

Achtung, der DAC hat einen großen Funktionsumfang, entsprechend umfangreich ist diese Dokumentation ausgefallen. Wenn Du Dir die 30-teilige Demonstration angeschaut hast, sollte es Dir leichter fallen, die verschiedenen Thematiken zu verstehen.

Möchtest Du „auf die Schnelle“ eine DAC-Zone erstellen, die auf Standard-Parameter zugreift, empfehle ich Dir dafür, die beiliegende Kurzanleitung zu lesen. Dort wird in wenigen Schritten erklärt, wie man eine Zone im DAC aufbaut.

Inhalt der Zip-Datei:

DAC Liesmich.pdf
DAC Kurzanleitung.pdf
DAC AI Behavior Plan.pdf (nur Infanterie und nur englisch)
DAC SectorFight Settings (Erläuterung der einzelnen Parameter)
DAC Demonstation (enthält die 30-teilige DAC Demonstration)
DAC_Source.pbo + DAC_Sound.pbo (die Standard DAC-Addons)
DAC Ordner (enthält alle Configs, für das Benutzen der „DAC_extern“ Logik)
DAC MP Missions (enthält 4 verschiedene MP-Missions-Beispiele)
DAC Script (Eine Skript-Version von DAC inkl. Beispielmission)

Themenübersicht:

02	Hinweis zu DAC + den DAC vorbereiten	39	DAC_Config_Units
04	Der DAC- Scriptaufruf für eine Zone	40	DAC_Config_Behaviour
07	Die Wahl der Seite, der Einheiten-, Verhaltens- und Camp-Konfiguration	43	DAC_Config_Waypoints
09	Mehrere DAC-Zonen erstellen (kopieren)	44	DAC_Config_Camp
10	DAC-Zonen verlinken	47	DAC_Config_Arti
11	Wegpunktzonen erstellen	49	DAC_Config_Events
12	Benutzerdefinierte Wegpunkte einbinden	51	DAC_Config_Marker
14	Logiken DAC-Zonen zuweisen	53	DAC_Config_Objects
15	DAC-Zonen aktivieren / deaktivieren	54	DAC_Config_Sound
16	Reduzierung von Infanterie-Gruppen	55	DAC_Config_Weapons
17	KI-Respawn aktivieren und konfigurieren	56	Danksagung
18	Camps mit Zonen verlinken	57	Anhang
19	Zonenwerte verändern (Größe, Position und Konfiguration)		
21	DAC-Zonen generieren		
22	DAC-Zonen löschen		
23	Die DAC-Arti		
24	Die DAC-Bodenunterstützung		
25	Editor-Gruppen einbinden		
26	DAC-Gruppen entlassen		
27	DAC-Objektgenerierung		
33	Die Konfigurations- Dateien		
34	DAC_Config_Creator		

Hinweis zu DAC

Diese Version von DAC V3.1 ist eine Beta-Version. Bitte beachte, dass durchaus Fehler auftreten können, oder das DAC unter bestimmten Umständen fehlerhaft reagiert.

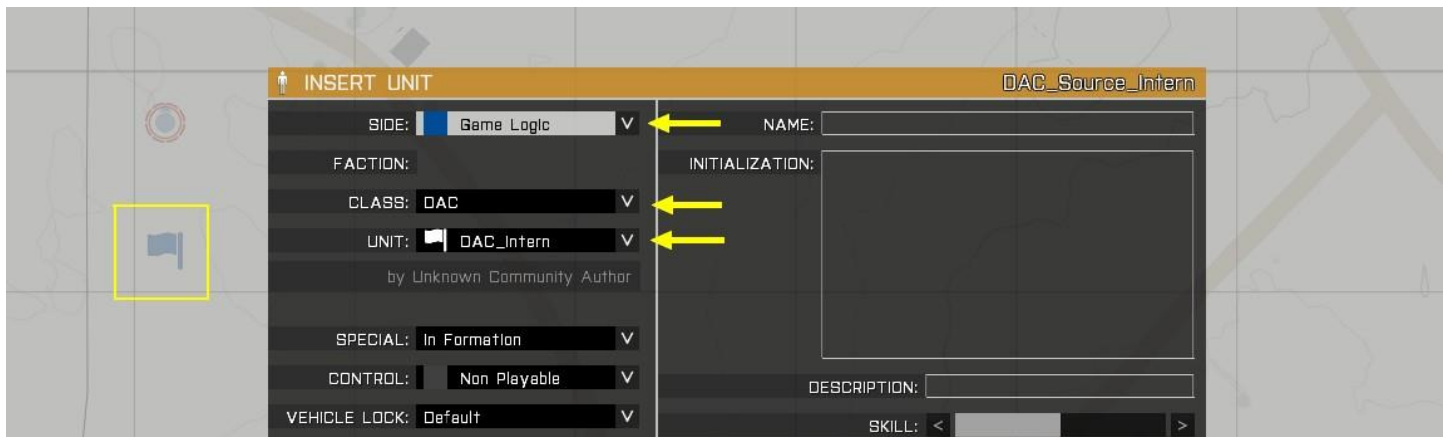
Es ist sehr wichtig, den DAC mit **gültigen** Daten zu füttern, die überwiegend in den **Konfigurations-Dateien** abgelegt sind. Kommt es zu Problemen, solltest Du immer erst versuchen, die Zonenkonstellation in Deiner Mission, mit Standard-Daten zu füttern (DAC_intern)

Es kann bei Problemen auch sehr hilfreich sein, wenn Du kritische Zonen für einen Test deaktivierst, in dem Du in die Bedingungszeile der Zone schreibst: **false** (Die Zone ist dann außer Betrieb)

Den DAC vorbereiten

Da der DAC in der Version 3.1 als Addon daher kommt, musst Du die DAC-PBO-Dateien, genau wie jedes Andere Addon auch, in Arma3 einbinden bzw. in einen eigenen Mod-Ordner installieren.
Im Notfall die PBO-Dateien einfach in einen vorhandenen Mod-Ordner kopieren.

Um den DAC zu starten bzw. zu initialisieren, musst Du einen Spieler mit dem Namen „S1“ im Editor platziert haben. Dann noch die DAC-Logik über F1 (Einheiten einfügen) im Editor platzieren:

Einheiten einfügen (F1) >>> Spiel-Logik >>> DAC >>> **DAC intern**

Die Logik **DAC_intern** benutzt, wie der Name schon sagt, interne Konfigurations-Dateien, in denen Standard-Verhalten und Standard-Einheiten hinterlegt sind. Für einen schnellen Test, oder auch einfach um sich mit dem DAC vertraut zu machen, ist diese Variante erste Wahl.

Möchtest Du dagegen die volle Kontrolle über alle Konfigurations-Dateien haben, die Dir ein hohes Maß an individueller Anpassung geben, musst Du die Logik **DAC_extern** benutzen. Dies Logik erfordert aber, dass Du den Ordner „**DAC**“ (im Zip-File enthalten) in Dein Missionsverzeichnis kopierst. DAC erwartet bei dieser Variante alle Konfigurations-Dateien in diesem DAC-Ordner. Welche Einstellungen Du in diesen Dateien vornehmen kannst, erfährst Du unter dem Thema [**Die DAC-Konfigurations-Dateien**]. Dort wirst Du auch erfahren, dass es noch eine dritte Variante gibt, mit der Du einzelne Konfigurations-Dateien auslagern kannst.

Ok, hast Du eine der DAC-Logiken platziert, wird DAC beim Missionsstart automatisch initialisiert. Was jetzt noch fehlt, ist eine DAC-Zone, also ein Bereich in dem Einheiten generiert werden sollen, denn das ist die Hauptaufgabe von DAC.

Ich kann Dir an dieser Stelle nur noch mal empfehlen, Dir **die 30-teilige Demo anzusehen**. Es wird für Dich dann wesentlich einfacher sein zu verstehen, was ich Dir hier erzähle :-)

Eine DAC-Zone wird über einen „einfachen“ Auslöser erstellt. Der Bereich des Auslösers ist gleichzeitig der Bereich, in dem die Einheiten generiert werden. Die Größe, die Form und die Drehung, kannst Du beliebig an Deine Bedürfnisse anpassen. Das Bild unten zeigt Dir eine typische DAC-Zone, so wie sie benötigt wird:

Die gelb markierten Optionen solltest Du genau wie angezeigt einstellen. Die grün markierten Bereiche sind variabel und von Dir anzupassen.

Name: der Name der DAC-Zone (beliebig)
Bedingung: diese muss bei allen DAC-Zonen gleich gesetzt sein (**time > 1** ist i.d.R. optimal)
Bei Akt.: der Skript-Aufruf, der den Auslöser zur DAC-Zone macht

Anmerkung:

Möchtest Du z.B. das die DAC-Zonen erst nach 10 Sekunden initialisiert werden, weil Du vielleicht vorher etwas anderes Skript-mäßig „erledigen“ musst, schreibst Du in alle DAC-Zonen: **time > 10**.

Möchtest Du zu Testzwecken eine bestimmte DAC-Zone mal deaktivieren, kannst Du in das Bedingungsfield folgendes eintragen: **false**.

Der Vorteil ist, dass Du eine bestimmte Zone dann nicht löschen musst, wenn sie mal nicht gebraucht wird, und Du kannst sie jederzeit wieder „scharf“ machen.

Achtung

Es ist zwingend erforderlich, dass alle Spieler-Einheiten einen Namen bekommen, damit DAC auf diese Einheiten richtig reagieren kann. Dafür vorgelegt sind die Namen **s1** bis **s10**.

Das bedeutet, im SP-Modus nennst Du Deine Spielfigur **s1**. Im MP-Modus musst Du die Spieler-Einheiten dann **s1**, **s2**, **s3**, **s4**, **s5** ... usw. benennen.

Selbstverständlich kannst Du die Spielernamen anpassen bzw. abändern. Dafür musst Du aber die Konfigurations-Datei „DAC_Config_Creator“ auslagern, in der die Spielernamen vordefiniert sind.

Wie das genau funktioniert, und welche Änderungen Du in dieser Datei sonst noch vornehmen kannst, kannst Du auch wieder unter dem Thema [**Die DAC-Konfigurations-Dateien**] nachlesen.

Der DAC- Scriptaufruf für eine Zone

Der DAC- Scriptaufruf enthält die gesamte Konfiguration einer Zone. Ohne diesen Eintrag im Aktivierungsfeld einer Zone, wird DAC nicht starten. Der Skriptaufruf sieht folgendermaßen aus:

```
["z1",[1,0,0],[ ],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

Wir befassen uns erstmal nur mit den 4 leeren Arrays, die ich farblich markiert habe. Dort werden die Mengen für Gruppen und Wegpunkte festgelegt, die in der Zone generiert werden sollen. Wie ich bereits in der Demonstration angemerkt habe, kann DAC bis zu 4 verschiedene Einheiten- Kategorien gleichzeitig je Zone generieren, wobei es insgesamt 5 verschiedene Kategorien gibt.

Jedes dieser Arrays steht für eine andere Einheiten- Kategorie:

Das 1. leere Array ist für die Generierung von **Infanterie** zuständig.

Das 2. leere Array ist für die Generierung von **Radfahrzeuge** zuständig.

Das 3. leere Array ist für die Generierung von **Kettenfahrzeuge** zuständig.

Das 4. leere Array ist für die Generierung von **Helis** oder **Spawncamps** zuständig.

Hier ein Beispiel für einen Skriptaufruf mit allen 4 Kategorien:

```
["z1",[1,0,0],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[1,1,1,1]] spawn DAC_Zone
```

8 Gruppen mit **Infanterie**

(Gruppengröße 2, 50 WP gesamt, 8 WP pro Gruppe)

2 Gruppen mit **Radfahrzeugen**

(Gruppengröße 2, 20 WP gesamt, 6 WP pro Gruppe)

5 Gruppen mit **Kettenfahrzeugen**

(Gruppengröße 1, 30 WP gesamt, 8 WP pro Gruppe)

2 Gruppen mit **Helis**

(Gruppengröße 2, 3 WP pro Gruppe)

Jetzt aber erstmal die Beschreibung zu den einzelnen Kategorien:

Infanterie:

1. **Die Anzahl der Gruppen die generiert werden soll**
2. **Die Gruppengröße** 4 Größenstufen sind standardmäßig verfügbar:
 $1 = 2 - 4$ Einheiten pro Gruppe
 $2 = 2 - 6$ Einheiten pro Gruppe
 $3 = 2 - 9$ Einheiten pro Gruppe
 $4 = 2 - 12$ Einheiten pro Gruppe
3. **Die Anzahl der Wegpunkte, die in dieser Zone generiert werden soll.**
Aus dieser Anzahl entsteht das WP-Pool, aus dem sich die Infanterie bedienen kann.
4. **Die Anzahl der Wegpunkte, die jede Gruppe aus dem WP-Pool zugewiesen bekommt.**
Diese Zahl entspricht nicht exakt der Anzahl der Wegpunkte, da hier +/-1 gerechnet wird.
Beispiel: Die Zahl 5 würde jeder Gruppe 4-6 Wegpunkte zuweisen.

Beispiel: ["z1",[1,0,0],[5,2,50,6],[],[],[],[0,0,0,0]] spawn DAC_Zone

In dieser Zone werden nach diesem Aufruf **5** Infanterie-Gruppen erstellt.
Die Gruppengröße beträgt **2 - 6** Einheiten.
Insgesamt wird ein Pool von **50** Wegpunkt in der Zone erzeugt (**nur für Infanterie**).
Jede erzeugte Gruppe bekommt **5 - 7** Wegpunkte aus diesem Pool zugewiesen.

Radfahrzeuge:

1. **Die Anzahl der Gruppen die generiert werden soll**
2. **Die Gruppengröße** 4 Größenstufen sind standardmäßig verfügbar:
 $1 = 2 - 4$ Einheiten pro Gruppe
 $2 = 2 - 6$ Einheiten pro Gruppe
 $3 = 2 - 9$ Einheiten pro Gruppe
 $4 = 2 - 12$ Einheiten pro Gruppe
3. **Die Anzahl der Wegpunkte, die in dieser Zone generiert werden soll.**
Aus dieser Anzahl entsteht das WP-Pool, aus dem sich die unbewaffneten Fahrzeuge bedienen können.
4. **Die Anzahl der Wegpunkte, die jede Gruppe aus dem WP-Pool zugewiesen bekommt.**
Diese Zahl entspricht nicht exakt der Anzahl der Wegpunkte, da hier +/-1 gerechnet wird.
Beispiel: Die Zahl 5 würde jeder Gruppe 4-6 Wegpunkte zuweisen.

Beispiel: ["z1",[1,0,0],[],[3,3,30,5],[],[],[0,0,0,0]] spawn DAC_Zone

In dieser Zone werden nach diesem Aufruf **3** Gruppen mit unbewaffneten Fahrzeugen erstellt.
Die Gruppengröße beträgt **2 - 9** Einheiten.
Insgesamt wird ein Pool von **30** Wegpunkt in der Zone erzeugt (**nur für Radfahrzeuge**).
Jede erzeugte Gruppe bekommt **4 - 6** Wegpunkte aus diesem Pool zugewiesen.

Unabhängig von der Gruppengröße, kann DAC maximal nur ein Fahrzeug komplett auffüllen. Wieviele Einheiten letztendlich generiert werden, ist also abhängig von der maximalen Crew-Grösse des Fahrzeugs.

Die Möglichkeit, mehrere Fahrzeuge pro Gruppe zu erzeugen habe ich deaktiviert, da die KI nicht in der Lage ist, mehrere Fahrzeuge vernünftig zu kontrollieren.

Kettenfahrzeuge:

1. Die Anzahl der Gruppen die generiert werden soll
2. Die Gruppengröße 4 Größenstufen sind standardmäßig verfügbar:

1 =	nur ein Fahrzeug pro Gruppe	\	
2 =	1 - 2 Fahrzeuge pro Gruppe	\	Hier entspricht die Zahl
3 =	1 - 3 Fahrzeuge pro Gruppe	/	der Anzahl der Fahrzeuge
4 =	1 - 4 Fahrzeuge pro Gruppe	/	
3. Die Anzahl der Wegpunkte, die in dieser Zone generiert werden soll.
Aus dieser Anzahl entsteht das WP-Pool, aus dem sich die bewaffneten Fahrzeuge bedienen können.
4. Die Anzahl der Wegpunkte, die jede Gruppe aus dem WP-Pool zugewiesen bekommt.
Diese Zahl entspricht nicht exakt der Anzahl der Wegpunkte, da hier +/-1 gerechnet wird.
Beispiel: Die Zahl 5 würde jeder Gruppe 4-6 Wegpunkte zuweisen.

Beispiel: ["z1",[1,0,0],[],[],[3,3,25,5],[],[0,0,0,0]] spawn DAC_Zone

In dieser Zone werden nach diesem Aufruf **3** Gruppen mit bewaffneten Fahrzeugen erstellt.
Die Gruppengröße beträgt **1 - 3** Fahrzeuge.
Insgesamt wird ein Pool von **30** Wegpunkt in der Zone erzeugt (**nur für Kettenfahrzeuge**).
Jede erzeugte Gruppe bekommt **4 - 6** Wegpunkte aus diesem Pool zugewiesen.

Helicopter:

1. Die Anzahl der Helis die generiert werden soll
2. Die Gruppengröße 4 Größenstufen sind standardmäßig verfügbar:

1 =	2 - 4 Einheiten pro Heli
2 =	2 - 6 Einheiten pro Heli
3 =	2 - 9 Einheiten pro Heli
4 =	2 - 12 Einheiten pro Heli
3. ~~Die Anzahl der Wegpunkte, die in dieser Zone generiert werden soll. (entfällt)~~
~~— Aus dieser Anzahl entsteht das WP-Pool, aus dem sich die Helis bedienen können. (entfällt)~~
3. Die Anzahl der Wegpunkte, die jeder Heli während des Fluges zugewiesen bekommt.
Diese Zahl entspricht nicht exakt der Anzahl der Wegpunkte, da hier +/-1 gerechnet wird.
Beispiel: Die Zahl 5 würde jedem Heli 4-6 Wegpunkte zuweisen, bis er wieder landet.

Beispiel: ["z1",[1,0,0],[],[],[],[2,2,4],[0,0,0,0]] spawn DAC_Zone

In dieser Zone werden nach diesem Aufruf **2** Helis erstellt.
Die Gruppengröße beträgt **2 - 6** Einheiten (nur bei unten genannten Bedingungen).
Jeder erzeugte Heli bekommt **3 - 5** Wegpunkte während des Fluges zugewiesen.
Nachdem alle Wegpunkte abgearbeitet wurden, fliegt der Heli zurück zu seiner Landeposition.

Die Angabe der Gruppengröße wird nur berücksichtigt, wenn der Heli Ladekapazität besitzt.
Ein AH64 z.B. wird keine zusätzlichen Einheiten aufnehmen können. Generiert wird dann nur die Besatzung.

Wenn Du diese 4 Arrays im Griff hast, ist das wirklich einfach damit umzugehen. Bis auf das Array mit den Helis, sind die Parameter ja eigentlich identisch. Möchtest Du eine bestimmte Einheiten-Kategorie nicht erstellen, musst Du nur das entsprechende Array leer lassen.

Die Wahl der Seite, der Einheiten-, Verhaltens- und Camp-Konfiguration

Diese Merkmale sind hier zusammengefasst, da sie auch im Scriptaufruf gemeinsam in einem Array stehen. Im Scriptaufruf befindet sich dieses Array am Ende (siehe **blaue** Markierung).

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```

Parameter 1 = Die Wahl der Seite für die Einheiten in dieser Zone
Parameter 2 = Die Wahl der Einheiten-Konfiguration in dieser Zone
Parameter 3 = Die Wahl der Verhaltens-Konfiguration in dieser Zone
Parameter 4 = Die Wahl der Camp-Konfiguration in dieser Zone
Parameter 5 = Die Wahl der Wegpunkt-Konfiguration in dieser Zone (optional)

Die Wahl der Seite

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```

Für die Wahl der Seite gibt es nur 4 Einstellmöglichkeiten (es gibt ja auch nur 4 verschiedene Seiten).

0 = Osten | **1** = Westen | **2** = Widerstand | **3** = Civil

Im Beispiel oben werden also Einheiten auf Seite **Ost** erstellt, da der Parameter mit **0** angegeben wurde.

Die Wahl der Einheiten-Konfiguration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```

Dieser Parameter bestimmt, welche Einheiten-Typen in der Zone generiert werden sollen. Wenn Du die Logik **DAC_intern** benutzt, stehen Dir folgende Einheiten-Konfigurationen zu Verfügung (Standard-Einheiten):

0	=	East (CSAT)		2	=	Independent (AAF)
1	=	West (NATO)		3	=	Civilian (Civilians)

Achtung,

für jede Einheiten-Konfiguration muß immer die richtige bzw. passende Seite eingestellt sein !!!

Mit anderen Worten: **Bitte keine West-Einheiten auf Seite Ost generieren usw.**

Solche Kombinationen sind zwar möglich, doch kommt es dann zu unkontrollierten KI-Aktionen.

Die Einheiten-Konfigurationen sind in der Datei **DAC_Config_Units** definiert.

Grundsätzlich gilt, dass beliebig viele Konfigurationen in dieser Datei angelegt werden können und das im Prinzip für jede DAC-Zone eine andere Konfiguration geladen werden kann.

Wie genau das funktioniert, erfährst Du unter dem Thema [**Die DAC-Konfigurations-Dateien**]

Die Wahl der Verhaltens-Konfiguration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

Dieser Parameter bestimmt, welche Verhaltens-Konfiguration für die Zone geladen wird.

Die Verhaltens-Konfigurationen sind in der Datei **DAC_Config_Behaviour** definiert.

Welche Parameter in einer solchen Verhaltens-Konfiguration definiert werden können, kannst Du auch wieder unter dem Thema [**Die DAC-Konfigurations-Dateien**] erfahren.

Benutzt Du die Logik **DAC_intern**, stehen Dir folgende Verhaltens-Konfigurationen zu Verfügung:

0	=	Grundverhalten für Seite Ost
1	=	Grundverhalten für Seite West

2	=	Grundverhalten für Seite Widerstand
3	=	Grundverhalten für Seite Zivil

Die Verhaltens-Konfiguration kann auch wieder um beliebig viele Einträge ergänzt werden, und für jede Zone wiederum, kann dann eine beliebige Konfiguration geladen werden.

Die Wahl der Camp-Konfiguration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

Der letzte Parameter ist zuständig für die Camp-Konfiguration. Insofern ist er nur relevant, wenn in der Zone mindestens ein Camp generiert wird, ansonsten bleibt dieser Parameter unberücksichtigt.

Analog zu der Verhaltens-Konfiguration, wird auch hier wieder eine Nummer eingetragen, die aus der Datei **DAC_Config_Camps** die entsprechende Konfiguration lädt. In der Camp-Konfiguration ist der komplette Aufbau eines Camps definiert.

Die Wahl der Wegpunkt-Konfiguration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

Dieser Parameter ist **optional** und muss nicht zwingend angegeben werden. Er legt fest, welche Wegpunkt-Konfiguration für die Zone benutzt wird. Wird dieser Parameter nicht gesetzt, wird Konfiguration **0** benutzt.

Der Parameter „NoReturn“

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

Dieser Parameter ist auch **optional** und bewirkt, dass die Gruppen ihre Masterzone verlassen und dorthin nie wieder zurückkehren.

Dieser Parameter erfordert aber mindestens eine Wegpunktzone, die mit der Masterzone verlinkt sein muss, damit die Einheiten einen Grund haben, ihre Zone zu verlassen.

Wenn Du hier einen Wert **> 0** angibst (max. 100), dann entspricht dieser Wert einer Wahrscheinlichkeit, mit der die Gruppen doch Wegpunkte in der Masterzone haben können.

Wenn Du diesen Parameter bei einer Helizone anwendest, dann bewirkt er, dass die Helis nur zum Landen in ihre Masterzone zurückkehren.

Mehrere DAC-Zonen erstellen (kopieren)

Hast Du erstmal eine Zone erstellt, z.B. eine Zone mit dem Namen **z1**, dann kannst Du ganz einfach weitere Zonen erstellen, in dem Du die Zone **z1** kopierst und an anderer Stelle wieder einfügst (Mauszeiger genau über Zone platzieren > Strg+C drücken > Mauszeiger auf neue Position bringen > Strg-V drücken)

Angenommen Du kopierst die Zone **z1**, dann hat die Kopie automatisch den Namen **z1_1**, dafür sorgt Arma.

Wichtig dabei ist, dass der neue Zonenname im Scriptaufruf eingetragen wird

Tipp: Verwende kurzen Namen, damit im Eingabefeld nicht zu viel vom Scriptaufruf verschwindet.

Die kopierte Zone kannst Du dann noch in der Größe ändern und an eine Position Deiner Wahl platzieren. Bei Bedarf noch die Parameter für die 4 verschiedenen Einheiten-Kategorien ändern, und schon ist die neue Zone bereit zum Einsatz.

So kannst Du innerhalb von wenigen Minuten mehrere Zonen aufbauen und konfigurieren und musst dabei nicht mal den Editor verlassen ☺

Mal angenommen, Du hast eine Zone 2 Mal kopiert und hast die Namen der Zonen und den Scriptaufruf schon angepasst. Die Scriptaufrufe aus den Zonen könnten dann so aussehen:

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



3 Zonen, die sich nur im Namen unterscheiden (müssen)

Das sind 3 Zonen (**z1**, **z2**, **z3**), in denen jeweils 3 Infanterie-Gruppen erstellt werden.

Möchtest Du, dass alle Zonen lokal arbeiten, also das die Einheiten sich nur in Ihren Zonen bewegen, musst Du noch einen Parameter verändern. Dieser Parameter ist sozusagen die **ID-Nr.** der Zone. Gib' jeder Zone eine andere Nummer, also so...

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[2,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[3,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



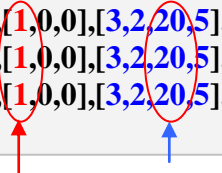
Das ist die **ID-Nr.** der Zonen

Es ist völlig egal, welche Zahl Du dort einträgst. Wichtig ist nur, dass jede Zone, die lokal arbeiten soll, eine eigene Nummer besitzt, die in keiner anderen Zone vorkommt.

DAC-Zonen verlinken

Zonen verlinken bedeutet nichts anderes als, die Wegpunkte mehrerer Zonen zusammenzulegen. Das bedeutet wiederum, dass die Einheiten auch Zugriff auf die Wegpunkte der verlinkten Zonen haben. Für das Verlinken wird auch wieder die **ID-Nr.** benutzt. Hier noch mal das Beispiel von ganz oben:

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



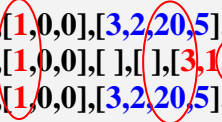
Die 3 Zonen sind über die **ID-Nr.** miteinander verlinkt. Gleiche **ID-Nr.** = gemeinsame Wegpunkte.

Das heißt, jede der 9 **Infanterie**-Gruppen hat Zugriff auf insgesamt **60** Wegpunkte. Welche Wegpunkte sich jede Gruppe letztendlich davon nimmt, wird per Zufall entschieden.

Durch diesen Zustand entsteht zwischen den Zonen ein reger Einheitenaustausch ;-)
In der **DAC-Demonstration** habe ich Dir bereits verlinkte Zonen vorgestellt.

Hier noch ein paar andere Konstellationen bzgl. verlinkten Zonen:

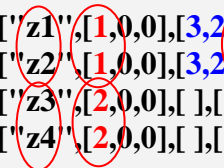
```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



Dieses Beispiel zeigt auch 3 verlinkte Zonen, allerdings werden in der Zone **z2** **Kettenfahrzeuge** erstellt. Obwohl diese Zone mit den beiden **Infanterie**-Zonen verlinkt ist, werden die **Kettenfahrzeuge** ihre Zone nicht verlassen, da ihnen in den Zonen **z1** + **z2** keine Wegpunkte zu Verfügung stehen. Im Prinzip hätte man diese Zone nicht mit den Zonen **z1** + **z2** verlinken brauchen.

Noch ein Beispiel:

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,15,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[2,0,0],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone  
["z4",[2,0,0],[ ],[3,1,35,8],[ ],[0,0,0,0]] spawn DAC_Zone
```



Auf der Map sind 4 Zonen mit den Namen **z1** bis **z4** platziert. Oben siehst Du die Scriptaufrufe dazu. Die Zonen **z1** + **z2** sind über die ID **1** verlinkt und haben insgesamt **35** Wegpunkte zu Verfügung. Die Zonen **z3** + **z4** sind über die ID **2** verlinkt und haben insgesamt **60** Wegpunkte zu Verfügung.

Wegpunktzonen erstellen

Die Wegpunktzonen haben den Zweck, den Einheiten aus der Master-Zone weitere Wegpunkte zu Verfügung zu stellen. In diesen Zonen werden keine Einheiten erzeugt.

Ein Beispiel findest Du auch wieder in der **DAC-Demonstration**.

Durch das Erstellen von Wegpunkt-Zonen erhöht sich die Dynamik noch mal deutlich. Hier ein Beispiel:

```
["z1",[1,0,0],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
["z2",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
["z3",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
["z4",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```

Das Beispiel zeigt wieder 4 Zonen **z1** bis **z4**. Die Zonen sind über die ID **1** miteinander verlinkt.

Die Zone **z1** ist die Master-Zone, da hier die 4 Parameter für die Einheiten eingetragen sind:

15 Infanterie-Gruppen, Gruppengröße **2** (2-6 Einheiten pro Gruppe).

In dieser Master-Zone werden **10** Wegpunkte generiert und jede Gruppe erhält **15** Wegpunkte für ihre Routen.

Die Wegpunktzonen **z2** – **z3** erkennst Du an dem einzelnen Eintrag in dem entsprechendem Array.

Dieser Eintrag gibt die Anzahl der Wegpunkte an, die in dieser Zone generiert werden sollen.

In unserem Beispiel sind das 3 Zonen mit je **20** Wegpunkten. Dazu kommt die Masterzone mit **10** Wegpunkten. Das bedeutet, dass die Gruppen aus der Zone Zugriff auf insgesamt **70** Wegpunkte hat.

Also noch mal zum Verständnis:

Ein Array mit 4 Parametern (oder 3 Parameter bei Helis) erzeugt Wegpunkte und Einheiten und ist sozusagen die Masterzone oder Ursprungszone der Einheiten. Ein Array mit nur einem Parameter erzeugt nur Wegpunkte für die entsprechende Kategorie.

Hier ein Beispiel, wie man 2 unterschiedliche Masterzonen auch kombinieren kann:

```
["z1",[1,0,0],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
["z2",[1,0,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone
```

Das hier ist noch nicht kombiniert. Die Zonen sind zwar über die ID **1** verlinkt, aber die Einheiten haben keine Möglichkeit in die andere Zone zu kommen, da es keine Wegpunkte der jeweiligen Kategorie in der anderen Zone gibt.

Kombiniert man die Zonen nun, sieht das ganze so aus:

```
["z1",[1,0,0],[15,2,10,15],[ ],[20],[ ],[0,0,0,0]] spawn DAC_Zone
["z2",[1,0,0],[20],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone
```

In der Zone **z1** werden zusätzlich **20** Wegpunkte für **Kettenfahrzeuge** generiert, und in Zone **z2** werden **20** Wegpunkte für **Infanterie** erzeugt. Durch diese kleine Änderung können sich die Zonen jetzt mit ihren Einheiten austauschen.

Benutzerdefinierte Wegpunkte einbinden

Es ist möglich, benutzerdefinierte Wegpunkte in das System zu integrieren. Diese Funktion ist immer dann sinnvoll, wenn man ganz bestimmte Positionen einbinden möchte, die dann den entsprechenden Einheiten zur Auswahl zu Verfügung stehen.

Das können z.B. Positionen in einer Ortschaft sein, oder auch Missionsrelevante Positionen. Durch diese Funktion ergeben sich 3 Möglichkeiten, Zonen mit Wegpunkten zu füllen:

1. **nur** generierte Wegpunkte
2. generierte **und** benutzerdefinierte Wegpunkte
3. **nur** benutzerdefinierte Wegpunkte

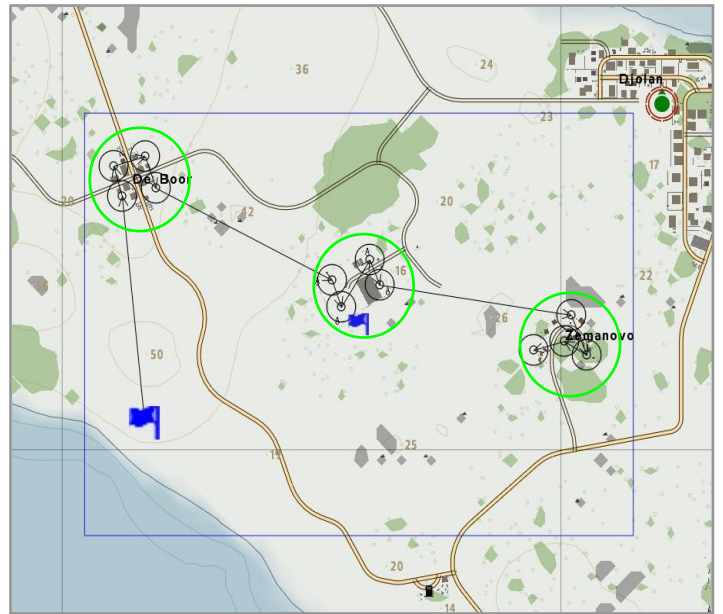
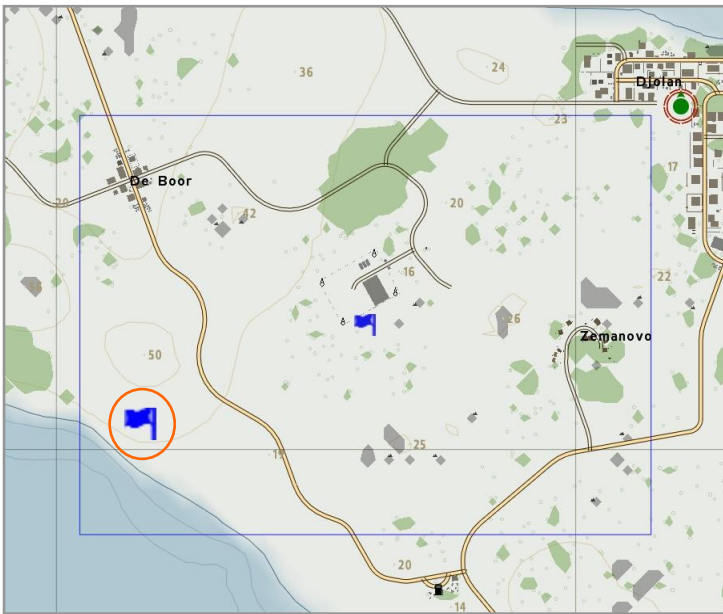
Das Einbauen von benutzerdefinierten Wegpunkten ist sehr einfach:

Du platzierst in der entsprechenden Zone eine Logic-Einheit.

Wo genau spielt dabei keine Rolle. Sie muss nur innerhalb der Zone liegen

Dieser Logic-Einheit gibst Du nun an den bestimmten Positionen ein paar Wegpunkt

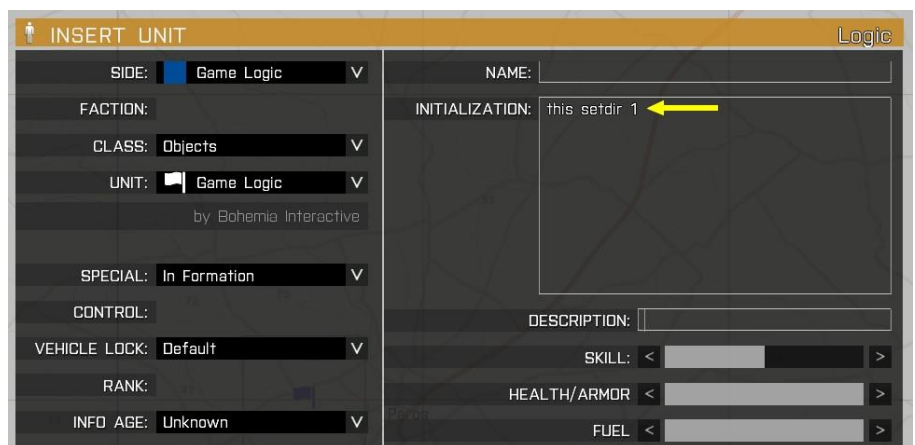
Auch die Wegpunkte müssen sich wieder innerhalb der Zone befinden



Jetzt musst Du nur noch bestimmen, welcher Einheiten-Kategorie diese Wegpunkte zugewiesen werden sollen.

Das machst Du ganz einfach mit einem Eintrag in der Init-Zeile der Logic-Einheit:

this setdir 1 = **Infanterie**
this setdir 2 = **Radfahrzeuge**
this setdir 3 = **Kettenfahrzeuge**
this setdir 4 = **Helis**
this setdir 5 = **Camps**



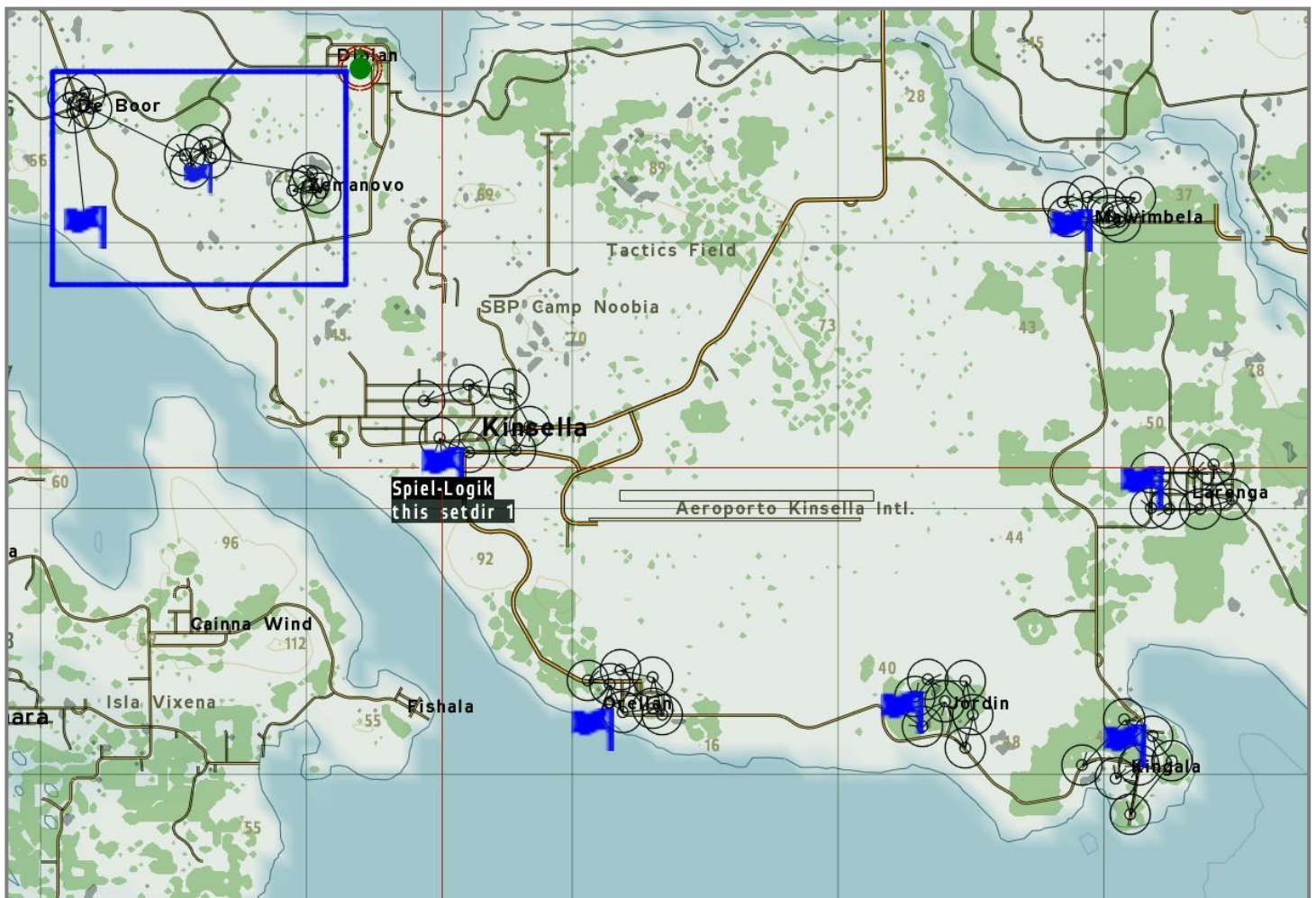
Zudem ist es möglich, benutzerdefinierte Wegpunkte **vorzudefinieren**. Das bedeutet, dass Du beliebig viele Logik-Einheiten irgendwo auf der Karte platzieren kannst, die sich nicht innerhalb einer DAC-Zone befinden und damit eigentlich keine Bedeutung haben.

Sobald aber eine dieser Logiken in den Einzugsbereich einer DAC-Zone kommen, werden sie bei der Wegpunktgenerierung berücksichtigt. Dabei spielt es keine Rolle, ob eine vorhandene Zone dorthin bewegt wird, oder ob dort eine neue Zone erzeugt wird.

Hinweis:

Jede dieser Logiken mit ihren vordefinierten Wegpunkten muss auch wieder den Eintrag für die Einheiten-Kategorie haben, z.B. **this setdir 1** (Wegpunkte für Infanterie)

Unten das Bild zeigt Dir eine DAC-Zone mit benutzerdefinierten Wegpunkten und weitere Logiken mit vordefinierten Wegpunkten an bestimmten Positionen.



Möchtest Du vordefinierte Wegpunkte einer ganz bestimmten DAC-Zone zuordnen, musst Du der entsprechenden Logik eine bestimmte Variable zuweisen. Diese Möglichkeit sorgt dafür, dass eine DAC-Zone wirklich nur die benutzerdefinierten Wegpunkte erkennt, die ihr über die Logik zugewiesen wurden.

Bitte beachte, dass jede DAC-Zone nur jeweils eine Logik pro Einheiten-Kategorie einlesen kann. Mit anderen Worten: Wenn Du einer Zone zwei Logiken mit Wegpunkten für Infanterie zuweist, wird DAC nur eine Logik verarbeiten können. Die andere bleibt unberücksichtigt.

Wie das mit dem „zuweisen“ funktioniert, erfährst Du auf der nächsten Seite.

Logiken DAC-Zonen zuweisen

Logiken mit Wegpunkten haben für den DAC unterschiedliche Bedeutung, je nach dem welcher Eintrag in der Init-Zeile der Logik eingetragen wird:

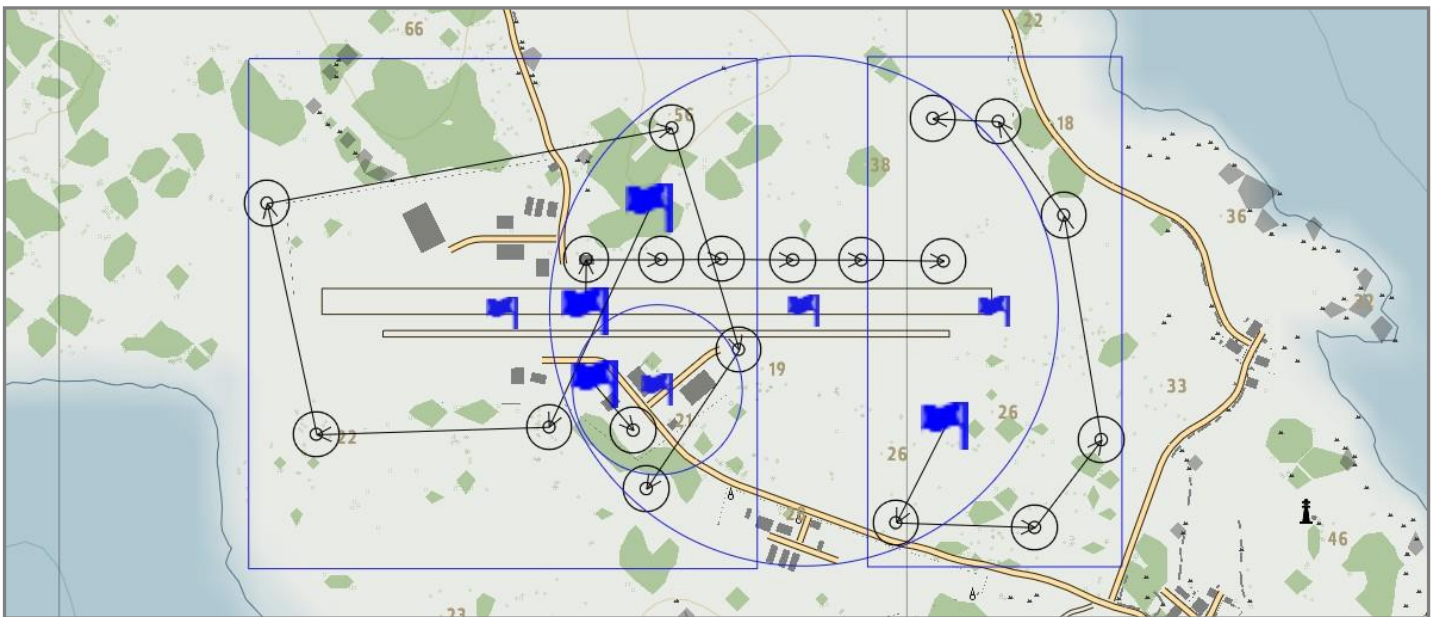
this setdir (1-5)	Die Wegpunkte der Logik definieren benutzerdefinierte Wegpunkte
this setdir 90	Die Wegpunkte der Logik definieren benutzerdefinierte Objekt-Positionen
this setdir 180	Die Wegpunkte der Logik definieren ein Polygon als DAC-Zone

Mit diesem Eintrag wird genau festgelegt, wie und wofür DAC die Wegpunkte einer Logik verarbeiten muss.

Es gibt zudem die Möglichkeit, Logiken einer bestimmten DAC-Zone zuzuweisen.

Dieses Vorgehen ist immer dann sinnvoll, wenn mehrere Zonen überlappen, oder sogar in einander liegen.

Schau Dir das folgende Bild an, damit Du verstehst was ich meine...



Wie Du siehst, sind dort einige Zonen ineinander platziert und jede Zone benutzt ihre eigenen Logiken, deren Wegpunkte z.B. ein Polygon definieren, oder ein paar benutzerdefinierte Wegpunkte markieren.

Solche Konstellationen von Zonen führen oft zu Problemen, da DAC die Logiken nicht eindeutig einordnen kann. Um dieses Problem zu umgehen, werden die jeweiligen Logiken einfach an ihre Zone „gelinkt“.

Dafür musst Du nur einen weiteren Eintrag in der Init-Zeile der Logik eintragen:

this setdir 1; this setVariable["Link", z1]	→	die Logik wurde mit der DAC-Zone z1 verlinkt.
--	---	--

Wie gesagt, ist diese Maßnahme nur nötig bei kritischen Zonen-Konstellationen, und auch nur dann, wenn mindestens eine Logik mit Wegpunkten zum Einsatz kommt.

DAC-Zonen aktivieren / deaktivieren

Jede Zone kann bei Bedarf, zu einem späteren Zeitpunkt aktiviert bzw. deaktiviert werden.

Später **Aktivieren** bedeutet, dass alles für die Zone vorbereitet wird, die Einheiten aber noch zurückgehalten werden, und erst bei einer bestimmten Bedingung erzeugt werden.

Deaktivieren bedeutet, dass alle Einheiten aus einer Zone bei einer bestimmten Bedingung gelöscht werden.

Beide Methoden haben in erster Linie den Zweck, Leistung einzusparen bzw. wieder freizugeben. Zonen, die z.B. erst im späteren Verlauf der Mission benötigt werden, da sie weit ab vom ersten Missionsziel liegen, können so problemlos auf „Standby“ gestellt werden. Das kann u. U. schon viel Leistung einsparen.

Genauso die Zonen, die für die Mission keine Bedeutung mehr haben, diese können über diese Funktion gelöscht werden und geben dadurch wieder Leistung frei.

DAC stellt dafür eine Funktion bereit, die mehrfach bei einer Zone angewendet werden kann. Mit anderen Worten, Du kannst eine Zone mehrmals Aktivieren und Deaktivieren.

Bitte beachte, dass eine Zone, die mehrfach deaktiviert / aktiviert wird, immer wieder in ihren Urzustand versetzt wird. Das bedeutet, wenn Du eine Zone deaktivierst, in der bereits einige Einheiten um's Leben gekommen sind, und Du diese Zone später wieder aktivierst, werden in dieser Zone wieder alle Einheiten generiert, die für die Zone definiert waren.

Eine Zone ist nicht automatisch deaktiviert, wenn alle Einheiten aus dieser Zone down sind. Du kannst eine solche Zone jeder Zeit wieder zum Leben erwecken, in dem Du sie einmal deaktivierst, und anschließend wieder aktivierst.

Die Funktion wird folgendermaßen aufgerufen:

[NameDerZone] call DAC_Activate	Beispiel: ([z1] call DAC_Activate oder [z1,z2,z3] call DAC_Activate)
[NameDerZone] call DAC_Deactivate	Beispiel: ([z1] call DAC_Deactivate oder [z1,z2,z3] call DAC_Deactivate)

Möchtest Du, dass eine Zone von Beginn an deaktiviert ist, weil sie erst im späteren Missionsverlauf benötigt wird, dann kannst Du folgenden Parameter dafür benutzen:

["z1",[1,0,0],[15,2,10,15],[],[],[0,0,0,0]] spawn DAC_Zone

↑
→ Die Zone ist von Beginn an **aktiv**

["z1",[1,1,0],[15,2,10,15],[],[],[0,0,0,0]] spawn DAC_Zone

↑
→ Die Zone ist von Beginn an **inaktiv**

Achtung, wenn Du Zonen **aktivierst/deaktivierst**, solltest Du immer zusammenhängende Zonen berücksichtigen. Also Zonen, die miteinander verlinkt sind, sollten auch gemeinsam **aktiviert/deaktiviert** werden. Bitte beachte auch, dass zwischen dem **Aktivieren / Deaktivieren** eine kleine Pause eingelegt wird (min. 3 Sek.)

Reduzierung von Infanterie-Gruppen

Das **Reduzieren** von **Infanterie**-Gruppen ist in erster Linie ein Mittel, um Gruppen außerhalb einer gewissen Reichweite auf ein Minimum zu reduzieren und so Rechenleistung einzusparen.

Da alle KI- bezogenen Scripte über den Leader einer Gruppe laufen, interessiert es in dem Moment nicht, wenn der Rest der Gruppe „nicht da“ ist. Nähert sich die „**reduzierte Gruppe**“ einer feindlichen Einheit, oder einer Spieler-Einheit, wird der Rest der Gruppe wieder erzeugt und die Gruppe ist wieder komplett.

Bevor die Gruppe **reduziert** wird, werden die benötigten Daten jeder Einheit gespeichert, das sind:

- **Einheitentyp**
- **Skill der Einheit**
- **Beschädigungsgrad der Einheit**
- **Waffen und Munition**

Wobei nicht jede einzelne Patrone gespeichert wird. Es wird „grob“ die Anzahl der Magazine gespeichert ;-)

Neuerdings werden auch die Gruppen der Kategorie **Radfahrzeuge** reduziert, allerdings mit einer kleinen Einschränkung: Es werden nur Gruppen reduziert, die mit maximal 1 Fahrzeug unterwegs sind. Die Gruppen, die mit mehreren Fahrzeugen unterwegs sind, werden von der Reduzierung nicht berücksichtigt.

Das bedeutet:

Eine Gruppe, bestehend aus 4 Einheiten und 1 HMMWV, **wird reduziert**

Eine Gruppe, bestehend aus 12 Einheiten und 1 Truck5t, **wird reduziert**

Eine Gruppe, bestehend aus 5 Einheiten und 2 HMMWV, **wird nicht reduziert**

Eine Gruppe, bestehend aus 10 Einheiten und 2 HMMWV + 1 Truck5t, **wird nicht reduziert**

Im Normalfall wird die Gruppe direkt bei ihrem Anführer wiederhergestellt. Dieser Umstand kann aber dazu führen, dass man mitbekommt, wie die Einheiten Ihre Waffen nachladen. Man hört dann je nach Abstand ein deutliches Klick-Geräusch (mehrfach bei mehreren Einheiten).

Um dieses Problem zu umgehen, bietet Dir DAC folgende Möglichkeit:

Platziere im Editor eine **Logic**-Einheit mit dem Namen **DAC_Pos_E**. Sobald diese **Logic** vorhanden ist, wird DAC alle feindlichen Einheiten an dieser Position wiederherstellen. Die Einheiten werden erst dann wieder in die Gruppe gebeamt, wenn sie komplett ausgerüstet sind und nachgeladen haben.

Damit sind dann auch die „störenden“ Nebengeräusche verschwunden ☺

Es ist sinnvoll, die **Logic** weit ab vom Missionsgeschehen zu platzieren, aber bitte nicht im Wasser ;-)

Um sicher zu gehen, dass die Einheiten sich an dieser Position nicht gegenseitig behindern, kannst Du die **Logic** etwas verdrehen, denn der Drehwinkel der **Logic** ist gleichzeitig Platzierungsradius der Einheiten.

Platziere dann noch eine Logic mit dem Namen **DAC_Pos_W**, um auch den freundlichen Einheiten eine Respawn-Position zu geben (bitte nicht direkt neben der Logic **DAC_Pos_E**).

Mit welchem Parameter die Gruppen-Reduzierung aktiviert bzw. justiert wird, kannst Du unter dem Thema **Die Konfigurations- Dateien** erfahren

KI-Respawn aktivieren und konfigurieren

Um den Respawn zu aktivieren, muss mindestens ein Camp auf der entsprechenden Seite generiert werden. Damit der Scriptaufruf nicht noch komplexer wird, habe ich mich dazu entschlossen, die Parameter für die Camps in das Array für Helis zu packen.

Das bedeutet also, dass in einer Zone entweder Helis generiert werden können, oder eben Camps. Ich denke, dass ist ein guter Kompromiss. Außerdem muss man auf die Helis ja nicht verzichten. Diese müssen dann eben in einer anderen Zone erstellt werden.

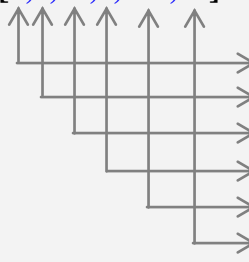
Respawned werden nur folgende Kategorien: **Infanterie**, **Radfahrzeuge**, **Kettenfahrzeuge**

Hier ein Beispiel für einen Scriptaufruf, in dem ein Spawn-Camp (Konfiguration **3**) generiert wird:

```
["z1",[1,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10],[0,0,0,3]] spawn DAC_Zone
```

Es werden **6** Parameter benötigt, um ein oder mehrere Camps in einer Zone zu erstellen. Für Helis waren es **3** optional **4** Parameter. Außerdem muss eine gültige **Camp-Konfigurations-Nr.** angegeben werden. Hier die Bedeutung der einzelnen Parameter:

[1,2,50,0,100,10]

- 
- Die Anzahl der Camps, die generiert werden sollen.
 - Die Größe der dort stationierten Gruppe (Standardmäßig 1 - 4 wählbar).
 - Der Bewegungsradius für die dort stationierte Gruppe.
 - Der Spawntyp: **0** = Soldaten + Fahrzeuge | **1** = nur Soldaten | **2** = nur Fahrzeuge.
 - Die Wahrscheinlichkeit für den Respawn in % Prozent.
 - Die Anzahl der Spawns pro Camp in der Zone, also nicht die Gesamtzahl in der Zone.

[3,2,50,0,100,5]

Das würde z.B. bedeuten, dass in der Zone **3** Camps generiert werden, und jedes Camp hat **5** Respawn zu Verfügung, egal von welchem Typ. Die Zone verfügt demnach über insgesamt **15** Respawn. Wird die Gruppe eines Camps ausgelöscht, gehen mit ihr auch die restlichen Spawns aus dem Camp verloren.

Achtung: je mehr Camps man in einer Zone generieren möchte, desto mehr Platz wird auch benötigt. Man sollte so einer Zone dann auch den Spielraum geben, den sie braucht. Es sei denn, du beabsichtigst, dass die Zonen eng zusammen liegen sollen.

Info:

Die bei einem Camp stationierte Gruppe löst sich auf, sobald die Respawn in einem Camp aufgebraucht sind.

Die Camp-Gruppe wird sich dann der nächstgelegenen Gruppe anschließen. Wird eine Camp-Gruppe zerstört, werden automatisch die noch verfügbaren Respawn in dem Camp auf **0** gesetzt.

Wie man ein Camp konfiguriert, wird ausführlich im Thema **DAC Config Camp** erklärt.

Camps mit Zonen verlinken

Werden mehrere Camps auf einer Seite generiert, entscheidet der Zufall, in welchem Camp ein Respawn ausgeführt wird. Dieser Zustand ist nicht immer optimal, da je nach Lage eines Camps, die neu generierten Einheiten einen sehr langen Weg in's Missionsgebiet zurücklegen müssen.

Um den Zufall etwas einzugrenzen, aber auch um mehr Anpassungsmöglichkeiten zu haben, ist es nun möglich, Camps mit Zonen zu verlinken. Das bedeutet, dass Du genau festlegen kannst, welche Camps für welche Zonen zuständig sind.

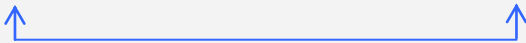
Stell Dir vor, Du hast 2 Camp-Zonen mit je einem Camp und 2 Master-Zonen, in denen nur Einheiten generiert werden. Im Normalfall würden die 2 Camps beide Master-Zonen gleichermaßen mit neuen Einheiten versorgen.

Wenn Du aber möchtest, dass jedes Camp für genau eine Master-Zone zuständig ist, kannst Du das mit dieser Funktion umsetzen.

Das Prinzip ist ganz einfach:

In der Camp-Zone musst Du nur die Zone angeben, die von der Camp-Zone versorgt werden soll:

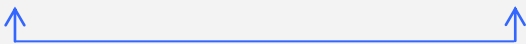
```
["z1",[1,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z2]],[0,0,0,3]] spawn DAC_Zone
```



Oben siehst Du, dass die Camps (es können ja auch mehrere sein) aus **z1** die Zone **z2** versorgen.

In dem Array für Camps, wurde ein optionaler Parameter angegeben. Dieser Parameter muss vom Typ „Array“ sein (damit man mehrere Zonen angeben kann) und es muss der echte Zonenname eingetragen werden, also nicht als „String“.

```
["z3",[1,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z4]],[0,0,0,3]] spawn DAC_Zone
```



Das wäre dann die andere Camp-Zone **z3**, die für Zone **z4** zuständig ist.

Wie ich schon angedeutet habe, können auch direkt mehrere Zonen eingetragen werden:

```
["z1",[1,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z4,z5]],[0,0,0,3]] spawn DAC_Zone
```

```
["z2",[2,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z6,z7]],[0,0,0,3]] spawn DAC_Zone
```

```
["z3",[3,0,0],[ ],[ ],[ ],[ 2,2,50,0,100,10,[z5,z7]],[0,0,0,3]] spawn DAC_Zone
```

Die Camp-Zone **z1**, versorgt die Zonen **z4 + z5**

Die Camp-Zone **z2**, versorgt die Zonen **z6 + z7**

Die Camp-Zone **z3**, versorgt die Zonen **z5 + z7**, die schon jeweils von den Zonen **z1 + z2** versorgt werden.

Achtung: Die Benutzung dieser Funktion, setzt den globalen Respawn außer Kraft. Das bedeutet, dass Camp-Zonen, die nicht verlinkt sind, nur noch sich selbst versorgen und keine andere Zone mehr

Zonenwerte (Größe, Position, Ausrichtung und Konfiguration) verändern

Es ist jederzeit möglich, die Größe und auch die Position einer DAC-Zone zu verändern.

Eine solche Änderung hat immer zu Folge, dass für die entsprechenden Einheiten aus der Zone neue Wegpunkte generiert werden.

Dabei werden die Positionen der alten Wegpunkte, mit den Positionen der neuen Wegpunkte überschrieben. Je nach Anzahl der Wegpunkte in der Zone, kann dieser Prozess mehrere Sekunden dauern.

Ist der Prozess abgeschlossen, haben alle betroffenen Gruppen anschließend neue Wegpunkte, die dann von ihnen abgearbeitet werden. Je nach DAC- Einstellung, bekommst Du die Änderungen per Sidechat mitgeteilt, und auch die Gruppen melden, dass sie neue Wegpunkte empfangen haben.

Bitte beachte, dass Camps bzw. die Gruppen, die bei den Camps stationiert sind, von einer Zonenänderung nicht betroffen sind, weil Camps feste Einrichtungen sind und nicht versetzt werden können.

Hier ein paar Hinweise, die Du bei einer Zonenänderung unbedingt beachten solltest:

Versuche möglichst eine Zone kontrolliert zu versetzen, um sicher zu gehen, dass an der neuen Position auch genügend Wegpunkte generiert werden können. Im Zweifelsfall, die in Frage kommenden Positionen kurz vorher testen und das am besten mit aktivierten Wegpunkt- und Zonen-Markern.

Außerdem die DAC-Systemmeldungen aktivieren. Diese zeigen Dir genau, welche Wegpunkte generiert wurden und welche nicht.

Bitte beachte in dem Zusammenhang, dass wenn eine Zone versetzt wird und an der neuen Position keine Wegpunkte generiert werden konnten, weil z.B. das Gelände nicht geeignet ist, oder eine zu hohe Objektdichte vorliegt, die Zone zwar auf der neuen Position angezeigt wird, die Einheiten sich aber zu ihren alten Wegpunkten weiterbewegen.

Es kann demnach auch passieren, dass sich Infanterie-Einheiten in Richtung der neu platzierten Zone bewegen, und z.B. die Kettenfahrzeuge zurückbleiben, weil für sie keine neuen Wegpunkte generiert werden konnten.

Du kannst nicht nur aktivierte Zonen verändern, sondern auch deaktivierte Zonen. So ist es z.B. möglich, eine Zone zu deaktivieren, dadurch werden alle Einheiten der Zone gelöscht, dann die Zone zu versetzen, um anschließend die Zone an ihrer neuen Position wieder zu aktivieren.

Werden Zonen zu weit weg versetzt, kann es passieren, dass die Einheiten sich nicht mehr bewegen, besonders dann, wenn viele Gruppen davon betroffen sind (eine Erklärung dafür habe ich nicht). Evtl. hilft es, die Zonen aufzusplitten und die Zonenänderung dann leicht zeitversetzt auszuführen.

Benutzerdefinierte Wegpunkte gehen bei einer Versetzung einer Zone verloren.

Der Skriptaufruf sieht folgendermaßen aus:

[[zonename](#), [position](#), [size&direction](#), [unitconfig](#), [behavconfig](#), [wpconfig](#)] call DAC_fChangeZone

- | | |
|---|--|
| 1. Der Name der Zone | Der richtige Name der Zone (nicht als String) |
| 2. Die Positionsangabe | wohin die Zone platziert werden soll [Positions-Array] |
| 3. Die Größe & Ausrichtung | Ändert Größe und Ausrichtung der Zone [x,y,d] |
| 4. Angabe der Einheiten-Konfiguration | eine gültige Nummer aus der DAC_Config_Units.sqf |
| 5. Angabe der Verhaltens-Konfiguration | eine gültige Nummer aus der DAC_Config_Behaviour.sqf |
| 6. Angabe der Wegpunkt-Konfiguration | eine gültige Nummer aus der DAC_Config_Waypoints.sqf |

Beispiel 1: Du möchtest nur die Größe einer Zone ändern, dann sieht der Skriptaufruf wie folgt aus:

```
[z1,[],[300,400,0],0,0,0] call DAC_fChangeZone
```

Die Zone **z1** wird auf die Größe 300x400 Meter geändert. Da keine Positionsänderung vorgesehen war, bleibt das Positions-Array leer. Die Werte für die verschiedenen Konfigurationen bleiben auf 0.

Beispiel 2: Die Position der Zone soll auf die Position des Spielers versetzt werden:

```
[z1,Position Player,[],0,0,0] call DAC_fChangeZone
```

Die Zone **z1** wird auf die Position des Spielers versetzt. Da keine Größenänderung vorgesehen war, bleibt das Größen-Array leer. Die Werte für die verschiedenen Konfigurationen bleiben auf 0.

Beispiel 3: Du möchtest sowohl die Größe, als auch die Position der Zone ändern:

```
[z3,Position Logic1,[500,1200,0],0,0,0] call DAC_fChangeZone
```

Die Zone **z3** wird auf die Position der Logic „**Logic1**“ versetzt, und die Größe wird auf 500x1200 Meter verändert. Die Werte für die verschiedenen Konfigurationen bleiben auf 0.

Eine weitere Möglichkeit besteht darin, dass Du für eine bestimmte Zone eine neue Konfiguration laden kannst. Folgende Konfigurationen können geändert werden:

DAC_Config_Unit	→	[z3,[],[],1,0,0] call DAC_fChangeZone
DAC_Config_Behaviour	→	[z3,[],[],0,4,0] call DAC_fChangeZone
DAC_Config_Waypoints	→	[z3,[],[],0,0,2] call DAC_fChangeZone

Um eine neue Konfiguration für eine Zone zu laden, musst Du in dem Skriptaufruf nur eine gültige Nummer aus der jeweiligen Konfiguration eintragen.

Folgendes solltest Du bei der Änderung einer Konfiguration beachten:

Wird die Konfiguration **Behaviour** verändert, werden die Einheiten der Zone die Änderung sofort umsetzen. Sofort bedeutet, innerhalb von etwa 10 Sekunden.

Änderst Du die Konfiguration **Unit**, wird die Änderung nicht sofort wirksam, sondern erst, wenn neue Einheiten für die Zone generiert werden. Das ist z.B. der Fall, wenn eine Gruppe zerstört wird und anschließend in einem Camp eine Ersatzgruppe generiert wird.

Du kannst eine Zone auch komplett durch eine andere Einheiten-Konfiguration ersetzen. Dafür musst Du aber die Zone einmal deaktivieren (alle Einheiten werden gelöscht), dann die neue Einheiten-Konfiguration laden, und anschließend die Zone wieder aktivieren.

Achtung, da die Seite einer Zone nicht verändert werden kann, musst Du unbedingt darauf achten, dass eine auf die Seite passende Einheiten-Konfiguration geladen wird, sonst gibt's Mord und Totschlag ;-)

Also bitte keine West-Einheiten auf Ost Seite generieren !

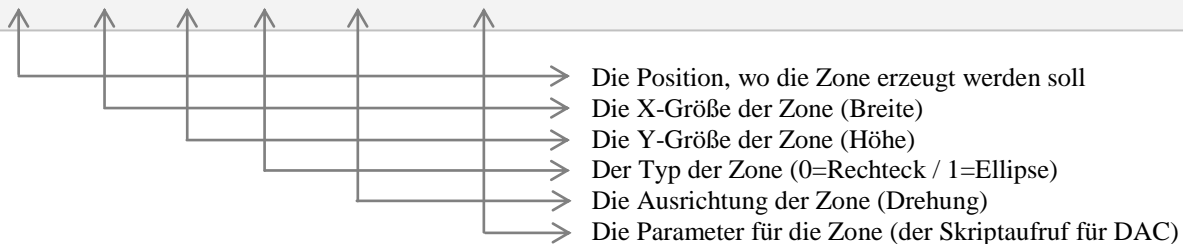
DAC-Zonen erzeugen (während einer laufenden Mission)

Ein ganz neues Feature ist das Erzeugen von DAC-Zonen während einer laufenden Mission, also „on the fly“. Dieses Merkmal eröffnet ganz neue Möglichkeiten und hilft Dir dabei, Missionen noch dynamischer zu gestalten.

Wird diese Funktion aufgerufen, erzeugt DAC im Hintergrund einen neuen Auslöser und übergibt die Startparameter. Die so erzeugte DAC-Zone integriert sich vollständig und ohne Einschränkung in das vorhandene DAC-System.

Aufgerufen wird die Funktion folgendermaßen:

[position,sizeX,sizeY,type,direction,parameter] call DAC_fNewZone



Beispiel:

[(position player),250,150,0,0,parameter] call DAC_fNewZone

Es wird eine rechteckige Zone auf der Position des Spielers erzeugt. Die Größe der Zone beträgt 250 x 150 m und ausgerichtet ist die Zone nach Norden (0°)

Der letzte Parameter entspricht dem Skriptaufruf, der den neuen Auslöser zu einer DAC-Zone macht, also im Prinzip ist das der Skriptaufruf, der normalerweise in die Init-Zeile eines Auslösers eingetragen wird.

Es macht Sinn, die Parameter in einer lokalen Variablen zwischenzuspeichern, und dann diese Variable in dem Funktionsaufruf einzutragen:

Beispiel:

```
_values = ["z1",[1,0,0],[5,2,50,8],[],[],[1,1,1,1]];
[(position player),250,150,0,0,_values] call DAC_fNewZone
```

Auf die Parameter der Variable **_values** möchte ich hier nicht weiter eingehen, da ich diese bereits weiter oben ausführlich erklärt habe, nur soviel, dass eine Zone mit dem Namen „z1“ erzeugt wird und diese Zone 5 Infanterie-Gruppen auf Seite West generiert.

Es gibt bei dieser Funktion die kleine **Einschränkung**, dass nicht mehrere Zonen gleichzeitig erzeugt werden können, Das bedeutet, dass Du warten musst bis eine Zone und deren Gruppen generiert wurden, bevor Du eine weitere Zone erzeugen kannst. Um festzustellen, wann eine neue Zone komplett fertig gestellt wurde, kannst Du die Variable **DAC_NewZone** abfragen:

Beispiel:

```
waituntil{DAC_NewZone == 0};
_values = ["z1",[1,0,0],[5,2,50,8],[],[],[1,1,1,1]];
[(position player),250,150,0,0,_values] call DAC_fNewZone
```

DAC-Zonen löschen (während einer laufenden Mission)

Es ist jederzeit möglich eine DAC-Zone zu löschen. Dabei kannst Du nicht nur Zonen löschen, die während der Mission mit der o.g. Funktion erzeugt wurden, sondern auch Zonen, die bereits im Editor platziert wurden.

Das Löschen von DAC-Zonen verhält sich ähnlich wie das Deaktivieren von DAC-Zonen:

Alle Einheiten der entsprechenden Zone werden gelöscht.

Zudem wird auch die DAC-Zone und alle damit verbundenen Variablen und Array-Einträge, sowie alle Wegpunkte, tatsächlich gelöscht. Insofern ist es nicht möglich, eine gelöschte Zone wieder zu aktivieren, da der DAC diese Zone nicht mehr kennt.

Der Funktionsaufruf zum Löschen einer Zone sieht folgendermaßen aus:

```
["zone"] call DAC_fDeleteZone
```

Sollte es Gruppen aus anderen Zonen geben, die Wegpunkte in der gelöschten DAC-Zone hatten, wird DAC automatisch dafür sorgen, dass die Wegpunkte dieser Gruppen dementsprechend reduziert werden.

Auch bei dieser Funktion ist zu beachten, dass der aufgerufene Lösch-Prozess erst beendet sein muss, bevor die nächste Zone gelöscht, bzw. eine neue Zone erstellt werden kann, und zwar mit der gleichen Abfrage:

Beispiel:

```
waituntil{DAC_NewZone == 0};  
["zone"] call DAC_fDeleteZone
```

Bei dieser Lösch-Funktion ist es aber möglich, gleich mehrere Zonen zu löschen, indem Du einfach mehrere Zonen dem Funktionsaufruf übergibst:

Beispiel:

```
waituntil{DAC_NewZone == 0};  
["zone1","zone2","zone3","zone4"] call DAC_fDeleteZone
```

Wie Du dem Funktionsaufruf entnehmen kannst, müssen die zu löschenden Zonen als "String" angegeben werden.

Die DAC-Arti (erfordert eine Logik mit dem Namen „DAC_Support_Logic“)

Die DAC-Arti ist komplett geskriptet und ist sowohl für die KI, als auch für den Spieler verfügbar. Für die DAC-Arti gibt es eine eigene Konfigurations-Datei, in der alle nötigen Parameter abgelegt sind.

Es gibt 3 Bedingungsstufen, die erfüllt sein müssen, damit die Arti losschlagen kann:

Bedingungen Stufe 1 > bezieht sich auf die **Target-Gruppe** (die feindliche Gruppe, die von der KI erfasst wurde)

Bedingungen Stufe 2 > bezieht sich auf die **Call-Gruppe** (die Gruppe, die den Arti-Schlag anfordert)

Bedingungen Stufe 3 > bezieht sich auf die **Arti-Einheit** (eine Einheit, die für einen Arti-Schlag in Frage kommt)

DAC-Arti für die KI

Die DAC-Arti(KI) kann grundsätzlich nur von Infanterie-Einheiten angefordert werden. Ausgelöst wird die Anforderung dadurch, wenn eine KI-Gruppe Feindkontakt hat und genügend über den Feind weiß.

Die Anforderung durchläuft dann die 3 Bedingungsstufen und wird bei Erfolg zu einem Arti-Schlag führen.

Die Konfiguration der Arti, und auch die Bedingungen, sind in der **DAC_Config_Arti.sqf** definiert.

Wie bei allen anderen Konfigurations-Dateien, sind auch hier wieder beliebig viele Einträge möglich.

Für jede Zone kann separat eine eigene Arti-Konfiguration geladen werden. Wird keine Konfiguration geladen, können die Einheiten aus dieser Zone keine Arti anfordern.

DAC-Arti für den Spieler

Die DAC-Arti(Player) kann im Prinzip jederzeit und an jedem Ort angefordert werden. Deshalb entfällt hier auch die **Bedingungsstufe 1**.

Im Gegensatz zu der DAC-Arti(KI), die automatisch durch die KI ausgelöst wird, muss die DAC-Arti(Player) logischerweise manuell ausgelöst werden. Dafür stellt DAC einen einfachen Skriptaufruf zu Verfügung:

[NameDesSpielers,PositionsArray,KonfigurationNr,ArtiRadius] spawn DAC_fCallArti

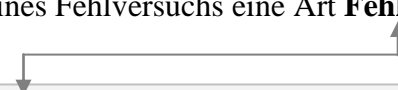
Beispiel: [Player,Position LogA,3,50] spawn DAC_fCallArti

Auf Position der Logik, mit dem Namen „LogA“, wird Arti angefordert.

Arti-Konfiguration = 3, Wirkungsradius maximal 50m

Dieser Aufruf führt nicht zwingend zu einem Arti-Schlag, denn er löst nur die Arti-Anforderung aus. Ob es letztendlich zu einer Ausführung kommt, hängt ganz von den Einstellungen der Konfiguration ab.

Wenn Du die DAC-Systemmeldungen aktiviert hast (siehe **DAC_Config_Creator**), wird Dir DAC im Falle eines Fehlversuchs eine Art **Fehlercode** anzeigen. Die Fehlercodes haben folgende Bedeutung:

- 
- 1 = **Target-Gruppe** bewegt sich zu schnell [nur relevant für DAC-Arti(KI)]
 - 2 = Anzahl Einheiten in **Target-Gruppe** zu gering [nur relevant für DAC-Arti(KI)]
 - 3 = Zu wenig Fahrzeuge in der **Target-Gruppe** [nur relevant für DAC-Arti(KI)]
 - 4 = Zu wenig Einheiten in der **Call-Gruppe**
 - 5 = Zu geringer Abstand der **Call-Gruppe** zur **Target-Gruppe** [oder zur Position, bei DAC-Arti(Player)]
 - 6 = Benötigter Einheitentyp in **Call-Gruppe** nicht vorhanden
 - 7 = Benötigter Skill der **Call-Gruppe** wird unterschritten
 - 8 = Angegebene Wahrscheinlichkeit wurde nicht erreicht
 - 9 = Freundliche Einheiten unterschreiten den Mindestabstand zur Zielposition
 - 10 = Keine gültige **Arti-Einheit** vorhanden, oder keine **Arti-Einheit** bereit bzw. in Reichweite
 - 11 = Globale Bedingung nicht wahr
 - 12 = **Arti-Einheit** außerhalb der maximalen Reichweite

Wie man die Arti konfiguriert, wird ausführlich im Thema **DAC_Config_Arti** erklärt.

Die DAC-Bodenunterstützung (erfordert eine Logik mit dem Namen „DAC_Support_Logic“)

Die DAC-Bodenunterstützung läuft auf KI-Ebene automatisch ab. Das bedeutet, dass DAC-Gruppen, wenn sie unter Beschuß kommen, versuchen werden Bodenunterstützung anzufordern.

Dieses Feature ist in der neuen DAC-Version auch auf Spieler-Ebene verfügbar. Das wiederum bedeutet, dass auch der Spieler bei Bedarf Bodenunterstützung anfordern kann.

Die Funktion für die Unterstützung kann jederzeit von Dir aufgerufen werden, allerdings bedeutet das nicht, dass jede Anforderung auch gleichzeitig eine Unterstützungs-Gruppe herbeiruft. Vielmehr durchläuft dieser Prozess die gleichen Bedingungen, als wenn die KI Unterstützung anfordert.

Folgender Funktionsaufruf fordert die DAC-Bodenunterstützung an (max. 1 Gruppe pro Anforderung):

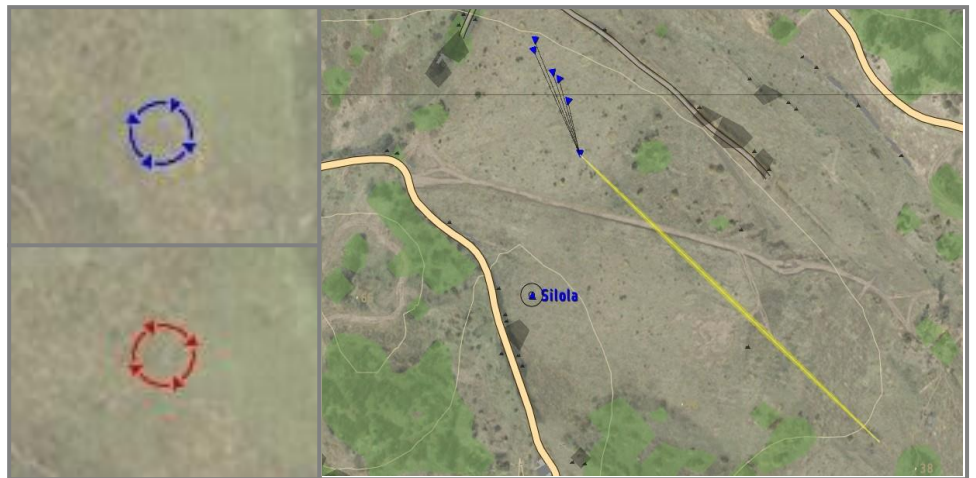
```
[player,position] spawn DAC_fCallHelp
```

Der erste Parameter muss eine gültige Spieler-Einheit sein. Wenn Du eine KI-Einheit einträgst, wird DAC die Anforderung ignorieren.

Der zweite Parameter muss eine gültige Position innerhalb eines bestimmten Radius zur Spieler-Einheit sein. Wo Du diesen Radius und andere Parameter definierst, kannst Du unter dem Thema **DAC Config Creator** nachlesen. Standard-mäßig ist eine Distanz von 1000m definiert.

Wenn Du die DAC-Marker aktiviert hast, erkennst Du eine erfolgreiche Anforderung an einem **blau-rotierenden** Marker. Deine Positionsangabe war dann innerhalb der Reichweite.

Ist die Position für eine Anforderung außerhalb der definierten Reichweite, wird das mit einem **rot-rotierenden** Marker quittiert.



Eine Gruppe die auf Unterstützungs-Mission unterwegs ist, erkennst Du an einem **gelb-animierten** Marker.

In der Regel wird DAC immer versuchen, die nächstgelegene Gruppe zu erreichen, egal zu welcher Einheiten-Kategorie die Gruppe gehört. Wie ich bereits sagte, bedeutet eine erfolgreiche Unterstützungs-Anforderung nicht, dass auch eine Gruppe dieser Anforderung nachkommt. Es gibt viele Gründe, warum eine Gruppe keine Unterstützung leisten kann: Sie steht unter Beschuss, sie leistet einer anderen Gruppe Unterstützung usw.

Um festzustellen, welche Gruppen zur Unterstützung eingetragen sind, kannst Du folgende Variable abfragen:

DAC_Support_Logic getVariable "**support_groups**" = (nur durch den Spieler angeforderte Gruppen)

Hinweis:

Zur Unterstützung können nur Gruppen kommen, die vom DAC-System generiert wurden. Andere Gruppen werden nicht unterstützt.

Editor-Gruppen einbinden

Es ist möglich, Gruppen die im Editor platziert wurden, in das DAC-System einzubinden. Das kannst Du sofort bei Missionsstart arrangieren, oder aber zu einem späteren Zeitpunkt.

Eine eingebundene Gruppe kann zudem jederzeit wieder aus dem DAC-System „entlassen“ werden. Hat die Editor-Gruppe Wegpunkte, wird DAC diese Wegpunkte speichern und für den Fall, dass die Gruppe später wieder aus dem DAC-System entlassen wird, der Gruppe ihre alten Wegpunkte wieder zurückgeben.

Folgende Kategorien werden dabei unterstützt:

Infanterie, **Radfahrzeuge**, **Kettenfahrzeuge**

Sobald eine Gruppe in das DAC-System eingebunden wurde, werden folgende DAC-Merkmale übertragen:

- Die Gruppe benutzt DAC-Wegpunkte bzw. die entsprechenden Bewegungsroutinen.
- Die Gruppe wird reduziert.
- Tote Einheiten der Gruppe werden gelöscht (je nach Einstellung der Löschroutine).
- Die Gruppe kann Unterstützung + Arti anfordern.
- Die Gruppe kann selbst zur Unterstützung angefordert werden.
- Die Gruppe kann Gebrauch von den DAC_Events machen
- Die Gruppe wird eine DAC-Verhaltens-Konfiguration benutzen.

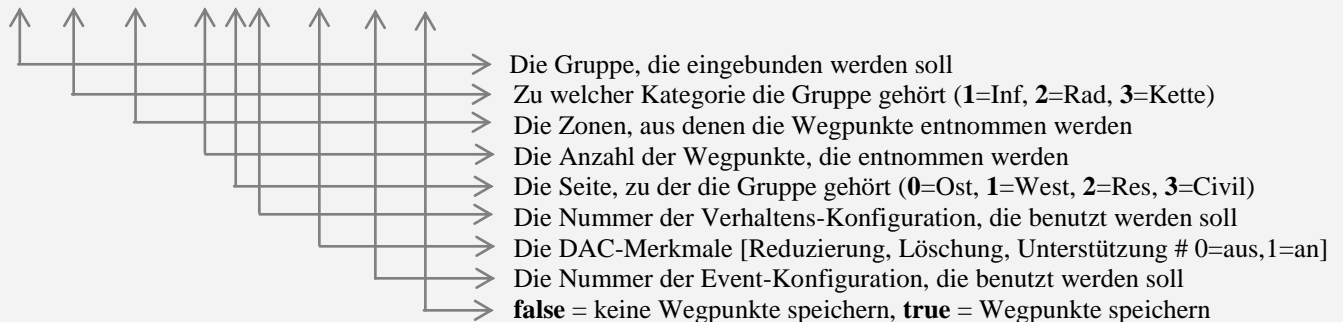
Es gibt aber auch eine Einschränkung:

- Eine eingebundene Gruppe kann nicht respawned werden.

Die Einbindung einer Editor-Gruppe wird mit folgendem Skriptaufruf vollzogen:

Beispiel:

`[group,1,[z1,z2],10,1,1,[1,1,1],0,false] spawn DAC_fInsertGroup`



Um eine Gruppe wieder aus dem DAC-System zu entfernen, benutzt Du folgende Funktion:

`[group] spawn DAC_fReleaseGroup`

Wurden für die Gruppe Wegpunkte gespeichert, sorgt die Funktion automatisch dafür, dass die Editor-Wegpunkte der Gruppe wieder hergestellt werden.

DAC-Gruppen entlassen

Es gibt neuerdings auch die Möglichkeit, DAC-Gruppen aus dem DAC-System zu entlassen bzw. zu entfernen. Diese Funktion wirft schnell die Frage auf: „wieso sollte man das tun?“

Nun, der Grund dafür ist sehr simpel: Es gibt sehr gute andere KI-Erweiterungen, die ich mit dieser Funktion unterstützen möchte ☺

Der Ablauf für eine derartige Konstellation könnte folgendermaßen aussehen:

1. DAC generiert dynamisch die Einheiten innerhalb von gesetzten Zonen
2. Nach der Initialisierung werden alle generierten Gruppen aus dem DAC-System entfernt.
3. Eine andere KI-Erweiterung übernimmt die Steuerung der Gruppen.

Es müssen selbstverständlich nicht alle generierten DAC-Gruppen entfernt werden. Genauso gut ist es möglich, nur DAC-Gruppen aus bestimmten Zonen aus dem DAC-System zu entfernen.

Beispiel:

Du generierst mit dem DAC insgesamt 4 Zonen. Davon lässt Du 2 Zonen unberührt, also das der DAC die Gruppen kontrolliert. Die Gruppen der anderen 2 Zonen entfernst Du aus dem DAC-System und überlässt die Kontrolle dieser Gruppen einem anderen KI-System.

Natürlich ist es auch möglich, einzelne Gruppe aus dem DAC-System zu entfernen. Der Funktionsaufruf für Das Entfernen von DAC-Gruppen aus dem DAC-System, ist der gleiche wie weiter oben beschrieben:

```
[group] spawn DAC_fReleaseGroup
```

Da die entlassenen DAC-Gruppen keine Wegpunkte mehr haben, bleiben sie nach Aufruf der Funktion stehen. Um dem entgegenzuwirken, gibt es ein paar optionale Parameter, damit die Gruppen „in Bewegung“ bleiben.

```
[group, [z1],1,8] spawn DAC_fReleaseGroup
```

- | | | |
|------|---|--|
| [z1] | = | eine (oder auch mehrere) gültige DAC-Zone(n), aus der/denen Wegpunkte entnommen werden |
| 1 | = | die Einheiten-Kategorie (1=Infanterie, 2=Radfahrzeuge, 3=Kettenfahrzeuge) |
| 8 | = | die Anzahl der Wegpunkte, die per Zufall aus der/den Zonen an die Gruppen übergeben werden |

Ein simples DAC- unabhängiges Move- Skript, lässt die Gruppen dann die Wegpunkte per Zufall abarbeiten.

Da ich das KI-System „**GroupLink4**“ sehr schätze, habe ich mich darum bemüht, dieses KI-System Standardmäßig zu unterstützen. Das bedeutet, dass bei Bedarf „entlassene“ DAC-Gruppen automatisch von GL4 erkannt und eingebunden werden können. Das setzt aber voraus, dass der entsprechende Parameter in GL4 gesetzt ist.

SNKMAN und ich haben dafür eine Gruppen-Variable ernannt, über die eindeutig erkannt wird, ob eine Gruppe vom DAC erzeugt wurde, und wenn ja, ob die Gruppe auch von DAC kontrolliert wird, oder ob sie freigestellt ist.

Erklärung:

Jede Gruppe die der DAC generiert, bekommt folgende Variable zugewiesen:

```
group setVariable ["DAC_Excluded", False]
```

Bei jeder Gruppe, die das DAC-System verlässt, wird die Variable neu gesetzt:

```
group setVariable ["DAC_Excluded", True]
```


DAC-Objekt-Generierung

Seit der Version 3.0 ist es möglich, in DAC-Zonen auch Objekte zu generieren. Das Generieren der Objekte erfolgt grundsätzlich nach dem Zufallsprinzip. Es gibt aber ein paar Möglichkeiten die Zufälligkeit zu steuern.

Die DAC-Zonen für die Objekt-Generierung können die gleichen Formen haben, die auch bei den DAC-Zonen für die Einheiten-Generierung verwendet werden: **Kreis, Ellipse, Rechteck und Polygon**.

Hinweis:

Das Generieren von Objekten erfolgt immer über eine eigene Zone. Es ist also nicht möglich, Objekte und Einheiten über eine gemeinsame Zone zu erzeugen.

Das Generieren von Objekten in einer Zone wird über einen eigenen Skriptaufruf gestartet. Im Prinzip ist diese Prozedur mit dem Generieren von Wegpunkten zu vergleichen, nur das zusätzlich Objekte generiert werden.

DAC kennt 2 Varianten der Objektgenerierung, welche man innerhalb des Skriptaufrufs anspricht. Damit DAC ein Objekt generieren kann, muss zuerst eine gültige Position innerhalb der Zone ermittelt werden, wo dann das Objekt platziert wird, und genau darin liegt der Unterschied der 2 Varianten:

Die Standard-Variante	:	Positionen werden nach den Kriterien der WP-Generierung geprüft.
Die Turbo-Variante	:	Keine Positionsüberprüfung. Jede zufällige Position wird benutzt.

Welche Variante Du letztendlich benutzt, musst Du entsprechend den Anforderungen entscheiden.

Zudem gibt es die Möglichkeit, die Funktion zum Generieren der Objekte in 2 Instanzen laufen zu lassen. Das bedeutet, dass DAC die Anzahl der zu generierenden Objekte aufteilt und dann die Prozedur zweimal parallel startet. Je nach Objekttyp und Anzahl der zu generierenden Objekte, führt diese Dual-Verarbeitung zu etwas Zeiteinsparung.

Die 2 nachfolgenden Bilder zeigen Dir noch mal die 2 möglichen Varianten (links = **Single** / rechts = **Dual**):



Hast Du die DAC-Marker aktiviert, wird Dir DAC für jedes generierte Objekt einen Marker zeichnen und zwar genau in der Größe des Objekts (bei sehr kleinen Objekten kann es sein, dass Du keine Marker mehr erkennst).

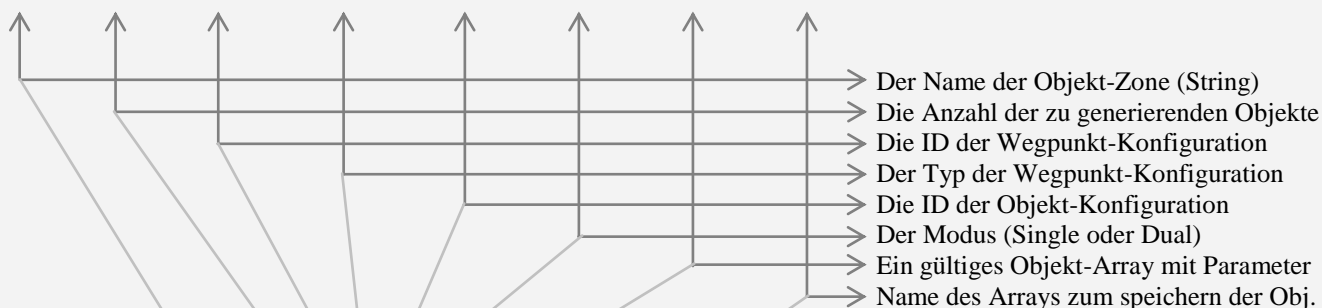
Standard-mäßig zeichnet DAC schwarze, rechteckige Marker für die generierten Objekte. Es ist aber möglich, diese Markereinstellungen für jeden Objekttyp separat zu konfigurieren (Form und Farbe).

Ein Zähler bzw. zwei Zähler (bei Dual Verarbeitung) zeigen Dir den Fortschritt der Generierung an. Sind die DAC-Systemmeldungen aktiviert, wird Dir zudem die Fertigstellung mit einen Sidechat quittiert:

```
1-1-A 1 (Silola): "ZONE '05' : ALL 250 OBJECTS WERE GENERATED."  
1-1-A 1 (Silola): "ZONE '05' : ALL 250 OBJECTS WERE GENERATED."  
1-1-A 1 (Silola): "ZONE '04' : ALL 500 OBJECTS WERE GENERATED."
```

Der Skriptaufruf, um Objekte in einer DAC-Zone zu generieren, sieht folgendermaßen aus:

```
nil = ["zone",count,wpconfig,wptype,objconfig,mode,[oArray],"sArray"] spawn DAC_Objects
```



Beispiel: nul = ["objZ1",300,0,0,5,0,[t1,t2,t3,25],"myTrucks"] spawn DAC_Objects

"zone" Wie unschwer zu erkennen, ist der erste Parameter der Name der Objekt-Zone. Es ist also notwendig, der Zone (Auslöser) einen Namen zu geben und genau diesen Namen im Skriptaufruf als „String“ einzutragen (das gleiche Prinzip wie bei den Einheiten-Zonen).

count Mit diesem Parameter wird die Anzahl der zu generierenden Objekte festgelegt. Bitte beachte, dass zu viele Objekte die Performance negativ beeinträchtigen können.

wpconfig Dieser Parameter wird immer in Verbindung mit dem Parameter **wptype** gesetzt. Ein Wert **größer 0** lädt eine gültige Konfiguration aus der **DAC_Config_Waypoints** und erfordert gleichzeitig einen Wert zwischen **0** und **4** bei dem Parameter **wptype**. Die Objektgenerierung wird dann nach den Kriterien der Wegpunktgenerierung erfolgen und entspricht der oben genannten **Standard-Variante**.

Werden beide Parameter auf **0** gesetzt, entspricht das der **Turbo-Variante**, da keinerlei Positionsprüfungen erfolgen und die Objekte ohne Verzögerung gesetzt werden.

wptype Dieser Parameter wird immer in Verbindung mit dem Parameter **wpconfig** gesetzt. Mögliche Werte sind **0** bis **4** und entsprechen der jeweiligen Einheiten-Kategorie.

Beispiele:	wpconfig	wptype	
	1	4	Standard-Variante - Positionen vom Typ Camp
	3	0	Standard-Variante - Positionen vom Typ Infanterie
	3	5	ungültig - wptype 5 nicht zulässig
	2	2	Standard-Variante - Positionen vom Typ Kettenfahrzeuge
	0	0	Turbo-Variante - keine Positionsüberprüfung
	0	1	ungültig - wptype 1 erfordert gültige wpconfig

Standard-Variante

Bei der Standard-Variante werden die Objekte nach den Kriterien der Wegpunktgenerierung platziert. Vegetation und Gelände werden berücksichtigt.



Turbo-Variante

Bei der Turbo-Variante werden die Objekte einfach zufällig platziert. Eine Positionsüberprüfung findet nicht statt. Gelände und Objekte werden nicht berücksichtigt.



objconfig Dieser Parameter lädt eine gültige Konfiguration aus der **DAC_Config_Objects**. Damit bestimmst Du, welche Objekte in der Zone generiert werden sollen. Wie Du eine solche Objekt-Konfiguration anlegst und welche weiteren Parameter Dir dort zu Verfügung stehen, erfährst Du unter dem Thema **DAC Config Objects**

mode Der Generierungs-Modus: **0** = **Single-Modus** / **1** = **Dual-Modus**

[oArray] Dieses Array kann dafür benutzt werden, bestimmte Objekte oder Positionen bei der Generierung zu berücksichtigen und zwar in Bezug auf einen Sicherheitsabstand. Mit anderen Worten: Du bestimmst, zu welchen Objekten bei der Generierung ein gewisser Sicherheitsabstand eingehalten werden soll.

Es gibt dabei mehrere Möglichkeiten das Array mit Daten zu füllen, wobei es folgende Regel gibt: **Der letzte Wert bestimmt immer den Sicherheitsabstand (numerisch).**

Die möglichen Varianten sind:

- | | | |
|----------------|---|--|
| Objekte | - | [obj1,obj2,obj3,obj4,25]
Du kannst die Objekte, durch ein Komma getrennt, direkt eingeben. Der letzte Parameter ist wie gesagt der Sicherheitsabstand (z.B. 25 m). Im oben genannten Beispiel bedeutet das, dass DAC zu den Objekten obj1 bis obj4 im Abstand von 25 Meter keine Objekte generiert. |
| Array | - | [Array,25]
Du kannst auch ein einzelnes Array angeben, in dem sich gültige Objekte befinden. Die Berücksichtigung dieser Objekte erfolgt dann nach dem gleichem Prinzip wie bei der Variante Objekte . |
| Logic | - | [Logic,25]
Möchtest Du, dass DAC Positionen anstatt Objekte berücksichtigt, kannst Du das über eine Logik-Einheit realisieren. Platziere dafür eine Logik auf der Map und gib dieser Logik einen Namen. Anschließend fügst Du dieser Logik beliebig viele Wegpunkte hinzu, an Positionen die Objektfrei bleiben sollen. Den Namen dieser Logik trägst Du dann in dem Array ein. |
| Zone | - | [Zone,25]
Die letzte Variante ermöglicht Dir Objekte zu berücksichtigen, die von einer anderen Objekt-Zone generiert wurden. Dazu ein kleines Beispiel: Du hast 2 Zonen. Die eine generiert ein paar Gebäude und die andere einige Bäume. Die Zonen liegen auf gleicher Position, Das bedeutet die Bäume werden zum Teil dort generiert, wo bereits Gebäude erstellt wurden. Um dieses Problem zu lösen, kannst Du diese Variante einsetzen. |

"sArray" Dieser letzte Parameter gibt Dir die Möglichkeit, die generierten Objekte in einem individuellen Array zu speichern. Der Parameter muss vom Typ „String“ sein und kann einen beliebigen Namen haben. DAC wird dann unter diesem Namen ein Array anlegen und die Objekte nach der Generierung darin speichern. Dieses Array steht dann zur weiteren Verarbeitung zu Verfügung.

Tip:

Benötigst Du nur zufällige Positionen in einer Zone, also ohne das DAC Objekte generiert, kannst Du das folgendermaßen machen: In dem Skriptaufruf gibst Du eine Objekt-Konfiguration an, in der nur eine Logik als Objekt zum generieren eingetragen ist. In dem Fall generiert DAC nur die Positionen und speichert diese in dem angegebenen Array **"sArray"** ab. Das Array kannst Du dann für eigene Zwecke benutzen z.B. um Missions-Marker zufällig zu platzieren.

Auf der nächsten Seite gibt's ein paar bebilderte Beispiele zum Thema Objektgenerierung:

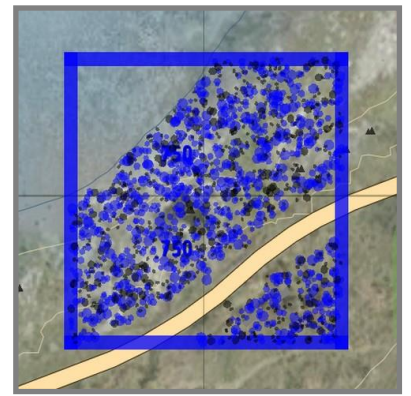
`["oz1",500,0,0,3,0,[],""]` spawn DAC_Objects

Das Bild zeigt Dir eine Zone, in der die Objekte mit der Turbo-Variante generiert wurden, also ohne Positionsüberprüfung.

Standard-mäßig wird DAC aber kritische Bereiche wie Wasser und Strassen bei der Objektgenerierung meiden (auf dem Bild gut zu sehen).

Dieses Verhalten kann aber jederzeit von Dir angepasst werden.

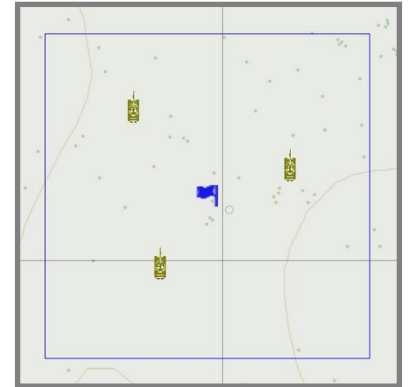
Die Parameter dafür findest Du in der Datei [DAC_Config_Objects](#).



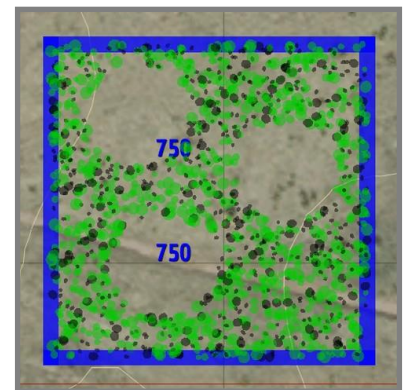
`["oz1",500,0,0,3,0,[t1,t2,t3,25],""]` spawn DAC_Objects

Hier das Beispiel mit 3 Editor-gesetzten Objekten (in dem Fall 3 Panzer) in deren Umkreis von 25m keine Objekte generiert werden sollen.

Die Panzer habe ich **t1**, **t2**, **t3** genannt und entsprechend im Skriptaufruf eingetragen. Der Sicherheitsabstand ist wie gesagt am Ende des Arrays eingetragen.



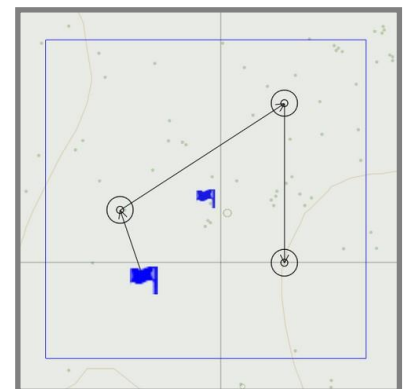
Das Ergebnis siehst Du rechts im Bild. Die Bereiche wo die Panzer platziert wurden, sind objektfrei.



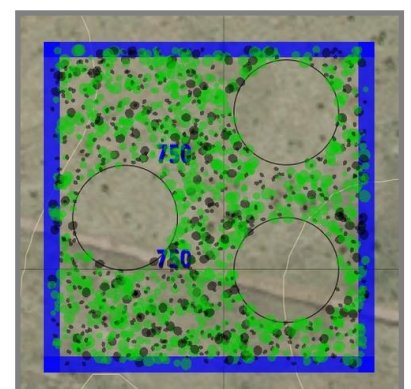
`["oz1",500,0,0,3,0,[Log1,25],""]` spawn DAC_Objects

Hier nun das gleiche Spiel, allerdings mit der Variante **Logik** umgesetzt. Wie auf dem Bild zu sehen, habe ich innerhalb der Zone eine Logik eingebaut und dieser Logik habe ich 3 Wegpunkte gegeben.

Dann nur den Namen der Logik (in dem Fall **Log1**) in dem Array eingetragen, und wieder den Sicherheitsabstand am Ende des Arrays ... das war's.



Wie Du siehst, ist das Ergebnis von der Sache her das gleiche: An den Positionen der Wegpunkte von der Logik, wurden keine Objekte generiert. Die Positionen bei dieser Variante werden automatisch durch einen Kreismarker kenntlich gemacht.



`["oz1",15,4,4,6,0,[],""]` spawn DAC_Objects

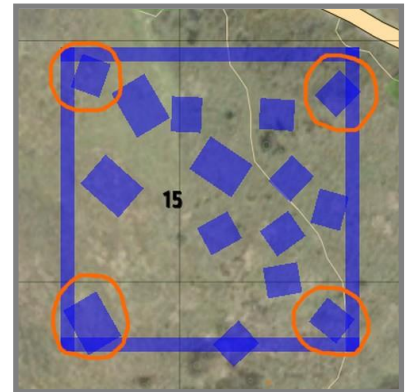
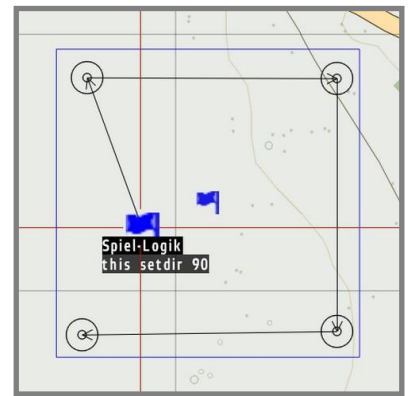
Natürlich gibt es auch bei der Objektgenerierung wieder die Möglichkeit, benutzerdefinierte Positionen einzubinden. Das Prinzip ist das gleiche wie bei den benutzerdefinierten Wegpunkten:

Platziere eine Logik innerhalb der Zone und gib' dieser Logik beliebig viele Wegpunkte, bei deren Positionen Objekte generiert werden sollen.

In der Init-Zeile der Logik trägst Du ein: **this setdir 90**

Durch diesen Eintrag weiß DAC genau, was er mit der Logik anstellen soll.

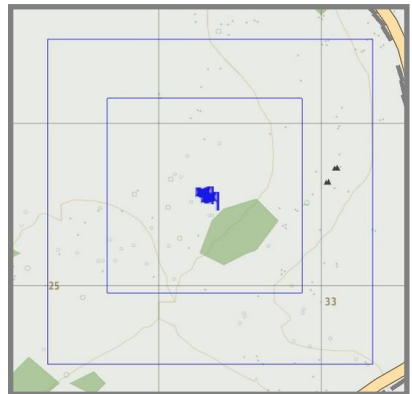
In diesem Beispiel habe ich 15 Gebäude generiert, wobei 4 Positionen durch die Wegpunkte der Logik fest vorgegeben sind (in dem Fall in den Ecken). Die restlichen Gebäude werden an zufälligen Positionen platziert.



Dieses letzte Beispiel verdeutlicht Dir die Variante **Zone**.

Damit ist es möglich, eine Objekt-Zone warten zu lassen, bis eine andere Zone alle Objekte generiert hat, um deren Objekte zu berücksichtigen.

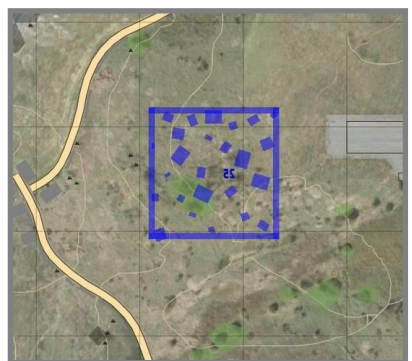
Ich verwende dafür folgendes Beispiel: Wir haben 2 Zonen. In der kleineren Zone sollen 25 Gebäude generiert werden. In der größeren Zone sollen 400 Bäume und Büsche generiert werden. Die Abbildung rechts zeigt Dir die Ausgangssituation.



Die Problematik, die sich daraus ergibt, siehst Du auf den folgenden Bildern rechts:

Die Gebäude alleine sind wunderbar generiert worden.

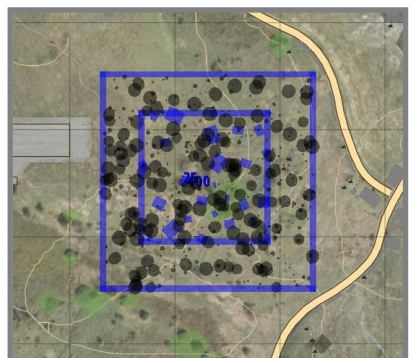
Durch die Wahl einer bestimmten Wegpunkt-Konfiguration, wurden die Gebäude alle in einem gewissen Abstand untereinander platziert, also sie berühren sich nicht.



Werden aber die Bäume und Büsche dazu generiert, endet das im Chaos, da die Bäume und Büsche planlos zum Teil in die Häuser generiert wurden.

Das liegt daran, dass die eine Zone nicht weiß, was die andere macht ;-)

Diesen Umstand können wir umgehen, in dem wir der Zone mit den Bäumen und Büschen mitteilen, dass sie warten soll, bis alle Gebäude generiert wurden und sie diese Gebäude anschließend mit einem gewissen Sicherheitsabstand berücksichtigt.



Betrachtest Du dieses Bild genauer, kannst Du feststellen, dass alle Bäume und Büsche einen bestimmten Sicherheitsabstand zu den Gebäuden einhalten (ich habe dafür 8m gewählt).

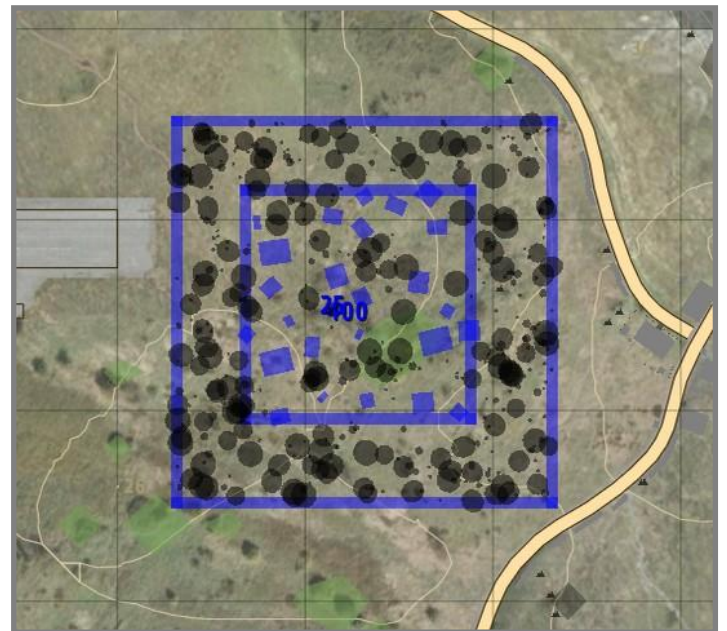
Eine Bedingung für eine solche Konstellation ist, dass die Gebäude in einem Array gespeichert werden, damit andere Zonen darauf zugreifen können.

Angenommen die Zonen heißen folgendermaßen:

U1 ist die Zone für die Gebäude

U2 ist die Zone mit der Vegetation

A3 ist das Array, in dem die Gebäude gespeichert werden sollen, dann sehen die Skriptaufrufe in etwa so aus:



```
["U1",25,4,4,6,0,[],"A3"] spawn DAC_Objects
```

U1 generiert 25 Gebäude und speichert diese Gebäude in dem Array **A3**

```
["U2",400,0,0,3,0,[U1,8],""] spawn DAC_Objects
```

U2 wird 400 Bäume und Büsche generieren, muss aber warten, bis **U1** seine Gebäude generiert hat. Die Bäume und Büsche werden dann in einem Sicherheitsabstand von 8 m zu den Gebäuden von **U1** generiert.

Sollte es mal dazu kommen, dass DAC nicht alle Objekte einer Zone generieren kann, wird Dir DAC dieses Problem auch per Sidechat anzeigen, vorausgesetzt die DAC-Systemmeldungen sind aktiviert.

```
1-1-A 1 (Silola): "ZONE '04' : ONLY 74 OF 150 OBJECTS COULD BE GENERATED."
```


Die Konfigurations-Dateien

Die Konfigurations-Dateien sind sozusagen das Daten-Pool für den DAC. Eine Sonderstellung dabei nimmt die Datei **DAC_Config_Creator** ein, denn dort sind alle relevanten globalen Variablen und Grundparameter abgelegt, die Du an Deine Bedürfnisse anpassen kannst.

Dann gibt es noch die **DAC_Config_Sound**, in der die Soundfiles eingetragen sind, die DAC in bestimmten Situationen abspielt. Diese Config-Datei ist nur eingeschränkt erweiterbar.

Alle anderen DAC_Config-Dateien sind von Dir beliebig erweiterbar. Der Aufbau und der Aufruf dieser Konfigurations-Dateien erfolgt dabei immer nach dem gleichen Schema.

Innerhalb dieser Config-Dateien gibt es so genannte „Daten-Blöcke“, die jeweils durch eine eindeutige ID getrennt sind. Soll DAC einen bestimmten Daten-Block verwenden, wird nur die entsprechende ID genannt und schon weiß DAC, welche Daten er laden muss. Wie und wo das genau passiert, erfährst Du in der Beschreibung der jeweiligen Konfigurations-Datei.

Um den DAC zu starten bzw. zu initialisieren, musst Du eine der 2 DAC-Logiken in Deiner Mission platzieren. Es gibt einmal die DAC-Logik „**DAC_intern**“ und zudem noch die DAC-Logik „**DAC_extern**“.

Die DAC-Logik „**DAC_intern**“ verarbeitet die internen Konfigurations-Dateien, welche sich innerhalb der PBO befinden. Dort enthalten sind z.B. alle Standard-Einheiten und auch eine Standard-Verhaltens-Konfiguration u.s.w. Also alles was Du brauchst um z.B. eine Test-DAC-Umgebung zu initialisieren.

Die „**DAC_intern**“ bietet Dir zudem die Möglichkeit, einzelne Konfigurations-Dateien in Dein Missions-Verzeichnis auszulagern, um daran Änderungen vorzunehmen, oder die Daten darin durch eigene Daten zu erweitern. Um einzelne Dateien auszulagern, schreibst Du in die Init-Zeile der DAC-Logik:

```
DAC_Single_Config = ["Units"]
```

In dem Fall bedeutet das, dass DAC die Datei „DAC_Config_Units.sqf“ in Deinem Missionsverzeichnis erwartet. Möchtest Du mehrere Dateien auslagern, musst Du nur weitere Daten angeben:

```
DAC_Single_Config = ["Units","Behaviour","Arti"]
```

DAC erwartet in diesem Fall, dass die Dateien DAC_Config_Units.sqf, DAC_Config_Behaviour.sqf und DAC_Config_Arti.sqf in Deinem Missionsverzeichnis zu finden sind.

Bei der Logik „**DAC_extern**“ sieht das etwas anders aus. DAC erwartet dann einen „DAC“ Ordner in Deinem Missionsverzeichnis, in dem alle Konfigurations-Dateien enthalten sind. Diesen Ordner findest Du übrigens in dem Zip-File. Diese Dateien können dann von Dir nach Belieben geändert oder erweitert werden.

Folgende Konfigurations-Dateien gibt es:

DAC_Config_Creator	Die Grundeinstellungen des DAC
DAC_Config_Units	Die Einheiten-Konfiguration - definiert die Einheitentypen der 4 Kategorien je Seite
DAC_Config_Behaviour	Die Verhaltens-Konfiguration - definiert das Verhalten der DAC-Gruppen
DAC_Config_Waypoints	Die Wegpunkt-Konfiguration - definiert die Parameter für den Wegpunktgenerator
DAC_Config_Camp	Die Camp-Konfiguration - definiert das Aussehen und die Einstellungen der DAC-Camps
DAC_Config_Arti	Die Arti-Konfiguration - definiert die Bedingungen und die Einstellungen der DAC-Arti
DAC_Config_Events	Die Ereignis-Konfiguration - erlaubt den Eingriff an verschiedenen Stellen im DAC
DAC_Config_Objects	Die Objekt-Konfiguration - definiert die Objekte für den Objektgenerator
DAC_Config_Weapons	Die Waffen-Konfiguration – erlaubt die Definition eigener Waffenzusammenstellungen
DAC_Config_Sound	Die Sound-Konfiguration – definiert Sprach- und Funksounds innerhalb des DAC
DAC_Config_Marker	Die Marker-Konfiguration – definiert das Aussehen und Verhalten der DAC-Marker

In der **DAC_Config_Creator.sqf** werden die Grundeinstellungen des DAC vorgenommen. Außerdem kannst Du dort eigene Skripte starten und bestimmen, ob die Skripte nur auf dem Server und/oder auf einem Client gestartet werden. Hier die Beschreibung der einzelnen Variablen und Arrays:

DAC_STRPlayers = ["s1","s2","s3","s4","s5","s6","s7","s8","s9","s10".....]

Das Array in dem die Namen der möglichen Spieler eingetragen werden. **Achtung, die Einträge müssen als String vorliegen.** Sobald die Mission gestartet wird, reduziert DAC dieses Array automatisch auf die gültigen Einträge.

DAC_AI_Count_Level = [[2,4],[2,6],[2,8],[2,12],[6,0]]

Hier kannst Du die Gruppengrößen definieren. Der erste Wert bestimmt die Mindestgröße, der zweite Wert bestimmt die Maximalgröße einer Gruppe. Die Gruppengröße wird dann im DAC-Skriptaufruf abgerufen:

["z1",[1,0,0],[5,2,50,6],[],[],[],[0,0,0,0]] spawn DAC_Zone

↑
→ Die **2** entspricht dem 2.Eintrag im Array **DAC_AI_Count_Level**, also 2-6 Einheiten

DAC_Dyn_Weather = [0,0,0,0] **!!! Bitte nur im SinglePlayer-Modus benutzen !!!**

Dieser Parameter steuert das dynamische Wettersystem im DAC. Möchtest Du dieses Merkmal nicht benutzen, musst Du nur den **1.** Wert auf **0** stellen.

1. Das Zeitfenster bis ein Wetterwechsel eintritt
2. Das Zeitfenster wie lange ein Wetterwechsel andauert
3. Der maximale Wetterwert
4. Der maximale Nebelwert

DAC_Reduce_Value = [600,650,0.1]

Hier sind die Parameter für die Einheiten-Reduzierung eingetragen. Möchtest Du dieses Merkmal nicht benutzen, musst Du nur den **3.** Wert auf **0** stellen.

1. Der Distanzwert, wann eine reduzierte Gruppe wieder aufgebaut wird
2. Der Distanzwert, wann eine Gruppe reduziert wird (dieser Wert sollte etwas größer sein als der erste).
3. Die Pause die DAC zwischen jeder Einheit einlegt, wenn eine Gruppe wieder aufgebaut wird.

DAC_AI_Spawn = [[10,30,10],[20,60,15],0,360,100,0]

Hier wird die Grundeinstellung und das Zeitverhalten vom KI-Respawn eingestellt. Grundsätzlich muss mindestens ein DAC-Camp generiert werden, damit der KI-Respawn überhaupt funktioniert.

1. Respawnzeit für **Infanterie** = Sek. mindestens + Sek. zufällig + Sperrzeit, bis Camp wieder freigegeben wird
2. Respawnzeit für **Fahrzeuge** = Sek. mindestens + Sek. zufällig + Sperrzeit, bis Camp wieder freigegeben wird
3. Respawn global = 0, Respawn lokal = 1
4. Zerstörungs-Option für Respawn-Camps, wenn alle Respawns aufgebraucht sind:
0 = Die Camp-Objekte bleibt erhalten, >0 = Sekunden bis Selbstzerstörung einsetzt
5. Mindestabstand aller Spielereinheiten zum Camp, damit ein Respawn in dem Camp ausgeführt werden kann.
6. Verhalten von Respawn-Camps, wenn alle Respawns aufgebraucht sind (z.B. um die Arti aufrecht zu erhalten).
0 = Die Camp-Gruppe gibt das Camp auf, 1 = Die Camp-Gruppe wird das Camp bis zum Ende bewachen

DAC_Delete_Value = [[120,150],[140,150],300]

Dieses Array bestimmt das Löschen von toten Einheiten bzw. von zerstörten Fahrzeugen. Du kannst die Bedingung dafür jeweils auf einen Parameter begrenzen, oder beide Parameter kombinieren:

[60,0] = nur auf Zeit, [0,200] = nur auf Distanz, [30,150] = erst Zeit dann Distanz

1. Zeit in Sek. + Distanz in Meter (zu Spielereinheiten), wann eine tote Einheit frühestens gelöscht werden darf
2. Zeit in Sek. + Distanz in Meter (zu Spielereinheiten), wann ein zerstörtes Fahrzeug frühestens gelöscht werden darf
3. Zeit in Sek. bis ein leeres Fahrzeug ohne Zugehörigkeit zerstört wird (damit es später gelöscht werden kann)

DAC_Del_PlayerBody = [10,150]

Hier kann man bei Bedarf die Löschfunktion für Spieler (im MP) aktivieren. Je nach MP-Respawn der benutzt wird, kann es passieren, dass übermäßig viele (Spieler-) Leichen auf dem Schlachtfeld liegen. Diese können dann mit dieser Funktion gelöscht werden.

1. Zeit in Sekunden, wann eine tote Spielereinheit frühestens gelöscht werden darf
2. Mindestabstand zu allen Spielereinheiten, bevor eine tote Spielereinheit gelöscht wird

DAC_Com_Values = [1,2,0,0]

Die Werte in diesem Array bestimmen, wie bzw. ob DAC Dir bestimmte Meldungen anzeigt.

Die DAC Systemmeldungen zeigen Dir z.B. wann eine Gruppe reduziert wurde, eine Einheit gelöscht wurde, eine Zone versetzt wurde, eine Gruppe respawned wurde u.s.w.

Der DAC Initialisierungs-Hint zeigt Dir den Fortschritt während der Initialisierung von DAC an, wie Wegpunkte und Einheiten generiert werden und wie viel Zeit DAC dafür benötigt hat.

Die DAC Funkmeldungen zeigen Dir Aktionen/Reaktionen der KI an, wie Gruppen Feindkontakt melden, Verstärkung anfordern, den Feindkontakt wieder verlieren u.s.w.

1. DAC Systemmeldungen: 0 = deaktiviert, 1 = aktiviert
2. DAC Initialisierungs-Hint: 0 = deaktiviert, 1 = minimal, 2 = maximal
3. DAC Funkmeldungen: 0 = Meldungen deaktiviert, 1 = feindlich aktiviert, 2 = freundlich aktiviert, 3 = alle aktiviert
4. DAC Monitor: 0 = Monitor deaktiviert, 1 = Monitor für alle aktiviert, „Name“ = Nur „Name“ hat Monitor

DAC_AI_AddOn = 1

Zur Zeit ist nur diese Einstellung möglich. Die AI lässt sich auch nicht mehr deaktivieren.

Folgendes ist aber möglich: Mach eine Kopie von dem Ordner **AI_1** und benenne ihn um in **AI_2**.

Es wird dann dieser Ordner benutzt, wenn Du **DAC_AI_AddOn** = 2 einstellst.

Das macht Sinn, wenn Du Änderungen an den AI-Scripten vornehmen möchtest, die original Scripte aber nicht verändern willst. Auf diese Weise kannst Du bequem zwischen mehreren AI-Ordern wechseln.

DAC_AI_Level = 4

Werte von 1,2,3 und 4 sind möglich, wobei der Wert 4 die stärkste Einstellung darstellt.

Dieser Wert beeinflusst das Verhalten der KI. Je kleiner der Wert ist, umso träger reagiert die KI und sie bemerkt dich nicht so schnell. Auch die maximale Anzahl von Verstärkungseinheiten wird durch diesen Wert beeinflusst.

DAC_Res_Side = 1

Diese Variable legt fest, auch welcher Seite die **Independents** kämpfen. Diese Einstellung musst Du unbedingt analog zu den Einstellungen im Editor vornehmen.

0 = Independent kämpft auf Seite Ost, **Editor**: Independent freundlich zu OSTEN einstellen

1 = Independent kämpft auf Seite West, **Editor**: Independent freundlich zu WESTEN einstellen

2 = Independent kämpft auf eigener Seite, **Editor** Independent freundlich zu NIEMANDEM einstellen

Achtung, bei dieser Einstellung kämpfen die Independent-Einheiten alleine gegen OST + WEST, wobei OST und WEST befreudet ist !!!

DAC_VehAllowed = [1,2]

Hier werden die Fahrzeugtypen bestimmt, die von Infanterie-Einheiten eingenommen werden können, wenn sie leer sind. Trage hier beliebig viele Config-Nr. aus der **DAC_Config_Units** ein.

DAC liest alle Fahrzeuge aus den angegebenen Konfigurationen aus und fasst diese zu einem Pool zusammen. Infanterie-Einheiten werden dann nur die leeren Fahrzeuge "erkennen", die in dem Pool vorhanden sind.

DAC_Marker = 1

Mit diesem Parameter werden die DAC-Marker aktiviert / deaktiviert.

Du kannst jederzeit eine andere Marker-Konfiguration laden, indem Du die Marker kurzfristig deaktivierst, und sie anschließend mit einer neuen Marker-Konfiguration wieder aktivierst.

Info: Im MP werden nur Einheiten- und Zonen-Marker angezeigt.

0 = DAC-Marker deaktiviert, **>0** = gültige Konfigurations-Nr. aus der **DAC_Config_Marker**.

Achtung, zwischen einem Deaktivieren, und einem Aktivieren der Marker, müssen mindestens 3 Sekunden Pause liegen.

DAC_WP_Speed = 0.01

Die Geschwindigkeit, mit denen die Wegpunkte generiert werden. Dieser Wert betrifft nur die Initialisierungs-Phase.

Achtung, der Wert 0 führt zu einem kurzen Aussetzer während der Initialisierung !

Beim Versetzen einer Zone wird immer der Standard-Wert 0.01 benutzt.

DAC_Join_Action = false

Diese Einstellung ist zur Zeit nur ein Versuchsparameter und zudem nur im Singleplayer-Modus verfügbar.

Ist diese Einstellung aktiv (true), dann bekommst Du, wenn Du Dich einer freundlichen DAC-Einheit nähert, ein Action-Menue angezeigt, mit dem Du diese Einheit in Deine Gruppe beitreten lassen kannst.

DAC_Fast_Init = false

Diese Einstellung ist nur im Singleplayer-Modus verfügbar.

Ist diese Einstellung aktiv (true), dann wird während der Initialisierungs-Phase von DAC, zum Einen die Sichtweite auf Minimal-Wert reduziert, und zudem der Nebel auf Maximal-Wert erhöht.

Diese Maßnahme führt dazu, dass zusätzliche Prozessorleistung freigegeben wird und so der Initialisierung-Phase von DAC zu Gute kommt. Die Zeit der Initialisierung kann dadurch um bis zu 40% verkürzt werden.

Ist die Initialisierung von DAC abgeschlossen, werden die alten Werte für Sichtweite und Nebel wieder hergestellt.

DAC_Player_Marker = false

Diese Einstellung aktiviert für jeden Spieler einen Marker sowohl im Singleplayer- als auch im Multiplayer-Modus.

DAC_Direct_Start = false

Standard-mäßig erwartet DAC mindestens eine DAC-Zone, um sich initialisieren zu können. Wenn Du aber diese Einstellung aktivierst (true), kannst Du diese Bedingung umgehen und DAC initialisiert sich nur anhand der DAC-Logik.

Diese Einstellung benötigst Du, wenn Du DAC-Zonen erst während der Mission erzeugen willst, oder auch wenn Du nur Objekte generieren möchtest.

DAC_Activate_Sound = false

Dieser Parameter aktiviert / deaktiviert die DAC-Sounds, die in bestimmten Situationen abgespielt werden.

DAC_Auto_UnitCount = [8,10]

Diese Einstellung kannst Du dafür benutzen, die Anzahl der DAC-Gruppen dynamisch an die Anzahl der Spieler-Einheiten anzupassen. Mit anderen Worten: Je mehr menschliche Spieler an einer Mission teilnehmen (COOP), umso mehr DAC-Gruppen werden generiert und umgekehrt.

1. Die Anzahl der Spieler, für die die Mission ausgelegt ist
2. Prozentsatz, um den sich die DAC-Gruppenanzahl erhöht bzw. verringert, für jeden Spieler der mehr oder weniger an der Mission teilnimmt.

Das passiert nicht automatisch, vielmehr kannst Du die Zonen bzw. Einheitentypen exakt bestimmen, welche von dieser Automatik betroffen sein sollen.

Angenommen Du hast eine DAC-Zone mit folgendem Skriptaufruf:

```
["z1",[1,0,0],[18,2,75,10],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

Das bedeutet, dass bei jedem Missionsstart exakt 18 Gruppen Infanterie generiert werden, egal wie viel Spieler an der Mission teilnehmen.

Um diese 18 DAC-Gruppen nun dynamisch zu reduzieren bzw. aufzustocken, je nach dem ob mehr oder weniger Spieler an der Mission teilnehmen, musst Du diese Anzahl einfach in Anführungszeichen setzen:

```
["z1",[1,0,0],[ "18",2,75,10],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

Du kannst diese Einstellung in jeder Zone separat für jede Einheiten-Kategorie vornehmen

Ein paar kleine Rechenbeispiele, wie sich das mit den 18 Gruppen verhält, ausgehend von der o.g. Einstellung [8,10]

Nehmen **8** Spieler an der Mission teil, wird DAC **18** Gruppen generieren (als Standard definiert)

Nehmen **6** Spieler an der Mission teil, wird DAC **14** Gruppen ($18 - (2 \times 10\%)$) generieren.

Nehmen **4** Spieler an der Mission teil, wird DAC **10** Gruppen ($18 - (4 \times 10\%)$) generieren.

Nehmen **9** Spieler an der Mission teil, wird DAC **20** Gruppen ($18 + (1 \times 10\%)$) generieren.

Nehmen **11** Spieler an der Mission teil, wird DAC **23** Gruppen ($18 + (3 \times 10\%)$) generieren.

DAC_Player_Support = [10,[10,2000,3,1000]]

Diese Einstellung aktiviert und konfiguriert die Boden- und Artilleryunterstützung für den Spieler.

Um dieses Merkmal nutzen zu können, muß eine Logik-Einheit mit dem Namen „**DAC_Support_Logic**“ vorhanden sein.

1. Maximale Anzahl Artillery-Anforderungen.
2. Maximale Anzahl Bodenunterstützungs-Anforderungen.
3. Maximale Reichweite, in der ein Spieler Bodenunterstützung anfordern kann (ausgehend von der Pos. des Spielers).
4. Maximale Anzahl von Unterstützungsgruppen, die gleichzeitig Bodenunterstützung leisten.
5. Maximale Reichweite, in der nach Unterstützungsgruppen gesucht wird.

Details zu der jeweiligen Unterstützungsform findest Du auf den Seiten:

23 = Die DAC-Artillery

24 = die DAC-Bodenunterstützung

DAC_SaveDistance = [500,["DAC_Save_Pos"]

Hier kannst Du Bereiche auf der Karte festlegen, wo keine DAC-Wegpunkte generiert werden sollen und somit auch keine Einheiten in diesen Bereichen erzeugt werden.

Angenommen Du möchtest eine Mission inmitten einer großen DAC-Zone starten. Um nun sicherzustellen, dass in Deiner näheren Umgebung keine DAC-Einheiten generiert werden, kannst Du diese Einstellung verwenden.

1. Die Distanz zu den angegebenen Objekten, wo keine DAC-Wegpunkte generiert werden sollen
2. Die Liste der Objekte (als String), die bei der Distanz-Messung berücksichtigt werden sollen (z.B. eine Logik)

Es sind auch mehrere Objekte möglich: **DAC_SaveDistance** = [500,["savePos1","savePos2","savePos3"]

DAC_GunNotAllowed = ["B_Mortar_01_F","O_Mortar_01_F"]

DAC_VehNotAllowed = ["O_MBT_02_arty_F","I_APC_Wheeled_03_cannon_F"]

Hier müssen die Typen der statischen Waffen bzw. der Fahrzeuge eingetragen werden, die (wenn sie leer sind) von der KI nicht benutzt werden dürfen.

DAC_Locked_Veh = [player_car_1,player_car_2]

Hier kannst Du Editor gesetzte Fahrzeuge eintragen, die von der KI nicht benutzt werden dürfen.

DAC_BadBuildings = ["CampEmpty","CampEast","Land_dum_istan4"]

Hier können Gebäudetypen eingetragen werden, die sich entweder nicht dazu eignen, von der KI besetzt zu werden, oder die Du Missions bedingt ausschließen möchtest.

DAC_SP_Soldiers = ["B_soldier_AR_F","B_G_soldier_AR_F","O_soldier_AR_F"];

Hier werden die Einheitentypen festgelegt, die Unterdrückungsfeuer geben können.

Die meisten Einstellungen der **DAC_Config_Creator** können auch direkt über eine Logic vorgenommen werden.

Also einfach eine Logic auf die Karte platzieren und in die Init-Zeile dieser Logic schreibst Du z.B. rein:

DAC_AI_Level = 2 (Der Wert im Skript **DAC_Config_Creator** wird dabei beschrieben).

[DAC_Config_Units]

In der **DAC_Config_Units** sind die Einheiten-Klassen definiert, die Du mit DAC generieren kannst. Diese Klassen sind in "Blöcke" aufgeteilt und jeder Block besitzt eine eindeutige Nummer.

Innerhalb von so einem Block sind dann die 4 Einheiten-Kategorien aufgeführt, die dann durch den DAC-Scriptaufruf angesprochen werden. Dieses Script lässt sich beliebig erweitern, so dass Du auch Deine „Lieblings-Einheiten“ dort ablegen kannst.

Du kannst Einheiten aus verschiedenen Addons kombinieren, und so individuelle Zusammenstellungen von Einheiten speichern. Bitte achte aber darauf, dass Du die entsprechenden Addons auch geladen hast.

So ein „Block“ sieht folgendermaßen aus...

```
case 0:
{
    _Unit_Pool_S = [
        "RU_Soldier_Crew", "RU_Soldier_Pilot", "RU_Soldier_SL", "RU_Soldier", "RU_Soldier_GL",
        "RU_Soldier_MG", "RU_Soldier2", "RU_Soldier_Medic", "RU_Soldier_AT",
        "RU_Soldier_Sniper", "RU_Soldier_AR", "RU_Soldier_Marksman",
        "RUS_Soldier_Sab", "RUS_Soldier_Marksman"
    ];
    _Unit_Pool_V = [ "UAZMG", "UralOpen_INS", "UAZ_RU", "UAZ_AGS30_RU", "BRDM2_INS", "UAZ_MG_INS" ];
    _Unit_Pool_T = [ "T72_RU", "ZSU_INS", "BMP3", "2S6M_Tunguska", "T90", "BMP3", "BTR90" ];
    _Unit_Pool_A = [ "Mi17_rockets_RU", "Ka52", "Mi24_V", "Mi24_P" ];
};
```

Figure based on Arma2

Hier ein DAC-Skriptaufruf, der Einheiten aus der Einheiten-Konfiguration **0** generiert:

```
["z1",[1,0,0],[5,3,30,8],[ ],[ ],[ ],[0,0,3,4]] spawn DAC_Zone
```

Die Verschiedenen Pools haben folgende Bedeutung:

_Unit_Pool_S	=	Die Einheitenklassen für die Kategorie (1) Infanterie
_Unit_Pool_V	=	Die Einheitenklassen für die Kategorie (2) Radfahrzeuge
_Unit_Pool_T	=	Die Einheitenklassen für die Kategorie (3) Kettenfahrzeuge
_Unit_Pool_A	=	Die Einheitenklassen für die Kategorie (4) Helikopter

Achtung, im **_Unit_Pool_S** gibt es eine Bedingung, die immer erfüllt sein muss, auch wenn keine Infanterie-Einheiten erzeugt werden: **Es müssen dort immer mindestens 3 Einträge vorhanden sein!**

Das hat folgenden Grund: Diese ersten 3 Einträge sind für DAC reserviert!

Der 1. Eintrag bestimmt den Einheitentyp für die **Crew** von **Kettenfahrzeugen**.

Der 2. Eintrag bestimmt den Einheitentyp für die **Piloten** von **Helikoptern**.

Der 3. Eintrag bestimmt den Einheitentyp für den **Leader** einer **Infanterie**-Gruppe.

Der Rest der Gruppe wird per Zufall generiert (dafür nimmt DAC alles, was nach dem 3. Eintrag kommt)

Tipp: Du kannst Einheiten-Klassen auch mehrfach innerhalb eines Pools eintragen (wie oben zu sehen). Das erhöht die Wahrscheinlichkeit, dass dieser Einheitentyp erzeugt wird.

[DAC_Config_Behaviour]

In diesem Konfigurations-Script lassen sich verschiedene Grundverhaltensarten abspeichern, die dann auch wieder

für jede Zone separat geladen werden können. Mit „Grundverhalten“ ist gemeint, wie sich die generierten Einheiten einer Zone verhalten, wenn sie Ihre Wegpunkte abarbeiten, ohne Feindkontakt zu haben.

Hier mal ein „Block“ aus diesem Script...

```
case 1:
{
    _setSkill      = [0.2,0.7];
    _setCombat     = ["green","white","yellow"];
    _setBehav      = ["careless","safe","aware"];
    _setSpeed      = ["limited","normal","full"];
    _setForm       = ["line","vee","column","wedge","stag column","ech left","ech right","file","diamond"];
    _setFleeing    = [0,100];
    _setHeliVal    = [45,100,0.7,1];
    _setPause      = [[5,10],[5,10],[5,10],[20,30,5,5],[1,3],[0,0]];
    _setBldgBeh    = [0,50,120,600,1];
    _setPatrol     = ["45 + (20 * (skill _leader))","(60 + (random 60)) + ((skill _leader) * 50)"];
    _setSearch     = ["40 + ((skill _leader) * 150)","50 + ((skill _leader) * 50)"];
    _setSupport    = [1,2];
    _setJoin       = 2;
    _setEmpVeh     = [[0,100],[0,100]];
    _setSupTime    = ["5 + ((skill _unit) * (5 * DAC_AI_Level))","2,5"];
    _setHidTime    = ["((10 * DAC_AI_Level) + ((skill _leader) * 50)) / ((count units _group) + 1)"];
};
```

Hier ein DAC-Skriptaufruf, der die Einheiten aus der Zone **z1** mit der **Verhaltens-Konfiguration 1** füttert :

```
["z1",[1,0,0],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Die Einstellungen verhalten sich folgendermaßen:

_setSkill	<p>[0.3,0.9] oder alternativ [[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8]]</p> <p>Dieses Array bestimmt den Bereich des Skill-Wertes, mit denen die Einheiten in der Zone generiert werden.</p> <ol style="list-style-type: none">1. Eintrag = der Minimal-Skill, den die Einheiten in dieser Zone bekommen.2. Eintrag = der Maximal-Skill, den die Einheiten in dieser Zone bekommen. <p>Bei der alternativen Variante können die Min/Max Werte für jeden Sub-Skill eingegeben werden.</p>
_setCombat	<p>["white","yellow"]</p> <p>Dieses Array kann mit den möglichen Combatmode-Werten gefüllt werden und benötigt mindestens einen Eintrag. Die Einheiten wechseln dann an jedem ihrer Wegpunkte per Zufall diesen Combatmode. Ist nur ein Eintrag vorhanden, wird nur dieser eine Wert benutzt.</p>
_setBehav	<p>["aware","combat"]</p> <p>Dieses Array kann mit den möglichen Behaviour-Werten gefüllt werden und benötigt mindestens einen Eintrag. Die Einheiten wechseln dann an jedem ihrer Wegpunkte per Zufall ihr Behaviour. Ist nur ein Eintrag vorhanden, wird nur dieser eine Wert benutzt</p>
_setSpeed	<p>["normal","full"]</p> <p>Dieses Array kann mit den möglichen Speed-Werten gefüllt werden und benötigt mindestens einen Eintrag. Die Einheiten wechseln dann an jedem ihrer Wegpunkte per Zufall ihren Speed. Ist nur ein Eintrag vorhanden, wird nur dieser eine Wert benutzt.</p>

_setForm	<p>["line","vee","column"]</p> <p>Dieses Array kann mit den möglichen Formations-Werten gefüllt werden und benötigt mindestens einen Eintrag. Die Einheiten wechseln dann an jedem ihrer Wegpunkte per Zufall ihre Formation. Ist nur ein Eintrag vorhanden, wird nur dieser eine Wert benutzt.</p>
_setFleeing	<p>[0,100]</p> <p>Diese Parameter bestimmen das Fluchtverhalten und die Deckungssuche der Einheiten.</p> <ol style="list-style-type: none"> 1. Eintrag = Fluchtverhalten. Der Wert muss zwischen 0 und 1 liegen. 2. Eintrag = Dieser Wert bestimmt, bei welchem Abstand zum Anführer, eine Einheit ein stationäres Geschütz wieder verlässt, um sich der Gruppe wieder anzuschließen. Sollen z.B. die Geschütze grundsätzlich besetzt bleiben, müsste hier ein großer Wert rein.
_setHeliVal	<p>[45,100,0.7,0]</p> <p>Dieses Array regelt die Flughöhe, sowie das Startverhalten von Heli-Einheiten je nach Wetter. Außerdem ist es möglich, dass bei jeder Heli-Startposition ein EmptyH generiert wird. Diese Maßnahme führt dazu, dass die Helis immer Punkt genau landen. Ansonsten suchen sich die Helis eigenständig eine geeignete Landeposition, meistens in der Nähe ihrer Startposition.</p> <ol style="list-style-type: none"> 1. Eintrag = Minimalflughöhe für Helis in der Zone 2. Eintrag = Maximalflughöhe für Helis in der Zone 3. Eintrag = Maximaler Wetterwert, bei dem Helis noch fliegen, ansonsten landen sie vorzeitig 4. Eintrag = EmptyH generieren AUS(0) / EmptyH generieren AN(1)
_setPause	<p>[[3,5],[3,5],[5,10],[20,30,5,5],[3,5],[1,30]]</p> <p>In diesem Array sind die Pausen für die verschiedenen Einheiten-Kategorien definiert, die sie jeweils an ihren Wegpunkten einlegen.</p> <ol style="list-style-type: none"> 1. Array = Wegpunktpausen für Infanterie [mindestens + zufällig] 2. Array = Wegpunktpausen für Radfahrzeuge [mindestens + zufällig] 3. Array = Wegpunktpausen für Kettenfahrzeuge [mindestens + zufällig] 4. Array = Wegpunktpausen für Helis und (wenn verfügbar) für die Cargo-Gruppe [mindestens + zufällig] für Helis am Boden [zufällig] für Helis in der Luft [zufällig] für die Cargo-Gruppe (wenn auf Patrouille) 5. Array = Wegpunktpausen für die Camp-Gruppe [mindestens + zufällig] 6. Array = Startverzögerung für alle Einheiten aus der Zone [mindestens + zufällig] Möchtest Du, dass sich die Einheiten aus der Zone nicht direkt zu ihren Wegpunkten bewegen, kannst Du hier eine Verzögerungszeit einstellen. Zum Beispiel bedeutet [1,30], dass jede Gruppe aus der Zone mit einer Verzögerung von 1 – 31 Sekunden startet. [10,0] würde bedeuten, dass jede Gruppe nach genau 10 Sekunden auf die Reise geht.
_setBldgBeh	<p>[2,50,120,600,1]</p> <p>Hier wird festgelegt, in wie weit Infanterie-Gruppen Gebäude besetzen können. Möchtest Du dieses Merkmal nicht benutzen, musst Du den 1. Wert auf 0 stellen.</p> <ol style="list-style-type: none"> 1. Eintrag = Maximale Anzahl Einheiten aus der Gruppe, die ein Gebäude besetzen 2. Eintrag = Radius der Erkennung von Gebäuden 3. Eintrag = Zeitfenster wie lange die Einheiten sich in einem Gebäude aufhalten 4. Eintrag = Zeitfenster bis ein Gebäude ein weiteres Mal besetzt werden darf 5. Eintrag = Die Anzahl der Positionen, die ein Gebäude mindestens haben muss <p>Oben das Beispiel würde also bedeuten, dass 2 Einheiten aus einer Gruppe ein Gebäude besetzen, wenn ein geeignetes Gebäude mit mindestens einer Position in einem Radius von 50m gefunden wurde. Die Einheiten halten sich 120 Sek. darin auf, und das Gebäude wird anschließend 600 Sek. nicht wieder besetzt. Achtung, es gibt nicht viele Gebäude/Objekte, bei denen das vernünftig klappt. Gebäude, die Probleme bereiten, können ausgeschlossen werden (siehe Thema Tips+Tricks).</p>

_setPatrol	<p>["50","60"]</p> <p>Die Einheiten der Kategorie Radfahrzeuge, steigen bei ihren Wegpunkten aus, um dann für eine gewisse Zeit und in einem bestimmten Bereich, auf Patroullie zu gehen. Hier kannst Du dafür den Radius festlegen und das Zeitfenster bestimmen.</p> <ol style="list-style-type: none"> 1. Eintrag = Maximaler Radius für die Patroullie, ausgehend vom Fahrzeug des Anführers 2. Eintrag = Zeitfenster für die Patroullie, bis die Gruppe wieder in ihr Fahrzeug steigt, um ihren nächsten Wegpunkt anzufahren. <p>Oben das Beispiel bedeutet also: Die Gruppe wird für die Zeit von 60 Sek. in einem Radius von 50m auf Patroullie gehen, und wird dann wieder in ihr Fahrzeug einsteigt.</p> <p>Hier noch 2 weitere Beispiele: ["50 + random 50","60 + random 30"] ["30 * DAC_AI_Level","60 * DAC_AI_Level"]</p> <p>Eine Besonderheit ist hier, dass über die Variable _leader, der Anführer der Gruppe angesprochen werden kann. Das Array könnte also auch folgende Einträge haben (Beispiel): ["25 + (30 * (skill _leader))","(30 + (random 30)) + ((skill _leader) * 50)"]</p>
_setSearch	<p>["100","120"]</p> <p>Einheiten aus den Kategorien Infanterie + Radfahrzeuge werden, wenn sie den Feind aus den Augen verloren haben, für eine gewisse Zeit die Gegend nach Feinden absuchen. Die Handhabung der Parameter, ist exakt die gleiche wie bei _setPatrol.</p> <ol style="list-style-type: none"> 1. Eintrag = Maximaler Suchradius für die Gruppe 2. Eintrag = Zeitfenster für die Suche, bis die Gruppe wieder ihre normalen Wegunkte aufnimmt.
_setSupport	<p>[1,2]</p> <p>Diese Einstellung regelt zum einen das Unterstützungsverhalten der Einheiten, und zum anderen wird hier die DAC-Arti-Anforderung aktiviert/deaktiviert. Möchtest Du, dass Einheiten ihre Zone(n), zwecks Unterstützung von Einheiten anderer Zonen, nicht verlassen, dann musst Du den 1. Wert auf 0 setzen. So ist sichergestellt, dass eine Zone nicht „leer laufen“ kann, weil alle Gruppen auf Unterstützungs-Mission sind ;-)</p> <ol style="list-style-type: none"> 1. Eintrag = Zone gibt Gruppen zur Unterstützung frei (0=nein, 1=ja) 2. Eintrag = Aktiviert in der Zone die Arti-Anforderung (0=deaktiviert) <p>Um die Arti-Anforderung durch Infanterie-Gruppen in der Zone zu aktivieren, muss der 2. Wert einer gültigen Nummer aus der DAC_Config_Arti entsprechen.</p>
_setJoin	<p>2</p> <p>Dieser Parameter legt fest, ab welcher Gruppengröße eine Gruppe sich auflöst und sich der nächstgelegenen Gruppe anschließt. Dieses Verhalten kann zu einem Respawn einer neuen Gruppe führen, wenn ein entsprechendes Camp verfügbar ist.</p>
_setEmpVeh	<p>[[150,100],[100,50]]</p> <p>Diese Arrays aktivieren das Besetzen von leeren Fahrzeugen und Geschützen. Möchtest Du eines dieser Merkmale in der Zone nicht benutzen, musst Du den jeweils 1. Wert auf 0 stellen, ansonsten wie folgt:</p> <ol style="list-style-type: none"> 1. Array = Radius der Erkennung für leere Fahrzeuge um sie zu besetzen, Wahrscheinlichkeit in % 2. Array = Radius der Erkennung für leere Geschütze um sie zu besetzen, Wahrscheinlichkeit in % <p>Welche leeren Fahrzeugtypen besetzt werden dürfen, wird festgelegt durch die Einstellung DAC_VehAllowed. Welche leeren Geschütze besetzt werden dürfen, wird festgelegt durch die Einstellung DAC_GunAllowed (siehe DAC_Config_Creator).</p>
_setSupTime	<p>["5 + ((skill _unit) * (5 * DAC_AI_Level))",2,5]</p> <ol style="list-style-type: none"> 1. Das Zeitfenster, in dem gültige Einheiten Unterstützungsfeuer leisten 2. Wie oft hintereinander diese Einheiten Unterstützungsfeuer leisten 3. Wieviel Sekunden Pause zwischen den Unterstützungsfeuer-Phasen eingelegt wird.
_setHidTime	<p>["(((10 * DAC_AI_Level) + ((skill _leader) * 50)) / ((count units _group) + 1))"]</p> <p>Die Zeit, die eine Gruppe in Deckung bleibt und nicht vorrückt, wenn sie Feinde geortet hat.</p>

[DAC_Config_Waypoints]

Die **DAC_Config_Waypoints** hält alle Parameter bereit, die zum generieren von Wegpunkten benötigt werden. Auch hier sind wieder beliebig viele „Blöcke“ erlaubt. Jeder Block enthält dabei alle Parameter für alle Wegpunkttypen.

Standard- mäßig wird immer die Konfiguration **0** (case 0) benutzt. Bei Bedarf kann man aber jede beliebige Zone mit einer eigenen Konfiguration ausstatten (siehe **Wahl der Wegpunkt-Konfiguration** auf Seite 7).

Unten die Abbildung zeigt Dir einen „Block“ aus der DAC_Config_Waipoints:

```
case 0: {  
    //----- #Sol----#Veh----#Tan----#Air----#Camp----;  
    _checkRadius1 = [ 10,    10,    10,    20,    20    ];  
    _checkRadius2 = [ 15,    20,    20,    40,    40    ];  
    _checkAreaH   = [ 40,    15,    20,    10,    10    ];  
    _checkMaxH    = [ 500,   500,   500,   500,   500   ];  
    _checkMinH    = [ 5,     5,     5,     5,     5     ];  
    _checkNear    = [ 0,     0,     0,     100,   200   ];  
    _checkObjH1   = [ 1.5,   0.5,   0.5,   0.2,   0.2   ];  
    _checkObjH2   = [ 30,    15,    15,    5,     4     ];  
    _checkCount   = [ 200,   200,   200,   500,   1500  ];  
    _checkResol   = [ 45,    36,    36,    12,    12    ];  
  
    _TempWPArray = call compile format["DAC_WP_Pool_%1",(_DACTemp select _DAC_WP_Typ)];  
};
```

Beispiel:

Für die Zone **z6** wurde die WP-Konfiguration **2** gewählt (dieser Parameter ist optional)

["z6",[1,0,0],[8,2,80,12],[],[],[],[0,0,0,2]] spawn DAC_Zone

Die einzelnen Parameter möchte ich hier nur grob ansprechen, bzw. erklären sie sich teilweise auch von selbst.

_checkRadius1	Der kritische Radius, in dem keine oder nur sehr kleine Objekte vorhanden sein dürfen
_checkRadius2	Der unkritische Radius, in dem nur Objekte einer gewissen Größe/Höhe vorhanden sein dürfen
_checkAreaH	Die maximale Höhendifferenz innerhalb von _checkRadius2
_checkMaxH	Die maximale Höhe, in der Wegpunkte generiert werden dürfen
_checkMinH	Die minimale Höhe, in der Wegpunkte generiert werden dürfen
_checkNear	Der Mindestabstand für Wegpunkte
_checkObjH1	Die maximale Objektgröße für den kritischen Bereich (_checkRadius1)
_checkObjH2	Die maximale Objektgröße für den unkritischen Bereich (_checkRadius2)
_checkCount	Die maximale Anzahl an Versuchen einen Wegpunkt zu finden, bevor DAC abbricht
_checkResol	Der Wert für die Raster-Auflösung beim Scannen der Höhenwerte (kleinerer Wert = höhere Belastung)

[DAC_Config_Camp]

Die **DAC_Config_Camp** sieht auf den ersten Blick sehr kompliziert aus...ist sie auch ;-)
Immerhin werden hier die kompletten DAC-Camps definiert und das in 7 möglichen Ausbaustufen.

Unten die Abbildung zeigt z.B. eine Camp-Konfiguration, die alle Ausbaustufen benutzt, deshalb sieht sie auch etwas chaotisch aus. Wenn Du Dir das Skript **DAC_Config_Camp.sqf** mal anschaust, findest Du dort aber auch weit weniger umfangreiche Camp-Konfigurationen. Von dem Array **_campObjInit** mal abgesehen, entsprechen die anderen Arrays jeweils genau einer Ausbaustufe.

["z6",[1,0,0],[8,2,80,12],[],[],[0,0,0,2,2]] spawn DAC_Zone

```
case 2:
{
    _campBasic      = ["FlagCarrierCDF",["Land_Campfire_burning",8,5,0],["Camp",5,0,0],["Logic",10,15,0],0];
    _campAmmo       = [["GuerillaCacheBox",10,2,0],["LocalBasicAmmunitionBox",10,0,0],["LocalBasicWeaponsBox",10,0,0]];
    _campStatic     = [["DSHKM_Gue",-7,25,0,"GUE_Soldier_1"],["DSHKM_Gue",25,25,0,"GUE_Soldier_1"],["DSHKM_Gue",45,25,0,"GUE_Soldier_1"]];
    _campAddUnit    = ["GUE_Soldier_AT","GUE_Soldier_AT","GUE_Soldier_AA","GUE_Soldier_Sniper"];
    _campUserObj    = [["Land_Antenna",10,-15,45],["Logic",17,35,0],["Logic",17,20,0],["Logic",47,0,0],["Logic",47,0,0]];
    _campRandomObj  = [];
    _campWall       = ["Land_BagFenceLong",[-10,30],[40,56,0],[5,5,5,5],[1,0.2],[0,0]];
    _campObjInit    = [[],[ ],[ ],[ ],[ ],[ ],[ ],[ ]];
};
```

Hinweis: Die Objekttypen auf den nachfolgenden Seiten stammen aus Arma2

_campBasic = ["FlagCarrierNorth",["Fire",5,10,0],["CampEast",7,0,0],["Logic",0,15,0],0];

Dieses Array ist die Basis für ein Camp und darf nicht erweitert oder reduziert werden.
Sehrwohl dürfen aber die vorhandenen Einträge verändert werden.

Der erste Eintrag ist immer die Flagge. Sie ist das Referenzobjekt für alle anderen Objekte, die im Camp generiert werden. Sie wird genau auf die Position gesetzt, die DAC für das Camp gefunden hat.
Dann kommen 3 Arrays, die jeweils 4 Einträge beinhalten.

Wichtig: Das letzte Objekt sollte vom Typ „Logic“ sein (wie oben zu sehen).
Sie dient als Position für einen Respawn, also die Stelle wo im Camp neue Einheiten generiert werden.

Die jeweils 4 Parameter haben folgende Bedeutung:

["CampEast",7,0,0]

- Der Objekt-Typ der generiert wird
- Die x-Abweichung vom Referenzobjekt (28 = 28m östlich / -28 = 28m westlich von der Flagge)
- Die y-Abweichung vom Referenzobjekt (10 = 10m nördl. / -10 = 10m südlich von der Flagge)
- Die Ausrichtung des Objekts (Beispiel: 0 = zufällige Ausrichtung, 90 = Osten, 180 = Süden, 270 = Westen, 360 = Norden)

_campBasic = ["FlagCarrierNorth",["Fire",5,10,0],["CampEast",7,0,0],["Logic",0,15,0],0];

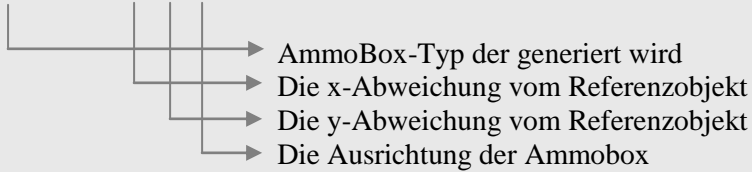
Der letzte Parameter in dem Array **_campBasic** steuert die Selbstreparatur (gilt nur für die 3 Basisobjekte)
0 = Selbstreparatur deaktiviert, >0 = Sekunden bis die Objekte wieder Instand gesetzt werden.
Info: Es kommt vor, dass respawnnte Fahrzeuge diese Basisobjekte zerstören. Deshalb dieser Parameter.


```
_campAmmo = [{"AmmoBoxEast",15,-2,90},{"WeaponBoxEast",20,-2,90},{"SpecialBoxEast",30,30,0}];
```

Dieses Array dient dazu, eine oder auch mehrere Waffenkisten im Camp zu platzieren.

Jedes Array besteht wieder aus 4 Werten, die analog zu den Basis-Objekten funktionieren:

```
["AmmoBoxEast",15,-2,90]
```



```
_campStatic = [{"D30",-3,18,270,"Soldiereb"},{"D30",5,38,0,"Soldiereb"},{"D30",38,25,90,"Soldiereb"}];
```

In diesem Array werden bei Bedarf beliebig viele statische Waffen definiert.

Das kann z.B. eine MG-Stellung sein, oder ein Flak- bzw. ein Arti- Geschütz. Voraussetzung für diese Statics ist, dass eine Gunner-Position vorhanden sein muss. Es können auch Fahrzeuge mit Gunner-Pos. angegeben werden, die Fahrer-Position wird dann aber (absichtlich) nicht besetzt.

Die Arrays haben hier einen weiteren Wert. Dieser Wert definiert den Einheiten-Typ für die Gunner-Position. Ansonsten ist der Aufbau wieder analog zu den anderen.

```
_campAddUnit = ["Soldiereaa","Soldiereaa","Soldieresniper"];
```

Hier können Infanterie-Einheiten definiert werden, die zusätzlich in die Camp-Gruppe generiert werden.

So kannst Du sicherstellen, dass in einem Camp z.B. genügend Luftabwehr-Einheiten vorhanden sind, da die Camp-Gruppe, wie alle anderen auch, per Zufall generiert werden und es keine Garantie für bestimmte Einheiten-Typen gibt. Bitte achte darauf, dass Du die richtigen Einheiten einträgst (auf die Seite bezogen).

```
_campUserObj = [{"UAZ",0,32,90},{"UAZ",0,25,90},{"Ural",30,0,270}];
```

Dieses Array enthält beliebig viele benutzerdefinierte Objekte aller Art. Das können leere Fahrzeuge sein, oder auch bestimmte Gebäude oder Einrichtungen wie z.B. Zelte oder Wachtürme.

Der Aufbau der Arrays ist wieder wie gehabt: Objekttyp, x-Pos, y-Pos, Drehung

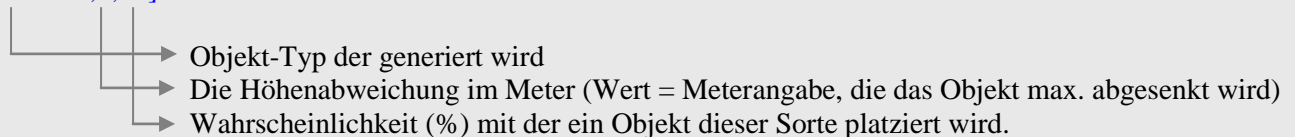
```
_campRandomObj = [{"AAPL048",1,60},{"AARO038",1,10},{"AARO041",1,10},{"AAPL068",3,20}],50,1,50,10];
```

Möchtest Du dem Camp und den Einheiten etwas natürlichen Schutz bieten, kannst Du hier einiges dafür tun.

Damit gemeint sind Objekte wie Bäume, Büsche, Steine usw.

Das Array kann mehrere Objekt-Arrays aufnehmen. Jedes dieser Objekt-Arrays benötigt 3 Werte:

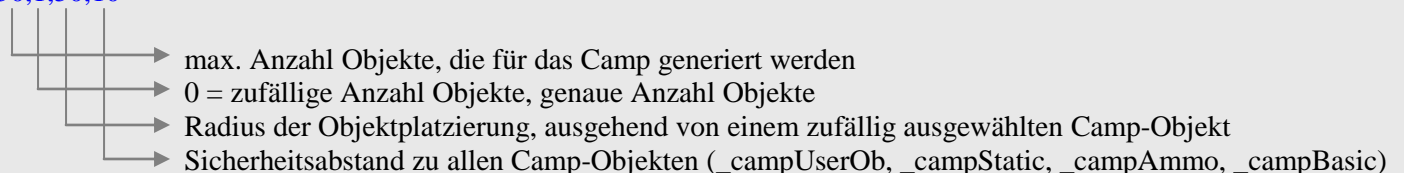
```
["AARO038",1,10]
```



```
_campRandomObj = [{"AAPL048",1,60},{"AARO038",1,10},{"AARO041",1,10},{"AAPL068",3,20}],50,1,50,10];
```

Die 4 Werte am Ende des Arrays bewirken folgendes:

```
50,1,50,10
```



```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Hier wird die Einfassung des Camps definiert, also die Mauern oder Sandsäcke, die um das Camp herum platziert werden.

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Der erste Wert bestimmt den Objekt-Typ, der für die Einfassung verwendet wird.

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Diese Werte bestimmen die Startposition [x,y], ausgehend wieder von der Flagge.

Von da aus werden die Objekte im Uhrzeigersinn generiert. In diesem Beispiel also 2 m westlich, und 35 m nördlich der Flagge.

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Hier wird die Grösse für die Einfassung festgelegt. Der 1. Wert definiert die x-Ausdehnung nach Osten, der 2. Wert definiert die y-Ausdehnung nach Süden (Meter). Der 3. Wert bestimmt, welche Objektachse DAC vermessen soll: **0** = x-Achse, **1** = y-Achse. Dadurch wird die genaue Objektgröße ermittelt. Das ist wichtig, damit DAC die Objekte exakt zueinander platzieren kann. Einfach ausprobieren, es gibt ja nur diese 2 Möglichkeiten ;-)

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Möchtest Du Ausgänge anlegen (macht ja Sinn), dann kannst Du diese hier definieren. Die Camp-Einfassung hat immer 4 Seiten, die Nord-, Ost-, Süd- und West-Seite. Genau diese 4 Seiten sind in dem Array abgebildet.

Der jeweilige Wert gibt an, wie viele Segmente Du an der entsprechenden Seite weglassen möchtest, immer von der Mitte ausgehend. Der Wert 0 bedeutet, dass die Einfassung an der Seite geschlossen ist.

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Hier wird festgelegt, wie weit die Objekte in dem Boden versenkt werden sollen. Das kann ein fester Wert sein, oder aber ein zufälliger Wert in einem bestimmten Bereich.

Der erste Parameter regelt diese Versenkung: 0 = zufälliger Wert, 1 = exakter Wert

Der zweite Parameter legt dann fest, wie weit das Objekt maximal versenkt wird.

```
_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];
```

Fehlt nur noch die Ausrichtung der Objekte, die Du mit diesem Array beeinflussen kannst.

Wert 1 legt fest, ob die Objekte eine feste Ausrichtung haben, oder ob ihre Ausrichtung in einem bestimmten Bereich zufällig abweichen soll: 0 = feste Ausrichtung, >0 = Wert für zufällige Abweichung.

Wert 2 legt die grundsätzliche Ausrichtung fest. Objekte werden grundsätzlich immer nach Norden ausgerichtet, wenn sie erstellt werden. Manche Objekte zeigen mit der „breiten“ Seite nach Norden, andere mit der schmalen Seite. Um diesen Umstand berücksichtigen zu können, muss hier evtl. eine Anpassung erfolgen.

Da die Einfassung immer im rechten Winkel aufgebaut wird, machen hier nur folgende Werte Sinn: 0, 90, 180, 270. (Um das Alles besser verstehen zu können, findest Du im Anhang ein paar bildliche Beschreibungen)

```
_campObjInit = [[],[],[],[],[],[],[]];
```

Hier kann man jedem generierten Objekt einen Zustand zuweisen, oder aber auch ein Skript starten. Jedes leere Array bezieht sich auf ein bestimmtes Objekt-Array:

```
[_campBasic, _campAmmo, _campStatic, _campAddUnit, _campUserObj, _campRandomObj, _campWall]
```

Möchtest Du z.B. die leeren Fahrzeuge, die Du im Objekt-Array `_campUserObj` definiert hast, abschliessen, dann würde das folgendermaßen aussehen:

```
_campObjInit = [[],[],[],[],["_x lock true"],[],[]];
```

Der Eintrag wurde im 5. Array gemacht, weil das `_campUserObj` – Array das 5. Objekt-Array ist.

Die Einträge müssen zudem unbedingt als String vorliegen. Die Objekte werden mit dem Platzhalter `_x` angesprochen

[DAC_Config_Arti]

Die **DAC_Config_Arti** enthält alle relevanten Parameter für eine Arti-Anforderung, sowie die 3 Bedingungsstufen. Um die Arti auf KI-Seite verfügbar zu machen, muss es eine Zone geben, die in ihrer **DAC_Config_Behaviour** einen gültigen Eintrag aus der **DAC_Config_Arti** eingetragen hat.

Diese Zone z.B. lädt die Verhaltens-Konfiguration **1** aus der **DAC_Config_Behaviour**.

["z1",[1,0,0],[5,3,30,8],[],[],[],[0,0,1,0]] spawn DAC_Zone

In dieser Konfiguration gibt es das Array **_setSupport = [1,3]** der den Parameter für die Arti-Konfiguration enthält.

Mit anderen Worten: Die Einheiten aus dieser Zone können Arti anfordern.

Wenn sie das tun, wird dabei die Arti-Konfiguration 3 verwendet (jede Zone kann eine andere Konfiguration benutzen).

```
case 3:
{
    _set0 = [10,2,0,30];
    _set1 = [0,0,100,[ ],2,30];
    _set2 = [100,100,100,100,4,1,0];
    _set3 = [["D30_RU"],["Sh_122_HE"],["T72_INS"],["Sh_125_SABOT"]];
    _set4 = [2,5,1];
    _set5 = [10,30,1];
    _set6 = [10,30,1];
    _set7 = [0.1,0.5,1];
    _set8 = [ ];
    _set9 = [ ];
    _set10 = 2000;
};
```

Oben die Abb. zeigt Dir einen kompletten „Block“ aus der **DAC_Config_Arti**. Die ersten 3 Arrays (**rot markiert**) definieren jeweils eine Bedingungsstufe. Die restlichen 7 Arrays (**grün markiert**) enthalten die Parameter für die Arti.

- _set0 = [20,2,0,30]** Bedingungsstufe 1
- Max. Geschwindigkeit der **Target-Group**, damit ein Arti-Schlag auf die Gruppe ausgeführt wird.
 - Min. Anzahl der Einheiten in der **Target-Group**, damit ein Arti-Schlag ausgeführt wird.
 - Min. Anzahl Fahrzeuge in der **Target-Group**, damit ein Arti-Schlag ausgeführt wird.
 - Min. Zeitspanne, bis ein weiterer Arti-Schlag auf die **Target-Group** ausgeführt werden kann.
- _set1 = [2,0,100,[],2,30]** Bedingungsstufe 2
- Min. Anzahl Einheiten in der **Call-Group**, damit Arti angefordert werden kann.
 - Min. Durchschnitts-Skill der **Call-Group**, damit Arti angefordert werden kann.
 - Min. Distanz der **Call-Group** zur **Target-Group**, damit ein Arti-Schlag
 - Einheitentypen, die in der **Call-Group** vorhanden sein müssen, damit ein Arti-Schlag ...
 - Anzahl der Versuche Arti anzufordern, wenn Bedingungen (z.B. Distanz) ungünstig.
 - Zeitspanne zwischen den Versuchen, Arti anzufordern.
- _set2 = [50,100,10,100,4,1,0]** Bedingungsstufe 3
- Wahrscheinlichkeit über alles in % für einen **Arti-Schlag**
 - Min. Distanz der **Arti-Einheit** zur **Target-Group**
 - Min. Distanz freundlicher Einheiten zur **Target-Group**
 - Max. Anzahl **Arti-Schläge** über alles
 - Max. Anzahl **Arti-Einheiten** pro Arti-Schlag
 - Max. Anzahl Schuß pro **Arti-Einheit**
 - Bewegliche **Arti-Einheiten** AN (1) / AUS (0)

`_set3 = [{"T72"}, {"Sh_105_HE"}];`

- Der Einheitentyp, der als Arti-Einheit zum Einsatz kommt
- Der Munitionstyp der bei einem Arti-Schlag verwendet wird.

Es sind hier auch mehrere Einträge möglich z.B.:

`[{"T72"}, {"Sh_105_HE"}], [{"D30"}, {"Sh_105_HE"}]`

Auch bei dem Munitionstyp, z.B. um zusätzlich Rauchgranaten zu zünden:

`[{"T72"}, {"Sh_105_HE"}, "Smokeshell"]]`

`_set4 = [2,5,1];`

- Min. Zeitfenster von der Arti-Anforderung bis zur Arti-Ausführung.
- Max. Zeitfenster von der Arti-Anforderung bis zur Arti-Ausführung.
- 0 = Skill-Abhängigkeit deaktiviert, 1 = Skill des Anführers der Gruppe wird berücksichtigt

Beispiel: Skill = 1.0 => Zeitfenster = 2 – 5 Sekunden
Skill = 0.5 => Zeitfenster = 4 – 10 Sekunden
Skill = 0.2 => Zeitfenster = 10 – 25 Sekunden

`_set5 = [3,10,1];`

- Min. Abweichung bei Positionsangabe durch **Call-Group**
- Max. Abweichung bei Positionsangabe durch **Call-Group**
- 0 = Skill-Abhängigkeit deaktiviert, 1 = Skill des Anführers der Gruppe wird berücksichtigt

Beispiel: Skill = 1.0 => Min. Abweichung = 3m | Max. Abweichung = 10m
Skill = 0.5 => Min. Abweichung = 6m | Max. Abweichung = 20m
Skill = 0.2 => Min. Abweichung = 15m | Max. Abweichung = 50m

`_set6 = [5,20,1];`

- Min. Abweichung bei Positionsangabe durch **Arti-Einheit**
- Max. Abweichung bei Positionsangabe durch **Arti-Einheit**
- 0 = Skill-Abhängigkeit deaktiviert, 1 = Skill des Anführers der Gruppe wird berücksichtigt

Beispiel: Skill = 1.0 => Min. Abweichung = 5m | Max. Abweichung = 20m
Skill = 0.5 => Min. Abweichung = 10m | Max. Abweichung = 40m
Skill = 0.2 => Min. Abweichung = 25m | Max. Abweichung = 100m

`_set7 = [0.1,0.5,1];`

- Min. Pause zwischen den Granatabschüssen
- Max. Pause zwischen den Granatabschüssen
- 0 = Skill-Abhängigkeit deaktiviert, 1 = Skill des Anführers der Gruppe wird berücksichtigt

Beispiel: Skill = 1.0 => Zeitfenster = 0.1 – 0.5 Sekunden
Skill = 0.5 => Zeitfenster = 0.2 – 1.0 Sekunden
Skill = 0.2 => Zeitfenster = 0.5 – 2.5 Sekunden

`_set8 = [];`

- Namen von Editor gesetzten Einheiten, die als Arti fungieren sollen.
Mehrere Einträge sind möglich.

`_set9 = [];`

- Globale Bedingung die erfüllt sein muss, damit ein Arti-Schlag ausgeführt werden kann.
Diese Bedingung bitte als String eingeben.
Beispiel: `["({ alive _x } count [unitName1, unitName2, unitName3]) > 0"]`

`_set10 = 2000;`

- Dieser Wert legt die maximale Reichweite der Arti-Einheiten fest.
Ziele außerhalb dieser Reichweite können von der Arti nicht unter Beschuß genommen werden.

[DAC_Config_Events]

Die **DAC_Config_Events** ermöglicht es Dir, an bestimmten Stellen im DAC einzugreifen. Dafür stellt DAC einige Events zu Verfügung, wo Du z.B. ein eigenes Skript starten kannst, oder Werte einer Gruppe/Einheit abfragen kannst. Genau wie bei den anderen **DAC-Konfigurationen**, kannst Du auch hier wieder beliebig viele Konfigurationen anlegen. So kannst Du im Prinzip jede Zone mit einer anderen Konfiguration ausstatten.

Event		Variable		Beschreibung
Create	>	_group	>	Nachdem eine Gruppe generiert wurde
ReachWP	>	_group	>	Sobald eine Gruppe einen ihrer Wegpunkte erreicht hat
NotAliveGroup	>	_group	>	Sobald eine Gruppe down ist
NotAliveUnit	>	_unit	>	Sobald eine Einheit down ist
BeforeReduce	>	_unit	>	Bevor eine Gruppe reduziert wird
AfterBuildUp	>	_unit	>	Nachdem eine reduzierte Gruppe wieder aufgebaut wurde
InitVehicle	>	_vehc	>	Nachdem ein Fahrzeug generiert wurde

Für jede Einheiten-Kategorie (s. unten) können in den jeweiligen Events Aktionen hinterlegt werden.

_Init_Unit_S	=	Kategorie Infanterie
_Init_Unit_V	=	Kategorie Radfahrzeuge
_Init_Unit_T	=	Kategorie Kettenfahrzeuge
_Init_Unit_A	=	Kategorie Helikopter
_Init_Unit_C	=	Kategorie Camp-Gruppe
_Init_Unit_H	=	Kategorie Helikopter-Gruppe
_Init_Unit_V	=	Alle leeren Fahrzeuge + Helis, die DAC generiert

In jeder dieser Kategorien findest Du mehrere Arrays, die jeweils einem Event entsprechen.
Hier zum Beispiel die Kategorie **Infanterie**:

```
_Init_Unit_S = [  
    [], <= für Event Create  
    [], <= für Event ReachWP  
    [], <= für Event NotAliveGroup  
    [], <= für Event NotAliveUnit  
    [], <= für Event BeforeReduce *for infantry, camps and wheeled vehicles only*  
    [], <= für Event AfterBuildUp *for infantry, camps and wheeled vehicles only*  
];
```

Wichtig: Alle Event-Einträge müssen als String angegeben werden !!

Hier ein paar Beispiele dazu, wie solche Einträge aussehen sollten (mögliche Variablen sind **fett** geschrieben):

```
Event Create :      "{_x Setskill 1} foreach units _group "  
                    "{if(format[""%1"",typeof _x] == "Soldierwsniper") then {_x Setskill 1} foreach units _group "  
                    "{if(!_x != vehicle _x) then {if(!((vehicle _x) in VehArray)) then { VehArray = VehArray + [vehicle _x]}} foreach units _group "  
                    "{[_x] execVM ""MyUnitScript.sqf""} foreach units _group "  
                    "{_x addeventhandler ["hit",{_this spawn MyHitScript}]} foreach units _group "  
  
Event ReachWP :     "{if((getdammage _x) > 0.8) then {_x commandmove (position HealthBuilding)}} foreach units _group "  
                    "{if((_x distance (leader (group _x))) > 50) then {_x commandmove (position (leader (group _x)))}} foreach units _group "  
                    "[_group] call MyGroupFunction"  
  
Event NotAliveGroup : "MyGroupArray = MyGroupArray - [_group]"  
  
Event NotAliveUnit : "MyUnitArray = MyUnitArray - [_unit]"  
                    "MyUnitCount = MyUnitCount - 1"  
  
Event BeforeReduce : "MyUnitArray = MyUnitArray - [_unit]"  
                    "MyUnitCount = MyUnitCount - 1"  
  
Event AfterBuildUp : "MyUnitArray = MyUnitArray + [_unit]"  
                    "_unit addeventhandler ["hit",{_this spawn MyHitScript}]"  
                    "MyUnitCount = MyUnitCount + 1"
```

Möchtest Du z.B. an jedem Wegpunkt für Infanterie ein Script starten, das Dir irgendwelche Daten aus den Gruppen einer Zone liefern soll, mußt Du zum einen die Konfigurations-Nr. aus der **DAC_Config_Events** in dem Skriptaufruf der Zone eintragen, und zum anderen muss dann in dieser Konfiguration der entsprechende Eintrag (als String) vorhanden sein.

Das hier wäre ein Skriptaufruf, der für die Zone "z1" die Konfiguration-Nr. **1** aus der **DAC_Config_Events** lädt: (Im Auslieferungszustand von DAC, ist nur die Nr. 1 vorhanden. Weitere Nr. können aber von Dir angelegt werden)

```
["z1",[1,0,1],[5,3,50,8],[],[],[1,1,0,6]] spawn DAC_Zone
```

Unter dieser Nummer in der **DAC_Config_Events.sqf**, mußt Du dann in dem Array für Infanterie, den entsprechenden Eintrag für den Skriptaufruf eintragen:

```
case 1:
{
    _Init_Unit_S = [
        [],
        ["[_group] execVM ""NameOfYourScript.sqf""],
        [],
        [],
        [],
        []
    ];

    _Init_Unit_V = [
        [],
        [],
        [],
        [],
        [],
        []
    ];

    _Init_Unit_T = [
        [],
        [],
        [],
        []
    ];

    _Init_Unit_A = [
        [],
        [],
        [],
        []
    ];

    _Init_Unit_C = [
        [{"_x SetSkill 1} foreach units _group "],
        [],
        [],
        [],
        [],
        []
    ];

    _Init_Unit_H = [
        [],
        [],
        [],
        []
    ];

    _Init_Vehicle = [
        [],
        ["[_vehc] execVM ""NameOfYourScript.sqf""],
        []
    ];
};
```

Infanterie

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)
Event [BeforeReduce](#)
Event [AfterBuildUp](#)

Radfahrzeuge

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)
Event [BeforeReduce](#)
Event [AfterBuildUp](#)

Kettenfahrzeuge

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)

Helibesatzung

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)

Campgruppe

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)
Event [BeforeReduce](#)
Event [AfterBuildUp](#)

Heligruppe

Event [Create](#)
Event [ReachWP](#)
Event [NotAliveGroup](#)
Event [NotAliveUnit](#)

Rad, Kette, Heli

Event [InitVehicle](#)
Event [InitVehicle](#)
Event [InitVehicle](#)

[DAC_Config_Marker]

Die **DAC_Config_Marker** enthält alle Parameter für die Erzeugung der DAC-Marker. Diese Marker können jederzeit und beliebig oft aktiviert / deaktiviert werden. Um die DAC-Marker zu aktivieren, musst Du in der **DAC_Config_Creator** nur eine gültige Nummer aus der **DAC_Config_Marker** angeben:

DAC_Marker = 3 würde zum Beispiel die Marker-Konfiguration 2 laden.

Um die DAC-Marker zu deaktivieren, musst Du dem DAC nur mitteilen:
Das kannst Du per Skript machen, oder z.B. auch durch einen Auslöser.

DAC_Marker = 0

Ein „Block“ in der **DAC_Config_Marker** ist folgendermaßen aufgebaut:

```
case 3:
{
    _setShowZones      = 2;
    _setShowWPs        = 1;
    _setShowUnit       = [1,1,1,1];
    _setGroupType      = 0;
    _setMarkerText     = ["if(isPlayer _unit) then {format[""%1"",_unit]}"];
    _setMarkerDel       = 1;
    _setMarkerRefresh   = [0.2,0.2];
    _setSizeWpLine      = [0.7,8];
    _setSizeLeaderLine  = 0.4;
    _setSizeZoneLine    = 2;
    _setSizeCampLine    = 2.5;
    _setSizeZoneBorder  = 4;
    _setArtiMarker      = 1;
    _setCampMarker      = 2;
    _setSideColor       = [
        "ColorRed",
        "ColorBlue",
        "ColorYellow",
        "ColorGreen",
        "ColorWhite",
        "ColorBlack",
        "ColorGreen",
        "ColorBlack"
    ];
};
```

Die einzelnen Einträge möchte ich hier nur in Kürze beschreiben:

_setShowZones	Zonen Marker: 0 = deaktiviert / 1 = nur bei Initialisierung / 2 = immer zeigen
_setShowWPs	Wegpunkt Marker: 0 = deaktiviert / 1 = nur bei Initialisierung / 2 = immer zeigen
_setShowUnit	Einheiten Marker: 0 = deaktiviert / 1 = aktiviert für Seite [West,Ost,Widerstand,Zivil]
_setGroupTyp	0 = ein Marker pro Gruppe / 1 = Marker für jede Einheiten einer Gruppe
_setMarkerText	Marker Text: formatierter Text als String (siehe oben). Variable _group benutzbar
_setMarkerDel	Einheiten-Marker: 0 = nicht löschen / 1 = löschen wenn Einheit down / 2 = löschen wenn Einheit gelöscht
_setMarkerRefresh	Aktualisierungszeit der Marker [Einheiten Marker, andere Marker]
_setSizeWpLine	Wegpunkt-Marker: [Linienstärke, Wegpunktgröße], Linienstärke = 0 deaktiviert die Wegpunkt-Marker
_setSizeLeaderLine	Leader-Verbindungs-Linien: Linienstärke, Linienstärke = 0 deaktiviert die Leader-Verbindungs-Linien
_setSizeZoneLine	Zonen-Verbindungs-Linien: Linienstärke, Linienstärke = 0 deaktiviert die Zonen-Verbindungs-Linien
_setSizeCampLine	Camp-Verbindungs-Linien: Linienstärke, Linienstärke = 0 deaktiviert die Camp-Verbindungs-Linien
_setSizeZoneBorder	Linienstärke Zonenumrandung
_setArtiMarker	Arti-Marker : 0 = deaktiviert / 1 = aktiviert
_setCampMarker	Camp-Marker : 0 = deaktiviert / 1 = minimal (nur Camp-Objekte) / 2 = maximal (alle Objekte)
_setSideColor	Die Markerfarben: [Ost, West, Res, Civil, Neutral, Wegpunkt-Linien, Wegpunkte]

Dann kommt noch die Definition der eigentlichen Marker, die verwendet werden sollen: **_setMarkerClass**

```

_setMarkerClass = [
    [ "STATICWEAPON" "Dot", [0.5,0.5], 1],
    [ "MAN" "mil_triangle", [0.6,0.6], 1],
    [ "CAR" "mil_box", [0.5,0.8], 1],
    [ "TRUCK" "mil_box", [0.5,0.9], 1],
    [ "TANK" "mil_box", [0.6,1.2], 1],
    [ "APC" "mil_box", [0.6,1.2], 1],
    [ "MOTORCYCLE" "Dot", [0.3,0.7], 1],
    [ "AIR" "mil_triangle", [0.7,1.5], 1],
    [ "HELICOPTER" "mil_triangle", [0.7,1.5], 1],
    [ "PLANE" "mil_triangle", [0.9,1.5], 1],
    [ "ParachuteBase" "mil_triangle", [0.9,0.3], 1],
    [ "SHIP" "Dot", [0.9,1.3], 1],
    [ "OTHER" "Dot", [0.7,0.7], 1]
];

```

Die Abbildung oben zeigt Dir die 8 **übergeordneten** Einheiten-Klassen (orange markiert), die grundsätzlich vorhanden sein müssen. Unterhalb dieser übergeordneten Einheiten-Klassen, kannst Du weitere Klassen angeben, die automatisch benutzt werden, wenn eine Einheit einer dieser angegebenen Klasse entspricht (wie oben zu sehen).

Kann die Einheit innerhalb ihrer Klasse nicht zugeordnet werden, wird automatisch die übergeordnete Klasse verwendet. Kann eine Einheit keiner übergeordneten Klasse zugeordnet werden, benutzt DAC die Einstellung, die unter "Other" definiert ist.

Jeder Eintrag in der **DAC_Config_Marker** entspricht einem Array mit 4 Einträgen:

[**"EinheitenKlasse"**, **"Markerklasse"**, [Markergröße x,y], **Markerdrehung** (0 = AUS / 1 = AN)]

[DAC_Config_Objects]

In der **DAC_Config_Objects** sind die Objekte definiert, die mit dem Skript **DAC_Objects** generiert werden können. Die Daten-Blöcke hier, enthalten die jeweiligen Objekt-Definitionen.

Auch hier gilt wieder, dass Du die Datei um beliebig viele Daten-Blöcke erweitern kannst und das jeder Daten-Block eine eindeutige ID bekommen muss (case Nr).

Um einen solchen Daten-Block zu benutzen, musst Du nur die entsprechende **ID** in dem Skriptaufruf angeben:

nul = ["objZ1",300,0,0,**13**,0,[]," "] spawn DAC_Objects

Figure based on Arma2

```
case 13:
{
    _Object_Pool = [
        [0,1],
        ["MAP_R2_Boulder1",5,0,0.05,0,0,"",[0,"ColorBlack"]],
        ["MAP_t_picea1s",2,0,0.05,0,0,"",[0,"ColorGreen"]],
        ["MAP_t_picea2s",1,0,0.05,0,0,"",[0,"ColorGreen"]],
        ["MAP_R2_Boulder2",5,0,0.05,0,0,"",[0,"ColorBlack"]],
        ["MAP_R2_Stone",5,0,0.1,0,0,"",[0,"ColorBlack"]],
        ["MAP_b_betulaHumilis",6,0,0,0,0,"",[0,"ColorGreen"]],
        ["MAP_b_canina2s",6,0,0,0,0,"",[0,"ColorGreen"]]
    ];
};
```

In jedem Daten-Block (den Namen „_Object_Pool“ bitte nicht ändern) kannst Du beliebig viele Objekte definieren, die dann in der entsprechenden Zone generiert werden.

Bitte beachte, dass hier der erste Eintrag immer ein Array mit 2 Zahlenwerten sein muss, denn diese Werte definieren in jedem Daten-Block folgendes:

1. Den Mindestabstand (Meter) zu Straßensegmenten
2. Die Mindesthöhe (Meter) über dem Meeresspiegel

Jede Datenreihe die dann folgt, hat diese Parameter:

Beispiel:

["MAP_c_fern",1,0,0,0,0,"",[0,"ColorGreen"]]

- Ein gültiger Objekttyp (String)
- Wahrscheinlichkeit der Platzierung. Je größer dieser Wert ist, umso höher ist die Wahrscheinlichkeit, dass dieses Objekt benutzt wird.
- Die Objektdrehung. 0 = zufällige Drehung / >0 = exakte Drehung.
- Die Schrägstellung des Objekts. Nur Werte von 0 bis 1 gültig.
- Die Objektversenkung. <0 = exakter Wert / >0 = zufälliger Wert
- Die Bodenanpassung. 0 = am Boden angepasst / 1 = vertikal ausgerich.
- Funktionsaufruf / Skriptaufruf für das generierte Objekt (String)
- Der Markertyp. 0 = Ellipse / 1 = Rechteck
- Die Markerfarbe

- Bitte beachte, dass sich nicht alle Objekte an dem Boden anpassen lassen.
- Um mit dem Parameter **Funktionsaufruf / Skriptaufruf** ein generiertes Objekt an ein Skript zu übergeben, kannst Du die Variable **_obj** als Referenz benutzen: " [**_obj**] execVM "myObjFunc.sqf" " ... zum Beispiel
- Die Markereinstellungen (letztes Array) sind **optional** und müssen nicht angegeben werden.

[DAC_Config_Sound]

In der **DAC_Config_Sound** sind die Sound-Files eingetragen, die in der DAC_Sound.pbo enthalten sind.

```
DAC_SayArrayE = [
    /* reach waypoint */ [100,"r01","r02","r03","r04","r05","r06","r07","r08","r09","r01_0","r01_1","r01_2","r01_3","r01_4","r01_5","r01_6","r01_7","r01_8","r01_9","r01_10","r01_11","r01_12","r01_13","r01_14","r01_15","r01_16","r01_17","r01_18","r01_19","r01_20","r01_21","r01_22","r01_23","r01_24","r01_25","r01_26","r01_27","r01_28","r01_29","r01_30","r01_31","r01_32","r01_33","r01_34","r01_35","r01_36","r01_37","r01_38","r01_39","r01_40","r01_41","r01_42","r01_43","r01_44","r01_45","r01_46","r01_47","r01_48","r01_49","r01_50","r01_51","r01_52","r01_53","r01_54","r01_55","r01_56","r01_57","r01_58","r01_59","r01_60","r01_61","r01_62","r01_63","r01_64","r01_65","r01_66","r01_67","r01_68","r01_69","r01_70","r01_71","r01_72","r01_73","r01_74","r01_75","r01_76","r01_77","r01_78","r01_79","r01_80","r01_81","r01_82","r01_83","r01_84","r01_85","r01_86","r01_87","r01_88","r01_89","r01_90","r01_91","r01_92","r01_93","r01_94","r01_95","r01_96","r01_97","r01_98","r01_99"],
    /* detect enemys */ [100,"r11","r12","r13","r14","r15","r16","r17","r18","r19","r20","r21","r22","r23","r24","r25","r26","r27","r28","r29","r30","r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* attack enemys */ [100,"r21","r22","r23","r24","r25","r26","r27","r28","r29","r30","r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* call for help */ [100,"r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* lost contact */ [100,"r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* call for arti */ [100,"r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* help positiv */ [100,"r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
];

DAC_SayArrayW = [
    /* reach waypoint */ [100,"u11","u12","u13","u14","u15","u16","u17","u18","u19","u20","u21","u22","u23","u24","u25","u26","u27","u28","u29","u30","u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* detect enemys */ [100,"u21","u22","u23","u24","u25","u26","u27","u28","u29","u30","u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* attack enemys */ [100,"u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* call for help */ [100,"u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* lost contact */ [100,"u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* call for arti */ [100,"u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* help positiv */ [100,"u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
];

DAC_SayArrayD = /* soldier die */ [100,"d01","d02","d03","d04","d05","d06","d07","d08","d09","d10","d11","d12","d13","d14","d15","d16","d17","d18","d19","d20","d21","d22","d23","d24","d25","d26","d27","d28","d29","d30","d31","d32","d33","d34","d35","d36","d37","d38","d39","d40","d41","d42","d43","d44","d45","d46","d47","d48","d49","d50","d51","d52","d53","d54","d55","d56","d57","d58","d59","d60","d61","d62","d63","d64","d65","d66","d67","d68","d69","d70","d71","d72","d73","d74","d75","d76","d77","d78","d79","d80","d81","d82","d83","d84","d85","d86","d87","d88","d89","d90","d91","d92","d93","d94","d95","d96","d97","d98","d99"];
```

Es können Sounds auf Seite Ost (**DAC_SayArrayE**) und West (**DAC_SayArrayW**) eingetragen werden, die dann jeweils der Anführer einer Gruppe in bestimmten Situationen von sich gibt.

DAC hält dafür **7** verschiedene Situationen bereit. Deshalb sind es auch je 7 Arrays, die zu Verfügung stehen. Die jeweilige Situation kannst Du dem entsprechenden Kommentar vor jedem Array entnehmen.

Dann gibt es noch das Array **DAC_SayArrayD**. Dort sind Sounds eingetragen die eine Einheit von sich gibt, wenn sie um's Leben kommt ;-)

```
DAC_RadioArrayW = [
    /* reach waypoint */ [100,"c103","c104","c105","c106","c107","c108","c109","c110","c111","c112","c113","c114","c115","c116","c117","c118","c119","c120","c121","c122","c123","c124","c125","c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* getin order */ [100,"c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* getout order */ [100,"c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* detect infantry */ [100,"c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* detect vehicle */ [100,"c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* attack enemy */ [100,"c012","c013","c014","c015","c016","c017","c018","c019","c020","c021","c022","c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* call for help */ [100,"c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* call for arti */ [100,"c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* help positive */ [100,"c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* help negative */ [100,"c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* arti positive */ [100,"c001","c002","c003","c004","c005","c006","c007","c008","c009","c010","c011","c012","c013","c014","c015","c016","c017","c018","c019","c020","c021","c022","c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* lost contact */ [100,"c092","c093","c094","c095","c096","c097","c098","c099"],
    /* start searching */ [100,"c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* back to zone */ [100,"c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* use smoke genade */ [100,"c118","c119","c120","c121","c122","c123","c124","c125","c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* soldier down */ [100,"c094","c095","c096","c097","c098","c099"],
    /* arti negative */ [100,"c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
];
```

Neu sind die Arrays **DAC_RadioArrayW** (für die Seite West) und **DAC_RadioArrayE** (für die Seite Ost), in denen Funksprüche zu verschiedenen Situationen eingetragen sind. Kommt eine Gruppe in eine solche Situation, wird der Anführer der Gruppe einen entsprechenden Funkspruch loslassen, den Du dann hören kannst.

Je mehr Einträge in einem der Arrays eingetragen sind, umso mehr Variation gibt es.

Bei allen Arrays ist der erste Wert ein numerischer Wert, der überall die gleiche Bedeutung hat. Er bestimmt die Wahrscheinlichkeit, mit der ein Sound wiedergegeben wird. Ein Wert von **100** entspricht dabei 100%. Insofern kannst Du auch den Sound zu einer bestimmten Situation ganz deaktivieren, indem Du dort eine **0** einträgst.

[DAC_Config_Weapons]

In der **DAC_Config_Weapons** kannst Du Dir beliebig viele Waffenkonfigurationen anlegen und diese bestimmten Einheitentypen zuweisen. Bei den entsprechenden Einheiten werden dann, sobald sie generiert wurden, die Standard-Waffen, die Munition und Ausrüstungsgegenstände gelöscht, und durch die Konfiguration Deiner Wahl ersetzt.

Um einem bestimmten Einheitentyp mit einer neuen Waffenkonfiguration auszustatten, musst Du in der Datei **DAC_Config_Units** einen kleinen Eintrag machen, denn dort sind ja die Einheitentypen definiert.

```
case 1:
{
    _Unit_Pool_S = [
        "USMC_Soldier_Crew", "USMC_Soldier_Pilot", "USMC_Soldier_SL", "USMC_Soldier_HAT",
        "USMC_Soldier_AR", "USMC_Soldier_MG", "USMC_Soldier_Medic", "USMC_Soldier_GL",
        "USMC_Soldier_AT", "USMC_Soldier_LAT", ["USMC_Soldier_Medic", 2], "USMC_SoldierS_Sniper", "USMC_SoldierS_SniperH", "USMC_SoldierS_Spotter",
        "USMC_SoldierS_Engineer", "USMC_SoldierM_Marksman", "USMC_Soldier_TL", "USMC_Soldier_MG"
    ];
    _Unit_Pool_V = [ "HMMWV", "HMMWV_M2", "MTVR", "HMMWV_MK19", "HMMWV_Armored", "MTVR", "LAV25", "HMMWV_TOW", "HMMWV_Avenger" ];
    _Unit_Pool_T = [ "M1A1", "LAV25", "M1A2_TUSK_MG", "MLRS", "AAV" ];
    _Unit_Pool_A = [ "AH1Z", "UH1Y", "MH60S" ];
};
```

```
case 2:
{
    _Weapon_Pool = [ "M16A4_ACG_GL", "NVGoggles", "Binocular", "ItemCompass", "ItemMap" ];
    _Magazine_Pool = [ ["30Rnd_556x45_Stanag", 6], ["1Rnd_HE_M203", 6], ["HandGrenade_West", 4] ];
};
```

Wie oben in der Abbildung zu sehen, muss der entsprechende Einheitentyp in ein Array eingetragen werden, und dieses Array enthält dann zusätzlich die entsprechende **ID** aus der **DAC_Config_Weapons**:

Normaler Eintrag in der DAC_Config_Units	:	"USMC_Soldier_Medic"
Erweiterter Eintrag in der DAC_Config_Units	:	["USMC_Soldier_Medic", 2]
Mit Angabe der zu ladenden Waffenkonfiguration		

Es ist zudem möglich, über einen Skriptaufruf einzelnen Einheiten (auch Spieler-Einheiten) eine neue Waffenkonfiguration zuzuweisen (Beispiel):

```
[player, "2"] spawn DAC_Weapons
```

Die Parameter:

1. Die Einheit, die eine neue Waffenkonfiguration bekommen soll
2. Die ID aus der Waffenkonfigurations-Datei (String)

Achtung,

diese Konfigurations-Datei habe ich aus Zeitgründen noch nicht an Arma3 angepasst. Ob die Funktion noch richtig ausgeführt wird, kann ich nicht sagen.

Das Benutzen dieser Funktion erfolgt demnach auf eigene Gefahr.

Tips, Tricks und Regeln

Bitte achte immer darauf, dass für eine DAC-Zone (Auslöser) die richtigen Grundeinstellungen gemacht wurden: **Rechteck, mehrfach, vorhanden, Spiellogik** (siehe Seite 2)

Die Auslöserbedingung muss bei allen Zonen gleich sein, mit Ausnahme von **true** / **false**

In der Regel ist die Auslöserbedingung **true** ausreichend. Du kannst aber auch, um z.B. die DAC-Initialisierung zu verzögern, bei allen Zonen folgendes eintragen: **time > 15**

Der DAC startet dann erst, wenn die Mission 15 Sek. alt ist. Eine andere Variante wäre, den DAC erst zu starten, wenn eine bestimmte Bedingung erfüllt ist. Diese Bedingung muss dann in allen DAC-Zonen eingetragen werden.

Es ist sinnvoll, eigene (intensive) Skripte erst **nach** der Initialisierung von DAC zu starten. Um festzustellen, wann genau der DAC mit seiner Arbeit fertig ist, musst Du folgende Variable abfragen:

waituntil{DAC_Basic_Value == 1} innerhalb von **sqf**-Skripten

In der **DAC_Config_Camp** gibt es die Konfiguration **3**, die den KI-Respawn auch ohne Camp und ohne Camp-Gruppe möglich macht. DAC generiert dabei nur eine Position, die grundsätzlich für einen Respawn benötigt wird. Die Position kannst Du bei Bedarf, und in Verbindung mit einem benutzerdefinierten Wegpunkt, exakt bestimmen.

Benutzerhinweis & Danksagung

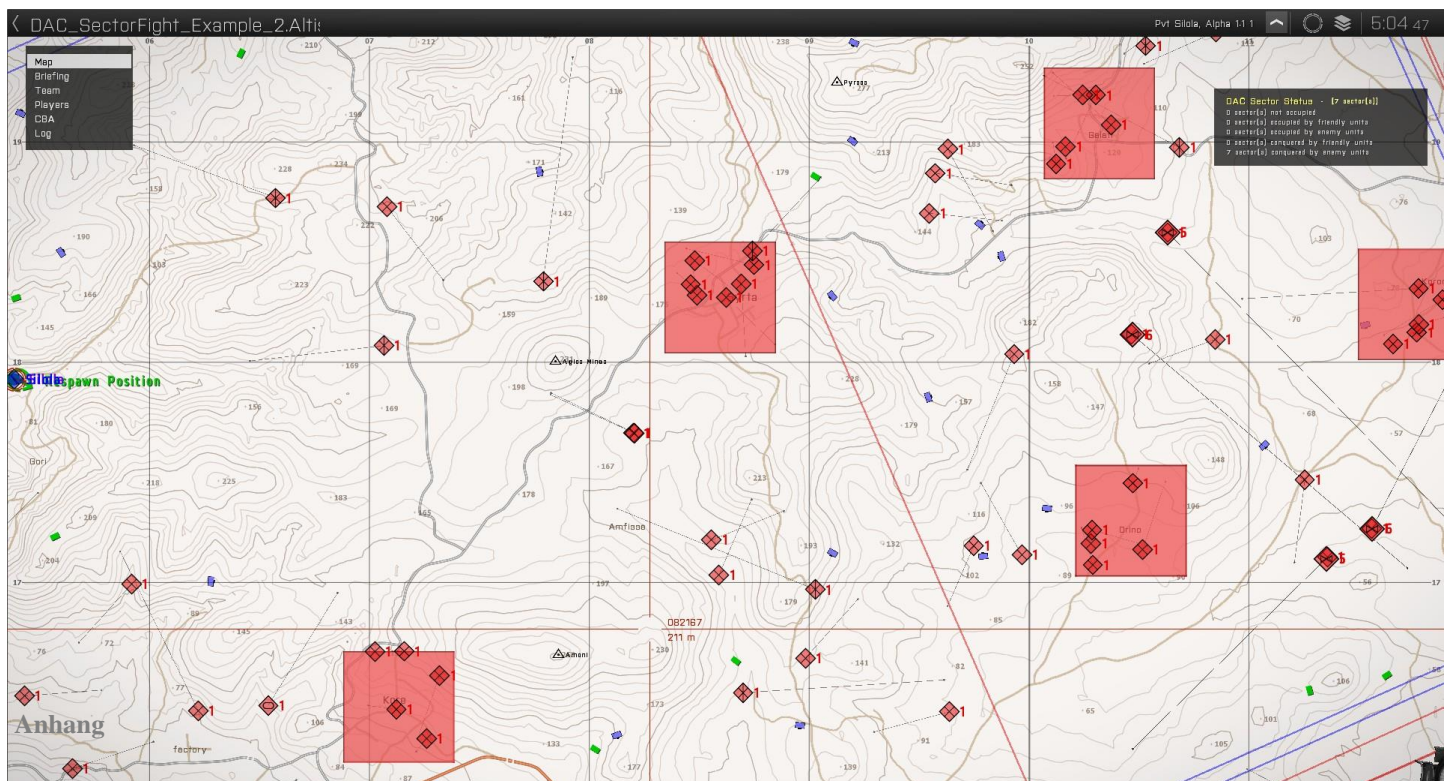
Der DAC ist **kein offizielles Addon**. Bitte keine Beschwerden an BIS oder deren Publisher.

Achtung - Die Benutzung von DAC V3.1 für Arma3 erfolgt auf eigene Gefahr!

An dieser Stelle noch ein **großes Dankeschön** an meine Tester und Unterstützer, mit denen ich wochenlang DAC-Missionen getestet habe und dadurch viele Fehler beheben konnte.

Danke an: **MCPXXL** | **SKH|Flip** | **SKH|Cyborg** | **SKH|Hydra** | **t-800a** | **Lester**

Vielen Dank auch an Bohemia, die mit Arma3 wieder ein wunderbares Stück Software entwickelt haben!



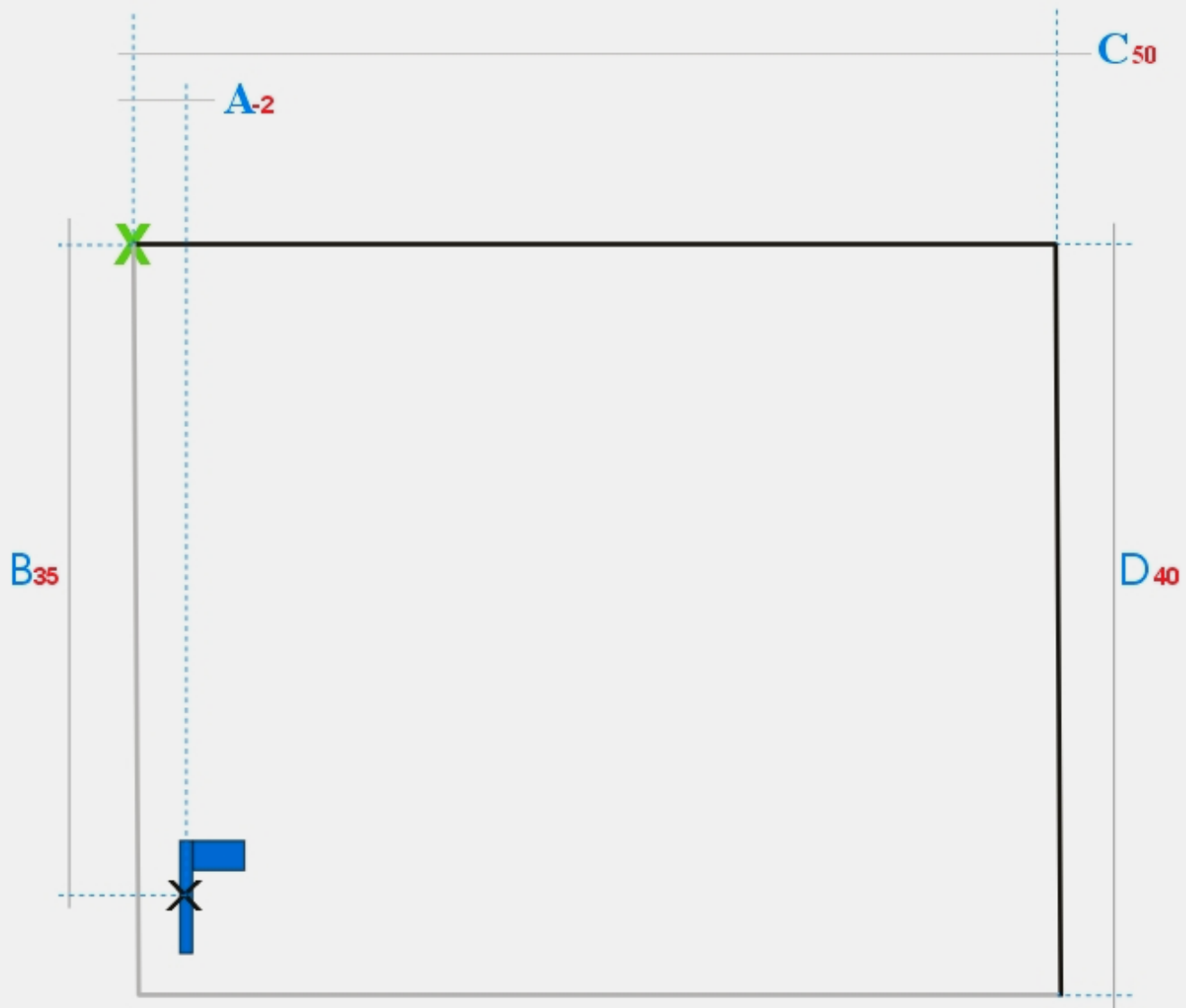
Hier die erwähnte „bildliche“ Beschreibung zu dem Array `_campWall` aus dem Skript `DAC_Config_Camp`:

$\begin{array}{c} \text{X} \\ \swarrow \quad \searrow \\ \text{A} \quad \text{B} \quad \text{C} \quad \text{D} \end{array}$

```
_campWall = ["FenceWood",[-2,35],[50,40,1],[7,0,0,4],[1,0.1],[1,90]]
```

Die Parameter **A+B** bestimmen die Startposition **X** für die Einfassung des Camps. Diese Einfassung wird immer im Uhrzeigersinn generiert. Referenzposition ist immer die Flagge (bei dem **X**).

Die Parameter **C+D** sind für die Größe der Einfassung zuständig. **C** = **x**-Ausdehnung, **D** = **y**-Ausdehnung

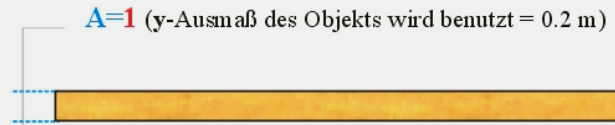


$_campWall = ["FenceWood", [-2,35], [50,40, \overset{A}{\underset{0/1}{0}}, 1], [7,0,0,4], [1,0.1], [\overset{BC}{\underset{0}{0}}, \underset{0}{0}]]$

Das erste Beispiel zeigt ein Mauer-Segment, das mit der breiten Seite nach Norden zeigt. Dieser Zustand ist für uns optimal, da wir nur wenig anpassen müssen. Alle Parameter (A+B+C) können auf 0 stehen bleiben.

(Achtung, um festzustellen wie ein Objekt grundsätzlich ausgerichtet ist, mußt Du es nur einmal im Editor platzieren.)

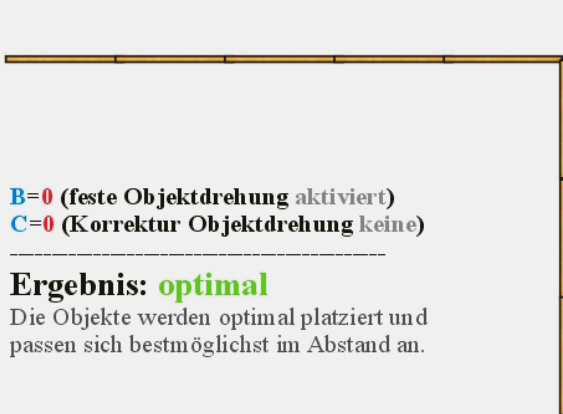
Das Ergebnis siehst Du im linken Bild. Das rechte Bild zeigt das Ergebnis, wenn der Parameter A den Wert 1 enthält. Die Objekte werden zwar richtig aufgebaut, jedoch mit einem viel zu geringen Abstand. Der Abstand entspricht dem y-Ausmaß.



$B=0$ (feste Objektdrehung aktiviert)
 $C=0$ (Korrektur Objektdrehung keine)

Ergebnis: optimal

Die Objekte werden optimal platziert und passen sich bestmöglichst im Abstand an.



$B=0$ (feste Objektdrehung aktiviert)
 $C=0$ (Korrektur Objektdrehung keine)

Ergebnis: sehr schlecht

Es werden sehr viele Objekte generiert. Anna kann auf schwachen Rechnern u.U. zusammenbrechen.



$_campWall = ["FenceWood", [-2,35], [50,40, \overset{A}{\underset{0/1}{0}}, 1], [7,0,0,4], [1,0.1], [\overset{BC}{\underset{0}{0}}, \underset{0}{0}]]$

Dieses Beispiel zeigt genau die gleichen Einstellungen, allerdings ist das Objekt nun mit der schmalen Seite nach Norden ausgerichtet. Du kannst deutlich sehen, welche Auswirkungen ein solches Objekt auf das Generieren hat.

Im linken Bild werden die Objekte zwar exakt aneinander gereiht, allerdings stimmt die Objektausrichtung nicht. Im rechten Bild ist im Prinzip der Abstand ok, nur fehlt auch hier die richtige Objektdrehung.



$B=0$ (feste Objektdrehung aktiviert)
 $C=0$ (Korrektur Objektdrehung keine)

Ergebnis: extrem schlecht

Es werden sehr viele Objekte generiert. Anna kann auf schwachen Rechnern u.U. zusammenbrechen.



$B=0$ (feste Objektdrehung aktiviert)
 $C=0$ (Korrektur Objektdrehung keine)

Ergebnis: schlecht

Die Objekte werden im richtigen Abstand generiert, aber die Objektdrehung passt nicht.



$_campWall = ["FenceWood", [-2,35], [50,40, \overset{A}{\underset{1}{1}}, 1], [7,0,0,4], [1,0.1], [\overset{BC}{\underset{0}{0}}, \underset{90}{90}]]$

```
_campWall = ["FenceWood",[-2,35],[50,40,0],[0,0,0,0],[1,0.1],[0,0]]
```

