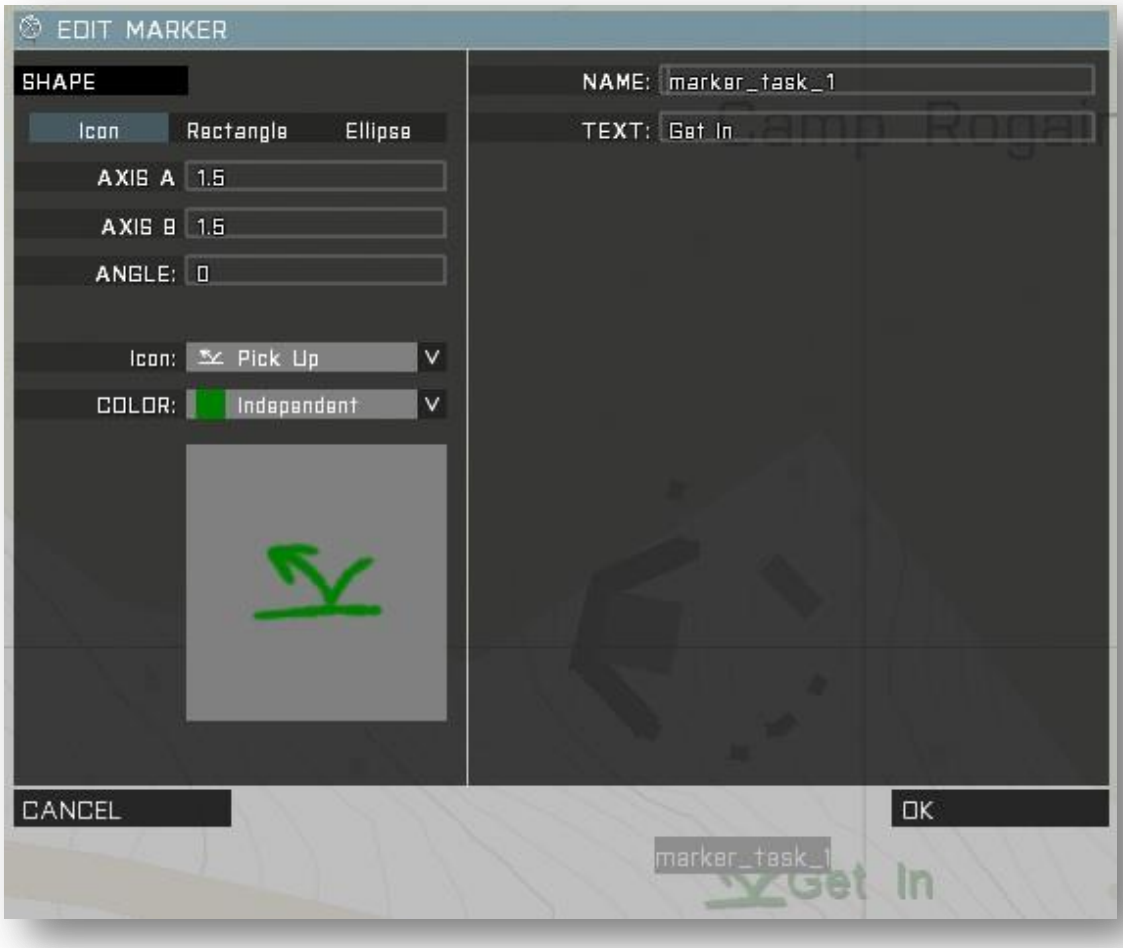
Anschließend kann der Spieler mittels der Tastenkombination **0-8-1** den CAS rufen. Achtet darauf, dass sich das Provider-Modul etwas abseits befindet da an dieser Position die Helikopter erzeugt werden.

Ähnliche Abläufe werden bei allen anderen Modulen benötigt. Mehr dazu gibt es unter **1.9 Module** auf Seite 39 zu lesen.

1.8 Markierungen

Markierungen sind natürlich ein wichtiges Stilmittel in jeder Mission. Erst mit den Markern wird dem Spieler bildlich klar gemacht worum es in der Mission überhaupt geht. Die Marker können also einfach nur einen für die Mission entscheidenden Ort markieren oder einen kompletten taktischen Schlachtplan widerspiegeln.

Marker können ganze Flächen in diversen Farben eindecken oder aus bestimmten Symbolen bestehen. Die Symbolpalette die uns ArmA anbietet ist gigantisch. Von einem simplen Punkt über komplexe NATO Symbole bis hin zu Flaggen verschiedener Nationalitäten ist alles vorhanden.



Dabei sollte man sich dann auch gleich die Frage stellen welcher Typ von Markern für die eigene Mission am geeignetsten ist. Erstellt ihr eben einen Guerillaauftrag und kämpft mit Milizen? Da werden sich wohl kaum die NATO Symbole anbieten sondern wohl eher die „handschriftliche“ Symbolik.

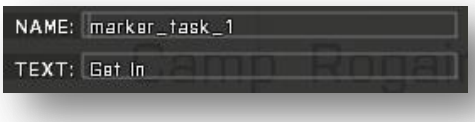
Oft sieht man auch Missionen die von Markern so überschüttet sind, dass das wesentliche unter dieser Flut von Informationen verloren geht. Weniger ist oft mehr! Bedenkt das bei der Vergabe von Markern und überlegt was wirklich einer Markierung bedarf und was nicht.

Um einen Marker auf der Karte zu platzieren wählt ihr einfach „Markierungen“ (F6) aus und öffnet das Menü mit einem Doppelklick auf der gewünschten Position in der Karte. Die verschiedenen Bereiche in diesem Dialog (siehe vorh. Bild) sind selbsterklärend und euch bereits aus den bisher kennengelernten Dialogen geläufig.

Oben links könnt ihr entscheiden ob ihr aus einem der vielen **Symbole** wählen wollt oder lieber eine rechteckige oder ellipsenförmige **Fläche** definiert.

Anschließend kann die Größe bzw. die Fläche und Winkel definiert werden. Tipp: definiert den Winkel erst wenn ihr wieder auf der Karte seit. Klickt dazu einmal auf den Marker um ihn zu markieren. Anschließend drückt und haltet die linke Shift-Taste gedrückt. Wenn ihr jetzt noch die linke Maustaste gedrückt haltet und die Maus dazu bewegt wird sich der Marker drehen und ihr könnt ihn perfekt ausrichten.

Danach entscheidet man sich für ein Symbol und eine passenden **Farbe**. Habt ihr statt dem Symbol eine Fläche gewählt könnt ihr hier unter diversen Flächenfüllmustern wählen.



Der **Name** eines Markers ist seine Variable mit der ihr den Marker später, falls gewünscht, ansprechen bzw. abfragen könnt. Der Name eines Markers kann nur einmal vergeben werden.

Der **Text** wird euch auf der Karte neben dem Marker angezeigt und kann alles mögliche beinhalten oder beschreiben.

Markierungen beeinflußen

Es ist möglich während des Spielverlaufes auf einen Marker Einfluß zu nehmen und sein Erscheinungsbild zu ändern. Folgende Dinge können an einem Marker verändert werden:

- Farbe, Winkel, Größe, Markertyp, Position, Text, Sichtbarkeit oder auch Löschen

also eigentlich alles.

Wozu ist das gut? Na klar – um den Fortschritt in euer Mission auch visuell hervorzuheben oder eine Statusänderung kenntlich zu machen. Man könnte auch erst unsichtbare Marker im späteren Missionsverlauf aufdecken wenn man den Spielern nicht vorher alles verraten möchte. Simpelstes Anwendungsbeispiel ist wohl das färben eines roten Markers auf grün nachdem ein Missionsziel erledigt wurde.

Im Folgenden ein paar Syntaxbeispiele wie man einen Marker beeinflussen kann. Um einen Marker ansprechen zu können benötigt man seinen Namen. In den folgenden Beispielen ist der Markername immer **Marker_1**.

Die Farbe eines Markers verändern:

```
“Marker_1“ setMarkerColor “ColorGreen“;
```

Die Ausrichtung eines Markers verändern:

```
“Marker_1“ setMarkerDir 45;
```

Die Größe eines Markers verändern:

```
“Marker_1“ setMarkerSize [5,20];
```

Den Typ eines Markers verändern, also sein Erscheinungsbild:

```
“Marker_1“ setMarkerTyp “Destroy“;
```

Die Position eines Markers beeinflußen:

```
“Marker_1“ setMarkerPos [x,y];
```

Koordinaten angeben (2D)

```
“Marker_1“ setMarkerPos getMarkerPos “Marker_2“;
```

Position auf einen anderen Marker beziehen

```
“Marker_1“ setMarkerPos getPos Heli1;
```

Position auf ein anderes Objekt beziehen

Den Text eines Markers verändern:

```
“Marker_1“ setMarkerText “neuer Plan“;
```

unsichtbar / sichtbar machen eines Markers: (0 ist unsichtbar – 1 ist voller Kontrast – 0.5 ist transparent)

```
“Marker_1“ setMarkerAlpha 0;
```

einen Marker komplett löschen:

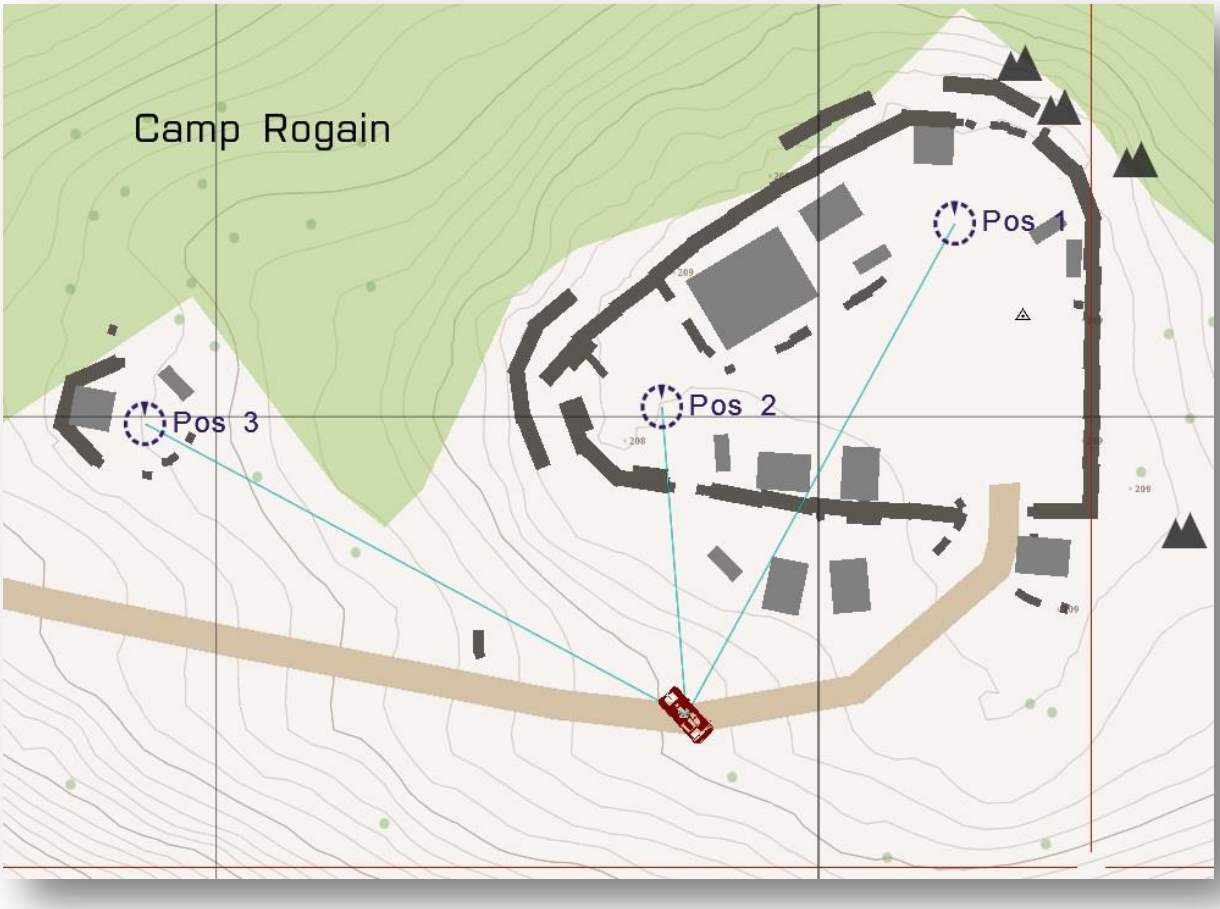
```
deleteMarker “Marker_1“;
```

Von letzterem würde ich jedoch abraten wenn es nicht unbedingt sein muss. Soll ein Marker verschwinden macht ihn lieber unsichtbar und vermeidet so undefinierte Variablen.

Weitere Anwendungsmöglichkeiten mit Markern

Lineal - Platziert ihr gerne Objekte besonders ordentlich? Ich schon, daher verwende ich z.B. einen rechteckigen Marker als Lineal wenn ich Soldaten in Reihe und Glied ausrichten möchte. Außerdem kann man damit super Abstände prüfen oder ausmessen da man ja die Größe eines Markers direkt bestimmen kann.

dynamische Objektplatzierung – platziert mehrere Marker auf der Karte vom Typ „Leer“ (engl. Empty). Diese Marker haben die Eigenschaft während des Spiels nicht sichtbar zu sein. Es würde natürlich auch mit jedem anderen Markertyp funktionieren.



Platziert nun ein beliebiges Objekt auf der Karte und verbindet das Objekt als Gruppe (F2) mit den drei Markern. Das ist zugegebener Maßen etwas umständlich weil die Marker nicht sichtbar sind solange ihr euch nicht im Marker-Menü befindet.

Wenn ihr die Mission jetzt in der Vorschau startet wird sich das Fahrzeug an einer der drei Markerpositionen befinden. Startet die Mission erneut und das Fahrzeug wird eventuell wieder auf einer anderen Position sein. Der Standort wird unter den drei Markern also zufällig ausgewählt. Eine extrem einfache Möglichkeit einen zufälligen Aufenthaltsort für ein Objekt zu generieren – und das ganz ohne komplizierte Skripte.

1.9 Module

Module verfolgen das Ziel immer wiederkehrende oder besonders komplexe Aktionen zu vereinfachen und mit wenigen Mausklicks durchführbar zu machen. Mit ArmA 3 sind es wesentlich mehr Module geworden. Wobei viele von den neuen Modulen auch ganz einfache Aufgaben übernehmen die ein erfahrener User mit Scriptkenntnissen auch mit einem Kommando erledigen kann.

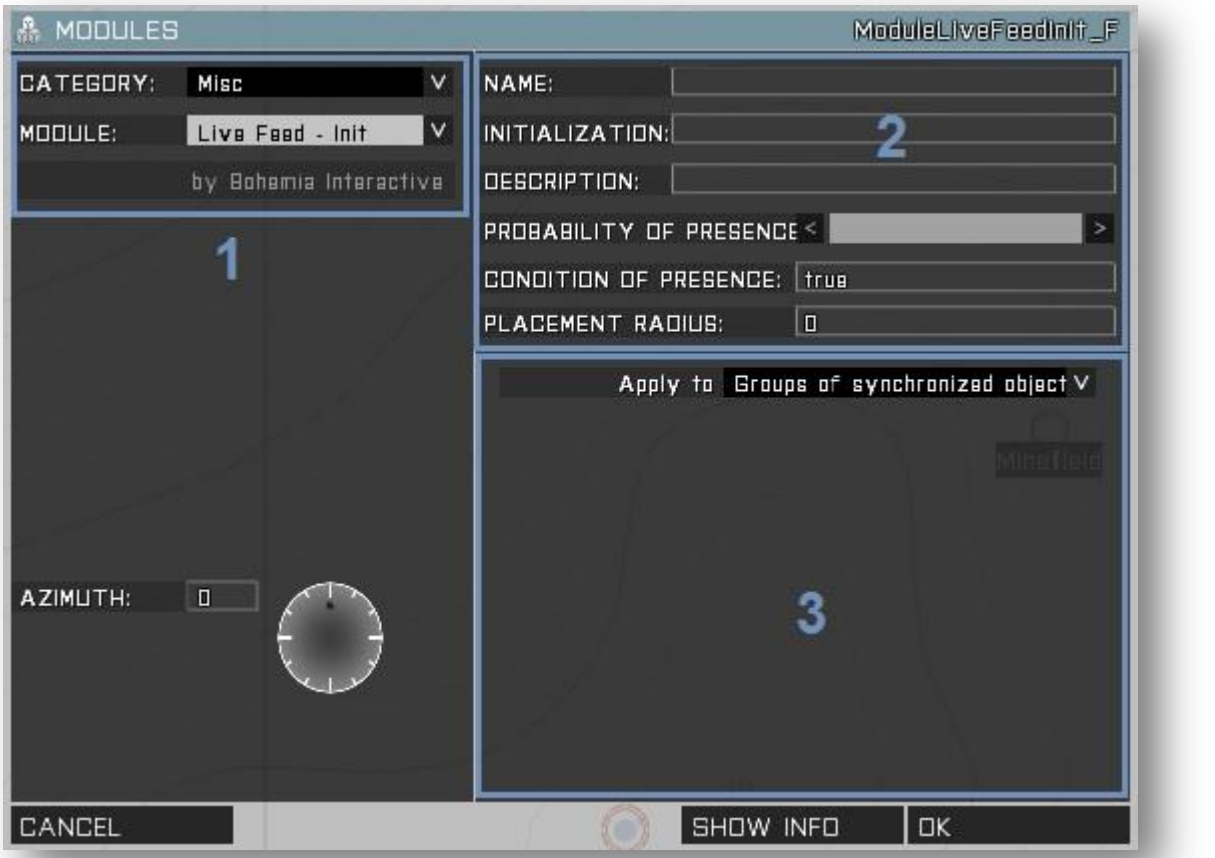
Beispiel: das „Set Callsign“ (Rufnamen vergeben) Modul soll euer Gruppe einen eigenen Rufnamen zuweisen. Dafür muß das Modul platziert und konfiguriert und mit einer Gruppe synchronisiert werden. Alternativ könnte man auch einfach folgende Kommandozeile in die Init des Gruppenführers schreiben:

```
this setGroupID [“Wolf“];
```

Statt z.B. „Alpha-1-1“ ist der Rufname euer Gruppe jetzt „Wolf“. Für den der den Befehl kennt ist die Variante mit dem Modul die eindeutig umständlichere Lösung.

Ein anderes Beispiel in dem sich ein Modul lohnt: Platziert das „Minefield“ Modul in einem beliebigen Bereich um diesen zu vermineren. Diverse Einstellungen innerhalb des Modules ermöglichen euch Größe des Feldes, Anzahl der Minen und vieles mehr zu bestimmen. So lassen sich schnell große Minenfelder anlegen für welche ihr sonst ewig im Editor hin und her klicken müßtet wenn ihr diese händisch platzieren wollt. Zumal die Verteilung der Minen dann statisch wäre und nicht zufällig so wie mit dem Modul.

Schauen wir uns wieder den Aufbau des Dialoges an. Ihr gelangt wie bei allen anderen Gruppen in den Dialog indem ihr „Module“ (F7) auswählt und einen Doppelklick auf die gewünschte Position der Karte durchführt.



Im oberen linken Bereich (1) wählt ihr das Modul aus welches ihr platzieren wollt. Zur besseren Übersicht wurden die Module in verschiedene Kategorien eingeteilt.

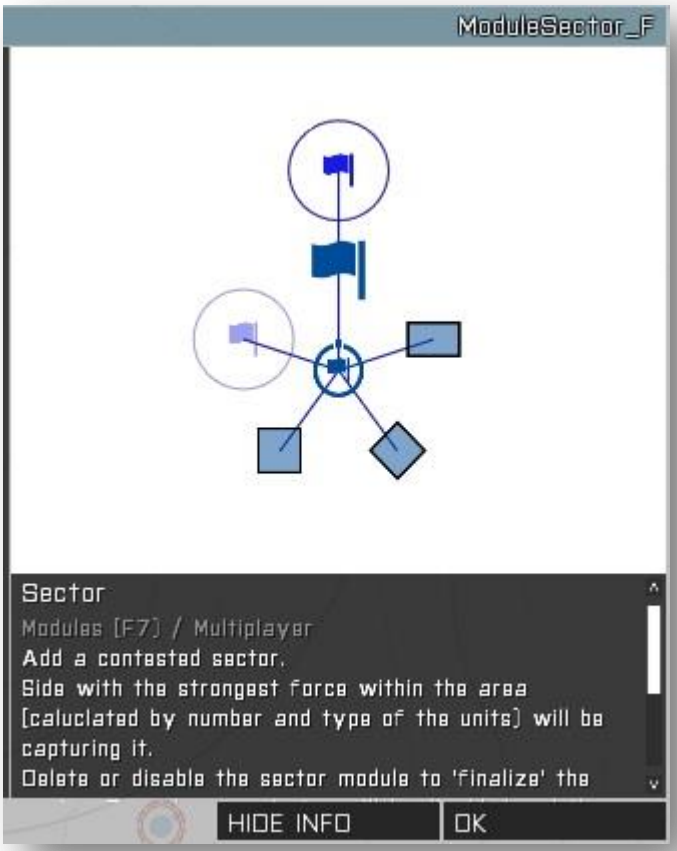
Oben rechts (Bereich 2) könnt ihr alle grundlegenden Einstellungen vornehmen. Diese sind bei jedem Modul gleich. Der Name stellt wieder die Variable des Moduls dar falls es später nochmal „angesprochen“ werden soll. Die Funktion von Platzierungsradius, Anwesenheitsbedingung und Anwesenheitswarscheinlichkeit kennen wir bereits von den Auslösern (engl. Trigger) und sind hier identisch. Der Sinn und die Funktion von „Initialization“ und „Description“ bleibt mir in diesem Fall leider verborgen.

Der 3. Bereich ist der interessanteste Bereich in diesem Dialog. Denn dieser bietet uns alle spezifisches Parametierungen für das jeweils ausgewählte Modul. Das heißt, je nachdem welches Modul ihr gerade bearbeitet, werden sich hier andere Setup-Optionen zeigen.

Ganz unscheinbar verbirgt sich am unteren Rand des Dialoges noch der Button „**Show Info**“. Klickt darauf und euch wird mit einer kurzen Beschreibung und einem Bild *versucht* die Funktion des Moduls zu beschreiben bzw. die nötigen Verknüpfungen erklärt und bildlich dargestellt.

Leider muß man sagen das es eben nur *versucht* wird da diese Beschreibungen oft sehr knapp gehalten, unvollständig oder sogar Fehler beinhalten. Das ist schade denn der Ansatz ist ein recht Guter.

Trotzdem empfehle ich den Blick auf die Info denn bei manchen Modulen, wie dem hier beispielhaft dargestellten Modul, kann man doch eine ganze Menge entnehmen. Mit einem Klick auf die verschiedenen Symbole auf dem Bild werden sogar noch detaillierte Informationen zu den einzelnen verknüpften Markern, Triggern und Objekten angezeigt.



Module im Multiplayer

Sämtliche Module funktionieren im Single Player wunderbar. Leider haben die Erfahrungen aus früheren Teilen der ArmA Serie gezeigt, dass viele Module im Multiplayer nicht mehr korrekt funktioniert haben oder sogar komplett ausgefallen sind.

Daher ist jedes Modul im Multiplayer mit Vorsicht zu genießen und dessen Funktion in der Multiplayerumgebung zu testen bevor es tatsächlich eingesetzt werden sollte.

Jedes Modul welches mit einem Spielerobjekt verbunden (synchronisiert) werden muß macht in der Regel spätestens dann Probleme wenn der Spieler gestorben ist, aber auch wieder respawnen soll. Denn nach dem Respawn ist die Synchronisation zwischen dem Modul und dem Spieler verloren gegangen. Die Verbindung wieder herzustellen, evtl. das Modul neu zu generieren und zu initialisieren ist ein komplizierter Prozess der gutes Wissen über Lokalitäten im Multiplayer und über Scripting bedarf.

„Feste“ Objekte die mit einem Modul agieren oder Module die komplett selbstständig arbeiten stellen hingegen kaum Probleme dar.

2. Dateien und Missionsordner

Alles was euch im 1. Grundlagenkapitel erklärt wurde drehte sich um den Editor. Damit waren bisher keine weiteren Dateien für eure Mission nötig. Doch damit ihr so richtig los legen könnt im Editing bedarf es meistens noch etwas mehr als nur der Arbeit im Editor. Dann werden weitere Dateien nötig die euren Missionsordner und damit die Mission mit Leben füllen. Außerdem sollte eure Mission nach der Vollendung in eine .pbo Datei gepackt werden.

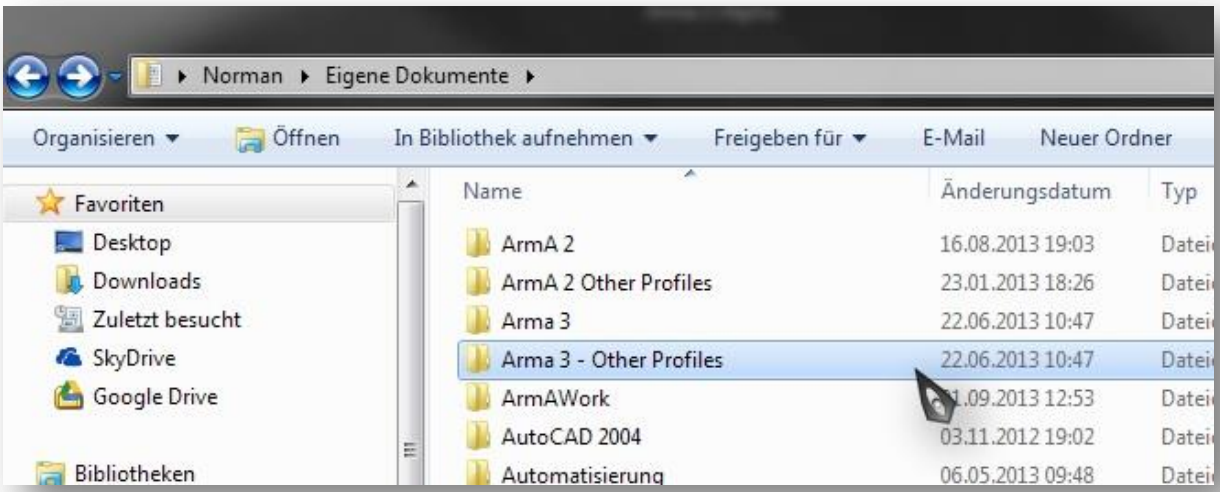
Dieses Kapitel wird euch alles rund um den Missionsordner und weitere Dateiformate in ArmA erklären. Außerdem zeige ich euch ein paar nützliche Tools die euch dabei helfen werden die Dateien zu erstellen und zu verwalten.

Hinweis: Alle hier angegebenen Pfade wurden unter Windows 7 aufgenommen. Die Pfade sollten jedoch unter Vista und Windows 8 äquivalent sein. Selbiges gilt für die Verwendung von dargestellter Software.

2.1 Speicherpfade

ArmA 3 legt nachdem ihr es das erste Mal gestartet habt eine ganze Menge an Dateien an die für euch relevant sein können. Spätestens wenn euch Fehler beim Scripten passieren und ihr bei der Fehlersuche nicht mehr weiter kommt, sollte man wissen wo man zu suchen hat. Denn ArmA hilft euch bei der Fehlersuche – sofern man weiß welche Datei man öffnen muss und den Inhalt auch deuten kann.

Das Spielerprofil

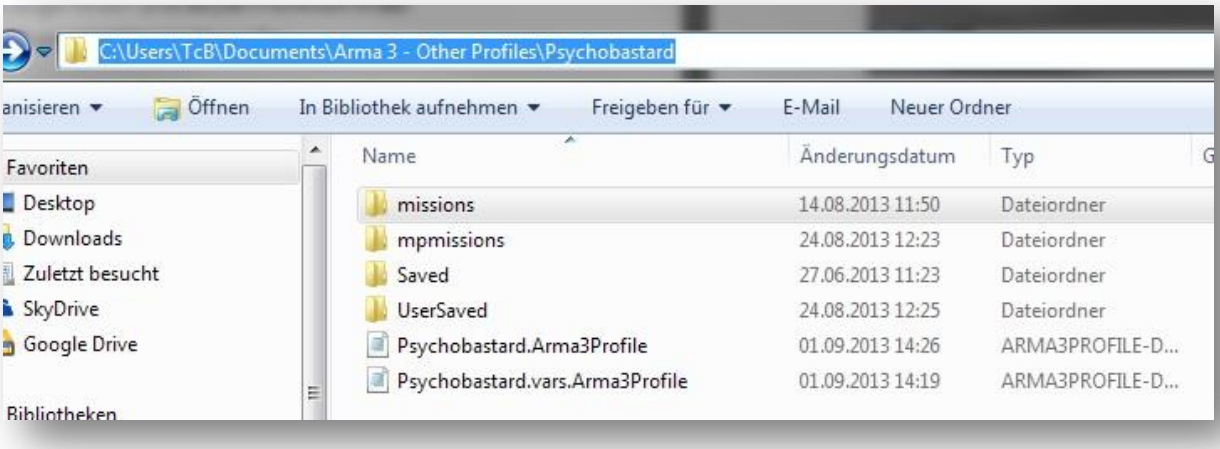


Der wohl wichtigste Bereich für euch befindet sich in den eigenen Dateien. Der Ordner „**Arma 3**“ wird bei der Installation automatisch angelegt. Nachdem ihr im Spiel ein eigenes Profil mit Spielernamen angelegt habt, wird auch der zweite Ordner, „**Arma 3 – Other Profiles**“ angelegt.

Der Pfad im Explorer lautet wiefolgt:

C:\UserseuerPC**\Documents\Arma 3 - Other Profiles**euerArmaName****

Wenn ihr diesen Ordner öffnet, findet ihr gleich eine ganze Reihe weiterer Ordner und auch noch zwei Dateien. Im folgenden werden die Inhalte erläutert.



missions – Hier befinden sich alle Missionen die ihr im Singleplayer Editor angelegt habt. Immer wenn ihr im Editor eine Mission zum ersten Mal speichert werdet ihr dazu aufgefordert der Mission, also der Datei, einen Namen zu geben. Den Namen den ihr hier vergebt wird auch der Name des Ordners sein. Hinweise zur Namensgebung findet ihr im Kapitel 3.1.

mpmissions – In diesem Ordner werden Missionsdateien abgelegt die ihr im Multiplayer Editor angelegt habt. Das wird eher selten der Fall sein. Jedoch bietet es sich an, dass wenn man eine Multiplayer Mission erstellt, man diese nachdem man die Arbeiten im Editor abgeschlossen hat, von dem **missions** Ordner in den **mpmissions** Ordner kopiert. Dieser Vorgang ermöglicht euch das laden der eben erstellten Mission im Multiplayer Bereich, in einer selbst gehosteten Umgebung ohne weiteren Aufwand.

Saved – Ihr habt während eine Mission läuft gespeichert? Dann wird die Missionsdatei genau hier abgelegt. Dabei unterteilt ArmA sauber in hauseigene und fremde Missionen. Achtung, dieser unscheinbare Ordner kann mit der Zeit mehrere hundert Megabyte beanspruchen. Wenn ihr den gesamten Ordner jedoch unkontrolliert löscht verschwinden natürlich auch alle eure Speicherstände. Das kann besonders bei Kampagnen sehr ärgerlich sein. Spielt in diesem Ordner also nur rum wenn ihr wißt was ihr tut.

UserSaved – fast genauso wie zuvor, aber nur manuell abgelegte Speicherslots.

*****.Arma3Profile** – In dieser Datei befinden sich alle Informationen über die Einstellungen die ihr im Spiel für das jeweilige Spielerprofil festgelegt habt. Wer mit dem Inhalt umgehen kann, kann hier genauso seine Settings beeinflussen wie über das Ingame Menü. Meistens findet man hier sogar noch weiterführende Möglichkeiten um die Grafik des Spieles zu beeinflussen die so nicht in den Videooptionen zu finden sind. Weiterführende Informationen und Diskussionen findet man meist zuhauf in den hiesigen Community-Foren. Auch wenn ihr Probleme mit den Grafikeinstellungen habt und Hilfe in einem etablierten Forum sucht, werdet ihr dort oft aufgefordert den Inhalt dieser Datei zu posten. Profis können euch so viel leichter und unkompliziert helfen.

(*deutsche Community Foren*: Arma2Base.de / hx3.de)

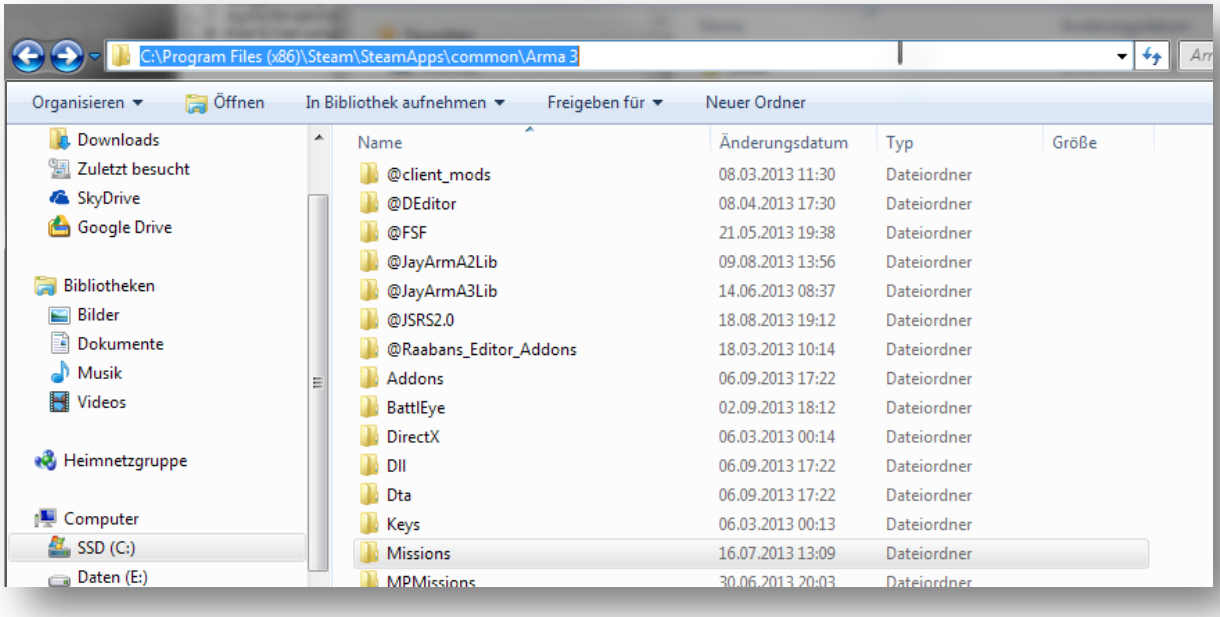
*****.vars.Arma3Profile** – Der Inhalt dieser Datei ist nicht lesbar und für den normalen Nutzer uninteressant

Das Installationsverzeichnis

Seit ArmA 3 besteht Steampflicht. Daher ist der Standardinstallationspfad in einen Unterordner von Steam gesetzt sofern ihr bei der Installation oder in den Steam Settings nichts anderes angegeben habt. In diesem Beispiel befindet sich Steam und somit auch ArmA 3 auf dem Laufwerk C.

ArmA 3 Installationspfad:

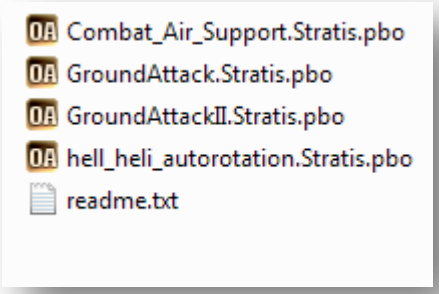
C:\Program Files (x86)\Steam\SteamApps\common\Arma 3



Hier werden Modifikationen und Addons abgelegt. Zu diesem Thema findet ihr mehr auf der Seite xx unter dem Kapitel 9.1. An dieser Stelle reicht erst einmal die Information, dass ihr auch hier wieder einen **Missions** und **MPMissions** Ordner findet.

Der Unterschied zu den Ordnern mit dem selben Namen im Spielerprofil ist, dass die hier abgelegten Missionen nicht im Editor Browser erscheinen wenn ihr dort eine Mission laden wollt. Hier abgelegte Missionen sollten im *.pbo Format sein und erscheinen dann im Ingame Spielmenü wenn dort eine Mission gestartet werden soll.

Schwer zu erklären, Bilder beschreiben es besser!



Das Bild rechts zeigt den Inhalt des ***Missions*** Ordner. Darin sind im Moment vier *.pbo's abgelegt. Jede Datei stellt eine Mission dar. Und diese werden dann wiederum im Ingame Menü unter **Scenarios** dargestellt wie im Bild unten zu sehen ist.

Reportdateien und Crash Reportes

Der dritte und letzte interessante Ort an dem euch ArmA 3 Dateien ab- und anlegt befindet sich unter:

C:\Users*euerPC*\AppData\Local\Arma 3

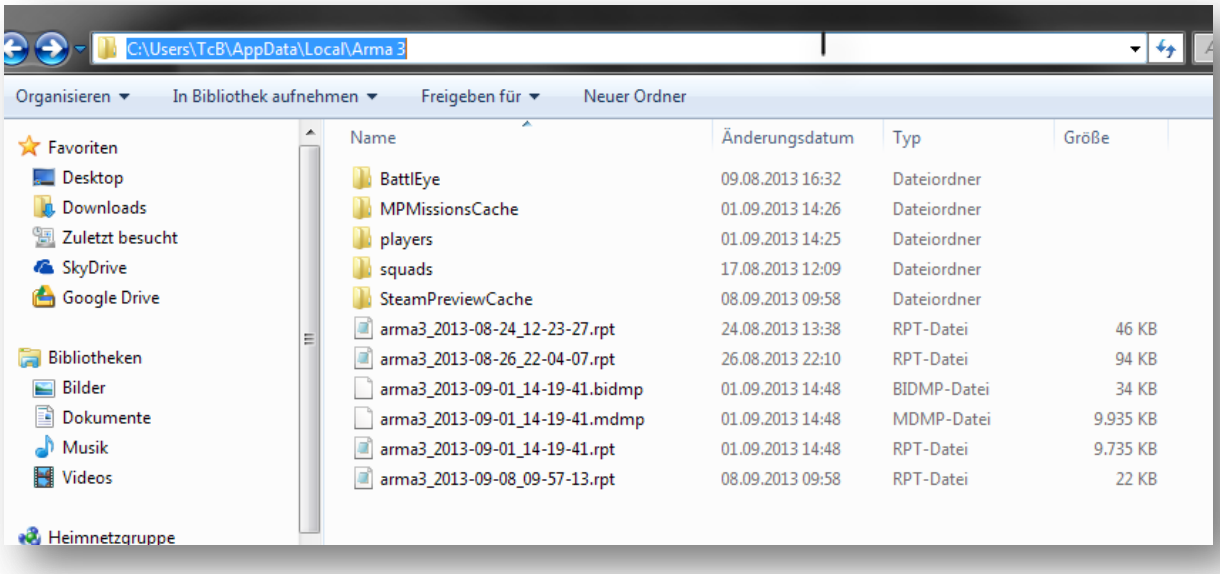
(Der Ordner **AppData** mit Inhalt ist aber nur sichtbar wenn im Datei Explorer " Versteckte Dateien" angezeigt werden.)

Hier „merkt" sich ArmA alles was so während des Spiels passiert, besonders im Multiplayer.Wenn ihr einen Server betretet um eine Mission zu spielen wird diese im Ordner ***MPMissionsCache*** abgelegt. Auf dem Server befinden sich bestimmt auch noch andere Spieler. Benutzen diese Spieler eine Squad-XML wird das darin enthaltene Clanlogo im Ordner ***squads*** abgespeichert. Benutzt der Spieler auch noch eine modifizierte Face Datei wird diese im Ordner ***players*** angelegt. Der ***SteamPreviewCache*** speichert z.B. Bilder von Missionsdateien die ihr euch aus dem Steam Workshop organisiert habt.

Warum speichert ArmA diese ganzen Daten? Ganz einfach – es soll das laden von Dateien beschleunigen. Denn jede Datei die hier abgelegt wurde muß beim nächsten Mal nicht mehr aus dem Netz geladen werden sondern wird direkt von euer Festplatte bereitgestellt.

Wenn ihr das erste Mal eine Mission im Multiplayer spielt wird diese von dem Server, der die Mission bereit stellt, an euch als Client (Spieler) übertragen. Ihr bekommt diesen Vorgang in Form einer Statusbar mit die euch den Fortschritt und die Größe der Datei anzeigen. Betretet ihr einen anderen Server auf dem die selbe Mission läuft ist dieser Ladevorgang nicht mehr nötig, da die Mission bereits auf euer Festplatte vorhanden ist. Der Beitritt zum Spiel erfolgt jetzt also wesentlich schneller.

Oft ließt man in Foren auch: *„Ich habe da eine Mission gespielt. Die war toll und ich will die auch haben. Wo bekomme ich die her???“.* Die Frage könnt ihr euch jetzt selbst beantworten. Denn die Mission befindet sich bereits auf euer Festplatte im Ordner ***MPMissionsCache***.



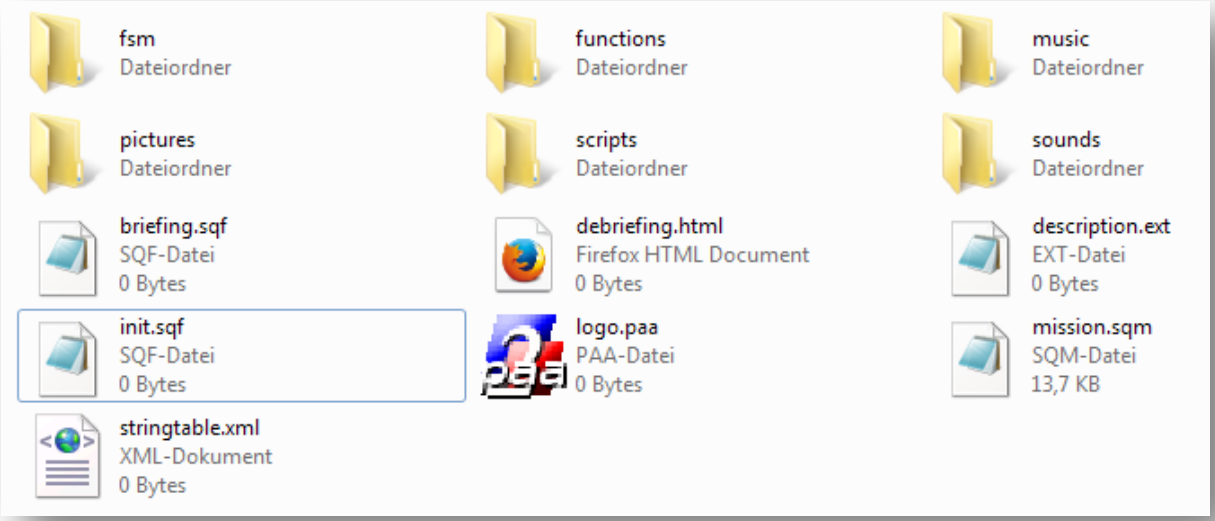
Viel wichtiger, für euch ist hier wohl aber die ***.rpt** Datei (Report-Datei).

Diese Datei wird einmal pro ArmA 3 Start angelegt. In dieser Datei werden dann ein paar Grundlegende Informationen zum Spielstart abgelegt. Darin enthalten z.B. der Spielpfad, geladene Dateien, Versionsnummer und vieles mehr. Außerdem werden hier Warnungen und Fehlermeldungen abgelegt die während des Spieles auftreten. Das ist für euch dann besonders wichtig wenn in euer Mission etwas nicht funktioniert und ihr nicht wisst wo der Fehler sein kann. Dann hilft in 90% der Fälle der Inhalt der ***.rpt** weiter. Man muß den Inhalt nur lesen und verstehen können. Und weil das so wichtig ist findet ihr sogar einen extra Abschnitt zu dieser Datei im Kapitel 9.4 auf der Seite xx in dem der Umgang mit dieser kostbaren Hilfe genauer erklärt wird.

2.2 Dateiformate unter ArmA

Im folgenden Abschnitt werden die am häufigsten verwendeten Dateiformate vorgestellt die für eine komplette Mission relevant sind. Das unten gezeigte Bild zeigt einen vollständigen Missionsordner der aufgeräumt und strukturiert ist. Die Benennung der Ordner ist dabei natürlich nur beispielhaft. Die Namensgebung der im Folgenden aufgelisteten Dateien hingegen nicht. Diese müssen exakt diesen Namen und diese Dateienden tragen. Außerdem müssen diese Dateien zwingend im Hauptverzeichnis des Missionsordners liegen.

- **Description.ext**
- **Init.sqf**
- **Mission.sqm**
- **Stringtable.xml**
- **Debriefing.html**



Die Ordner dienen der Übersicht und vereinfachen euch das Auffinden bestimmter Dateien in euer Mission um einiges, wenn ihr später etwas ändern wollt.

*.pbo (Profiler Binary Output)

Das wohl wichtigste Dateiformat in (fast) allen Produkten von Bohemia Interactive und somit auch von ArmA 3. Die .pbo's sind Dateicontainer, vergleichbar mit einer „**RAR**“ oder „**ZIP**“ Datei. In einer .pbo sind alle weiteren Dateien enthalten die für eine Mission benötigt werden. Auch die Spieldateien selber sind in .pbo's zusammen gefasst und gepackt. Details zum erstellen und öffnen von .pbo's findet ihr auf Seite 66 im Kapitel 2.7.

Die Stringtable

Die Stringtable ermöglicht es euch eine Mission mehrsprachig zu gestalten oder lange Textblöcke in kurze definierte Wörter zu stecken. Das ist nützlich um in Scripten nicht den Überblick zu verlieren oder wenn der selbe Text an verschiedenen Stellen wieder verwendet werden soll. Die Stringtable kann in zwei unterschiedlichen Formaten verwendet werden. Entweder als stringtable**.xml** oder als stringtable**.csv**.

Mehr Informationen sind auf Seite 68 im Kapitel 2.8 dazu enthalten.

Die init.sqf

Die init.sqf ist die Initialisierungsdatei für alle folgenden Scripte oder Funktionen die in der Mission aufgerufen werden sollen. Die Init wird automatisch bei jedem Missionsstart für jeden Spieler und den Server einmal ausgeführt. Daher werden hier auch alle zu verarbeitenden Variablen deklariert um die Ausgangswerte und bestimmte Paramater zu bestimmen.

Die mission.sqm

Diese Textdatei enthält alle relevanten Informationen zu Einheiten, Objekten, Triggern, Markern, Positionen usw. die durch speichern im Editor hier hinterlegt werden. Die Datei wird im folgenden Kapitel ausführlich beschrieben.

Die description.ext

Hier werden entscheidende Missionsparamter gestellt und definiert die jede Mission benötigt. Auch Ressorucen, gleich welcher Art, werden hier das erstemal definiert um diese später im Spielverlauf oder in Scripten ansprechen zu können. Syntaxfehler in dieser Datei führen zum sofortigen CTD (Crash to Desktop).

Die Datei wird ausführlich in den folgenden Kapiteln beschrieben.

2.3 Die Mission.sqm

Diese Datei beinhaltet alle wichtigen Informationen die in euer Mission enthalten sind und im Editor angelegt wurden. Wenn ihr zum ersten Mal eine neue Mission im Editor speichert wird diese Datei angelegt. Wenn ihr eine bereits vorhandene Mission editieren wollt und diese im Editor öffnet greift ArmA auf diese Datei zurück. Enthalten sind alle Informationen zu Wegpunkten, Objekten, Markern, Auslösern, Modulen und deren Parametern und Positionen. Theoretisch ist es möglich eine komplette Mission ohne den Editor zu erstellen wenn man diese Datei manuell erstellen und bearbeiten kann. Aber Vorsicht! Kleinste Syntax- und Formatfehler führen beim laden der Datei zum Absturz des Spieles.

Im folgenden wird kurz der Umgang mit der Datei erklärt. Denn in manchen Situationen ist es hilfreich zu wissen wie diese Datei aufgebaut ist um Änderungen vornehmen zu können.

Der Kopf – hier sind verwendete Addons (auch externe) eingetragen, der Missionsname, Wetter-, Zeitdaten und alle anderen Informationen die auch im Bereich ***Intel*** (siehe Kapitel 1.2) konfiguriert werden können.

Wenn man während dem editieren einer Mission mit Addons arbeitet, kann es vorkommen das sich diese Addons hier eintragen. Wenn diese aber nicht zwingend für den Missionsbetrieb benötigt werden ist der Eintrag unerwünscht und führt bei anderen die eure Mission laden nur zu Fehlermeldungen.

Profis kontrollieren daher vor dem veröffentlichen einer Mission diese Einträge und entfernen bei Bedarf unterwünschte Einträge.

```
addOns[]=
{
	"A3_Characters_F_BLUFOR",
	"a3_map_stratis",
	"A3_Air_F_Heli_Light_01",
	"a3_characters_f_beta",
	"A3_Air_F_Beta_Heli_Transport_01"
};
addOnsAuto[]=
{
	"A3_Characters_F_BLUFOR",
	"A3_Air_F_Beta_Heli_Transport_01",
	"a3_modules_f_multiplayer",
	"A3_Air_F_Heli_Light_01",
	"a3_map_stratis"
};
randomSeed=14579098;
class Intel
{
	briefingName="Psychos Editing Guide Example";
	overviewText="by Psychobastard";
	startWeather=0.29999998;
	startWind=0.099999994;
	startWaves=0.099999994;
	forecastWeather=0.29999998;
	forecastWind=0.099999994;
	forecastWaves=0.099999994;
	forecastLightnings=0.099999994;
	year=2035;
	month=7;
	day=6;
	hour=8;
	minute=0;
	startFogBase=0.001;
	forecastFogBase=0.001;
	startFogDecay=0.0049999999;
	forecastFogDecay=0.0049999999;
};
```

Gruppen – Jedes Objekt wird in einer Gruppe als Klasse dargestellt mit allem ihm zugehörigen Informationen, wie z.B. Wegpunkten oder Synchronisierungen.

In diesem Beispiel ist *class item0* eine Gruppe, bestehend aus einer Infanterieeinheit der Seite „*WEST*“. Außerdem sind dieser Einheit zwei Wegpunkte zugeordnet.

Der Einheit werden hier die Informationen über ihre Position auf der Karte, der Klassenname, die Fähigkeiten und die Definition als Gruppenführer zugeschrieben.

```
class Groups
{
	items=11;
	class Item0
	{
		side="WEST";
		class Vehicles
		{
			items=1;
			class Item0
			{
				position[]={1675.1456,5.5,5308.603};
				id=0;
				side="WEST";
				vehicle="B_soldier_exp_F";
				leader=1;
				skill=0.60000002;
			};
		};
		class Waypoints
		{
			items=2;
			class Item0
			{
				position[]={1661.1971,5.5,5325.5718};
				type="GETIN";
				synchronizations[]={11};
				class Effects
				{
				};
				showWP="NEVER";
				syncId=9;
			};
			class Item1
			{
				position[]={1849.3241,5.5099306,5417.2788};
				synchronizations[]={12};
				class Effects
				{
				};
				showWP="NEVER";
				syncId=10;
			};
		};
```

In der 3. Zeile steht *items=11;* - das sagt aus, dass sich in Summe 11 Gruppen auf der Karte befinden. Möchte man manuell in der mission.sqm eine Gruppe hinzufügen oder löschen muss auch dieser Wert angepasst werden. Sonst stürzt das Spiel ab. Das gleiche gilt für das entfernen oder hinzufügen von Einheiten innerhalb der Gruppe.

Weitere Parameter können sein:

Class Item0	Class Item0 ist Leader der Gruppe. Der dem Leader unterstellte Soldat ist Class Item1, der nächste Class Item2 usw.
Presence	Wahrscheinlichkeit der Anwesenheit
Position	Koordinaten des Spielers in der Anordnung X Z Y
Azimut	Blickrichtung der Einheit (Wert definierbar von 0 bis 359)
ID	ID der Einheit (fortlaufende Zahl)
Side	Spielerseite („WEST“,“ EAST“, …)
Vehicle	Art der Einheit (classname)
Player	Ist die Einheit ein Spieler bzw. Spielbar?
Leader	Gibt an, ob die Einheit ein Leader ist (Zahl 0 oder 1)
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 (Text Variable)
Init	Die Initzeile der Einheit (Angabe einer Syntax etc.)

Fahrzeuge – wie schon bei den Einheiten, jedoch alle leeren Fahrzeuge. Es werden also alle Objekte die keiner Gruppe angehörig sind zusammen gefasst. Wenn ein Fahrzeug mit Besatzung einer bestimmten Seite platziert wird, befindet sich dieses ebenfalls unter den Gruppen-Klassen.

```
class Vehicles
{
        items=5;
        class Item0
        {
                position[]={1767.0123,5.5,5580.8853};
                special="NONE";
                id=20;
                side="EMPTY";
                vehicle="B_Heli_Light_01_F";
                skill=0.60000002;
        };
        class Item1
        {
                position[]={1650.9839,5.5,5330.3569};
                azimut=58.0508;
                id=21;
                side="EMPTY";
                vehicle="Land_HelipadSquare_F";
                leader=1;
                skill=0.60000002;
        };
```

Marker – die nächste Klasse sind die Marker. Wieder werden alle Marker als *items* zusammen gefasst und der Reihe nach mit den diversen Informationen abgehandelt.

```
class Markers
{
        items=3;
        class Item0
        {

position[]={5022.7563,208.86714,5932.1328};
                name="marker";
                text="Pos 1";
                type="Empty";
                a=1.5;
                b=1.5;
                angle=-1.11217;
        };
```

Auslöser – werden in der Klasse *Sensors* sortiert.

```
class Sensors
{
        items=2;
        class Item0
        {

position[]={2623.3413,138.68729,1624.3838};
                activationBy="ANY";
                interruptable=1;
                age="UNKNOWN";
                class Effects
                {
                };
                synchronizations[]={2};
                syncId=13;
        };
```

Die gesammte mission.sqm macht auf den ersten Blick einen recht unübersichtlichen und komplizierten Eindruck. Lasst euch aber davon nicht sofort abschrecken und lasst das Funktionsprinzip und die Anordnung auf euch wirken. Es reicht vorerst aus das Grundprinzip zu verstehen und was man mit der mission.sqm machen kann und was diese beinhaltet.

Das bearbeiten der mission.sqm ist nur für fortgeschrittene Nutzer zu empfehlen. Es wird auch empfohlen sich vor dem bearbeiten dieser Datei eine Sicherungskopie anzulegen.

2.4 Die Description.ext

Diese Datei „beschreibt“, definiert also, die verschiedensten Dinge und Ressourcen die in der Mission eine Rolle spielen sollen. Das können Bilder, Musik, Sounds oder eigene Text- und Grafikressourcen sein.

Die wichtigsten Parameter und deren Funktion werden im folgenden vorgestellt. Außerdem wird beispielhaft die Definition von einem Bild und einer Sounddatei dargestellt. Detaillierte Informationen über die Definitionen in dieser Datei werden in den entsprechenden Abschnitten in diesem Guide erklärt.

Eine vollständige Auflistung der möglichen Parameter der description.ext findet ihr im BIS-Wiki: description.ext (https://community.bistudio.com/wiki/Description.ext)

onLoadName	Ist der angezeigte Name der Mission während des Ladebildschirm‘s Beispiel: *onloadName = “Attack Igor“;*
overviewText	Kann eine kurze Beschreibung der Mission sein oder sonstige Informationen liefern. Beispiel: *onloadMission = “Go and arrest Igor!“;*
author	Der Name des Authors. Wird der Parameter nicht definiert schreibt ArmA einfach „made by: Community Author“. Beispiel: *author = “Psychobastard“;*
overviewPicture	Der Pfad zu einem Bild welches während der Ladebildschirm dargestellt wird angezeigt werden soll. Beispiel: *loadScreen = “logo.paa“;*

Und damit ihr auch versteht von was ich hier eigentlich schreibe noch ein kleines Bild. Aufgenommen während des ladens einer aus der Community stammenden Mission.

Der Author dieser Mission hat genau diese Bausteine in der description.ext verwendet um das gezeigte Ergebnis zu erreichen.

Übrigens erhaltet ihr den selben Inhalt in abgewandelter Darstellung auch in der Scenarios-Übersicht.

Folgende Parameter sind nur für Multiplayer Missionen wichtig.

respawn	Wird als Zahl angegeben und regelt die Art des Respawns in einer Mission. Folgende Arten sind möglich:
	• 0 – kein Respawn
	• 1 – Respawn als Schmetterling oder Vogel (freie Kamera)
	• 2 – Instant Respawn (an Ort und Stelle nach dem Tod)
	• 3 – in der Basis (benötigt entsprechende Marker)
	  ○ "respawn_west"
	  ○ "respawn_east"
	  ○ "respawn_civilian"
	  ○ "respawn_ guerilla"
	• 4 – Respawn in ein KI Gruppenmitglied (wenn noch frei)
	• 5 – Respawn in ein KI Seitenmitglied (wenn noch frei)
	
	Für einen Fahrzeug Respawn wird ebenfalls ein Marker benötigt. Dieser nennt sich dann "respawn_vehicle_west" usw.
	
	Beispiel: *respawn = 3;*

respawnDelay	Gibt den Wert in Sekunden an bis ein Spieler wieder respawnt. Natürlich nur sofern ein Respawn verfügbar ist. Beispiel: *respawnDelay = 20;*
respawnVehicleDelay	Gibt den Wert in Sekunden an bis ein Fahrzeug wieder respawnt. Achtung! Dieser Parameter beeinflusst keine leeren Fahrzeuge. Beispiel: *respawnVehicleDelay = 20;*
diabledAI	Wenn dieser Parameter auf 1 steht, werden alle spielbaren KI Slots in der MP Lobby deaktiviert und damit auch die KI. Beispiel: *disabledAI = 1;*
header	Definiert den Spieltyp der Mission welcher auch im Serverbrowser angezeigt und filterbar ist. Der Parameter hilft anderen Spielern also eure Mission zu finden. Folgende Typen sind seit ArmA 3 möglich:

Class	Name	Shortcut
Unknown	Unknown	Unknown
Coop	Cooperative Mission	Co-op
DM	Deathmatch	DM
TDM	Team Deathmatch	TDM
CTF	Capture the Flag	CTF
SC	Sector Control	SC
CTI	Capture the Island	CTI
RPG	Role-Playing Game	RPG
Sandbox	Sandbox	Sandbox
Seize	Seize	Seize
Defend	Defend	Defend

Musik einbinden

Das folgende Beispiel definiert ein eigenes Musikstück welches in einer Mission abgespielt werden soll. Dafür ist es natürlich erforderlich erstmal ein Musikstück zu haben. Wie eine eigene Sound- oder Musikdatei auszusehen hat wird in Kapitel 2.6 beschrieben.

```
class CfgMusic
{
        tracks[]={};

        class intro_music
        {
                name = "AC/DC - Highway to Hell";
                sound[] = {"\music\intro.ogg", db+0, 1.0};
        };
        class outro_music
        {
                name = "Helden wie Wir";
                sound[] = {"\music\outro.ogg", db+10, 1.0};
        };
};
```

Die Klasse ***CfgMusic*** umschließt mit seinen geschweiften Klammern die anderen Klassen, wobei *CfgMusic* eine Hauseigene Config-Class von Bohemia ist. Diese kann also als gegeben angesehen werden. Die beiden anderen Klassen ***intro_class*** und ***outro_class*** sind die Klassen die wir in Form zweier Musikstücke definieren wollen.

Der ***name*** kann frei gewählt werden und stellt lediglich den angezeigten Namen des Titels dar welcher in einem Auslöser unter Effekte angezeigt werden würde. Denn eine richtig definierte Sound- oder Musikdatei wird dann auch dort dargestellt.

Unter ***sound*** wird dann der Pfad zu der Musikdatei definiert. Diese liegt in diesem Beispiel im Missionsordner, Unterordner ***music*** mit dem Namen ***intro*** und ***outro*** und dem „.ogg“ Format. Die anderen beiden Parameter, jeweils getrennt durch ein Komma, stellen die Lautstärke und den Pitch (Tonhöhe) dar.

Wie wird die Musik jetzt abgespielt? Wie schon erwähnt könnte man die Musik jetzt über einen Auslöser triggern da das Musikstück jetzt in den Effekten aufgelistet wird. Oder man benutzt folgenden Scriptbefehl an beliebiger Stelle:

```
playMusic “intro_music“;

playMusic [“intro_music“, 60]; → Musik hört nach 60 Sekunden auf
```

Noch ein letzter Hinweis zu allen Änderungen in der description.ext. Es handelt sich bei dieser Datei um eine Art Config-Datei für eure Mission. Daher werden alle Änderungen erst wirksam nachdem ihr die Mission einmal komplett neu geladen habt. Einfach auf „Vorschau“ klicken zeigt euch nicht die Änderungen.

Achtet außerdem peinlich genau auf Syntaxfehler. Der kleinste Fehler führt zu einem CTD (Crash to Desktop).

2.5 Scripte und deren Formate

Was ist überhaupt ein Script? Dieser Begriff wird in der Community fälschlicherweise für fast alles benutzt was in einem Missionsordner liegt. Grob umschrieben ist ein Script ein Ablauf von Kommandozeilen in einer bestimmten Reihenfolge.

Man kann im Moment wohl vier Formate aufzählen die in diese große Oberkategorie fallen:

- .sqf Funktionen, Scripte, ähnlich c++
- .sqs veraltetes Scriptformat
- .fsm Script in Blockdarstellung
- .hpp für Dialoge

Gleich vorab „.sqs“ wird hier nur der Vollständigkeit halber erwähnt. Dieses Format ist veraltet und sollte nicht mehr verwendet werden. Es hat gegenüber den anderen Formaten nur Nachteile. Wer sich mit .sqs auskennt der lernt auch schnell .sqf und .fsm zu nutzen. Alte .sqs Scripte können meistens schnell und einfach in das .sqf Format übersetzt werden. Hilfreich hierfür ist folgende Seite:

SQS to SQF Conversation (http://community.bistudio.com/wiki/SQS_to_SQF_conversion)

Betrachten wir kurz die drei übrig gebliebenen Formate. Eine detaillierte Ausführung und der Anwendung der Formate findet ihr im Kapitel 8 was sich ausschließlich mit dem Thema Scripten beschäftigt.

.sqf Format

Das wichtigste und am häufigsten verwendete Format in ArmA ist SQF. Die Sprache ist ähnlich wie c++ aufgebaut. Wer diese Sprache schon beherrscht wird es leicht haben mit .sqf umzugehen. SQF wird meistens als Script verwendet, kann und soll aber eigentlich als Funktion verwendet werden. Funktionen sind gegenüber Scripten bereits vorkompiliert und werden bei Bedarf blitzschnell ausgeführt um Berechnungen oder Anweisungen durchzuführen und ein Ergebnis zurück zu geben. Ein Script führt eine Folge von Befehlen aus, gibt aber in der Regel kein Ergebnis zurück.

Um SQF erstellen oder bearbeiten zu können benötigt man lediglich einen Texteditor. Wesentlich effektiver und komfortabler ist aber die Verwendung spezieller Bearbeitungsprogramme wie z.B. Notepad++. Dieses ist kostenlos und kann Scriptsprachen erkennen um diese dann besser lesbar mit Hilfe von Farben darzustellen. Außerdem erkennt es Befehle und kann Hilfestellung zu diesen anbieten oder sogar automatisch Befehle vervollständigen. Diese Eigenschaft wird Syntaxhighlight genannt. Mehr dazu findet ihr im Abschnitt 2.9. Dort wird erklärt wie ihr Syntaxhighlight für SQF in Notepad++ aktivieren könnt.

Beispiel von Syntaxhighlight in Notepad++ und einer SQF Datei:

```
13  _agony = false;
14
15  switch _bodypart do {
16      case "" : {
17          _damage = damage vehicle _unit + _return; //(_unit getVariable "bon_ais_overall") + _return;
18          _unit setVariable ["bon_ais_overall",_damage,false];
19          if(_damage >= 0.9) then {
20              _agony = true;
21              if((_damage >= _revive_factor) && tcb_ais_can_die) then {_unit setVariable ["bon_ais_unit_
22          } else {_unit setDamage _damage};
23      };
24      case "body" : {
25          _damage = (_unit getVariable "bon_ais_bodyhit") + _return;
26          _unit setVariable ["bon_ais_bodyhit",_damage,false];
27          if(_damage >= 0.9) then {
28              _agony = true;
29              if((_damage >= _revive_factor) && tcb_ais_can_die) then {_unit setVariable ["bon_ais_unit_
30          } else {_unit setHit ["body",_damage]};
31      };
32      case "head_hit" : {
33          _damage = (_unit getVariable "bon_ais_headhit") + _return;
34          _unit setVariable ["bon_ais_headhit",_damage,false];
35          if(_damage >= 0.9) then {
36              _agony = true;
```

Eine SQF Datei wird mit folgendem Befehl geöffnet:

```
[] execVM   "meinScript.sqf";
```

aus einem anderen Script

```
nul = [] execVM   "meinScript.sqf";
```

aus einem Trigger

```
myFunc = compile preprocessFileLineNumbers "meinScript.sqf";
```

Kompiliert als Funktion

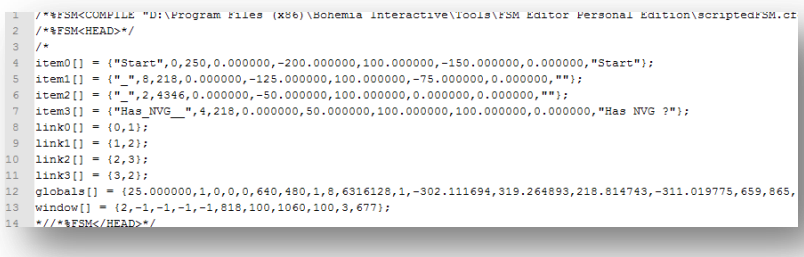
```
call myFunc;
```

die Funktion „rufen“

.fsm Format

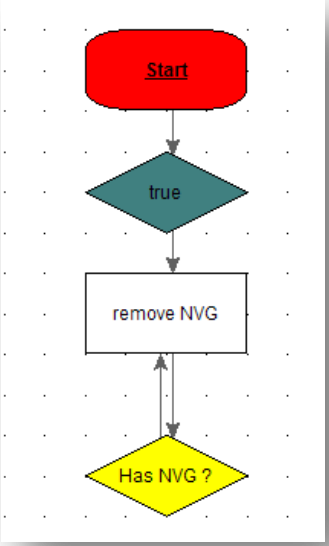
Eine FSM Datei eignet sich besonders gut um Schleifen (engl. Loops), also immer wieder kehrende Abläufe abzuhandeln. Die meisten Missionsbauer scheuen sich davor dieses Dateiformat zu verwenden. Ein Grund dafür könnte sein, dass ein richtiges Bearbeitungsprogramm für eine FSM benötigt wird. Ein solches Programm stellt den Ablauf sehr schön übersichtlich und grafisch dar. Erst beim Speichern wird dann eine Datei generiert die auch mit dem Texteditor geöffnet und bearbeitet werden kann. Diese ist dann aber extrem unübersichtlich.

Auch hier hilft euch ein Blick in den Abschnitt 2.9 weiter in dem das Bearbeitungsprogramm für eine FSM vorgestellt wird. Das folgende Bild zeigt eine FSM als Textdatei – sehr schlecht lesbar.



Diese FSM ist dann 70 Zeilen groß.

Und jetzt mal die gleiche Datei im speziellen Bearbeitungsprogramm für FSM's. ☺



Schön bunt und mit Pfeilen. Das Script erklärt sich jetzt fast von alleine. Und genauso einfach wie es jetzt aussieht ist auch die Bearbeitung und Erstellung dieser Dateien. Aber dieses Format hat auch seine ganz speziellen Eigenschaften und Besonderheiten auf die man achten muß. Details findet ihr im Kapitel 8.

Der Aufruf einer FSM ist fast identisch der einer SQF. Dieses Format kann aber nicht kompiliert werden wie das SQF Format, es sind also keine Funktionen möglich.

```
[] execFSM   "removeNVG.fsm";
```

.hpp Format

Die Sprache in diesem Format ist identisch mit dem der description.ext. Daher kann die description.ext auch den Inhalt dieser Dateien lesen und verarbeiten. Oft werden von Missionsbauern daher .hpp Dateien in der description.ext verlinkt. Und warum die .hpp – diese Dateien sind einfach viel besser lesbar.

Genutzt werden diese um Definitionen anzulegen und Dialoge zu erstellen. Dialoge sind z.B. Bild- oder spezielle Textdarstellungen in Missionen. Dialoge sind auch eigens angelegte Fenster mit Buttons wie z.B. der Artilleriedialog.

```
#define BON_BACKPACK_DIALOG 220000
#define BON_BACKPACK_PICTURE 220001
#define BON_BACKPACK_EQUIPBUTTON 220002
#define BON_BACKPACK_LIST 220003

                                //missionConfigFile >> "getBackpackDialog"
class getBackpackDialog {
    idd = BON_BACKPACK_DIALOG;
    movingEnable = true;
    enableSimulation = true;
    onLoad = "[] execVM 'dialogs\backpack_dialog\dialog_bp_update.sqf'";


    class controlsBackground {

        class Mainback : HW_RscPicture {
            idc = 0;
            x = 0.2; y = 0.25;
            w = 0.9; h = 0.6;
            text = "\ca\ui\data\ui_background_video_ca.paa";
        };

        class Title : RscTitle {
            idc = -1;
            moving = false;
            colorBackground[] = { 0, 0, 0, 0 };
            colorText[] = { 1, 1, 1, 1 };
```

Eine HPP Datei wird im Gegensatz zu den anderen Formaten nicht direkt geöffnet sondern nur „inkludiert“. Die Implementierung erfolgt stets in der description.ext und sonst nirgendwo.

```
#include "dialog/rscTitles.hpp";
```

Hier ein Beispielbild aus einer description.ext wie die Einbindung mehrer HPP’s aussieht:

```
#include "dialogs\rscTitles.hpp"
#include "common\spect\spectating.hpp"
#include "dialogs\backpack_dialog\getbackpack.hpp"
#include "dialogs\mission_settings\common.hpp"
#include "dialogs\mission_settings\mission_settings.hpp"
```

2.6 Sound- und Bilddateien

Musik, Sounds und Bilder lassen eine Mission leben und ansehnlich werden. Die Verwendung dieser optischen und akustischen Effekte ist daher im richtigen Moment und am richtigen Ort elementar für eine gute Mission.

Bilder verwenden

Zugegeben, es wird einem nicht gerade leicht gemacht ein Bild in einer Mission zu verwenden. Man muss bestimmte Formate verwenden und bestimmte Abmessungen einhalten. Das Format PAA wurde aus pragmatischen Gründen gewählt, denn die Dateiperformance ist hier sehr hoch. ArmA kann auch mit .jpg Dateien umgehen, jedoch hat dieses Format den entscheidenden Nachteil keine Alphakanäle darstellen zu können. PAA ist daher immer priorisiert zu behandeln.

Das Format lässt sich mit TexView2 öffnen und auch in andere Formate (.png, .pga) umwandeln um es mit gängigen Bildbearbeitungsprogrammen öffnen zu können. Das Erstellen einer PAA Datei ist auch in umgekehrter Reihenfolge möglich. (siehe Abschnitt 2.9 für mehr Details)

Abhängig von der Verwendung des Bildes in der Mission können die Abmaße des Bildes entscheidend für dessen Wiedergabe sein. Soll das Bild in einem Dialog oder in einem formatierten Text dargestellt werden kann das Seitenverhältnis in der Regel frei gewählt werden. Soll das Bild jedoch z.B. als Overview verwendet werden müssen bestimmte Seitenverhältnisse und Pixelbreiten eingehalten werden. Zulässig sind dann nur noch Seitenverhältnisse von 1:1 oder 2:1 unter der Potenz von 64 Pixeln.

Zulässige Formate sind dann z.B.: 128x128, 256x128, 1024x512, 2048x1024 oder auch 256x512.

Darstellen von Bildern

Die einfachste Form der Darstellung kennt ihr bereits aus dem Abschnitt 2.4. Dort wird bereits ein Bild als Loadscreen (Overview) geöffnet. Dabei wird immer der zu öffnende Pfad angegeben in dem sich das Bild befindet. In diesem Beispiel liegt das Bild ordnungsgemäß aufgeräumt im Missionsordner, Unterordner „pictures“.

```
loadScreen =  "pictures\myOverview.paa";
```

Das folgende Beispiel zeigt einen kleinen Dialog welcher ein Bild definiert was dann dem Spieler angezeigt werden soll nachdem die Mission begonnen hat. Dazu wird hier die Datei ***intro.hpp*** in die description.ext „inkludiert“.

Initialisierung in der description.ext:

```
#include   "dialog/intro.hpp";
```

Intro.hpp:

```
// ---------------------------------- Standard Definitions:

class RscStdText {
        type=0;
        idc=-1;
        style=2;
        colorBackground[]={0,0,0,0};
        colorText[]={1,1,1,1};
        font="TahomaB";
        size=1;
};

class RscPicture {
    access=0;
    type=0;
    idc=-1;
    style=48;
    colorBackground[]={0,0,0,0};
    colorText[]={1,1,1,1};
    font="TahomaB";
        sizeEx = 0.023;
        x = 0.0; y = 0.2;
        w = 0.2; h = 0.2;
    lineSpacing=0;
    text="";
};


//--------------------------------- define a Dialog:

class RscTitles {
        class intro_Title {
                idd=-1;
                movingEnable=0;
                duration=12;
                fadein=1;
                name="Mission_Title";
                controls[] = {Titelt1,Picture};
                class Titelt1 : RscStdText {
                        text = "Welcome Soldier!";
                        colorText[] = {0.8, 0.9, 0.4, 1};    //green
                        sizeEx = 0.07;
                        x = -0.1;
                        y = -0.6;
                        w = 1;
                        h = 1;
                };
                class Picture : RscPicture {
                        x =  0.05;
                        y = - 0.3;
                        w = 0.20; h = 0.2;
                        colorText[] = {1,1,1,0.7};
                        text="pictures\psy_logo.paa";
                        sizeEx = 256;
                };
        };
};
```

Aufruf des Dialoges „intro_Title“ nach 3 Sekunden in der init.sqf:

```
sleep   3;
titleRsc [“intro_Title“,“PLAIN“] ;
```

Ergebnis ist ein Bild und ein kurzer Willkommenstext:

Das nächste Beispiel zeigt die Darstellung eines Bildes als aufgesetzte Objekttextur auf ein bereits vorhandenes Objekt aus ArmA 3. Das funktioniert jedoch leider nicht mit allen Objekten. Folgender Code wurde in die Init eines Schildes geschrieben:

```
this setObjectTexture [0, "pictures\psy_logo.paa"];
```

… und das Ergebnis:

Das letzte Beispiel zeigt wie wir das Bild auf einer Flagge darstellen können. Dazu wurde ein Fahnenmast auf der Karte platziert mit folgendem Code in der Init:

```
this setFlagTexture "pictures\psy_logo.paa";
```

Und das Ergebnis ist unsere wehende Flagge im Wind –dass lässt jedes ArmA Herz höher schlagen!

Bei Objekten und Flaggen spielt das Seitenverhältnis übrigens keine Rolle. Ihr solltet trotzdem „angemessene“ Seitenverhältnisse wählen da das Bild sonst total verzerrt dargestellt wird.

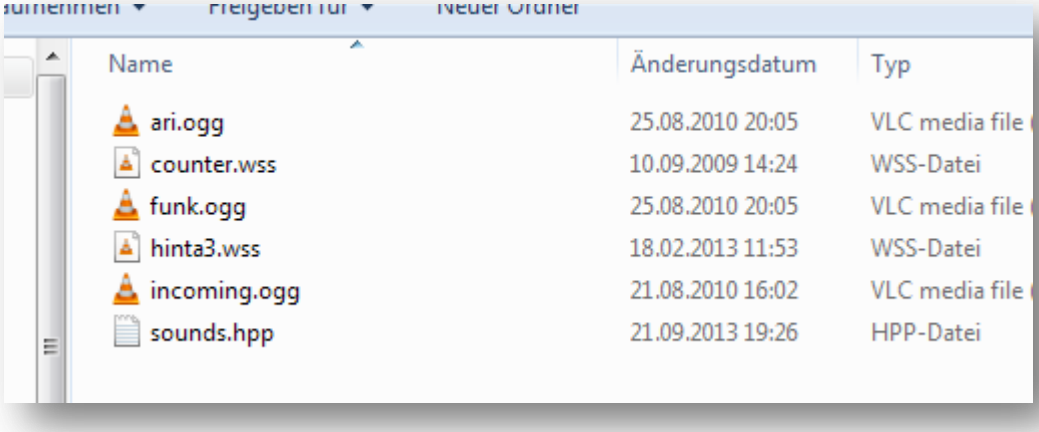
Sound- und Musikdateien

Bereits unter Kapitel 2.4 wurde darauf eingegangen wie Musik in einer Mission eingebunden wird.

Grundsätzlich wird zwischen Sounddateien und Musikdateien unterschieden. Der Unterschied macht sich am meisten bemerkbar anhand der spezifischen Soundeinstellungen der Spieler in ihrem Profil. Denn dort kann jeder Spieler die Musiklautstärke unabhängig von der Soundlautstärke einstellen. Und das sollte man als Missionsdesigner wirklich ernst nehmen und berücksichtigen.

Das Format von Sound- und Musikdateien muss entweder als **.wss, .wav** oder als **.ogg** ausgeführt sein. Das Format .WAV ist als Außenseiter zu betrachten da dieses Format wesentlich mehr Speicherplatz belegt als die anderen beiden.

Programme die euch eine MP3-Datei in eines der beiden Formate wandeln können findet man im Internet zuhauf. Dabei ist **OGG** das gängigere Format der beiden.



Die Definition von Sounddateien erfolgt wie auch schon bei der Musik. Unterschied ist hier statt **cfgMusic** findet die Definition in der Klasse **cfgSounds** statt.

Sounds.hpp:

```
class CfgSounds
{
        sounds[] = {};
        class funk
        {
        // how the sound is referred to in the editor (e.g. trigger effects)
                name = "Radio Request";
                // filename, volume, pitch
                sound[] = {"sounds\funk.ogg", 1, 1};
                // subtitle delay in seconds, subtitle text
                titles[] = {1, "*order Arty*"};
        };
};
```

Zum Abspielen des Sounds wird jetzt der eben definierte classname *funk* verwendet. Es gibt eine Vielzahl von Möglichkeiten einen Sound abzuspielen.

```
playSound "funk";
```
← einfach, ohne Effekte

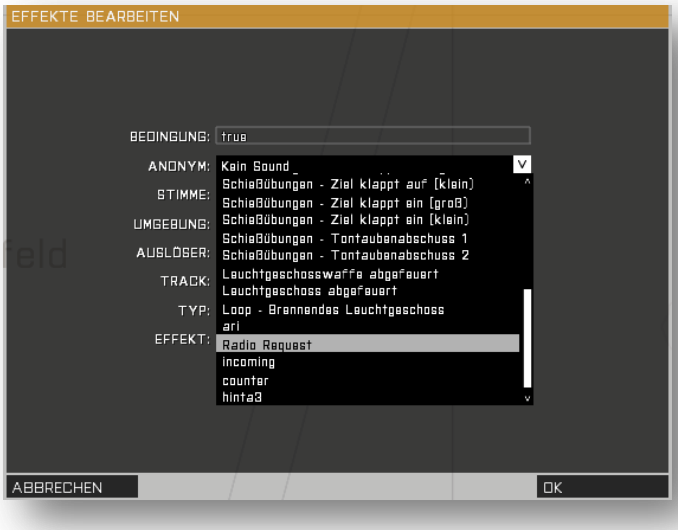
```
player say3D "funk";
```
← 3D-Sound von einer Quelle

```
playSound3D ["funk", player, false, getPos player, 1, 1, 0];
```

Syntax: playSound3D [filename, source, isInside, position, volume, frequency, distance];

Neuer Befehl seit Arma 3 – sehr nützlich da hier vorher viel gescriptet werden musste

Der Sound kann auch aus einem Trigger heraus abgerufen werden. Unter „Effekte“ tauchen die ebend efinierten Sounds wieder auf und können dort verwendet werden.



.cfgRadio

Eine besondere Form der Soundwiedergabe ist die Möglichkeit einen Sound als Funkspruch in Kombination mit einer Textausgabe zu verwenden. Dieser Sound stellt dann meistens eine Sprachaufzeichnung dar. Während die Stimme einen Funkspruch darstellt wird zeitgleich ein Text ausgegeben der in der Regel nochmal visualisiert was die Stimme gerade sagt. Dieser Text kann dann in der stringtable.csv Mehrsprachig definiert werden um so z.B. eine englische Stimme direkt ins Deutsche zu übersetzen. Um diese Art der Soundwiedergabe zu verwenden zieht man sich die Klasse **cfgRadio** heran.

z.B. in der description.ext:

```
class CfgRadio
{
        sounds[] = {};
        class RadioMsg1
        {
                name = "";
                sound[] = {"\sounds\filename1.ogg", db-100, 1.0};
                title = "I am ready for your orders.";
        };
        class RadioMsg2
        {
                name = "";
                sound[] = {"\sounds\filename2", db-100, 1.0};
                title = {$STR_RADIO_2};
        };
};
```

Verwendung in der Mission:

```
[west, ”BASE”] sideRadio "RadioMsg1";
```

Entsprechend das Ergebnis:

Passend zu dem Text wird die definierte Sounddatei, in diesem Beispiel *filename1.ogg* genannt abgespielt.

Pro-Tipp – Papa Bear:

Ihr seid alt? Ihr mögt keine Veränderungen denn alles ist gut so wie es war? Dann seid ihr wohl auch OFP-Fans und steht total auf den Rufnamen „*Papa Bear*“ statt diesem neumodischen, seit Jahren in der Verwendung befindlichen „*Crossroad*“. ☺

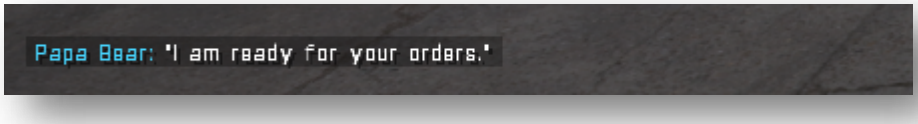
Folgender Code in der *init.sqf* generiert eine Logik die euch einen virtuellen Papa Bear generiert der euch künftig über Chat neue Befehle zukommen lassen wird:

```
if (isServer) then {
    _grp = createGroup west;
    hq_logic_papa = _grp createUnit ["Logic",[0,0,0],[],0,"NONE"];
    [hq_logic_papa] joinSilent _grp;
    hq_logic_papa enableSimulation false;
    hq_logic_papa setGroupId ["Papa Bear"];
};
```

Der Aufruf des Radio-Chats sieht dann wie folgt aus:

```
hq_logic_papa sideRadio "RadioMsg1";
```

…und natürlich noch das Resultat:



Und schon könnt ihr wieder in der Vergangenheit leben. ☺

2.7 PBO-Dateien erstellen und entpacken

Das PBO-Format ist ein Containerformat wie RAR oder 7z. Jede Mission sollte nach ihrer Fertigstellung als PBO gepackt werden bevor diese weiter verbreitet wird. Genauso wichtig ist es auch PBO's entpacken zu können.

Es gibt viele Möglichkeiten und Tools eine PBO zu packen/entpacken. Einige Leute erledigen das angeblich immer noch über DOS Kommandos. In diesem Guide zeige ich wie man mit dem Tool **cpbo** ein Kontextmenü nach Rechtsklick auf eine zu entpackende PBO oder einen zu packenden Ordner erstellt.

Aber vorher noch kurz eine kleine Liste der im Umlauf befindlichen PBO-Helfer:

PBOView – leichte Bedienung, gute Oberfläche aber veraltet (kann nicht alle PBO's entpacken)

Mikero's Tools – sehr gutes Tool, jedoch für Laien schwer zu bedienen da DOS-basiert

BinPBO – Hauseigenes Tool von BIS, kann PBO's nur packen nicht entpacken, wird benötigt um PBO_s zu signieren da es Keys erstellen kann (nötig bei Addons/Mods)

Eliteness – ein sehr altes Tool welches Mikero's DLL's zum Arbeiten benötigt, kann dafür noch weitaus mehr als nur Dateien packen/entpacken

Kegetys cpbo – einmal eingerichtet bietet dieses Tool eine sehr einfache und schnelle Bedienung

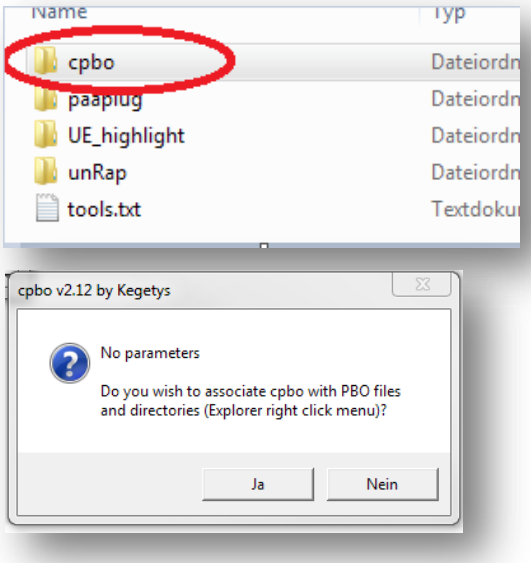
cpbo einrichten:

Download: http://www.kegetys.fi/dl.php/tools07072010.zip

Entpackt die Datei und öffnet den Ordner cpbo. Darin befindet sich die cpbo.exe.

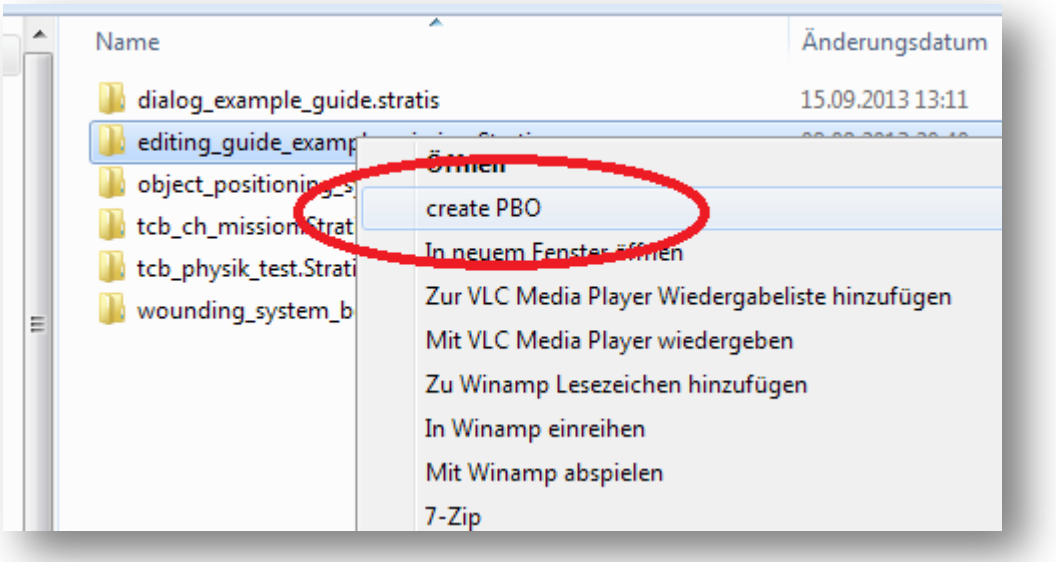
Führt diese .exe mit Doppelklick aus.

Ein DOS-Fenster und eine Warnmeldung öffnet sich. Bestätigt die Frage mit einem Klick auf „Ja".



PBO Packen

Sucht den Missionsordner im Dateibrowser und macht einen Rechtsklick auf den Ordner. Jetzt sollte sich ein neuer Shortcut in dem Dropdown-Menü von Windows eingetragen haben welcher euch die Möglichkeit gibt den Ordner zu einer PBO-Datei zu packen. Die PBO Datei wird im selben Überordner angelegt.



PBO entpacken

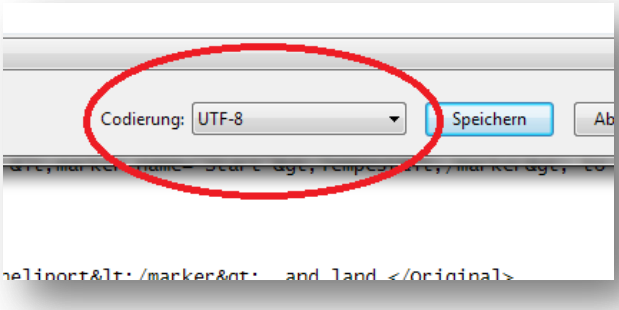
Auf dieselbe Art und Weise kann jetzt eine PBO entpackt werden. Dazu einfach per Rechtklick auf eine PBO-Datei im Kontextmenü „unpack PBO“ wählen und der PBO-Container wandelt sich wieder in einen Ordner.

2.8 Sonderformate

Die Stringtable

Die Stringtable ermöglicht es euch eine Mission mehrsprachig zu gestalten oder lange Textblöcke in kurze definierte Wörter zu stecken. Das ist nützlich um in Scripten nicht den Überblick zu verlieren oder wenn der selbe Text an verschiedenen Stellen wieder verwendet werden soll. Die Stringtable kann in zwei unterschiedlichen Formaten verwendet werden. Entweder als stringtable**.xml** oder als stringtable**.csv**.

Für welches Format ihr euch letztendlich entscheidet bleibt euch überlassen. Beide Formate haben ihre Vor- und Nachteile. Ich selbst bevorzuge das .csv-Format da es übersichtlicher ist und weniger Speicherplatz benötigt. Daher wird im folgenden auch nur das .csv Format in den Beispielen betrachtet.



Windows schlägt zur Bearbeitung dieses Formats Excel vor, bearbeitet die Datei jedoch mit einem Texteditor. Beim Speichern der Datei sollte man darauf achten, dass die Datei UTF-8 formatiert wird. Standardmäßig formatiert euch Windows die Datei nämlich im ANSI-Format. In diesem Format können Ingame aber kein ü,ö oder ä dargestellt werden. Um die Formatierung zu ändern wählt „Speichern unter“ und wählt im Fenster unten rechts die gewünschte Formatierung. (siehe Bild)

Der Aufbau einer .csv ist simpel. In der ersten Zeile werden die Sprachen der Reihenfolge nach definiert. Das bedeutet ein Spieler der die deutsche Version des Spiels hat wird automatisch den deutschen Text angezeigt bekommen und jemand mit der französischen Spielversion den französischen Text. Ihr kennt nicht alle Sprachen? Kein Problem – nicht definierte Sprachen weichen automatisch auf die erste definierte Sprache aus. Die erste Sprache sollte also immer Englisch sein!

LANGUAGE,English,German,Czech

Anschließend wird der Text sprachlich sortiert in der selben Reihenfolge niedergeschrieben. Vor dem jeweiligen Text wird jedoch zuerst immer die Adresse definiert. Dabei erhält die Adresse immer ein STR_ vor dem frei wählbaren Titel. In der Praxis sieht das wie folgt aus:

STR_Titel,“english example text“,“deutscher Beispieltext“

Aufrufen der Textadresse – es gibt drei verschiedene Methoden, wobei jeder Anwendungsbereich seine eigene Methode benötigt. Grundsätzlich lautet die Adresse für den Aufruf jetzt **STR_Titel.**

Aufruf aus dem Editor, z.B. einem Trigger	→	**@**STR_Titel
Aufruf aus einer Config	→	**$**STR_Titel
Aufruf aus einer .sqf	→	**localize** “STR_Titel“;

Der Aufruf der Textadresse muss natürlich im Kontext mit einem textfähigen „Baustein“ stehen. Soll z.B. ein Wegpunkt benannt werden muss @STR_Titel in das Description Textfeld geschrieben werden.

Wird der Aufruf mit Hilfe von ***localize*** in einem Script ausgeführt, sieht es in Kombination mit ***hint*** so aus:

```
hint localize “STR_Titel“;
```

Und noch ein Beispiel für eine Config bzw. in dem Fall in der description.ext um den Missionsnamen mehrsprachig darstellen zu können:

```
onLoadName $STR_Titel;
```

Will man innerhalb des Textes einen Zeilenumbruch vorgeben muss lediglich ein **n/** an die gewünschte Stelle geschrieben werden.

STR_Order,Deliver fast the packet to your boss**n/**Move out!

Auch das einbringen variabler Wörter, z.B. einem Namen ist möglich. Das **%1** ist hierbei der Platzhalter. Jeder weitere Platzhalter innerhalb des selben Strings würde fortlaufenden nummeriert werden. (%2, %3, …)

STR_Order_Personal,Hey %1, you have to deliver something.

Dann der entsprechende Aufruf aus einem Script dazu:

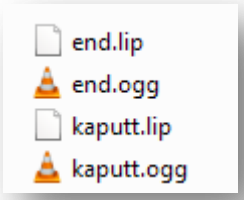
```
hint format [localize "STR_Order_Personal",name player];
```

Lip-Dateien

Die Lip-Datei sorgt dafür dass sich die Lippen eines Charakters anfangen zu bewegen. Jede Sounddatei die von einer Einheit mit Hilfe der Sprachausgabe (Befehl „say“) wieder gegeben wird kann mit einer solchen .lip-Datei hinterlegt werden um die Bewegung der Lippen herbei zu führen.

Dazu muß die .lip Datei lediglich denselben Dateinamen tragen wie die Sounddatei. Es sind keine weiteren befehle nötig um die Datei zustarten. Sounddatei und Lipdatei sollten sich im selben Ordner befinden.

Beispielhaft zeigt das Bild die Sound- und Lipdateien mit gleicher Namensgebung im Ordner der Sounddateien.



Der Inhalt einer Lipdatei ist im Textblock am rechten Seitenrand dargestellt und ist sehr einfach gehalten. Frame = 0.040 ist ein konstanter Wert der als gegeben betrachtet werden kann. Anschließend folgen viele Zeilen die jeweils zwei Zahlen getrennt durch ein Komma darstellen.

Die erste Zahl ist die Zeit in Sekunden entsprechend der Abspieldauer der Sounddatei beginnend bei 0. Die Lipdatei in diesem Beispiel ist also für eine Sounddatei mit einer Spieldauer von 2,52 Sekunden ausgelegt. Die Zahl hinter dem Komma gibt den Öffnungsgrad der Lippen an. **0** steht für einen geschlossenen Mund und **7** für weit geöffnet. Der Öffnungsgrad der Lippen kann also mit einem Wert zwischen 0 und 7 bestimmt werden.

Die letzte Zeile hat den Öffnungsgrad **-1** zugewiesen bekommen. Das bedeutet hier endet die Lipdatei.

➔ Syntax: **Zeit , Öffnungsgrad**

```
frame = 0.040
0.000, 1
0.040, 3
0.080, 5
0.120, 1
0.280, 2
0.320, 1
0.360, 3
0.400, 1
0.480, 2
0.520, 3
0.600, 7
0.640, 1
0.680, 2
0.840, 1
0.880, 3
0.920, 1
1.000, 2
1.080, 1
1.160, 0
1.400, 1
1.440, 2
1.560, 3
1.600, 4
1.640, 5
1.680, 3
1.720, 2
1.760, 5
1.800, 4
1.840, 1
1.920, 2
2.000, 1
2.080, 0
2.120, 1
2.280, 2
2.320, 1
2.440, 0
2.520, -1
```

So eine Lipdatei ist ein fantastisches Stilmittel, macht jedoch auch sehr viel Schreibarbeit wenn man solche Dateien manuell erstellen will. Daher hat BIS dem ambitionierten Missionsdesigner ein Tool beiseite gestellt welches einem diese Arbeit abnehmen soll. In den BIS-Tools (nähere Erläuterungen findet ihr im folgenden Kapitelabschnitt) existiert das Tool **Wave2Lip**.

Um eine Lipdatei zu erstellen muss dafür lediglich eine OGG-Sounddatei auf das Tool per Drag & Drop gezogen werden und im selben Ordner wird eine Lipdatei generiert. Mit dem Tool also ein Kinderspiel und ein absolut wichtiges Stilmittel wenn ihr Sprachdialoge in einer Mission verwenden wollt.

2.9 Nützliche Helfer

Wow, eine Menge Dateiformate und damit eine Menge Möglichkeiten eine Mission zu erstellen und zu bearbeiten. Damit kommt aber auch die Frage auf wie man diese ganzen Dateiformate händeln und verwalten soll. Prinzipiell ist es möglich jeder Dateiformat mit dem Texteditor der mit jeder Windows Version geliefert wird zu öffnen und zu bearbeiten. Aber wie sieht das aus wenn man eine SQF Datei mit mehreren hundert Zeilen im Texteditor öffnet? Richtig, einfach nur katastrophal unübersichtlich und strukturlos. Noch viel schlimmer sieht es aus wenn man sich den Spaß macht und eine FSM Datei im Texteditor betrachtet. Nahezu unmöglich solche Dateien händisch zu erstellen oder zu bearbeiten.

Daher existieren natürlich viele Programme und Tools die einem hier die Arbeit erleichtern und für eine einfache Bearbeitung sorgen. Die wichtigsten Tools, besonders für Addon Ersteller, liefert Bohemia Interactive selbst. Das sind die unter dem Oberbegriff zusammen gefassten **BI-Tools**.

Neben den offiziellen BI-Tools existieren natürlich auch andere Programme welche nicht speziell für Arma Entwickelt wurden sich aber hervorragend zum Editieren von Arma verwenden lassen. Dazu sei gesagt dass jedes hier erwähntes und gesammeltes Tool als sogenannte Light- oder Free- Version im World-Wide-Web zur Verfügung steht.

Zum vereinfachten Umgang und für eine bessere Übersicht werden im Rahmen dieses Editing Guides aber trotzdem alle Programme und Tools in einem Packet gebündelt und zur Verfügung gestellt. Daher noch ein wichtiger Hinweis: Bitte achte immer auf den Stand der letzten Aktualisierung des Packet um stets die richtige Programmversion zu verwenden. Hier steht mit vollem Bewusstsein die „richtige“ und nicht die „aktuellste“. Nicht immer ist die aktuellste Version auch die richtige Version!

Downloadpfad Arma 3 Editing Guide Tools:

Arma 3 Editing Guide Tools

(http://upload.g-g-c.de/tcb/files/Editing_Guide_Tools.7z)

Auf den folgenden Seiten wird kurz auf die Funktion der wichtigsten Tools eingegangen und deren Installation erläutert. Die meisten Tools haben einen eigenen Installer welcher die Anwendung der Tools einfach gestaltet.

Die BI-Tools

Die Installation aller BI-Tools wird über einen gemeinsamen Installer abgewickelt welcher im Umgang selbsterklärend ist.

Inhalt:

Oxygen 2 – Modelle editieren und Animationen erstellen (nur für Addon-Maker)

Visitor 3 – Terrains und eigene Karten erstellen

TexView 2 – Texturen und Bilder konvertieren und ansehen

BinPBO – das Haustool um PBO's zu erstellen und zu signieren (kann leider nichts entpacken)

Sound Tools – Wave2Lip und WavetoSSS (WSS) sind hier zu finden

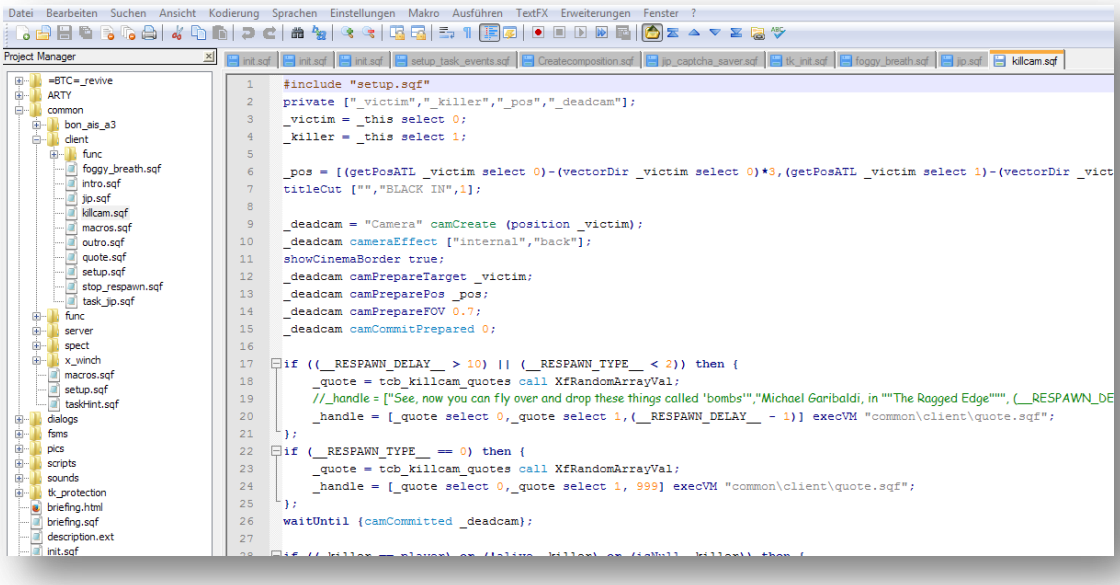
FSMEditor – zum Erstellen, editieren und kompilieren von FSM Dateien

Notepad++

Dieser sehr gute Texteditor ist das wohl wichtigste Werkzeug für jeden Missionsbauer der auch Scripte erstellen möchte. Notepad wird in den Guide Tools in zwei Versionen geliefert. Enthalten ist die aktuelle Version und die alte Version 5.7. Dazu gleich mehr.

Die Installation wird wieder einfach mit einem Installer erledigt. Die Bedienung von Notepad erscheint im ersten Moment als herausfordernd da einem hier viele Optionen und Möglichkeiten geboten werden. Nach ein paar Stunden Eingewöhnungszeit sollte es aber jedem recht einfach fallen mit dem Programm umzugehen. Hat man einmal die Vorteile dieses Editors erkannt will man ihn garantiert nicht mehr missen.

Was leistet Notepad? → Syntaxhighlight für fast alle Programmiersprachen, Projektverwaltung, Einschübe erstellen, Makros ausführen, Codierungen, Auto-Completition, Hilfe-Popups, Schlagwortsuche, …. Die Liste kann ewig weitergeführt werden.



Leider beinhaltet Notepad nicht von Haus aus Syntaxhighlights für SQF Scripts. Daher muss an dieser Stelle mit Plug-Ins nachgeholfen werden. Zwei verschiedene Plug-Ins sind in den Guide Tools enthalten. Und jetzt die Erklärung warum auch zwei unterschiedliche Versionen von Notepad selbst geliefert werden. Denn jedes der beiden Plug-Ins kann nur mit der entsprechend passenden Version arbeiten. Die Plug-Ins wurden in der Ordnerstruktur schon der jeweiligen Version zugeordnet.

Welche Version man benutzt hängt wohl von der persönlichen Vorliebe ab wie sich die Farbgestaltung der Highlights darstellt. Die aktuelle Notepad Version mit dem dazugehörigen Plug-In hat einen großen Vorteil gegenüber der alten Version 5.7 mit dem entsprechenden Plug-In:

Die neue Version ist sehr gut gepflegt und auf dem neuesten Stand in Bezug auf die Script-Kommandos unter Arma 3. Denn neue Kommandos erkennt Notepad nur wenn diese auch von jemand eingearbeitet werden. An dieser Stelle engagiert sich das Community Mitglied „*GossamerSolid*“ sehr stark und pflegt dieses Plug-In regelmäßig.

Das „alte“ Plug-In mit der alten Notepad Version ist im Ursprung von dem Community Mitglied „*Kegetys*“ erstellt worden und wurde das letzte mal im Jahr 2009 aktualisiert. Daher ist diese Version nicht perfekt gepflegt. Das Plug-In wurde daher von mir nach dem Erscheinen von Arma 3 selber aktualisiert. Jedoch ist es trotzdem nicht 100% vollständig. Dieses Plug-In pflege ich aus ganz pragmatischen Gründen: Ich verwende es selber! Und weil ich dies seit Jahren so handhabe möchte ich mich nicht auf eine andere Farbgebung einlassen. ☺

Nach objektiver Betrachtung sollte dem aktuellen Plug-In von „*GossamerSolid*“ an dieser Stelle aber der Vortritt gewährt werden. Daher wird hier kurz beschrieben wie dieses Plug-In zu installieren ist.

Installation:

1. Öffne Notepad++ 6.3 oder höher
2. Klick im Menü auf „Language“ → „Define your own Language“ → „Import“
3. Importiere die Datei „syntaxhighlighting/SQF.xml“ (Speicherort der Guide Tools)
4. Starte Notepad++ erneut
5. Öffne eine SQF Datei, diese ist jetzt farbig profiliert

Weiterführende Informationen sind in der Readme des Plug-Ins zu finden.

Total Commander

Dieses Programm ist nichts anderes als ein verbesserter Windows Explorer. Mit dem Tool lassen sich zwei Fenster parallel öffnen und die Dateien darin verwalten. Außerdem können Schlagwörter innerhalb ganzer Ordnerstrukturen gesucht werden. Der Suchbegriff wird dann aber nicht nur in den enthaltenen Dateinamen sondern auch in deren Inhalt gesucht. Eine unschlagbar hilfreiche Funktion zum Aufspüren von Variablen in komplexen Dateigebilden. Darüber hinaus kann dieses Tool auch als FTP-Client verwendet werden und bringt eigene ZIP-Utilities mit sich.

Die Installation lässt sich über den Installer einfach und schnell realisieren.

Soundtools

Die Soundtools sind eine kleine Sammlung von nützlichen Helfern welche Soundformate konvertieren können oder die Sounddatei selbst bearbeiten können.

Inhalt:

dbPowerAmp – konvertiert MP3 zu WAV und bietet diverse Kompressionsraten

Power Sound Editor – kann WAV Dateien bearbeiten (Fadeout, Fadein, div. Effekte, Schneiden)

OFP Sound Lab – kann WAV in OGG und/oder WSS wandeln und erstellt LIP-Dateien

Die Kombination dieser drei Helfer macht alles möglich was man benötigt um selber Sounds und Musik für eine Mission zusammenstellen zu können. Leider ist mir kein Programm bekannt welches alle Aufgaben gemeinsam bewältigen kann und als Freeware zu haben ist.

Weitere Programme

Hier sammeln sich weitere kleine „Nice to Have“ Tools. Keines davon ist wirklich entscheidend, wird dem ein oder anderem aber extrem weiter helfen.

Inhalt:

CompareIt – vergleich zwei Dateien miteinander und zeigt die unterschiede

SynchronizeIt – synchronisiert Dateien miteinander

PAA Plug-In – damit lassen sich Bilder im PAA Format auch in Photoshop öffnen

Kegetys Tools – ein Schatz für sich!

SQF Editor – Plug-Ins für Notepad zu umständlich? Dann friss das!

Farbtabelle – mischt jede Farbe und gibt den Farbcode aus (HTML, RGB, HEX, ARMA)

Der Nutzen und die Verwendung dieser Tools sollte sich jedem nach Bedarf selber erschließen. ☺

Aufruf: Du kennst einen weiteren Nützlichen Helfer der hier nicht aufgeführt wurde dir aber schon oft „das Leben“ gerettet hat? Feedback ist immer willkommen und die Liste der Helfer ist nie vollständig. Zögere also nicht und lasse andere an deinem Wissen teilhaben. Eine kurze Info an den Autor reicht und die Liste wird bei entsprechender Zweckmäßigkeit erweitert.

3. Eine eigene Mission erstellen

Vorwort und Denkanstoß

Gehörst du zu den aufmerksamen Lesern die bis an diese Stelle alles gelesen haben und nebenbei die Dinge die beschrieben wurden selber ausprobiert haben? Dann hast du bis hier hin schon einmal alles richtig gemacht.

Die ersten beiden Kapitel waren eher handwerklicher und technischer Natur. Nachdem die Werkzeuge verteilt wurden und die Baustelle definiert ist soll dieses Kapitel dem ambitionierten Missionsbauer dabei helfen eine Richtung zu finden. Außerdem soll es dazu anregen darüber nachzudenken was für eine gute Mission wichtig ist und was zu beachten ist.

Hier wird der Versuch gewagt anhand von schlechten Beispielen zu zeigen wie es nicht laufen sollte. Im Gegenzug werden Aspekte ausgeleuchtet über die man sich als Missionsdesigner Gedanken machen sollte. Täglich erscheinen neue Missionen aus der Community. Leider sind 95% davon schlecht organisiert, nicht durchführbar oder schlicht weg durchgehend fehlerhaft, was das Spielen teilweise völlig unmöglich macht.

Was macht eine gute Mission eigentlich aus? Diese Frage sollte man sich stellen bevor man mit der Arbeit beginnt. Eine gute Mission sollte in aller erster Linie originell sein! Die Einzigartigkeit ist es welche eine gute Mission ausmacht. Dieser Zustand wird nur durch gute Ideen hervor gebracht. Erst in zweiter Linie spielt die technische Umsetzung eine Rolle. Wenn man seine Gedanken mal quer durch diese besagten 95% ziehen lässt, kann man ein simples Muster erkennen. Eigentlich hat fast jede Mission andere Missionsziele und damit auch eigentlich andere Aufgaben.

Ich persönlich behaupte jetzt aber dass jede Mission dieselben zwei Missionsziele hat, die da lauten:

„Alle erschießen und alles in die Luft jagen!“

Das ist stark verallgemeinert, trifft aber wie ich finde genau den Punkt. Was soll damit ausgedrückt werden? Dass die meisten Missionen alles andere als originell sind, weil man immer und immer wieder dieselben Aufgaben und damit dieselben Handlungen durchführt. Gesellt sich jetzt noch eine schlechte technische Umsetzung dazu oder sonst ein negativer Faktor macht sich unter den Spielern schnell Frust breit. Und das war bestimmt nicht das Ziel des Missionsbauers gewesen als er seine Idee umsetzte.

Jetzt kommt vielleicht der Gedanke auf was man denn sonst in einer Militärsimulation machen soll. Natürlich werden auch weiterhin alle Missionen Gegner beinhalten auf die geschossen wird. Eventuell verteidigen diese auch eben eine Radarstation die gesprengt werden soll. Das „Wie“ ist aber der entscheidende Unterschied welche eine Mission originell oder langweilig werden lässt.

„Starte in Basis Z und gehe von Punkt A nach B um alles was sich dir in den Weg stellt platt zu machen, dafür bekommst du alle Waffen des Spiels mit unendlich Munition und Leben sowieso, zur Verfügung gestellt.“ Ist *wie* man es nicht machen sollte.

Und wie könnte das jetzt aussehen um denselben Auftrag interessant zu gestalten?

Zum Beispiel durch die Begrenzung der Leben des Spielers. Gibt es keinen Respawn überlegt sich der Spieler vorher zweimal was genau sein nächster Schritt sein wird statt blind über eine offene Fläche zu laufen. Begrenze Munition oder Waffenauswahl ist ebenso ein simples Stilmittel um den Spieler in bestimmte Situationen oder Handlungsmuster zu zwingen die nicht dem einfachsten und damit dem geradlinigsten Weg entsprechen.

Das Beispiel zeigte jetzt zwei ganz einfache Stilmittel auf die jeder umsetzen kann ohne viel über das Editing wissen zu müssen oder über einen reichhaltigen Erfahrungsschatz verfügen zu müssen. Und trotzdem würde die Mission völlig anders verlaufen und ein komplett anderes Spielgefühl hervor bringen. Und das Beispiel war ja nun wirklich alles andere als originell. Trotzdem werden schon alleine diese beiden Aspekte, ich nenne sie gerne Stilmittel, in kaum einer Mission angewendet.

„Weniger ist oft mehr."

Das sollte man sich besonders beim Erstellen von Mission oft zu Herzen nehmen. Es entsteht oft der Eindruck, dass ein großer Anteil der Missionsbauer in der Community die Auffassung oder das Denken vertritt, dass eine Mission umso besser ist umso mehr Fahrzeuge sich auf dem Stützpunkt befinden. Gespart wird in der Regel auch nie an Waffen, Ausrüstung und sonstigen Gegenständen. Nicht selten kommt es vor das da dann mehrere Tausend (!) Sandsäcke verbaut wurden. An diejenigen die so etwas bauen ein Appel: Denkt bitte daran das die Sandsäcke allein eine Mission nicht schöner machen sondern höchstens den Grafikspeicher der Spieler schön belasten.

Gut zu sehen ist die Bauwut und der Hang zum Überfluss auch immer an Missionen die in ihrem Originalzustand sehr gut gelungen sind, daher auch sehr beliebt in der Community, dann aber von anderen Spielern „*verbessert*" werden weil diese der Meinung sind das da noch 20 schwere Panzer und 10 Kampfhubschrauber in der Basis gefehlt haben. Auch an diese „Verschlimmbesserer" ein Appel: Der ursprüngliche Missionsdesigner hat eine gute Mission gebaut und sich das Design, worin enthalten auch die eigenen Kampfkraft der Truppe ist, etwas gedacht oder es sogar in vielen Stunden des Testens genau ausbalanciert. Also verändert doch bitte nicht das gesamte Missionslayout.

Dass hier nicht übertrieben wird kann man täglich im Serverbrowser sehen. Denn in der Regel ist keine „Insurgency" oder „Domination" zu sehen die nicht völlig verkrautet wurde. Daher soll die harsche Kritik an der herrschenden Kultur der Missionen nicht ungerechtfertigt erscheinen oder beleidigen. Die Kritik soll den angehenden Missionsbauern unter euch nur als Denkanstoß dienen und dazu animieren einen gewissen Weitblick zu bilden wenn es darum geht was eine Mission ausmacht.

Die folgenden Abschnitte in diesem Kapitel werden den dafür stecken und die Ecken markieren die so oft vernachlässigt werden.

3.1 Missionsname und Beschreibung

Blablabla…