Dynamic-AI-Creator [DAC] Version 2.0 Beta

Benötigt wird mindestens ArmA Version 1.08 / Addons: EditorUpgrade V1.02, MAP_Misc

Achtung, der DAC hat einen großen Funktionsumfang, entsprechend umfangreich ist diese Dokumentation ausgefallen. Wenn Du Dir die 5 Kapitel der 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 in DAC aufbaut.

Inhalt der Zip-Datei:

DAC Liesmich.pdf
DAC Kurzanleitung.pdf
DAC AI Behavior Plan.pdf (nur Infanterie und nur englisch)
DAC Demonstation (enthält die 5 Kapitel der DAC Demonstration)
DAC Examples (enthält mehrere DAC Beispiele)
DAC Core (enthält den DAC Ordner und eine Init-Datei)
EditorUpdate V1.02

Themenübersicht:

02 Hinweis zu DAC + den DAC vorbereiten
03 Der DAC- Scriptaufruf für eine Zone
06 Die Wahl der Seite, der Einheiten-, Verhaltens- und Camp-Konfiguration
08 Mehrere DAC-Zonen erstellen (kopieren)
09 DAC-Zonen verlinken
10 Wegpunktzonen erstellen
11 Benutzerdefinierte Wegpunkte einbinden
12 DAC-Zonen aktivieren / deaktivieren
13 Reduzierung von Infanterie-Gruppen
14 KI-Respawn aktivieren und konfigurieren
15 Camps mit Zonen verlinken
16 Zonenwerte (Größe, Position und Konfiguration) verändern
18 Die DAC-Arti
19 Editor-Gruppen einbinden
20 Die Konfigurations- Dateien
20 DAC_Config_Creator
24 DAC_Config_Units
25 DAC_Config_Behaviour
28 DAC_Config_Waypoints
29 DAC_Config_Camp
32 DAC_Config_Arti
34 DAC_Config_Events
36 DAC_Config_Marker
38 Tips, Tricks + Regeln
38 Danksagung
39 Anhang

Hinweis zu DAC

Diese Version von DAC V2.0 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.

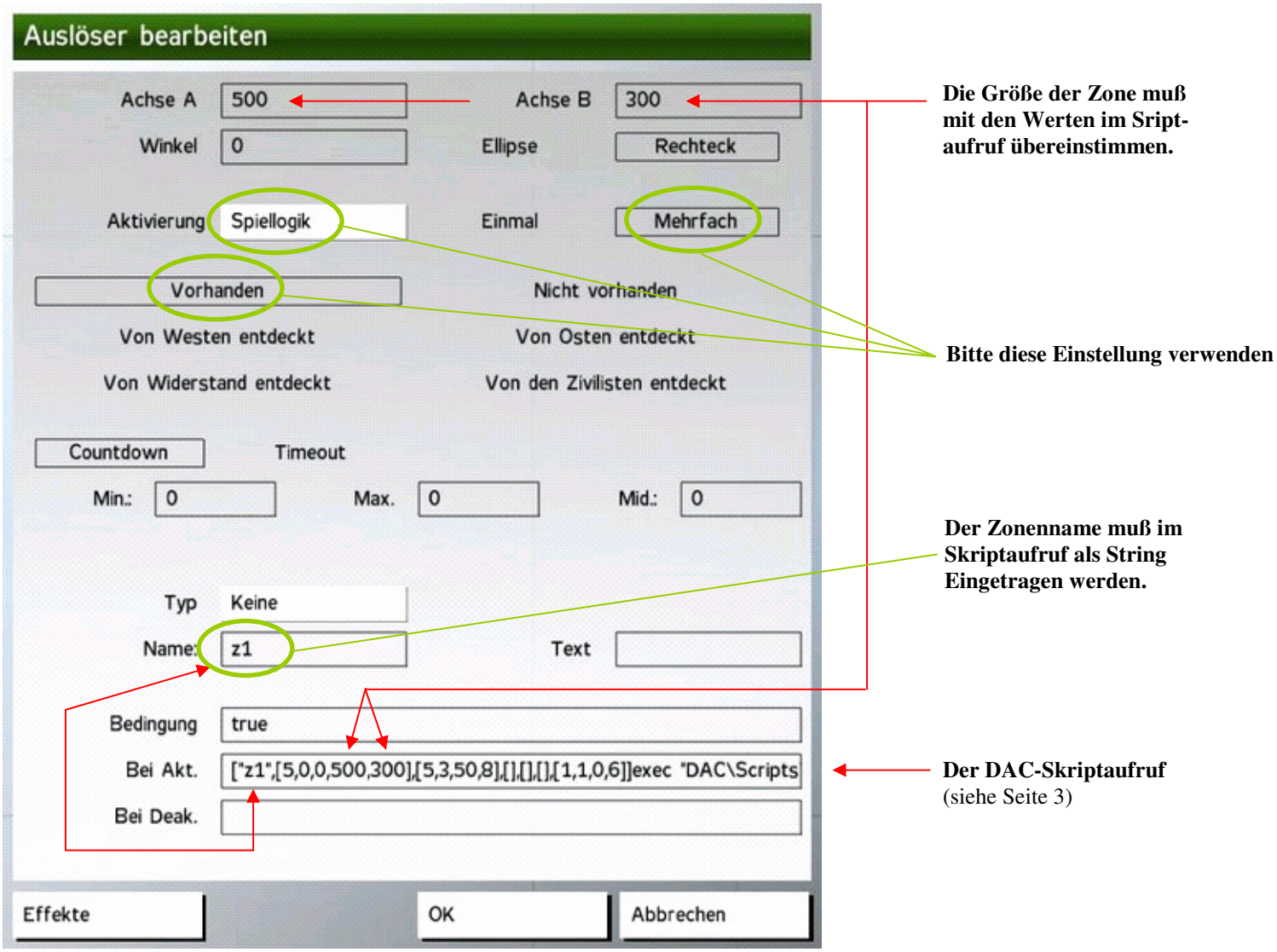
Es kann bei Problemen auch sehr hilfreich sein, wenn Du kritische Zonen für einen Test deaktivierst, in dem Du in die Bedingungszeile der Zone schreibst: **false** (Die Zone ist dann außer Betrieb)

Den DAC vorbereiten

DAC V2.0 benötigt keine Dummy-Einheiten mehr, und es müssen auch keine Marker mehr platziert werden. Du musst lediglich den DAC-Ordner in Deinen Missionsordner kopieren (**Nicht den Ordner aus den Demos**)

DAC muss mit dem Scriptaufruf **execVM "DAC\DAC_Config_Creator.sqf"** initialisiert werden. Ob der Scriptaufruf in der Init.sqs eingetragen wird, oder ob er im Initfeld einer Logic eingetragen wird, ist egal.

Zu dem Skriptaufruf braucht DAC noch mindestens einen Auslöser, also eine Zone, um starten zu können. So eine Zone muss wie folgt eingestellt werden (Die Größe, der Name und der Scriptaufruf sind variabel):



Achtung

Damit die Gruppenreduzierung und die Einheitenlöschung 100%ig funktionieren, musst Du die Spielereinheit bzw. alle möglichen Spielereinheiten einer MP-Mission, im Editor benennen und in einem Array eintragen.

Standard mäßig sind die Namen **s1** bis **s16** dort eingetragen. Das bedeutet, wenn Du Deinen Spieler „**s1**“ nennst, und weitere Spieler z.B. „**s2**,**s3**,**s4**“ musst Du keine Anpassung vornehmen.
Möchtest Du andere Namen verwenden, musst Du diese Namen im Array **DAC_STRPlayers** als String eintragen. Das Array findest Du im Script **DAC\DAC_Config_Creator.sqf**

Das Array **DAC_STRPlayers** bereinigt sich automatisch, dass heißt es können beliebig viele (auch ungültige) Einträge in dem Array stehen z.B. **DAC_STRPlayers = [“s1“,“s2“,“s3“,“w1“,“w2“,“w3“,“ap1“,“ap2“,”ap3”]**. Sobald die Mission gestartet wird, reduziert DAC dieses Array automatisch auf die gültigen Einträge.

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"[1,0,0,400,200],[ ],[ ],[ ],[ ],[1,1,0,6]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Wir befassen uns erstmal nur mit den 4 leeren Arrays, die ich farblich markiert habe.
Dort werden die Mengen für Einheiten 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",[1,0,0,800,500],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[0,0,0,0]exec "DAC\Scripts\DAC_Init_Zone.sqs"

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",[1,0,0,300,200],[5,2,50,6],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

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",[1,0,0,300,200],[ ],[3,3,30,5],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

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.

Die Anzahl der Fahrzeuge werden bei dieser Kategorie dynamisch erzeugt. Hier ein Beispiel:
Es wird eine Gruppe mit 9 Einheiten generiert. Erst nachdem dieser Vorgang abgeschlossen ist, werden die Fahrzeuge erzeugt. Wird z.B. erst ein Jeep generiert, passen da 4 Einheiten rein. Jetzt wird das nächste Fahrzeug generiert. Passen die restlichen 5 Einheiten dort rein, werden keine weiteren Fahrzeuge generiert.
Würde aber z.B. wieder ein Jeep generiert, bleibt immer noch eine Einheit übrig, die keinen Platz findet. Diese Situation führt dazu, dass ein weiteres Fahrzeug generiert werden muss.

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",[1,0,0,300,200],[ ],[ ],[3,3,25,5],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

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",[1,0,0,300,200],[ ],[ ],[ ],[2,2,4],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

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 hellblaue Markierung).

["z1",[1,0,0,300,200],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[0,0,0,0,0]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

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

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

Die Wahl der Einheiten für eine Zone wird auch über eine Nummer geregelt. Eine Nummer entspricht dabei einem Eintrag in dem Script **DAC\Configs\DAC_Config_Units.sqf**. Siehe folgende Abbildung:

```
//--------------------------------------------------------------------------------------------
    case 0:
    {
        _Unit_Pool_S =  [
                            "SoldierECrew","SoldierEPilot","SquadLeaderE","Soldiereb",
                            "Soldiereg","Soldiereat","Soldieremg","Soldiereaa","Soldiereat",
                            "Soldiereb","Soldieremedic","Soldieremg","Soldieresniper",
                        ];
        _Unit_Pool_V =  [   "UAZMG","Ural","UAZ","UAZ_AGS30","BRDM2","UAZMG"];
        _Unit_Pool_T =  [   "BMP2","T72","T72","ZSU","BMP2","ZSU","BMP2"];
        _Unit_Pool_A =  [   "MI17","KA50"];
    };
//--------------------------------------------------------------------------------------------
    case 1:
    {
        _Unit_Pool_S =  [
                            "SoldierWCrew","SoldierWPilot","SquadLeaderW","Soldierwb",
                            "Soldierwg","Soldierwmg","Soldierwar","Soldierwmg","Soldierwat",
                            "Soldierwmedic","Soldierwat","Soldierwsniper","SoldierWSaboteur","SquadLeaderW",
                        ];
        _Unit_Pool_V =  [   "HMMWV50","HMMWV","Truck5t","HMMWVMK","Stryker_TOW","Truck5tOpen","Truck5tMG"];
        _Unit_Pool_T =  [   "M1Abrams","Vulcan","M113","Vulcan","Stryker_ICV_M2"];
        _Unit_Pool_A =  [   "AH1W","UH60","MH6"];
    };
//--------------------------------------------------------------------------------------------
```

In der Abbildung oben siehst Du z.B. die Konfiguration **0** (case 0:) und die Konfiguration **1** (case 1:)
Dieses Script kann beliebig erweitert werden. Man muss nur darauf achten, dass jeder Einheitenblock eine eigene eindeutige (Case-) Nummer hat. Näheres findest Du unter dem Thema : **Die Konfigurations- Dateien**.

Um in einer Zone bestimmte Einheiten zu erstellen, musst Du also nur die entsprechende Konfig-Nr. in dem Scriptaufruf angeben. In jeder Zone kann dabei eine andere Konfig-Nr. angesprochen werden.

Die Standard-Einheiten findest Du unter folgenden Nummern:

0 = Standard Ost-Einheiten | **1** = Standard West-Einheiten | **2** = Standard Res-Einheiten | **3** = Zivil-Einheiten

Achtung:
Bitte beachte, dass Du bei Addon-Einheiten das Addon geladen haben musst, um die Einheiten mit dem DAC generieren zu können. Außerdem gibt es Probleme, wenn Du Einheiten auf falscher Seite generierst.

Die Wahl der Verhaltens-Konfiguration
Dieser Parameter bestimmt das Grundverhalten der Einheiten in einer Zone. Grundverhalten bedeutet, wie sich die Einheiten auf Ihren Wegen zu den Wegpunkten verhalten, ohne Einfluss durch Feindeinheiten.

Auch hier wird wieder durch eine Nummer die Konfiguration für das Verhalten gewählt.
In dem Script **DAC\Configs\DAC_Config_Behaviour.sqf** findest Du verschiedene Konfigurationen dafür.
Dieses Script lässt sich auch wieder um beliebig viele Einträge erweitern.

Jede Zone kann dann mit einem individuellen Grundverhalten konfiguriert werden.

Die Wahl der Camp-Konfiguration
Der letzte Parameter ist zuständig für die Camp-Konfiguration. Insofern ist er nur relevant, wenn in der Zone mindestens ein Camp generiert wird (Der Parameter muss auch vorhanden sein, wenn kein Camp generiert wird).

Analog zu der Verhaltens-Konfiguration, wird auch hier wieder eine Nummer eingetragen, die aus dem Skript **DAC\Configs\DAC_Config_Camps.sqf** die entsprechende Konfiguration lädt.

In der Camp-Konfiguration ist der komplette Aufbau eines Camps definiert.

Die Wahl der Wegpunkt-Konfiguration
Dieser Parameter ist **optional** und muss nicht zwingend angegeben werden. Er legt fest, welche Wegpunkt-Konfiguration für die Zone benutzt wird.

Standard- mäßig wird immer die Wegpunkt- Konfiguration **0** benutzt, die in den meisten Fällen ausreichend ist. Nur wenn erforderlich (wenn z.B. Wegpunkte in einer gewissen Höhe generiert werden sollen) kannst Du hier eine andere Konfiguration angeben, also einen gültigen Wert aus der **DAC\Configs\DAC_Config_Waypoints.sqf**.

Hier noch mal der Scriptaufruf mit Kurzerklärung:

["z1",[1,0,0,300,200],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[1,1,2,7,2]]exec "DAC\Scripts\DAC_Init_Zone.sqs"



Eine ausführliche Beschreibung, wie man das Grundverhalten bzw. die Camps + Wegpunkte konfiguriert und welche Möglichkeiten es gibt, findest Du auch wieder unter dem Thema **Die Konfigurations- Dateien**.

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

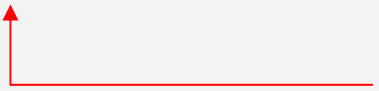
Tipp: Verwende kurzen Namen, damit im Eingabefeld nicht zu viel vom Scriptaufruf verschwindet.

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",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

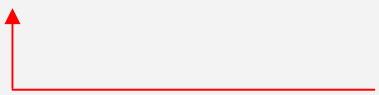


3 Zonen, die sich nur im Namen unterscheiden (müssen)

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 abändern. Dieser Parameter ist sozusagen die **ID-Nr.** der Zone.
Gib' jeder Zone eine andere Nummer, also so…

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[2,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[3,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```



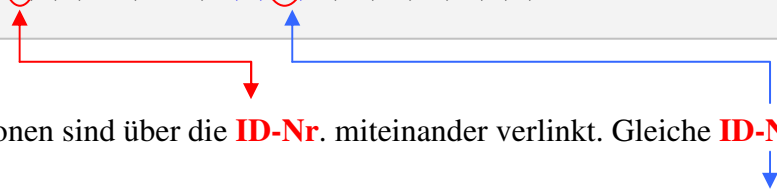
Das ist die **ID-Nr**. der Zonen

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",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```



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",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,15,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[2,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z4",[2,0,0,300,200],[ ],[ ],[3,1,35,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z4",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,300,200],[15,2,10,15],[ ],[20],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[20],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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 feste Positionen einbinden möchte, die dann den entsprechenden Einheiten zur Auswahl zu Verfügung stehen.

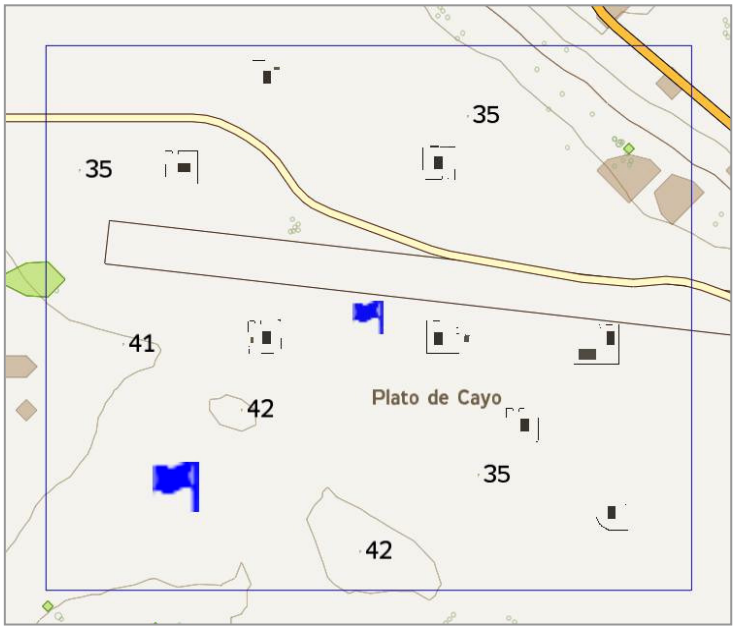
Das können z.B. Positionen in einer Ortschaft sein, oder auch Missionsrelevante Positionen.
Durch diese Funktion ergeben sich 3 Möglichkeiten, Zonen mit Wegpunkten zu füllen:

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

Das Einbauen von benutzerdefinierten Wegpunkten ist sehr einfach:

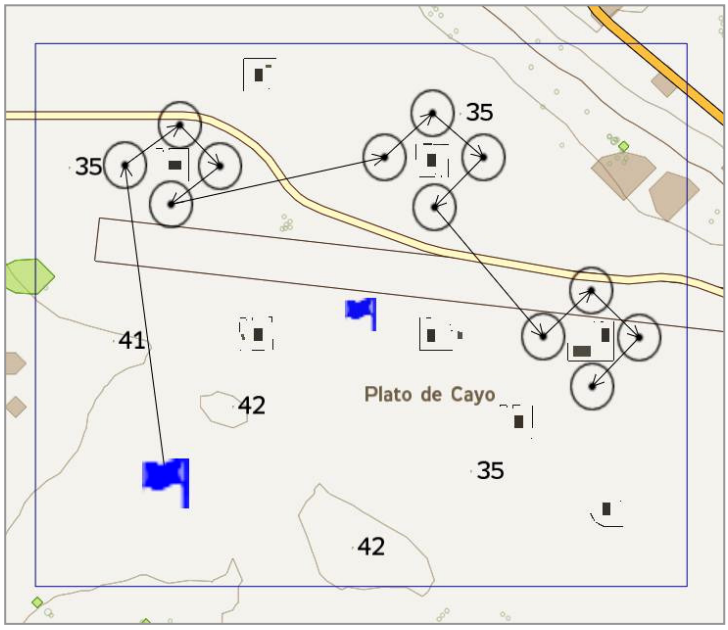
Du platzierst in der entsprechenden Zone eine Logic-Einheit.

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



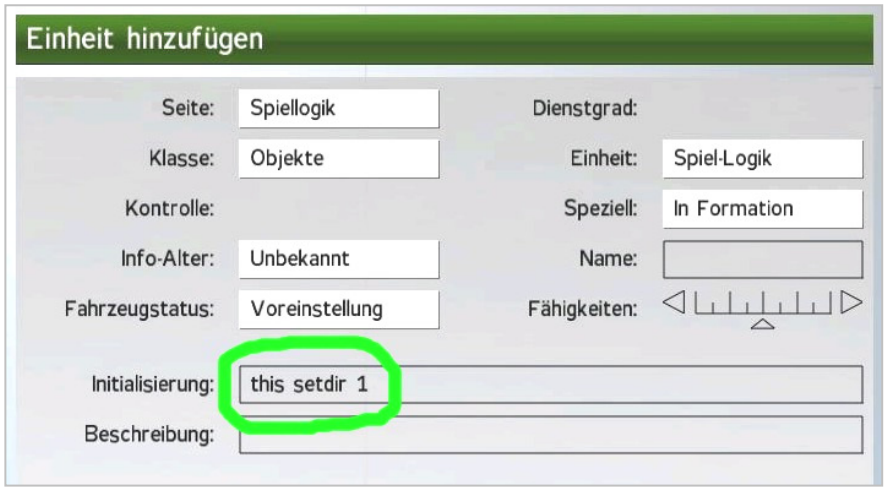
Dieser Logic-Einheit gibst Du nun an den bestimmten Positionen ein paar Wegpunkt

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**

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.

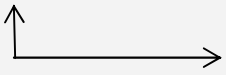
Eine Zone ist nicht automatisch deaktiviert, wenn alle Einheiten aus dieser Zone down sind. Du kannst eine solche Zone jeder Zeit wieder zum Leben erwecken, in dem Du sie einmal deaktivierst, und anschließend wieder aktivierst.

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)

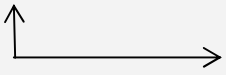
Möchtest Du, dass eine Zone von Beginn an deaktiviert ist, weil sie erst im späteren Missionsverlauf benötigt wird, dann kannst Du folgenden Parameter dafür benutzen:

["z1",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"



Die Zone ist von Beginn an **aktiv**

["z1",[1,1,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"



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.

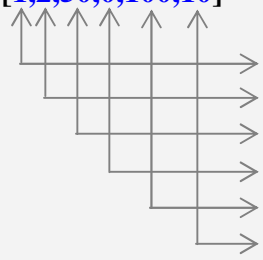
Respawned werden nur folgende Kategorien: **Infanterie**, **Radfahrzeuge**, **Kettenfahrzeuge**

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

```
["z1",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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** Respawns zu Verfügung, egal von welchem Typ. Die Zone verfügt demnach über insgesamt **15** Respawns. 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 Respawns 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 Respawns 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",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z2]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z4]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z4,z5]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

["z2",[2,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z6,z7]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

["z3",[3,0,0,800,400],[],[],[],[ 2,2,50,0,100,10,[z5,z7]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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 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 Infatrerie-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, 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 Größenangabe	für eine mögliche Größenänderung [x,y]
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] 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] 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 !

Die DAC-Arti

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 Logic, 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

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

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.

Folgende Kategorien werden dabei unterstützt:
Infanterie, **Radfahrzeuge**, **Kettenfahrzeuge**

Sobald eine Gruppe in das DAC-System integriert 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.

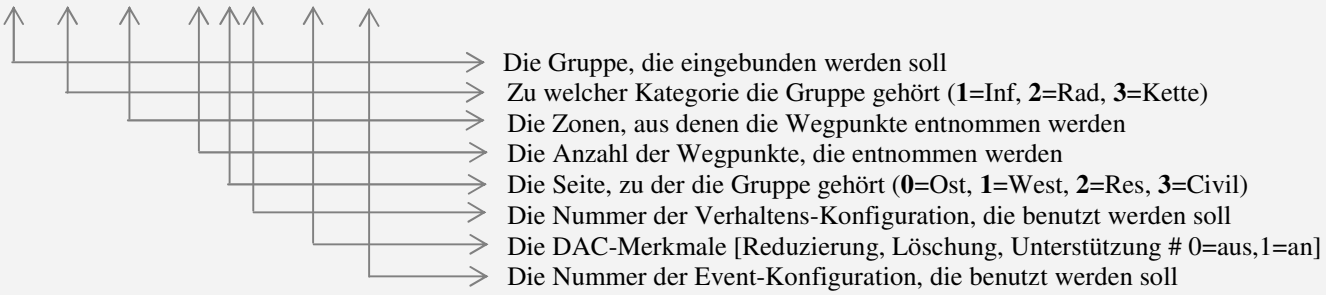
Es gibt aber auch ein paar Einschränkungen:

- Die Einbindung in das DAC-System, lässt sich nicht wieder rückgängig machen.
- Die einzubindende Gruppe darf keine eigenen Wegpunkte besitzen.
- 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] **spawn DAC_fInsertGroup**



Ein kleines Anwendungsbeispiel:

Du platzierst im Editor einen Heli, mit Platz für eine Infanterie-Gruppe (evtl. den Treibstoff auf 0 setzen, damit der Heli erstmal da bleibt, wo er ist ;-)
Dann platzierst Du daneben noch eine Infanterie-Gruppe, die Du per MoveIn-Befehl in den Heli beamst.
In die Init-Zeile des Anführers trägst Du ein: **HeliGroup1 = group this**
In einer bestimmten Situation lässt Du den Heli in's Missionsgebiet einfliegen (Treibstoff nicht vergessen), und lässt dort die Infanterie-Gruppe aussteigen. Das wäre dann der richtige Zeitpunkt, die Gruppe in DAC einzubinden (damit sie nicht blöd rumsteht ☺):

[HeliGroup1,1,[z1],10,1,1,[1,1,1],0] **spawn DAC_fInsertGroup**

Die Konfigurations- Dateien

Die Konfigurations-Dateien sind nichts anderes als „stinknormale“ Skripte die, bis auf die **DAC_Config_Creator**, von Dir beliebig erweitert, bzw. individuell angepasst werden können.

Der DAC wird von ihnen sozusagen mit Daten und verschiedenen Parametern gefüttert :-)
Insofern sind sie ein sehr wichtiger Bestandteil des ganzen Systems.

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 Event-Konfiguration, erlaubt den Eingriff an verschiedenen Stellen im DAC

Nachfolgend die jeweiligen Konfigurations-Dateien im Detail:

[DAC_Config_Creator]

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 Spieler eingetragen werden. Achtung, die Einträge müssen als String vorliegen.
Ungültige Einträge werden automatisch gelöscht

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",[1,0,0,300,200],[5,**2**,50,6],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

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

DAC_Dyn_Weather = [240,60,1,0.6**]**

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**]**

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** = Sekunden mindestens + Sekunden zufällig + Sperrzeit, bis Camp wieder freigegeben wird
2. Respawnzeit für **Fahrzeuge** = Sekunden mindestens + Sekunden 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.

DAC_Delete_Value = [30,150**]**

Dieses Array bestimmt das Löschen von toten Einheiten. Du kannst die Bedingung dafür 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 Sekunden, wann eine tote Einheit frühestens gelöscht werden darf
2. Mindestabstand zu allen Spielereinheiten, bevor eine tote Einheit gelöscht wird

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**]**

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

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-Ordnern 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 RACS kämpfen. Diese Einstellung musst Du unbedingt analog zu den Einstellungen im Editor vornehmen.

0 = Widerstand kämpft auf Seite Ost, Editor: RACS freundlich zu OSTEN einstellen
1 = Widerstand kämpft auf Seite West, Editor: RACS freundlich zu WESTEN einstellen
2 = Widerstand kämpft auf eigener Seite, Editor: RACS freundlich zu NIEMANDEM einstellen
Achtung, bei dieser Einstellung kämpfen RACS 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_GunAllowed = ["M2StaticMG","M119","DSHKM","AGS","D30"**]**

Hier müssen die Typen der statischen Waffen eingetragen werden, die (wenn sie leer sind) besetzt werden können.
Die Funktion selbst, wird in der **DAC_Config_Behaviour** eingestellt.

DAC_SayArrayE = [[ ],[ ],[ ],[ ],[ ],[ ],[ ]]
DAC_SayArrayW = [[ ],[ ],[ ],[ ],[ ],[ ],[ ]]

Hier können Sounds auf Seite Ost und West eingetragen werden, die dann jeweils der Anführer einer Gruppe in bestimmten Situationen von sich gibt. (Achtung, die Sounds müssen in der Description.ext definiert sein). Die Sounds werden übrigens auch im MP wiedergegeben ☺

DAC hält dafür **7** verschiedene Situationen bereit. Deshalb sind es auch je 7 Arrays, die zu Verfügung stehen.

Folgende Situationen gibt es:

1. Die Gruppe erreicht einen ihrer Wegpunkte
2. Die Gruppe entdeckt Feindeinheiten
3. Die Gruppe steht unter Beschuss
4. Die Gruppe fordert Hilfe an
5. Die Gruppe hat Feindkontakt verloren
6. Die Gruppe fordert Arti an
7. Die Gruppe bestätigt Unterstützung

Hier ein Beispiel (Info: hinter jedem Eintrag verbirgt sich ein Soundfile):

DAC_SayArrayE = [[“rus1“,“rus2“],[“rus3“, “rus4“ “rus5“],[ ],[“rus6“],[“rus9“, “rus11“],[ ],[“rus12“, “rus14“, “rus15“]**]**

- Der Leader der Gruppe ruft **rus1** oder **rus2** wenn er einen seiner Wegpunkte erreicht hat.
- Der Leader der Gruppe ruft **rus3**,**rus4** oder **rus5**, wenn seine Gruppe Feindeinheiten entdeckt.
- Der Leader der Gruppe ruft **nichts** wenn seine Gruppe unter Beschuss steht (das Array ist leer).
- Der Leader der Gruppe ruft **rus6** wenn seine Gruppe Hilfe anfordert.
- Der Leader der Gruppe ruft **rus9** oder **rus11** wenn seine Gruppe den Feindkontakt verliert.
- Der Leader der Gruppe ruft **nichts** wenn seine Gruppe Arti anfordert (das Array ist leer).
- Der Leader der Gruppe ruft **rus12**,**rus14** oder **rus15**, wenn seine Gruppe Unterstützung bestätigt.

Je mehr Einträge zu einer bestimmten Situation eingetragen sind, umso mehr Variation gibt es.

DAC_BadBuildings = ["CampEmpty","CampEast","Land_dum_istan4","Land_dum_istan4_detaily1"]

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

Bis auf die Arrays **DAC_SayArrayE + DAC_SayArrayW + DAC_BadBuildings**, kannst Du alle Einstellungen auch direkt über eine Logic vornehmen. Also einfach eine Logic auf die Karte platzieren und in die Init-Zeile dieser Logic schreibst Du z.B. rein:

DAC_AI_Level = 2

Dadurch wird automatisch der Wert im Skript **DAC_Config_Creator** übergangen.
So kannst Du ganz bequem einige Einstellungen direkt im Editor vornehmen.

[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 =  [
                        "SoldierECrew","SoldierEPilot","SquadLeaderE","Soldiereb",
                        "Soldiereg","Soldiereat","Soldieremg","Soldiereaa","Soldiereat",
                        "Soldiereb","Soldieremedic","Soldieremg","Soldieresniper",
                        "Soldiereaa","Soldiereminer","SoldierESaboteur","SoldierESaboteurPipe",
                        "SoldierESaboteurBizon","SoldierESaboteurMarksman"
                    ];
    _Unit_Pool_V =  [   "UAZMG","Ural","UAZ","UAZ_AGS30","BRDM2","UAZMG"];
    _Unit_Pool_T =  [   "BMP2","T72","ZSU","BMP2","ZSU","BMP2"];
    _Unit_Pool_A =  [   "MI17","KA50"];
};
```

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

["z1",[1,0,0,300,200],[5,3,30,8],[ ],[ ],[ ],[0,0,3,4]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

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   = [0.3,0.9];
    _setCombat  = ["white","yellow"];
    _setBehav   = ["safe","aware"];
    _setSpeed   = ["normal","full"];
    _setForm    = ["line","vee","column"];
    _setFleeing = [0,50];
    _setHeliVal = [45,100,0.7,1];
    _setPause   = [[3,5],[3,5],[5,10],[20,30,5,5],[0,1],[1,5]];
    _setBldgBeh = [0,50,120,600];
    _setPatrol  = ["25 + (100 * (skill _leader))","(30 + (random 30)) + ((skill _leader) * 50)"];
    _setSearch  = ["25 + (200 * (skill _leader))","(30 + (random 60)) + ((skill _leader) * 50)"];
    _setSupport = [1,2];
    _setJoin    = 2;
    _setEmpVeh  = [[100,50],[100,80]];
};
```

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

```
["z1",[1,0,0,300,200],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Die Einstellungen verhalten sich folgendermaßen:

_setSkill	[0.3,0.9] Dieses Array bestimmt den Bereich des Skill-Wertes, mit denen die Einheiten in der Zone generiert werden. **1.** Eintrag = der Minimal-Skill, den die Einheiten in dieser Zone bekommen. **2.** Eintrag = der Maximal-Skill, den die Einheiten in dieser Zone bekommen.
_setCombat	["white","yellow"] 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.
_setBehav	["aware",“combat“] 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
_setSpeed	["normal","full"] 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

_setForm	["line","vee","column"] 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.
_setFleeing	[0,50] Diese Parameter bestimmen das Fluchtverhalten und die Deckungssuche der Einheiten. **1.** Eintrag = Fluchtverhalten. Der Wert muss zwischen 0 und 1 liegen. **2.** Eintrag = Deckungssuche. Der Wert bestimmt den Radius, in dem die Einheiten unter bestimmten Bedingungen Deckung suchen (0 = deaktiviert).
_setHeliVal	[45,100,0.7,0] 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. **1.** Eintrag = Minimalflughöhe für Helis in der Zone **2.** Eintrag = Maximalflughöhe für Helis in der Zone **3.** Eintrag = Maximaler Wetterwert, bei dem die Helis noch fliegen, ansonsten landen sie vorzeitig **4.** Eintrag = EmptyH generieren AUS(0) / EmptyH generieren AN(1)
_setPause	[[3,5],[3,5],[5,10],[20,30,5,5],[3,5],[1,30]] In diesem Array sind die Pausen für die verschiedenen Einheiten-Kategorien definiert, die sie jeweils an ihren Wegpunkten einlegen. **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	[2,50,120,600] Hier wird festgelegt, in wieweit Infanterie-Gruppen Gebäude besetzen können. Möchtest Du dieses Merkmal nicht benutzen, musst Du den 1. Wert auf **0** stellen. **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 Oben das Beispiel würde also bedeuten, dass 2 Einheiten aus einer Gruppe ein Gebäude besetzen, wenn ein geeignetes Gebäude 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**)

_setPatrol	["50","60"] 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. **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. Achtung, die Werte in diesem Array müssen als String eingetragen werden. 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. Hier noch 2 weitere Beispiele: ["50 + random 50","60 + random 30"] ["30 * DAC_AI_Level","60 * DAC_AI_Level"] Eine Besonderheit ist hier, dass über die Variable **_leader**, der Anführer der Gruppe angesprochen werden kann. Demnach könnte das Array auch folgende Einträge haben (Beispiel): ["25 + (30 * (skill _leader))","(30 + (random 30)) + ((skill _leader) * 50)"] oder ["25 + (10 * (count units (group _leader)))","(20 * DAC_AI_Level) + ((skill _leader) * 50)"]
_setSearch	["100","120"] 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**. **1.** Eintrag = Maximaler Suchradius für die Gruppe **2.** Eintrag = Zeitfenster für die Suche, bis die Gruppe wieder ihre normalen Wegunkte aufnimmt.
_setSupport	[1,2] 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 ;-) **1.** Eintrag = Zone gibt Gruppen zur Unterstützung frei (0=nein, 1=ja) **2.** Eintrag = Aktiviert in der Zone die Arti-Anforderung (0=deaktiviert) 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.
_setJoin	2 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.
_setEmpVeh	[[150,100],[100,50]] 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: **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 % Welche leeren Fahrzeugtypen besetzt werden dürfen, wird festgelegt durch die Einstellung **DAC_VehAllowed** (Seite 22). Welche leeren Geschütze besetzt werden dürfen, wird festgelegt durch die Einstellung **DAC_GunAllowed** (Seite 22).

[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 Thema **Die 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    = [ 5,      10,     10,     20,     20      ];
            _CheckRadius2    = [ 10,     30,     30,     40,     60      ];
            _checkAreaH      = [ 20,     15,     15,     10,     10      ];
            _checkMaxH       = [ 500,    500,    500,    500,    500     ];
            _checkMinH       = [ 5,      5,      5,      5,      5       ];
            _checkNear       = [ 0,      0,      0,      100,    200     ];
            _checkObjH1      = [ 0.5,    0.5,    0.5,    0.2,    0.2     ];
            _checkObjH2      = [ 30,     20,     10,     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 Wegpunkt-Konfiguration **2** gewählt (dieser Parameter ist optional)

["z6",[1,0,0,300,200],[8,2,80,12],[ ],[ ],[ ],[0,0,0,0,2]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

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",[1,0,0,300,200],[8,2,80,12],[ ],[ ],[ ],[0,0,0,6,2]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

```
case 6:
{
    _campBasic     = ["FlagCarrierSouth",["Fire",5,10,0],["CampEmpty",10,0,0],["Logic",0,15,0],0];
    _campAmmo      = [["AmmoBoxWest",16,10,0],["WeaponBoxWest",16,5,0],["SpecialBoxWest",16,0,0]];
    _campStatic    = [["M2StaticMG",7,42,0,"Soldierwb"],["M2StaticMG",23,42,0,"Soldierwb"]];
    _campAddUnit   = ["Soldierwsniper","Soldierwsniper","Soldierwsniper","Soldierwsniper"];
    _campUserObj   = [["Logic",20,55,315],["ShedBig",25,20,90],["HMMWV",0,32,90],["Truck5tOpen",25,5,0]];
    _campRandomObj = [["AAR0038",4,80],["AAR0041",1,80],["AAPL068",1,80],["AAPL076",2,80]],50,1,30,15];
    _campWall      = ["AAH0393",[-3,40],[40,45,0],[3,0,0,3],[1,0.8],[0,0]];
    _campObjInit   = [[],[],[],[],[],[],["[_x]execVM ""scripttest.sqf"""]];
};
```

_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 nördlich / -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 respawnte 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]
```

- AmmoBox-Typ der generiert wird
- Die x-Abweichung vom Referenzobjekt
- Die y-Abweichung vom Referenzobjekt
- Die Ausrichtung der Ammobox

_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]
```

- Objekt-Typ der generiert wird
- Die Höhenabweichung im Meter (Wert = Meterangabe, die das Objekt max. abgesenkt wird)
- Wahrscheinlichkeit (%) mit der ein Objekt dieser Sorte platziert wird.

_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
```

- max. Anzahl Objekte, die für das Camp generiert werden
- 0 = zufällige Anzahl Objekte, genaue Anzahl Objekte
- Radius der Objektplatzierung, ausgehend von einem zufällig ausgewählten Camp-Objekt
- Sicherheitsabstand zu allen Camp-Objekten (_campUserOb, _campStatic, _campAmmo, _campBasic)

_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",[1,0,0,300,200],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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 = [100,1,0,3];
    _set1 = [0,0,100,[],3,30];
    _set2 = [100,1,100,100,4,1,0];
    _set3 = [["MlAbrams",["Sh_105_HE"]]];
    _set4 = [2,5,0];
    _set5 = [20,50,0];
    _set6 = [ 5,15,0];
    _set7 = [ 1, 5,0];
    _set8 = [];
    _set9 = ["((player distance TestLogic) < 200)","Not possible"];
};
```

Oben die Abbildung 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 DAC-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 ein Arti-Schlag angefordert werden kann.
- Min. Durchschnitts-Skill der **Call-Group**, damit ein Arti-Schlag 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 Positionsberechnung durch **Arti**-Einheit
Max. Abweichung bei Positionsberechnung 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)"]

[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 (siehe 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 **Infantrie**:

```
_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",[1,0,1,400,200],[5,3,50,8],[],[],[],[1,1,0,6]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Unter dieser Nummer in der **DAC_Config_Events.sqf**, muß 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 = 2 würde zum Beispiel die Marker-Konfiguration 2 laden.

Um die DAC-Marker zu deaktivieren, musst Du dem DAC nur mitteilen: **DAC_Marker = 0**
Das kannst Du per Skript machen, oder z.B. auch durch einen Auslöser.

Ein „Block“ in der DAC_Config_Marker ist folgendermaßen aufgebaut:

```
case 2: {
            _setShowZones       = 2;
            _setShowWPs         = 1;
            _setShowUnit        = [1,1,1,1];
            _setGroupType       = 1;
            _setMarkerText      = ["format[""<--- %1 (%2)"",_group,count units _group]"];
            _setMarkerDel       = 0;
            _setMarkerRefresh   = [1,1];
            _setSizeWpLine      = [0.5,3];
            _setSizeLeaderLine  = 0.5;
            _setSizeZoneLine    = 0;
            _setSizeCampLine    = 2;
            _setSizeZoneBorder  = 6;
            _setArtiMarker      = 1;
            _setCampMarker      = 2;
            _setSideColor       = [
                                    "ColorRed",
                                    "ColorBlue",
                                    "ColorYellow",
                                    "ColorGreen",
                                    "ColorWhite",
                                    "ColorBlack","ColorGreen"
                                  ];
```

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",[0.4,0.4],0]
                        ],
                        [
                            ["Man","ARMA_MARKER_Man",[0.4,0.4],1],
                            ["SquadLeaderE","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SquadLeaderW","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SquadLeaderG","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerE","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerW","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerG","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SoldierESniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierWSniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierGSniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierEMedic","ARMA_MARKER_Medic",[0.4,0.4],1],
                            ["SoldierWMedic","ARMA_MARKER_Medic",[0.4,0.4],1],
                            ["SoldierGMedic","ARMA_MARKER_Medic",[0.4,0.4],1]
                        ],
                        [
                            ["Car","ARMA_MARKER_UAZ",[0.5,0.5],1],
                            ["Truck","ARMA_MARKER_TruckMG",[0.5,0.5],1]
                        ],
                        [
                            ["Tank","ARMA_MARKER_T72",[0.5,0.5],1],
                            ["APC","ARMA_MARKER_M113",[0.5,0.5],1]
                        ],
                        [
                            ["Motorcycle","Dot",[0.5,0.5],1]
                        ],
                        [
                            ["Air","Dot",[0.5,0.5],1],
                            ["Helicopter","ARMA_MARKER_Cobra",[0.5,0.5],1],
                            ["Plane","ARMA_MARKER_Harrier",[0.5,0.5],1],
                            ["ParachuteBase","Dot",[0.6,0.2],1]
                        ],
                        [
                            ["Ship","ARMA_MARKER_Gunboat",[0.5,0.5],1]
                        ],
                        [
                            ["Other","Dot",[0.6,0.6],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)]

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, oder
@(DAC_Basic_Value == 1)              innerhalb von sqs-Skripten
```

In der **DAC_Config_Camp** gibt es die Konfiguration **8**, 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.

Ich könnte hier noch wesentlich mehr Tips + Tricks aufzeigen, aber mir fehlt dafür einfach die Zeit :-/

Benutzerhinweis & Danksagung

Der DAC ist ein **inoffizielles** Skript-Paket. Bitte keine Beschwerden an BIS oder deren Publisher.

Achtung, die Benutzung von DAC V2.0 (Beta) erfolgt auf eigene Gefahr!

Ich danke dem ganzen **Mapfact-Team**, für die freundliche Unterstützung in allen Bereichen,
und insbesondere bedanke ich mich bei:

MCPXXL	für die Bereitstellung eines MP-Servers und für die extremen MP-Tests, die uns allen sehr viel Freude bereitet haben.
SKH-Team	für die Bereitstellung eines MP-Servers und für die unzähligen MP-Test-Stunden, die uns immer wieder gezeigt haben, wie schlecht wir doch sind ;-)

Durch die vielen Teststunden, konnten sehr viele Fehler aufgedeckt und bereinigt werden. Danke !

Ein ganz **besonderer Dank** geht auch an meine Übersetzer. Ohne diese Jungs hätte sich der Release-Termin vermutlich um 6 Monate verzögert. Übersetzt haben für mich:

zwobot, rocko, Burns, Memphis und gruftschreck … vielen, vielen Dank !!

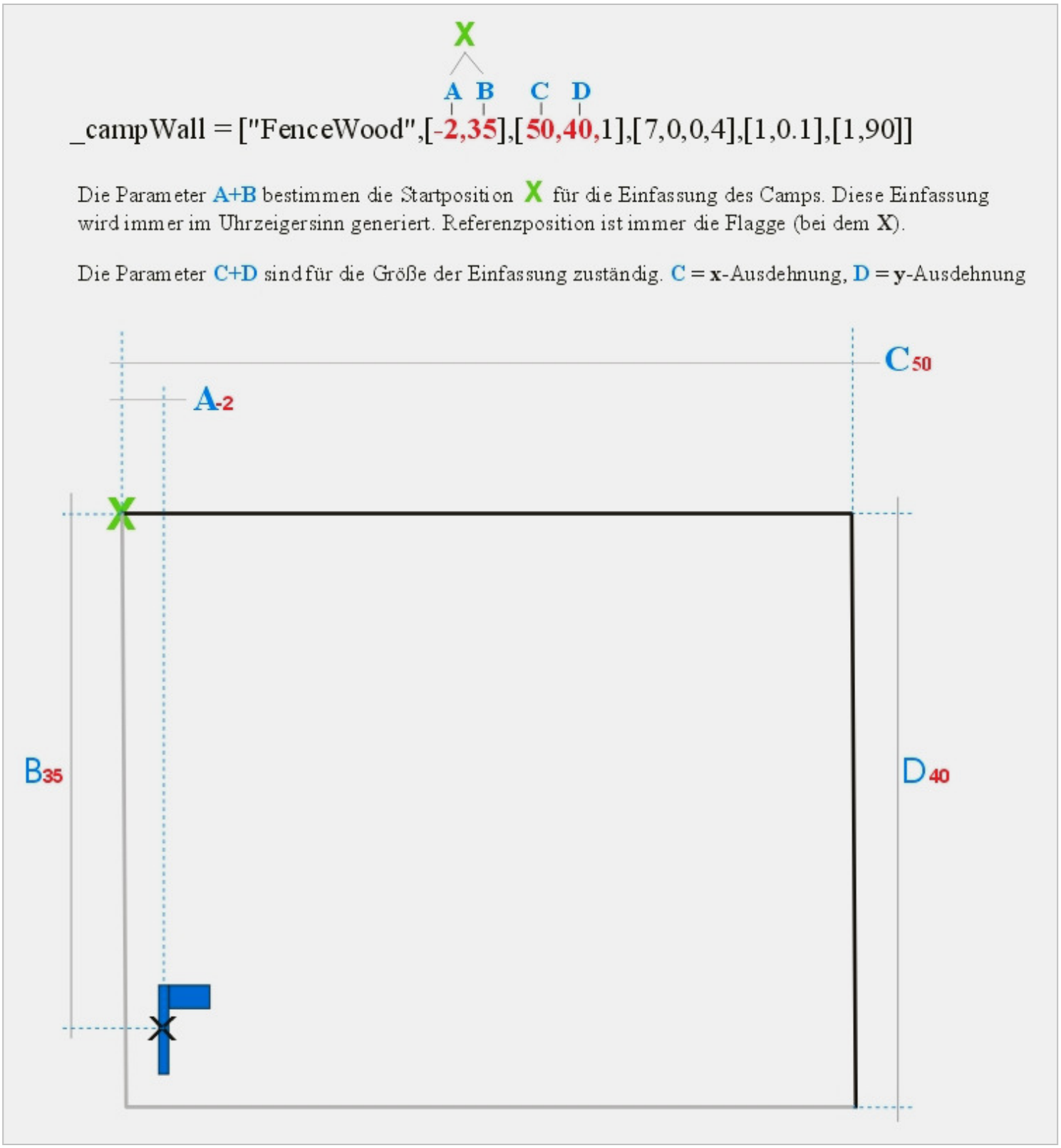
Weiterer Dank geht an: MattRochelle, PitPines, Sickboy, Thomas Niedermeier

Ich bitte um Verzeihung, wenn ich jemanden vergessen habe.

Zum Schluss noch eine dickes Dankeschön an BIS, die mit ARMA ein wunderbares Spiel entwickelt haben !

Anhang

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



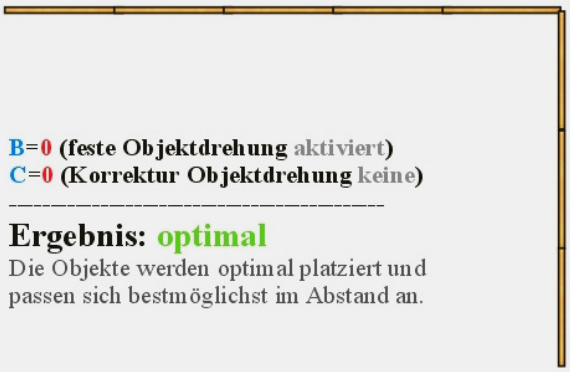
```
                                                A                   B C
_campWall = ["FenceWood",[-2,35],[50,40,0/1],[7,0,0,4],[1,0.1],[0,0]]
```

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.

A=0 (x-Ausmaß des Objekts wird benutzt = 2 m)



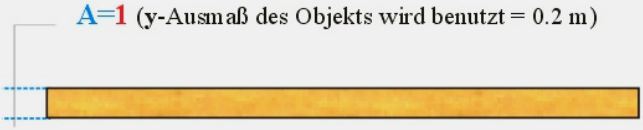


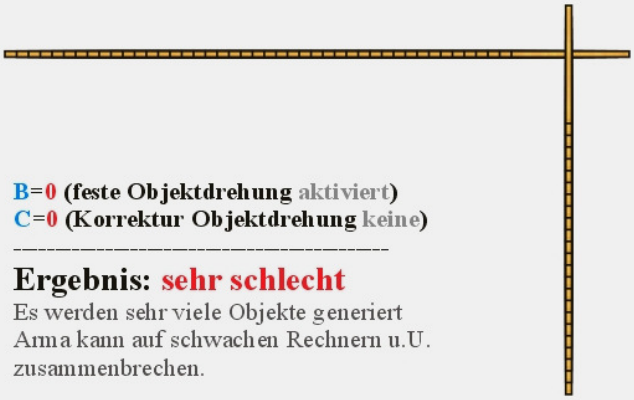
B=0 (feste Objektdrehung aktiviert)
C=0 (Korrektur Objektdrehung keine)

Ergebnis: optimal

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

A=1 (y-Ausmaß des Objekts wird benutzt = 0.2 m)





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

Ergebnis: sehr schlecht

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

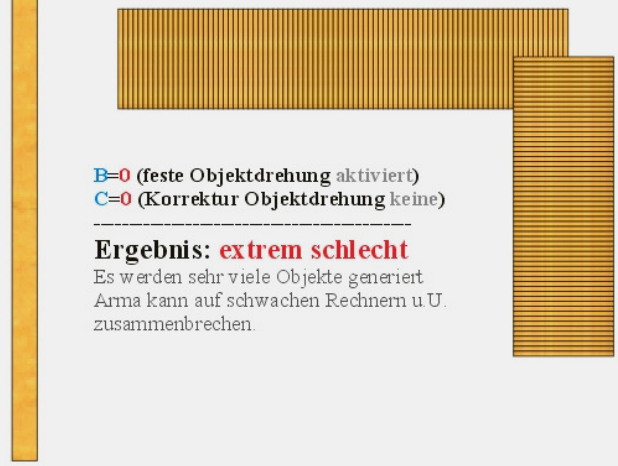
A B C

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

Dieses Beispiel zeigt genau die gleichenEinstellungen, 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.

A=0 (**x**-Ausmaß des Objekts wird benutzt = 0.2 m)



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

Ergebnis: extrem schlecht

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

A=1 (**y**-Ausmaß des Objekts wird benutzt = 2 m)



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

Ergebnis: schlecht

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

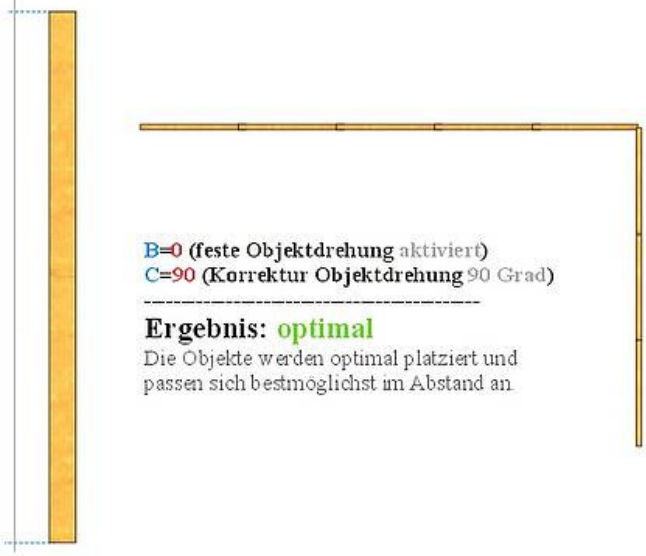
A B C

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

Hier kommt nun der Parameter C zum Einsatz, damit wir auch diese Art Objekte verarbeiten können,
also Objekte, die mit der schmalen Seite nach Norden zeigen.

Das Objekt bekommt in diesem Beispiel eine permanente Drehung von 90 Grad verpasst.
Das Ergebnis siehst Du unten im linken Bild.Im rechten Bild wurde die zufällige Objektdrehung (B) aktiviert,
indem dort ein Wert von 3 Grad (gilt in beide Richtungen) eingestellt wurde.

A=1 (y-Ausmaß des Objekts wird benutzt = 2 m)

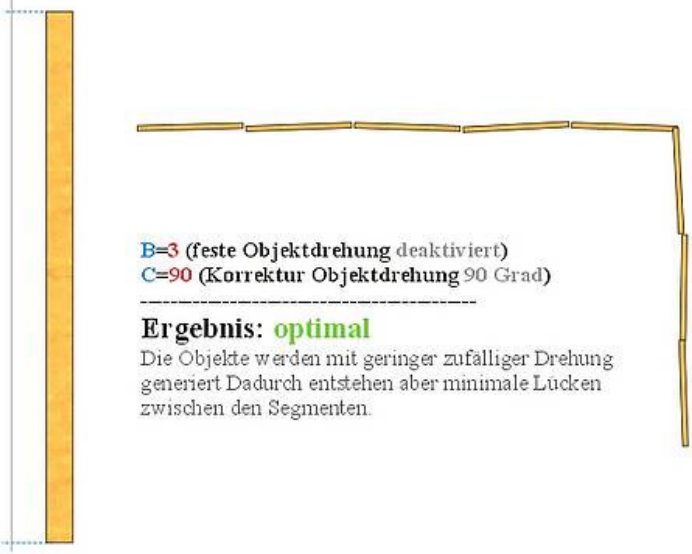


B=0 (feste Objektdrehung aktiviert)
C=90 (Korrektur Objektdrehung 90 Grad)

Ergebnis: optimal

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

A=1 (y-Ausmaß des Objekts wird benutzt = 2 m)



B=3 (feste Objektdrehung deaktiviert)
C=90 (Korrektur Objektdrehung 90 Grad)

Ergebnis: optimal

Die Objekte werden mit geringer zufälliger Drehung
generiert Dadurch entstehen aber minimale Lücken
zwischen den Segmenten.

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

Hier nochmal das Array für die Ausparungen in der Einfassung. Jeder Wert darin steht für einen Abschnitt: **Nord, Ost, Süd, West.** Der Wert selbst enspricht der Anzahl der Segmente, die entfernt werden sollen.

