Dynamic-AI-Creator [DAC] Beta Version 1.0

Benötigt wird mindestens OFP Version 1.75 / Addons: cti_marker

Bitte lass Dich nicht von dieser umfangreichen PDF abschrecken. Der Umfang war nötig, um alle Möglichkeiten von DAC aufzuzeigen. Es reicht aus, wenn Du Dir vorerst nur die ersten 3 bis 5 Themen durchliest, um mit dem Prinzip von DAC klarzukommen. Der Rest ergibt sich, wenn Du die PDF an den für Dich wichtigen Stellen weiter liest.

In der DAC_Kurzanleitung in vier Schritten, findest Du erstmal alle Infos, um direkt loslegen zu können.

Inhalt der Zip-Datei:

DAC_Vorschau.pdf
DAC_Liesmich.pdf
DAC_Kurzanleitung.pdf
DAC_Editor_Beispiele 1-11 auf Everon
DAC_MP_Demo
DAC_Ordner (zum kopieren in Deinen Missionsordner)
cti_marker.pbo

Themenübersicht:

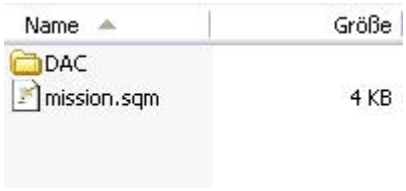
- Hinweis zu DAC [**1**]
- Was benötigt DAC innerhalb bzw. außerhalb des Editors [**2-3**]
- Erste Schritte mit der Demo1 (ein Beispiel mit einer Zone) [**3-6**]
- Kontrollmarker in DAC [**7**]
- Die Wahl der Seite, das Einheiten-Addon für eine Zone, das Grundverhalten der KI [**8-9**]
- Mehrere Zonen erstellen (Zonen kopieren) [**9-10**]
- Zonen verbinden bzw. verlinken [**10-11**]
- Wegpunktzonen erstellen [**11**]
- Benutzerdefinierte Wegpunkte einbinden [**12**]
- Zonen aktivieren / deaktivieren [**13**]
- Bestimmte Einheiten- Kategorien aus einer Zone speichern [**14**]
- AI-Addon auswählen / AI-Addon Level einstellen [**14**]
- AI-Spawn aktivieren und konfigurieren [**15**]
- Reduzierung von Infanterie-Gruppen und Cachen von Einheiten [**16**]
- Die Konfigurations- Dateien [**17-28**]
- Multiplayerbetrieb [**28**]
- Tipps und Tricks [**29-30**]
- Danksagung [**31**]

Hinweis zu DAC

Wer von DAC noch nichts gehört hat, sollte auf jeden Fall mal die beiliegende **DAC_Vorschau.pdf** lesen. Diese Vorschau gibt Dir einen ersten guten Einblick in DAC und beschreibt das Prinzip von DAC. Ansonsten nimm Dir etwas Zeit um diese **DAC_Liesmich.pdf** zu lesen. Nur so kannst Du alles aus DAC rausholen und kommst dabei hoffentlich voll auf Deine Kosten ;-)

Dieser Beta-Version liegen einige Kurz-Beispiele bei, auf die ich an entsprechender Stelle verweise.
So lassen sich einige Themen aus dieser PDF besser nachvollziehen.

Was benötigt DAC innerhalb bzw. außerhalb des Editors



Außerhalb des Editors ist DAC nicht sehr anspruchsvoll ;-)
Es wird lediglich der DAC-Ordner benötigt, den Du einfach in Deinen Missionsordner kopieren musst.

Dann noch die „cti_marker.pbo“ in dein AddOn-Verzeichnis kopieren…das war’s.
Es wird weder eine „Description“ noch eine „Init.sqs“ benötigt.

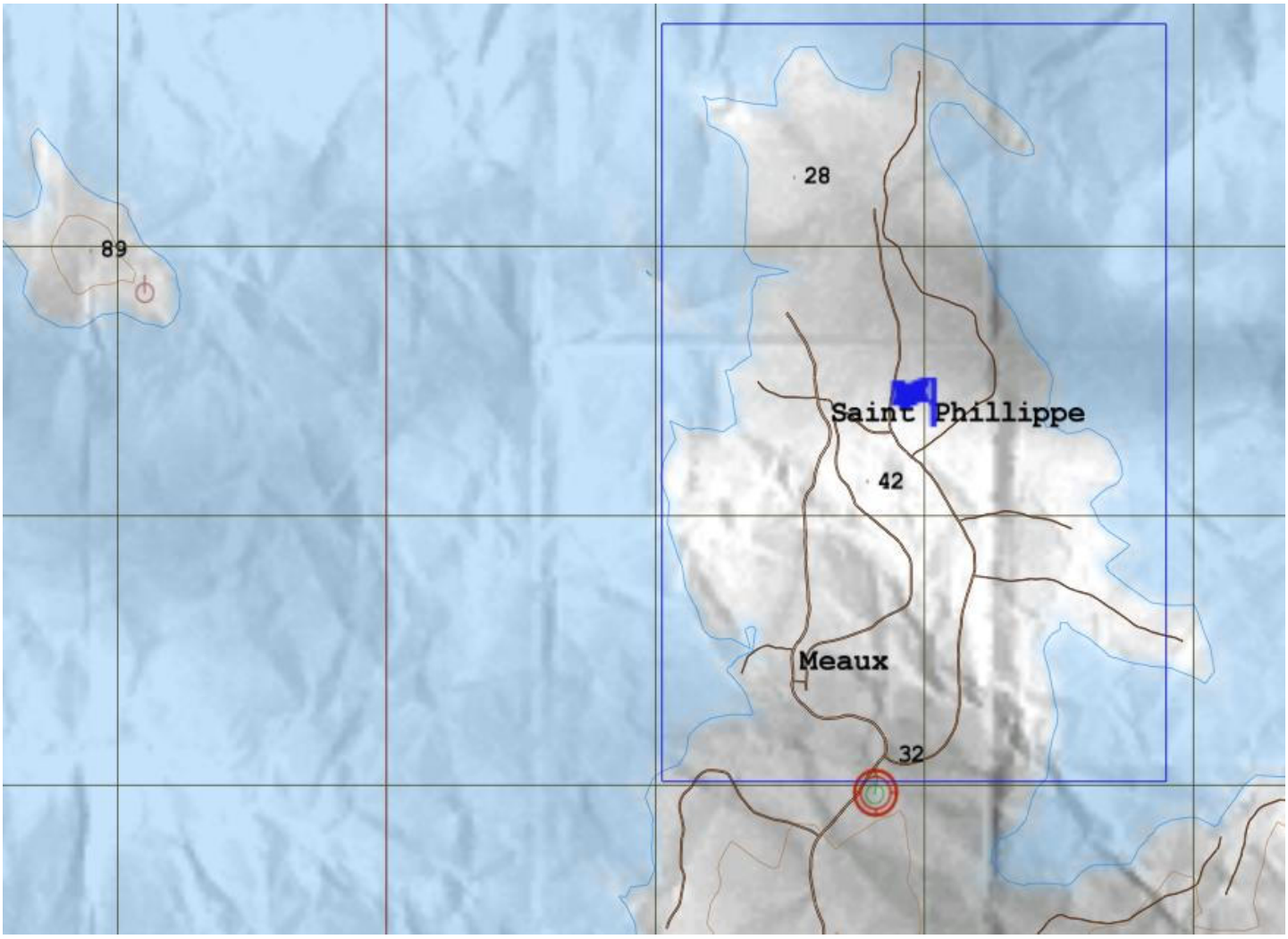
Prinzipiell wird auch kein Addon benötigt. Mit DAC ist es aber möglich mehrere Einheiten-Addons anzusprechen. Das klappt aber nur, wenn Du diese Addons dann auch installiert und geladen hast.

Unten das Bild (**Abb.1**) zeigt Dir die Ausgangssituation in der Demo **DAC_01_OneZone**.
Anhand von diesem Bild kannst Du sehr gut erkennen, was genau im Editor benötigt wird:

- Der Spieler mit dem Namen „s1“
- Eine Dummy-Einheit (zu sehen oben links)
- Mindestens ein Auslöser für eine Zone

Das ist im Prinzip schon alles was DAC benötigt, um loslegen zu können ☺

(Abb.1)



Anmerkung **[Player]**

Den Player musst Du benennen, damit die KI auf Deine bzw. auf die Anwesenheit Deiner Gruppe reagiert. Der Name „s1“ kommt daher, weil DAC so vorkonfiguriert ist. Weitere Spieler oder Einheiten auf Deiner Seite musst Du dann mit …s2,s3,s4… usw. benennen.

Möchtest Du andere Namen verwenden bzw,. wenn das eine bestehende Mission erfordert, dann musst du diese Namen in dem Array **DAC_Players** eintragen.
Diese Array findest Du im Script **DAC\DAC_Config_Creator.sqs**.

Dieses Array bereinigt sich automatisch, dass heißt es können beliebig viele (auch ungültige) Einträge in dem Array stehen z.B. **DAC_Players = [s1,s2,s3,s4,s5,s5,ap1,ap2,ap3.....]**.
Sobald die Mission gestartet wird, reduziert DAC dieses Array automatisch auf die benötigten Einträge.

Wie gesagt, betrifft das nur Einheiten die Du im Editor platzierst und Deiner Seite angehören.

Anmerkung **[Dummy Einheit]**

Es wird immer eine Dummy-Einheit pro Seite benötigt. Also insgesamt maximal 4 Dummy-Einheiten, wenn alle 4 Seiten generiert werden sollen. Möchtest Du nur Ost-Einheiten erzeugen, musst du nur eine Dummy-Einheit auf Ost-Seite im Editor platzieren.

Um zum Beispiel eine West- und eine Ost-Zone zu generieren, musst Du demnach eine Dummy-Einheit auf Ost-Seite und eine Dummy-Einheit auf West-Seite im Editor platzieren.

Achtung, die Dummy-Einheiten müssen folgendermaßen benannt werden:

Ost-Einheit =E_Dummy | West-Einheit =W_Dummy | Res-Einheit =R_Dummy | Civil-Einheit =C_Dummy

Erste Schritte mit der DAC_Demo1

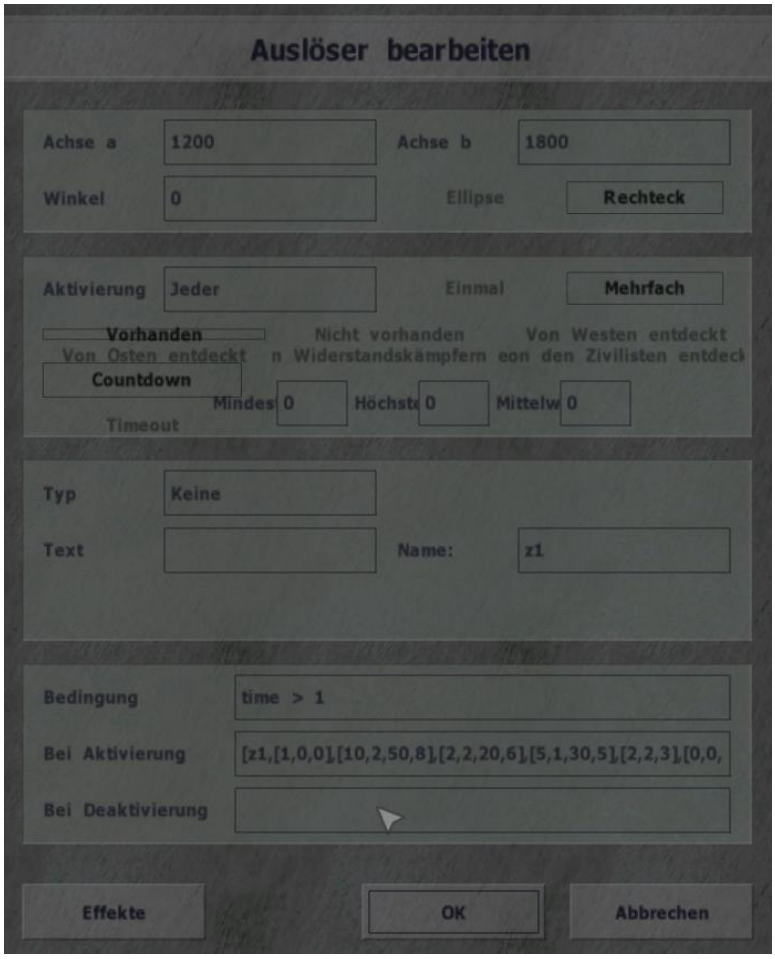
Die Demomission **DAC_01_OneZone**
ist keine richtige Mission, sondern nur eine DAC- Demonstration mit einer Zone.

Eine AI-Zone (Auslöser), wie in der Demo zu sehen, muss folgendermaßen eingestellt sein:

Position : **beliebig**
Achse a/b : **beliebig**
Winkel : **0**
Form : **Rechteck**
Aktivierung : **Jeder / Mehrfach / Vorhanden**
Typ : **Keine**
Name : **Name der Zone (z.B. ->** z1 **)**
Bedingung : **time > 1**

„Bei Aktivierung“ kommt dann der Scriptaufruf rein, der die Zone zum Leben erweckt ☺

Möchtest Du zu Testzwecken eine Zone mal kurzfristig deaktivieren, dann trag bei „Bedingung“ ein: **time > 10000000**



Das hat den Vorteil, dass alle Einstellungen der Zone erhalten bleiben und Du die Zone bei Bedarf jederzeit wieder aktivieren kannst.

Du kannst den Auslöser aus der Demo in der Größe ändern oder auch mal auf eine andere Position setzen. Diese Änderungen werden von DAC automatisch erkannt und Du brauchst nur wieder auf „Vorschau“ klicken, damit diese Änderung neu generiert wird. Überhaupt wird alles neu generiert.

Der schwierige Teil von DAC liegt bei der Konfiguration einer Zone. Schwierig insofern, weil dieser Scriptaufruf sehr viele Variationen und Möglichkeiten bietet. Die Hälfte dieser Liesmich.pdf befasst sich deshalb mit diesem einen Scriptaufruf. Aber keine Panik, dass ist locker zu schaffen ;-)

Wie gesagt, wird jede Zone durch einen Scriptaufruf konfiguriert und sieht folgendermaßen aus:

[z1,[1,0,0],[ ],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Für die Demo interessieren uns erstmal nur die 4 leeren Arrays, die ich farblich markiert habe.

Mit diesen Arrays werden die 4 nachfolgenden Einheiten- Kategorien erstellt:

Das 1. leere Array ist für die Generierung von Infanterie zuständig.
Das 2. leere Array ist für die Generierung von unbewaffneten Fahrzeugen zuständig.
Das 3. leere Array ist für die Generierung von bewaffneten Fahrzeugen zuständig.
Das 4. leere Array ist für die Generierung von Helis oder Spawncamps zuständig.

Diese Arrays brauchen folgende Parameter, damit DAC was damit anfangen kann….

Infanterie:

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

Beispiel: **[z1,[1,0,0],[5,2,50,6],[ ],[ ],[ ],[0,0,0]]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.

Unbewaffnete Fahrzeuge:

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

Beispiel: **[z1,[1,0,0],[ ],[3,3,30,5],[ ],[ ],[0,0,0]]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 unbewaffnete Fahrzeuge).
Jede erzeugte Gruppe bekommt **4 - 6** Wegpunkte aus diesem Pool zugewiesen.

Du fragst Dich bestimmt: Wieso muss man keine Anzahl für Fahrzeuge angeben?

Diese Anzahl wird dynamisch von DAC geregelt. 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.

Bewaffnete Fahrzeuge:

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

Beispiel: **[z1,[1,0,0],[ ],[ ],[3,3,25,5],[ ],[0,0,0]]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 bewaffnete Fahrzeuge).
Jede erzeugte Gruppe bekommt **4 - 6** Wegpunkte aus diesem Pool zugewiesen.

Helicopter:

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

Beispiel: **[z1,[1,0,0],[ ],[ ],[ ],[2,2,4],[0,0,0]]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 erstens die DAC-KI aktiv ist, und zweitens der Heli Ladekapazität besitzt. Ein AH64 z.B. wird keine zusätzlichen Einheiten aufnehmen können. Generiert wird dann nur die benötigte Besatzung.

Achtung, der Heli landet meistens nicht exakt auf seiner Startposition, also da wo er erstellt wurde, da er immer den für sich optimalen Landepunkt in unmittelbarer Nähe seiner Startposition sucht.

Schau Dir mal den Scriptaufruf in der Demo **DAC_01_OneZone**, der sieht folgendermaßen aus:

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

8 Gruppen mit **Infanterie**
(Gruppengröße 2, 50 WP gesamt, 8 WP pro Gruppe)

3 Gruppen mit **unbewaffneten Fahrzeugen**
(Gruppengröße 2, 20 WP gesamt, 6 WP pro Gruppe)

3 Gruppen mit **bewaffneten Fahrzeugen**
(Gruppengröße 1, 30 WP gesamt, 8 WP pro Gruppe)

2 Gruppen mit **Helis**
(Gruppengröße 2, 3 WP pro Gruppe)

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. Mach diesbezüglich ruhig mal ein paar Versuche mit der Demo **DAC_01_OneZone**, also z.B. die Anzahl der Gruppen oder der Wegpunkte mal ändern, oder auch mal eine Kategorie auslassen.

In fast allen DAC-Demos sind die Kontrollmarker aktiviert. Das bedeutet für Dich, dass Du Dir die Aktivitäten von DAC auf der Karte anschauen kannst und Deine Änderungen auch direkt nachvollziehen kannst.

Kontrollmarker

Die Kontrollmarker helfen Dir dabei in etwa nachzuvollziehen, was DAC während der Initialisierung alles zu erledigen hat ;-)

Wenn Du die Demo **DAC_01_OneZone** startest, kannst Du auf der Karte beobachten, wie DAC …

die Zonen berechnet,
die Wegpunkte generiert,
die Einheiten erzeugt.

Im weiteren Verlauf der Mission Siehst Du dann, wie die Einheiten Ihre Wegpunkte abklappern ;-)

Es gibt 5 verschiedene Markertypen:

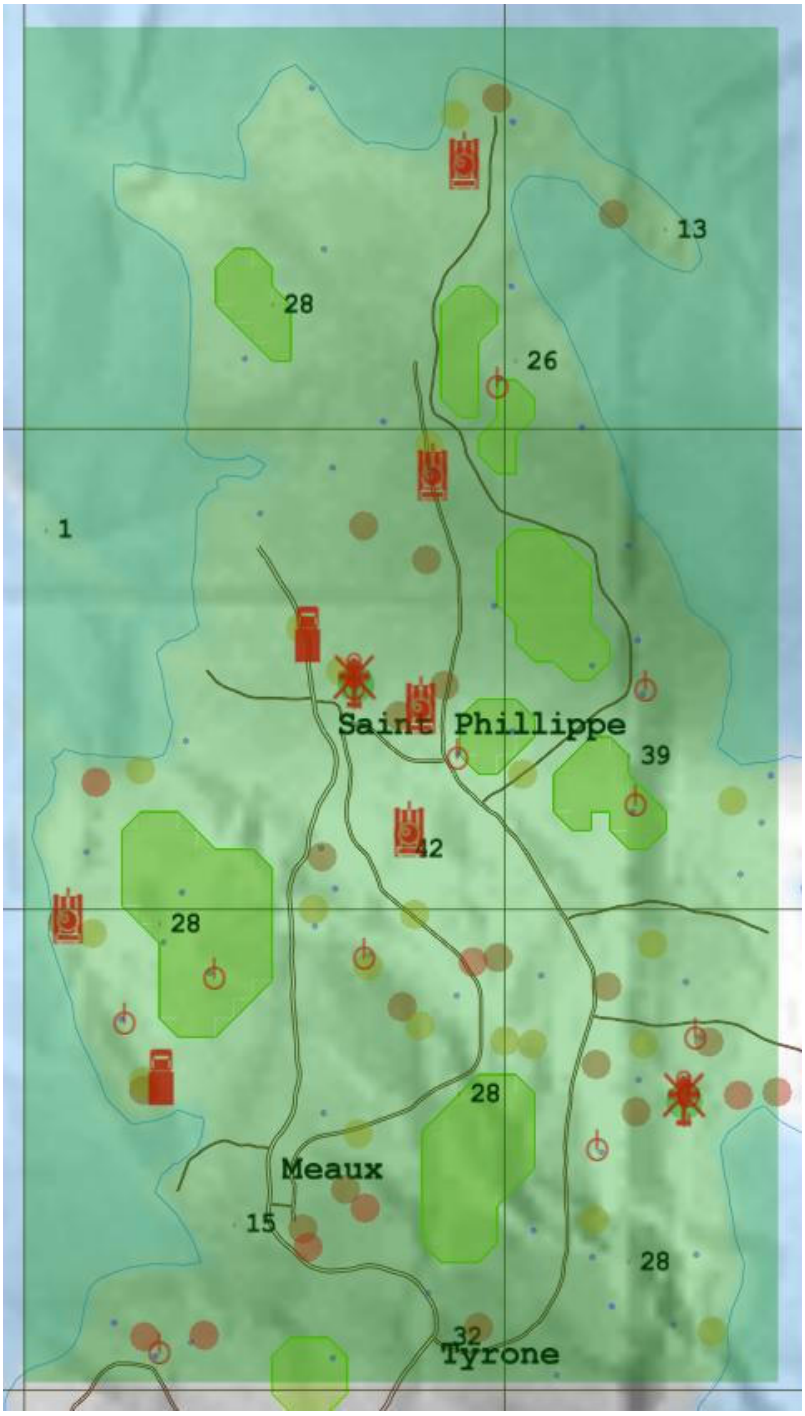
Blau
Wegpunkte für Infanterie

Orange
Wegpunkte für unbewaffnete Fahrzeuge

Red
Wegpunkte für bewaffnete Fahrzeuge

Grün
Start/Lade- Zonen für Helis

Grau
Standorte für Camps



Dann noch die jeweiligen Marker für die verschiedenen Einheiten in den Farben **Blau** (Westen), **Rot** (Osten)

Bitte beachte, dass die Marker für Einheiten, immer nur eine Gruppe anzeigen und nicht alle Einheiten einer Gruppe. Außerdem steht auch nur ein gewisses Kontingent an Markern zu Verfügung.

Möchtest Du dieses Merkmal von DAC deaktivieren, musst Du die Variable DAC_Marker auf false setzen. Die Variable findest Du im Script DAC\DAC_Config_Creator.sqs.

Möchtest Du die Marker in Deiner Mission einsetzen (zum Testen gut zu gebrauchen), musst Du Dir außerdem die Marker aus einer der Editor-Beispiele rauskopieren und in Deiner Map einfügen.

Ohne diese Marker kannst Du nur erahnen, wo sich Einheiten aufhalten und welche Wege sie gehen. Selbst für Dich als Missions-Ersteller ist jeder Missionsstart wieder ein neues Erlebnis und Du musst mit allem rechnen ☺

Die Wahl der Seite, das Einheiten-Addon für eine Zone, das Grundverhalten der KI

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).

(**Scriptaufruf aus der Demo1**)

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

Diese 3 Parameter haben folgende Bedeutung:

Parameter 1 = Die Wahl der Seite für die Einheiten in dieser Zone
Parameter 2 = Die Wahl des Einheiten-Addons in dieser Zone
Parameter 3 = Die Wahl des Grundverhaltens in dieser Zone

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** = Resistance | **3** = Civil

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

Zur Erinnerung: für Ost-Einheiten wird der E_Dummy benötigt!

Die Wahl der Addon-Einheiten

Die Wahl der Einheiten für eine Zone wird auch über eine Nummer geregelt. Eine Nummer entspricht dabei einem Eintrag in dem Script **DAC\DAC_Config_Units.sqs.**

Hier mal ein kleiner Auszug aus diesem Script:

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_0
;/////////////////////////////////////////

_Unit_Pool_S = ["SoldierECrew","SoldierEPilot","officerE","Soldiereb","Soldier
_Unit_Pool_V = ["Ural","UAZ","BRDM"]
_Unit_Pool_T = ["BMP","T72","T80","BMP","T72","ZSU","BMP","ZSU","BMP2"]
_Unit_Pool_A = ["MI17","MI24"]
_Unit_Pool_C = ["rus_vlajka.pac","CampEast","Shed"]

goto "SetPool"


;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_1
;/////////////////////////////////////////

_Unit_Pool_S = ["SoldierWCrew","SoldierWPilot","officerw","Soldierwlaw","Sold
_Unit_Pool_V = ["Jeep","Truck5t","Truck5tOpen"]
_Unit_Pool_T = ["M1Abrams","Bradley","M60","Vulcan","JeepMG"]
_Unit_Pool_A = ["Cobra","AH64","Ch47D","UH60"]
_Unit_Pool_C = ["usa_vlajka.pac","CampEmpty","Shed"]

goto "SetPool"
```

In diesem Script findest Du viele abgelegte Addon-Einheiten. Jedes dieser Addons ist dabei unter einer Nummer zu finden (eingekreist).

Dieses Script kann beliebig erweitert werden. Man muss nur darauf achten, dass jeder Einheiten-Block eine eigene eindeutige Nummer hat.

Nähere Informationen findest Du unter dem Thema : **Die Konfigurations- Dateien.**

In der Abbildung oben siehst Du z.B. die **#Unit_Config_0** und die **#Unit_Config_1.**

Um in einer Zone bestimmte Einheiten zu erstellen, musst Du also nur die entsprechende Config_Nr in dem Scriptaufruf angeben. In jeder Zone kann dabei eine andere Config_Nr angesprochen werden. Du musst nur darauf achten, dass Du die gewählten Einheiten auch installiert bzw. geladen hast.

Die Standard-Einheiten findest Du unter folgenden Nummern:

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

Die Wahl des Grundverhaltens

Der letzte Parameter dieses Arrays bestimmt das Grundverhalten der Einheiten in einer Zone. Grundverhalten bedeutet, wie sich die generierten 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\DAC_Config_Behaviour.sqs** 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.

Hier noch mal der Scriptaufruf aus der Demo **DAC_01_OneZone**:

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

Ost-Seite
Standard Ost-Einheiten
Grundverhalten Konfiguration 0

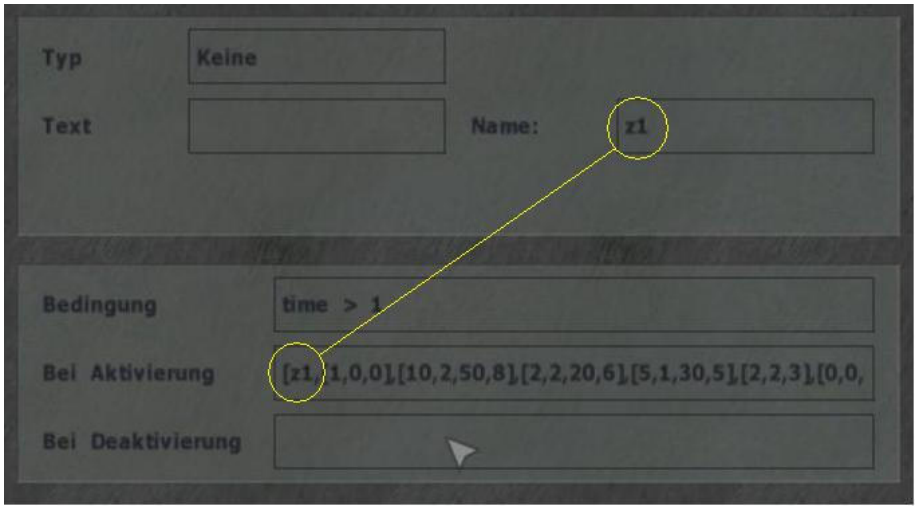
Eine ausführliche Beschreibung, wie man das Grundverhalten konfiguriert und welche Möglichkeiten es gibt, findest Du auch wieder unter dem Thema **Die Konfigurations- Dateien**.

Mehrere Zonen erstellen (kopieren)

Weitere Zonen zu erstellen,
ist eigentlich ganz simpel ☺

Du musst nur eine Zone kopieren
und an anderer Stelle wieder einfügen.

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



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

Du kannst den Namen stehen lassen, oder auch abändern z.B. auf „z2“. Egal für welchen Namen Du Dich entscheidest, der Name der Zone muss in den Scriptaufruf und zwar direkt an erster Stelle (siehe Abb.).

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

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

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

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

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

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 ändern. Diesen Parameter hab' ich **rot** markiert. Dieser Parameter ist sozusagen die **ID-Nr.** der Zone. Gib' jeder Zone eine andere Nummer, also so…

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

Die Demo **DAC_02_MoreZones** zeigt auch noch mal das Prinzip der lokalen Zonen.

Zonen verbinden bzw. verlinken

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

```
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[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 ;-)
Ein kleines Beispiel findest Du in der Demo **DAC_02_LinkedZones**.

Hier noch ein paar andere Konstellationen bzgl. verlinkten Zonen:

```
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

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

Noch ein Beispiel:

```
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,15,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,0,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,0,0],[ ],[ ],[3,1,35,8],[ ],[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

Diese 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 in der Demo **DAC_05_WPZones**.

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

```
[z1,[1,0,0],[15,2,10,15],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[20],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[1,0,0],[20],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[1,0,0],[20],[ ],[ ],[ ],[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. Also noch mal zum Verständnis: Ein Array mit 4 Parametern (oder 3 Parameter bei Helis) erzeugt Wegpunkte **und** Einheiten und ist sozusagen die Masterzone oder Ursprungszone der Einheiten.

Ein Array mit nur einem Parameter erzeugt **nur** Wegpunkte für die entsprechende Kategorie. Hier ein Beispiel, wie man 2 unterschiedliche Masterzonen auch kombinieren kann:

```
[z1,[1,0,0],[15,2,10,15],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[ ],[ ],[3,1,25,8],[ ],[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],[15,2,10,15],[ ],[20],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[20],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

In der Zone **z1** werden zusätzlich **20** Wegpunkte für bewaffnete Fahrzeuge 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.

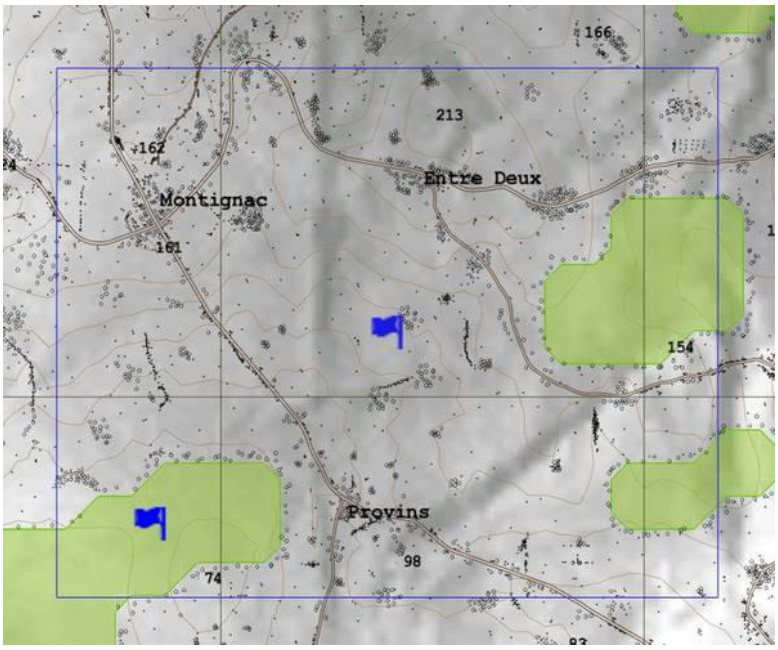
Das können z.B. Positionen in einer Ortschaft sein (DAC hat damit etwas Probleme, wenn Gebäude zu eng aneinander stehen), oder auch Missionsrelevante Positionen.

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:

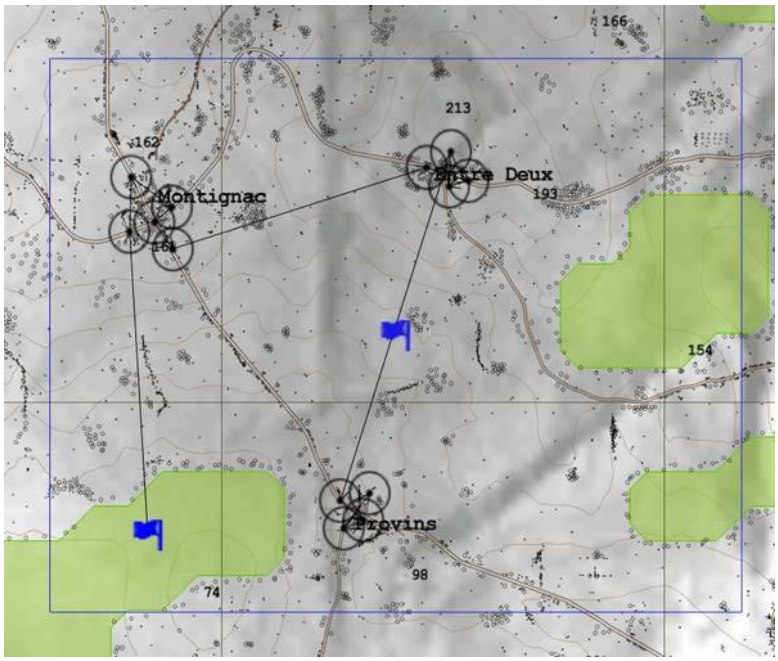
Du platzierst in der entsprechenden Zone eine Logic-Einheit. Wo genau spielt dabei keine Rolle. Sie muss nur innerhalb der Zone liegen →



Dieser Logic-Einheit gibst Du nun an den bestimmten Positionen ein paar Wegpunkte. →

Es ist völlig egal, in welcher Reihenfolge die Wegpunkte platziert sind, oder wie viel Wegpunkte Du platzierst, DAC regelt das und integriert die Wegpunkte automatisch.

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 über die Einstellungen der Logic-Einheit: →

Unter „Azimut“ gibst Du eine Nummer zwischen 1 und 5 an (Bedeutung s. rechts)



1 = **Infanterie**
2 = **unbewaffnete Fahrzeuge**
3 = **bewaffnete Fahrzeuge**
4 = **Helis**
5 = **Camps**

Zonen aktivieren / deaktivieren

Jede Zone kann, auch wieder über einen Zahlencode, 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.

Die Handhabung ist auch wieder sehr simpel - meiner Meinung nach ;-).

Dafür vorgesehen sind zwei Parameter, die ich **Grau** markiert habe…

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

Parameter zum **aktivieren** einer Zone
Parameter zum **deaktivieren** einer Zone

Diese beiden Parameter stehen in direkter Verbindung mit der Variable **DAC_Activate**. Diese Variable wird mit **0** initialisiert. Deshalb wurden die beiden Parameter bis jetzt auch immer mit 0 angegeben, damit die Zonen direkt zu Beginn generiert werden.

Bei dem folgendem Beispiel werden die Einheiten aus der Zone erst erstellt, wenn die Variable **DAC_Activate** den Wert **1** bekommt. Du musst also nur dafür sorgen, dass diese Variable **DAC_Activate** in irgendeinem Script den Wert 1 bekommt und schon werden alle Zonen, die diesen Wert besitzen, aktiviert.

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

Dann mal mit Parameter zum löschen der Zone. Bei **DAC_Activate** = **1**, werden die Einheiten aus dieser Zone erstellt, bei **DAC_Activate** = **2**, werden die Einheiten dieser Zone wieder gelöscht.

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

Und noch ein weiteres Beispiel: Die Zonen **z1** + **z2** werden direkt zu Beginn der Mission generiert, da der Wert zum aktivieren **0** beträgt. Beide Zonen werden gelöscht, wenn **DAC_Activate** den Wert **3** bekommt.

Die Zone **z3** wird aktiviert, wenn **DAC_Activate** den Wert **1** bekommt und wird nicht gelöscht.
Die Zone **z4** wird aktiviert, wenn **DAC_Activate** den Wert **2** bekommt und wird nicht gelöscht.

```
[z1,[1,0,3],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,3],[8,2,45,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,1,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,2,0],[ ],[ ],[2,1,35,6],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Achtung, bei dieser **Aktivierung** / **Deaktivierung** sollte man immer zusammenhängende Zonen berücksichtigen, also Zonen die verlinkt sind.

Bestimmte Einheiten- Kategorien aus einer Zone speichern

Es ist möglich, die generierten Einheiten einer Zone anzusprechen, bzw. die generierten Gruppen in ein benutzerdefiniertes Array zu speichern, um einen späteren Zugriff auf die Einheiten/Gruppen zu ermöglichen.

Umgesetzt wurde diese Möglichkeit mit einem „optionalen" Parameter für **jede** Einheiten-Kategorie.

Hier ein Beispiel: In der Zone **z1** werden **8** Infanterie-Gruppen erstellt. Um Zugriff auf die erzeugten Gruppen zu bekommen, wird in dem Array für Infanterie ein weiterer Parameter „angehängt"…

```
[z1,[1,0,0],[8,2,50,5,''MyArray''],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

!!! Achtung: der optionale Parameter muss in Anführungszeichen angegeben werden!!!

Aus diesem „Begriff" wird ein Array erzeugt, in dem Du dann die Gruppen dieser Kategorie aus dieser Zone findest. Das Beispiel oben würde demnach die **8** Infanterie-Gruppen in dem Array „MyArray" speichern.

Es ist natürlich sinnvoll, diese Arrays aussagekräftig zu benennen…z.B. „**Z1_inf"** oder „**Z4_veh"**, nur damit man die Zugehörigkeit direkt schon an dem Namen des Arrays erkennt.

Testen kannst Du das, indem Du Dir den Inhalt des Arrays anzeigen lässt (aber bitte erst, wenn DAC mit der Initialisierung fertig ist): **hint format[„%1",MyArray]**

Aus den Gruppen, die in den Arrays gespeichert werden, dürfte es dann kein Problem sein, die einzelnen Einheiten und/oder Fahrzeuge zu ermitteln.

Eine Anwendung wäre z.B. folgende:

Du erstellst eine Zone mit 3 Gruppen Shilkas, Gruppengröße 1-2 Fahrzeuge. Ein Missionsziel könnte jetzt sein: zerstören Sie alle Shilkas in dieser Zone.

Um festzustellen, ob und wann die Shilkas zerstört wurden, musst Du diese Einheiten/Gruppen auf ihren Zustand abfragen. Genau dafür ist dieser optionale Parameter gut.

AI-Addon auswählen / AI-Addon Level einstellen

Leider konnte das KI-Addon von KeyCat (**GroupLink II**) noch nicht in DAC implementiert werden, von daher gibt es zurzeit nur 2 Einstellmöglichkeiten was die KI betrifft:

DAC-KI-Addon → EIN (siehe Demomission **DAC_10_AI_Reaction**)
DAC-KI-Addon → AUS (die Scriptbelastung ist etwas geringer, die Einheiten verhalten sich wie Editor-platzierte Einheiten)

Dann gibt es noch die Möglichkeit, den Level des DAC-KI-Addons in 4 Stufen einzustellen Einzelheiten über Einstellmöglichkeiten findest Du wieder unter dem Thema: **Die Konfigurations- Dateien**

Eine Besonderheit der DAC-KI ist, dass die Einheiten nicht nur bei Deiner Anwesendheit reagieren, sondern auf alle feindlichen Einheiten. So wird z.B. ein generierter Ost-Trupp auch in Deckung gehen, wenn er auf einen generierten West-Trupp stößt. Er wird ihn angreifen und auch Verstärkung für sich anfordern (wenn er dazu kommt). Ist die DAC-KI aktiviert, werden die unbewaffneten Fahrzeuge an ihren Wegpunkten Aufklärungsausflüge machen und die generierten Helis werden zusätzlich eine Gruppe bei sich haben (vorausgesetzt der Heli hat die Kapazität dafür), die bei Bedarf auch mal über Dir abspringt ☺

AI-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 diese Camps in das Array für Helis zu packen.

Das bedeutet also, dass in einer Zone entweder Helis generiert werden können, oder eben Spawn-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.

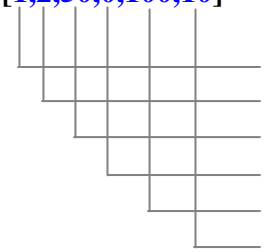
Respawned werden nur folgende Einheiten: **Infanterie**, **unbewaffnete Fahrzeuge**, **bewaffnete Fahrzeuge**

Gut, hier nun ein Beispiel für einen Scriptaufruf, in dem ein Spawn-Camp generiert werden soll:

```
[z1,[1,0,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0,0,0]]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. Hier jetzt die Bedeutung der einzelnen Parameter:

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



Die Anzahl der Camps, die generiert werden soll.
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 also 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.

Wird mehr als ein Camp in einer Zone generiert, wird der Respawn zufällig in einem der Camps stattfinden.

Ach ja, die bei einem Camp stationierte Gruppe löst sich wenn möglich auf, sobald die Respawns in einem Camp aufgebraucht sind. Sie wird das Camp noch kurz verminen, damit es sich wenig später selbst zerstört.

In jedem Camp findest Du eine Ammobox, deren Inhalt aus den Waffen der Einheiten der Zone generiert.

Angenommen, Du generierst eine Zone mit RHS-Einheiten und mit dieser Zone generierst Du auch ein oder mehrere Spawn-Camps, so findest Du in den Ammoboxen der Camps dann auch nur Waffen und Muni von RHS-Einheiten.

Bitte beachte, dass der Respawn einer Gruppe eine gewisse Zeit benötigt, also nicht der Respawn selbst, sondern die Zeit zwischen dem Ableben einer Gruppe und dem Zeitpunkt, wo die Gruppe neu aufgebaut werden kann… aus technischen Gründen sozusagen. Die Zeitspanne liegt in etwa zwischen 15 und 45 Sekunden, je nach Gruppengröße.

Weitere Einstellmöglichkeiten findest Du wieder unter dem Thema: Die Konfigurations- Dateien

Reduzierung von Infanterie-Gruppen und Cachen von Einheiten

Das sind zwei nicht zu unterschätzende Merkmale von DAC. 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, 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 ;-)

Das Reduzieren der Gruppen bewirkt außerdem noch, dass auch das Steuerscript für die Gruppen umschaltet und nicht mehr permanent die Gegnereinheiten abfragt. Das Script läuft dann sozusagen auf „Sparflamme“.

Zu diesem Thema liegen 2 Editor-Beispiele bei: **DAC_06_ReduceInfantry + DAC_07_ReduceBigZone**

Das **Cachen** von Einheiten steht im direkten Zusammenhang mit dem Respawn und der Gruppenreduzierung. Das heißt, machst Du weder Gebrauch von dem Respawn, noch von der Gruppenreduzierung, kannst Du dieses Merkmal auch getrost deaktivieren.

Jeder, der mal versucht hat, während einer Mission neue Einheiten zu „createn“, wird u.U. festgestellt haben, dass es dabei zu kleinen Aussetzern kommt, da OFP die „neuen Einheiten“ erst laden muss. Also so Sachen wie Modell, Texturen und Konfigurationen, und dabei kommt es dann etwas zum Stocken.

Um dem entgegenzuwirken, platziert man jeden möglichen Einheitentyp einmal im Editor, damit OFP die Daten der Einheiten bei Missionsbeginn bereits im Speicher hat.

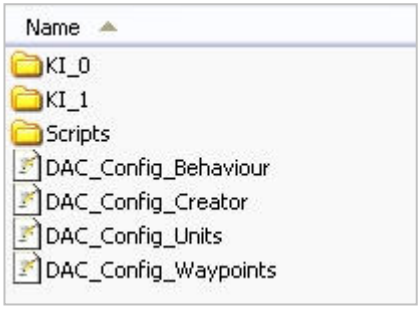
Beim Respawn und auch bei der Gruppenreduzierung, werden neue Einheiten erzeugt, und es macht Sinn in dem Zusammenhang das **Cachen** von Einheiten zu aktivieren.

Außer das **Cachen** per Parameter zu aktivieren, musst Du nichts weiter unternehmen. Diese Arbeit übernimmt DAC für dich. DAC ermittelt anhand der Zonen die möglichen Einheitentypen und erzeugt daraus jeweils eine Dummy-Einheit.

Bitte berücksichtige, dass bei zu vielen unterschiedliche Einheitentypen, die Du mit DAC problemlos erzeugen kannst, entsprechend viele Dummy-Einheiten erzeugt werden müssen. Das könnte sich ab einer bestimmten Anzahl dann wieder negativ auf die Performance auswirken. Das musst Du einfach mal ausprobieren.

Die Einstellmöglichkeiten erfährst Du im nachfolgenden Thema…

Die Konfigurations- Dateien



Die Konfigurations- Dateien liegen direkt im Hauptverzeichnis von DAC.

Es gibt 4 verschiedene Config-Scripte, mit denen man DAC an seine Bedürfnisse anpassen kann, oder auch erweitern kann.

Fangen wir mit den globalen Einstellungen an. Die befinden sich in der **DAC_Config_Creator.sqs**:

DAC_Config_Creator.sqs

```
DAC_Players             = [s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]
DAC_KI_Count_Level      = [[2,4],[2,6],[2,9],[2,12]]
DAC_Reduce_Value        = [600,700,0.1]
DAC_KI_Spawn            = [30,60,0,120]
DAC_Delete_Value        = [60,300,30]
DAC_KI_Cache            = [1,1]
DAC_Dyn_Weather         = [180,10,0.6]
DAC_Com_Values          = [1,1]
DAC_KI_AddOn            = 1
DAC_KI_Level            = 4
DAC_Res_Side            = 0
DAC_Marker              = true
```

DAC_Players = [s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]

Das ist das Array, in dem die Spieler und/oder die Einheiten der Spieler Gruppe abgelegt sind. Du kannst das Array um beliebige Einträge erweitern. DAC bereinigt dieses Array automatisch, und löscht überflüssige Einträge raus. Wichtig ist nur, dass die Editor-platzierten Einheiten auf Deiner Seite einen Namen aus dem Array bekommen, damit die generierte KI auch auf diese Einheiten reagiert.

DAC_KI_Count_Level = [[2,4],[2,6],[2,9],[2,12]]

Hier sind die Gruppengrößen abgelegt, die DAC generieren soll. Du erinnerst Dich? Also z.B. das Array für das Erzeugen von Infanterie, dass benötigt ja 4 Parameter: [5,2,50,6]

Der 2. Parameter greift auf diese Array **DAC_KI_Count_Level** zu. Eine **2** bedeutet also, es werden **2-6** Einheiten generiert, da der 2. Eintrag in dem Array die entsprechenden Werte dafür enthält. Es gibt immer 2 Werte in dem Array **DAC_KI_Count_Level.** Der 1. Wert ist die Anzahl der Einheiten, die mindestens erzeugt wird, und der 2. Wert entspricht der maximalen Anzahl, die generiert werden kann.

Beachte bitte die Limitierung der Gruppengröße von 12 Einheiten, wenn Du die Einträge ändern solltest. Bei bewaffneten Fahrzeugen wird der ermittelte Wert automatisch durch 3 dividiert, da erstmal von 3 Crewmitgliedern ausgegangen wird. Es können also nur max. 4 Fahrzeuge pro Gruppe erzeugt werden.

Es ist kein Problem, dieses Array um weitere Einträge zu ergänzen. Zum Beispiel könntest Du diesen Eintrag hinzufügen, wenn die Gruppen in einer Zone genau 6 Mann stark sein soll…

DAC_KI_Count_Level = [[2,4],[2,6],[2,9],[2,12],[6,0]]

Durch dieses Beispiel wurde das Array um einen 5. Eintrag ergänzt. Im Scriptaufruf der Zone musst Du jetzt nur die Nummer **5** angeben, damit Deine Gruppen genau 6 Einheiten enthält: [5,5,50,6]

DAC_Reduce_Value = [600,700,0.1]

Wie der Name schon sagt, sind das die Einstellungen für die Gruppen-Reduzierung. Der 1. Wert bestimmt die Distanz, bei der die Gruppe aufgebaut wird. Der 2. Wert bestimmt die Distanz, bei der die Gruppe reduziert wird. **Achtung**, der 2. Wert sollte dabei immer etwas größer sein als der 1. Wert.

Damit wird erreicht, dass das Reduzierungs-Script eine kleine Pause einlegt, und es nicht permanent mit „reduzieren“ und „aufbauen“ beschäftigt ist.

Der 3. Wert bestimmt die „Aufbaugeschwindigkeit“ der Gruppe. Der Wert **0.1** ist dabei eigentlich eine gute Wahl. Bei langsameren Rechnern könnte man den Wert z.B. auf **0.3** erhöhen.

Die Gruppen-Reduzierung wird deaktiviert, indem man die „Aufbaugeschwindigkeit“ auf **0** setzt.

DAC_KI_Spawn = [30,60,0,120]

Das Zeitverhalten und der Typ des Respawns werden über dieses Array eingestellt.

Der **1. Wert** bestimmt die minimale Zeit für den Respawn. Der **2. Wert** bestimmt die Zeit, die per Zufall auf den **1. Wert** hinzuaddiert wird. Oben der Eintrag würde also den Respawn einer Gruppe nach mindestens 30 Sekunden und maximal 90 Sekunden auslösen.

Achtung, wie schon weiter oben erwähnt, benötigt der Respawn eine gewisse Zeit, bis die alte Gruppe wieder komplett neu erstellt werden kann. Dieses Zeitfenster beträgt ca. 15-45 Sekunden, je nach Größe der Gruppe.

Es bringt also nichts, das Zeitverhalten des Respawns weit unter diese Grenze zu setzen.
DAC_KI_Spawn = [5,10,0] würde bedeuten, dass der Respawn einer Gruppe zwischen 5 und 15 Sekunden benötigt. Es ist aber sehr unwahrscheinlich, dass diese Zeit, aus genannten Gründen, überhaupt erreicht wird. Deshalb den **1. Wert** auf Minimum 30 Sek. belassen. Der **2. Wert** kann dann nach Belieben eingestellt werden.

Der **3. Wert** in diesem Array, bestimmt ob der Respawn lokal oder global funktioniert…

0 = Global : Jede Gruppe kann gespawned werden, egal wo sie herkommt, oder wo sie Wegpunkte hat.

Diese Einstellung ist sehr nützlich, wenn Du z.B. neue Einheiten von außerhalb in das Missionsgebiet schicken möchtest. Da jede neue Gruppe die Wegpunkte der alten Gruppe übernimmt, wird die neue Gruppe sich dorthin bewegen, wo die alte Gruppe ihr Einsatzgebiet hatte.

1 = Lokal : Nur die Gruppen, in deren Master-Zone mindestens ein Spawn-Camp generiert wurde, können gespawned werden. Natürlich nur so lange Spawns zu Verfügung stehen.

Der **4. Wert** bestimmt ob ein Camp, das keine Respawns mehr zu Verfügung hat, nach einer gewissen Zeit zerstört wird. Die Ammobox bleibt dabei aber erhalten. Oben sind z.B. **120** Sekunden eingetragen. Nach dieser Zeitspanne aktiviert sich die Selbstzerstörung. Der Wert **0** deaktiviert diese Selbstzerstörung.

DAC_Delete_Value = [60,300,30]

Diese Werte bestimmen das Löschen von Einheiten. Dieses Merkmal ist auch wieder nur dazu da, Performance freizugeben. Viele herumliegende „tote Körper“ können die Performance schon etwas beeinträchtigen.

Der **1. Wert** bestimmt das Zeitverhalten, also die Zeit in Sekunden bis eine „Leiche“ gelöscht wird.
Der **2. Wert** ist eine Distanzschwelle. Sie bestimmt, ab welcher Entfernung zu allen Spieler-Einheiten die „Leiche“ gelöscht wird.

Man kann beide Werte einzeln verwenden, also nur nach Zeit, oder nur nach Distanz löschen. Die beste Variante ist aber, wenn man beide Werte kombiniert, so wie im Beispiel oben: **[60,300,30]**

Die Einheit wird also erst gelöscht wenn 1. die angegebene Zeit abgelaufen ist, und 2. die Distanz zu allen gültigen Spielern überschritten wird. Somit wird sichergestellt, dass niemand mitbekommt, dass eine Einheit vom Spielfeld gelöscht wird. Außerdem kann man sich weiterhin an den Leichen „bedienen“, solange sie innerhalb des angegebenen Radius gestorben sind und der Spieler sich nicht weiter entfernt.

Der **3. Wert** behandelt einen Sonderfall, und zwar geht es dabei um leere nicht zerstörte Fahrzeuge. Wenn also z.B. ein Trupp sein Fahrzeug verlässt, weil er in Schwierigkeiten gerät, oder auch wenn er einen Erkundungsgang unternimmt, und dann dabei ums Leben kommt, würde ein leeres unbenutztes Fahrzeug zurück bleiben.

Unter Umständen, besonders wenn der Respawn zum Einsatz kommt, könnten so, nach einer gewissen Zeit, viele leere Fahrzeuge auf dem Schlachtfeld „umherstehen“, die auch wieder unnötige Performance verbrauchen.

Um das zu verhindern, wird das Fahrzeug einer Gruppe, die nicht mehr lebt, sich nach dieser angegebenen Zeit selbst zerstören. Denn erst wenn ein Fahrzeug zerstört ist, kann es auch gelöscht werden.

Das Beispiel von oben würde also bedeuten, dass ein Fahrzeug das seine Besatzung „verliert“, sich nach 30 Sekunden zerstört. Danach greifen wieder die Werte 1 + 2. Nach Ablauf der 60 Sekunden wird die Distanz geprüft und bei Überschreitung dieser Entfernung wird das Fahrzeug dann gelöscht.

DAC_KI_Cache = [1,1]

Hiermit wird das Cachen von Einheiten aktiviert bzw. deaktiviert. Wie schon gesagt, kann man diese Funktion deaktivieren, wenn man weder Gebrauch von der Gruppenreduzierung, noch von dem Respawn macht.

Der **1. Wert** legt fest, ob Einheiten vom Typ „Soldier“ gecached werden oder nicht.
Der **2. Wert** legt fest, ob Einheiten vom Typ „Vehicle“ gecached werden oder nicht.

1 … bedeutet, die Funktion ist aktiviert
0 … bedeutet, die Funktion ist deaktiviert

Wenn Du also eine Mission ohne Fahrzeuge erstellst, kannst Du den **2. Wert** auf **0** setzen, um das Cachen von Fahrzeugen zu deaktivieren und umgekehrt.

Wunder Dich nicht, wenn die Infoanzeige meldet, dass „Soldier“ gecached wurden, obwohl Du diesen Einheitentyp deaktiviert hast. Das liegt dann daran, dass für bewaffnete Fahrzeuge automatisch die Crew und damit eine Einheit vom Typ „Soldier“ gecached werden muss.

Zum Deaktivieren musst Du nur **beide Werte** auf **0** setzen.

DAC_Dyn_Weather = [180,10,0.6]

Hier findest Du die Einstellungen für das dynamische Wetter. Der kleinste einstellbare Wert in OFP, mit dem das Wetter geändert werden kann, beträgt **0.02**.

Das bedeutet, bei einer Spanne von **0** (bestes Wetter) bis **1** (schlechtestes Wetter) gibt es **50** Wetterabstufungen. Diese Abstufungen nutzt DAC aus, um einen Wetterwechsel in möglichst kleinen Schritten zu vollziehen.

Ein Beispiel: **Der aktuelle Wetterwert beträgt 0.4 und das Wetter soll auf 0.8 wechseln**

Diese Wetteränderung wird nicht mit einem Schritt vollzogen, also das das Wetter abrupt wechselt, sondern es werden alle dazwischen liegenden möglichen Abstufungen genutzt.

Bei dem oben genannten Wetterwechsel sind das immerhin 20 Abstufungen, die DAC durchführt.

Zurück zu den Einstellungen: **DAC_Dyn_Weather = [180,10,0.6]** wird von DAC folgendermaßen umgesetzt...

Alle **180** Sekunden wird ein neuer Wetterwechsel eintreten (wird per Zufall ermittelt). Die Abstufungen, die sich aus dem Wetterwechsel ergeben, werden mit einer Geschwindigkeit von **10** Sekunden ablaufen.

In unserem Beispiel, also der Wechsel von 0.4 auf 0.8, dauert es 200 Sekunden (10 sek. x 20 Stufen), bis der Wechsel vollzogen ist. Dann dauert es wieder 180 Sekunden, bis ein neuer Wetterwechsel per Zufall eintritt.

Der **3. Wert** bestimmt den maximalen Nebelwert, der bei einem Wetterwechsel auftreten kann. Der Nebel wird unabhängig vom Wetter ermittelt (auch per Zufall), so dass alle möglichen Wettervarianten entstehen können.

DAC_Com_Values = [1,1]

Das sind die Einstellungen für die Texteinblendungen bei der Initialisierung von DAC, die Systemmeldungen von DAC und die Funksprüche.

Bei der Initialisierung von DAC wird angezeigt, wie die Zonen berechnet werden. Du siehst den Fortschritt bei der Berechnung der Wegpunkte und auch bei der Generierung der Einheiten.

Diese Fortschrittsanzeigen kannst Du ausblenden, wenn Du den **2. Wert** von **DAC_Com_Values** auf **0** stellst. Möchtest Du, dass weder die DAC-Systemmeldungen, noch die Funksprüche angezeigt werden, dann setze auch den **1. Wert** auf **0**.

Soll DAC Dir nur die Funksprüche zeigen, also die Chatmeldungen, die von der KI bei bestimmten Reaktionen und Aktionen eingeblendet werden, dann setze den **1. Wert** auf **1**.

Um auch die DAC-Systemmeldungen angezeigt zu bekommen, setze den **1. Wert** auf **2**. DAC zeigt dann an, wenn Einheiten gelöscht werden, Einheiten respawed werden, Gruppen reduziert werden und Zonen aktiviert / deaktiviert werden.

DAC_KI_AddOn = 1

Geplant ist, dass dem DAC verschiedene KI-Addons zu Verfügung stehen. **Grouplink II** von *KeyCat* ist fest eingeplant, aber noch nicht umgesetzt. Möglich wäre damit in Zukunft, dass man die Auswahl hat, welches KI-Addon geladen wird. Hört sich doch gut an…oder?

Im Moment steht „nur" das DAC-Eigene-KI-Addon zu Verfügung, das aber auch sehr gute Dienste leistet ;-)

Mit **DAC_KI_AddOn = 1** wird die DAC-KI geladen und mit **DAC_KI_AddOn = 0** werden nur die nötigsten Scripte geladen, um z.B. die generierten Einheiten von WP zu WP zu schicken.

Was die KI so drauf hat, steht zum einen in der Vorschau.pdf und es gibt die Demo **DAC_10_AI_Reaction**, die Dich das auch „spüren" lässt. In der Demo **DAC_11_AI_Reaction** zeigt Dir DAC, dass die KI nicht nur auf Dich reagiert, sondern auf alle generierten feindlichen Einheiten.

Diese Demo ist überhaupt ein gutes Beispiel dafür, wie dynamisch eine DAC-generierte Mission ablaufen kann und sich während der Spielzeit „entwickelt". Das kannst Du alles sehr gut auf der Map verfolgen, und die entsprechenden Funksprüche zeigen auch sehr gut das dynamische Verhalten der KI. Die DAC-Systemmeldungen informieren Dich über die Aktionen, die DAC während der Mission erledigt.

DAC_KI_Level = 4

Den KI-Level kannst Du in 4 Stufen wählen, wobei **4** die stärkste Stufe ist und **1** die schwächste Stufe darstellt.

Auf Stufe **1** reagieren die Einheiten erst sehr spät auf feindliche Einheiten, die Reaktionen sind träge und Verstärkung kommt auch eher selten. Also eigentlich etwas für „blutige Anfänger".

DAC_Res_Side = 0

Achtung, DAC unterstützt in dieser Version nur **WESTEN** oder **OSTEN** als Seite für den Spieler!

Mit der Einstellung **DAC_Res_Side** gibst Du an, auf welcher Seite der Widerstand steht, also hier bitte das einstellen, was Du im Editor eingestellt hast. Die Einstellung im Editor „auf beiden Seiten", wird von DAC ebenfalls **nicht** unterstützt. Das bitte beachten!

DAC_Res_Side = 0 bedeutet...der Widerstand kämpft auf Seite der **Ost-Einheiten**
DAC_Res_Side = 1 bedeutet...der Widerstand kämpft auf Seite der **West-Einheiten**

Im Prinzip ist das eine gute Möglichkeit, die Seite **Ost** oder **West** mit mehr als 63 Gruppen zu versorgen, auch im Hinblick auf die Möglichkeit, dass Zonen später aktiviert werden können, und Du dadurch relativ groß angelegte Missionen planen kannst.

Ein Beispiel: Du spielst als West-Einheit gegen den bösen Osten. Du möchtest auf Ost-Seite 4 Zonen mit je 20 Gruppen generieren, die nach und nach aktiviert bzw. deaktiviert werden. In allen Zonen sollen die gleichen Einheiten generiert werden, sagen wir mal RHS-Einheiten. Nun ist das unter „normalen" Umständen nicht möglich, RHS-Einheiten auf Res-Seite einzusetzen, da sie im Editor der Seite Ost fest zugeordnet sind.

Mit DAC ist das aber möglich, Du kannst alle Einheiten auf jeder Seite generieren und somit auch RHS-Einheiten auf Res-Seite. Du musst also nur **DAC_Res_Side = 0** einstellen, damit der Widerstand auf Ost-Seite kämpft, und schon ist die oben genannte Variante mit 4 Zonen möglich.

Wie man das macht, wurde ja bereits unter folgendem Thema weiter oben erklärt…
Die Wahl der Seite, das Einheiten-Addon für eine Zone, das Grundverhalten der KI

DAC_Marker = true

Diese Variable aktiviert / deaktiviert die DAC-Marker. Diese Marker haben den Zweck, Dir die Initialisierung von DAC anzuzeigen und damit die generierten Wegpunkte und Einheiten.

Du kannst so schnell feststellen, ob das generierte Ergebnis in etwa Deinen Vorstellungen entspricht, also ob die Wegpunkte korrekt platziert werden, wo die Spawn-Camps erzeugt werden usw.

Dann siehst Du natürlich auch, wie die Einheiten sich innerhalb ihrer zugewiesenen Zonen bewegen und Du bekommst eher ein Gefühl dafür, wie DAC Deine Einstellungen umsetzt.

Bitte denke daran, dass Du die DAC-Marker benötigst, wenn Du dieses Merkmal benutzen möchtest.

Am besten ist, Du kopierst Dir diese Marker aus irgendeinem DAC-Editor-Beispiel raus und fügst sie dann in Deine Mission ein.

DAC_Config_Unit.sqs

Dieses Config-Script hat es wortwörtlich „in sich“. Hier findest Du sehr viele Einheiten-Klassen, die Du mit DAC ansprechen 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 wiederum 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.

So ein „Block“ sieht folgendermaßen aus…

```
;\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_0
;/////////////////////////

_Unit_Pool_S = ["SoldierECrew","SoldierEPilot","OfficerE","Soldiereb","Soldiereg","Soldierelaw".....]
_Unit_Pool_V= ["Ural","UAZ","BRDM"]
_Unit_Pool_T= ["BMP","T72","T80","ZSU"]
_Unit_Pool_A= ["MI17","MI24"]
_Unit_Pool_C= ["rus_vlajka.pac","CampEast","Shed",[100,3]]
```

[z1,[1,0,0],[ ],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Unter der Config-Nr. **0** sind die Standard Ost-Einheiten gespeichert, die in den meisten DAC-Demos zum Einsatz kommen. Ich habe hier noch mal den Scriptaufruf reinkopiert, damit Du den Parameter wieder vor Augen hast, der diese verschiedenen Unit-Config-Nr. anspricht.

Die Nr. **215** erzeugt z.B. BW-Mod-Einheiten, da unter dieser eindeutigen Nummer diese Einheiten in den 4 Kategorien abgelegt sind….

```
;\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_215
;/////////////////////////
```

```
_Unit_Pool_S = ["BWMOD_CrewT","BWMOD_HeloPilot","BWMOD_Officer","BWMOD_Soldier".....]
_Unit_Pool_V = ["BWMOD_MAN5tgl","BWMOD_MAN7tgl","BWMOD_MAN10tgl"]
_Unit_Pool_T = ["BWMOD_Dingo_MG3","BWMOD_Leo1A5","BWMOD_Wiesel","BWMOD_Marder...."]
_Unit_Pool_A = ["BWMOD_EC135","BWMOD_BO105PAH1","BWMOD_BO105VBH"]
_Unit_Pool_C = []
```

Folgende Kategorien findest Du hier wieder:

_Unit_Pool_S	=	Kategorie **Infanterie**
_Unit_Pool_V	=	Kategorie **unbewaffnete Fahrzeuge**
_Unit_Pool_T	=	Kategorie **bewaffnete Fahrzeuge**
_Unit_Pool_A	=	Kategorie **Helikopter**
_Unit_Pool_C	=	Kategorie **Spawn-Camps**

Klar dürfte sein, dass Du nur die jeweilige Kategorie erzeugen kannst, wenn dort auch gültige Einträge vorhanden sind.

Achtung, im **_Unit_Pool_S** gibt es eine Bedingung, die immer erfüllt sein muss, auch wenn Du Infanterie gar nicht erzeugen willst**: Es müssen dort immer mindestens 3 Einträge vorhanden sein!**

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

Der 1. Eintrag stellt den Einheitentyp für die **Crew** von **bewaffneten Fahrzeugen**.
Der 2. Eintrag stellt den Einheitentyp für die **Piloten** von **Helikoptern**.
Der 3. Eintrag stellt den Einheitentyp für den **Leader** einer **Infanterie**-Gruppe.

Schau Dir noch mal die 2 Config-Blöcke oben an, da siehst Du das auch noch mal schön.

Im Zweifelsfall, also wenn ein Addon keine passenden Einheiten enthält, ersetze diese dann durch Standard-Einheiten oder andere passende Einheiten Es spielt im Prinzip keine Rolle, welche Einheiten Du einträgst, es müssen halt nur welche vorhanden sein.

Alles was im **_Unit_Pool_S** nach dem 3. Eintrag kommt, ist dann für die zufällige Auswahl an Einheiten bestimmt. Je mehr Einheiten Du einträgst, umso vielfältiger werden die Gruppen generiert.

Wenn Du mal in einer Zone nur Infanterie-Gruppen erzeugen möchtest, die aus nur einem Einheitentyp bestehen sollen, z.B. aus Law-Einheiten, dann müsstest Du folgendes machen:

_Unit_Pool_S = ["SoldierECrew","SoldierEPilot","SoldierELaw","SoldierELaw"]

Die ersten beiden Einträge interessieren uns in diesem Fall nicht. Wichtig ist, dass der 3. Eintrag, also der Leader der Gruppe, auch eine Law-Einheit ist und das der 4. Eintrag, also ab da wo per Zufall generiert wird, eben auch die Law-Einheit eingetragen ist.

So bleibt DAC nichts anderes übrig, als ausschließlich Law-Einheiten zu „produzieren“ ;-)

Kleiner Tipp: **Wenn Du einen neuen „Block“ mit Einheiten-Klassen aus z.B. einem neuen Addon anlegst, dann leg’ ihn erstmal komplett an, also mit allen Einheitentypen die das Addon hergibt. Evtl. noch splitten, wenn das Addon mit verschiedenen „Bemalungen“ daherkommt.**
Aus diesem „kompletten Block“ kannst Du nun weitere Blöcke „ableiten“, die Du dann mit DAC verwendest. So hast Du immer Zugriff auf alle Einheiten des Addons und brauchst bei Bedarf nur aus diesem „kompletten Pool“ die passenden Einheiten rauskopieren. Die Ableitungen müssen natürlich auch wieder eine eigene Config-Nr. haben

Noch’n Tipp: **Du kannst Einheiten-Klassen auch mehrfach innerhalb eines Pools eintragen. Das erhöht die Wahrscheinlichkeit, dass dieser Einheitentyp erzeugt wird**.

Das hätte ich doch fast noch vergessen: _Unit_Pool_C = **["rus_vlajka.pac","CampEast","Shed",[100,3]]**

In diesem Array kannst Du zu jedem Config-Block die Angaben für die Spawn-Camps machen:
Der 1. Eintrag enthält die Textur für die Flagge, die in dem Camp aufgestellt wird.
Der 2. Eintrag enthält „Gebäude 1“, dass im Camp aufgestellt wird. Das Gebäude hat keine Funktion.
Der 3. Eintrag enthält „Gebäude 2“, dass im Camp aufgestellt wird. Hier werden die Einheiten respawned
Der 4. Eintrag enthält die Einstellungen für die „Ammobox“, die im Camp generiert wird:
1. Wert = Prozentangabe, wie viel Waffensets (incl. Muni) aus der Zone in dieser Ammobox enthalten sind
2. Wert = Wie viel Sets von jedem Typ in die Ammobox kommen.

DAC_Config_Behaviour.sqs

In diesem Config-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, also ohne Feindkontakt zu haben.

Hier mal ein „Block“ aus diesem Script…

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Behaviour_Typ_3
;////////////////////////////
_setSkill      = [0.8,1]
_setCombat     = ["white","yellow"]
_setBehav      = ["aware","combat"]
_setSpeed      = ["normal","full"]
_setForm       = ["line","vee","column","wedge","stag column","ech left","ech right"]
_setFleeing    = [0]
_setFlighth    = [50,150]
_setPause      = [5,5,10,25,5]
_setSay        = []
_setSupport    = 1
```

[z1,[1,0,0],[ ],[ ],[ ],[ ],[0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Hier auch wieder der Scriptaufruf und die entsprechende Verbindung mit dem Parameter.
Diese Zone wird also mit dem **Behaviour_Typ_3** gefüttert. Die Einstellungen verhalten sich folgendermaßen:

_setSkill | Dieses Array benötigt immer 2 Einträge
| 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 | 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 | 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 | 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 | 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 | Dieser Wert bestimmt das **Fluchtverhalten** der Einheiten. Der Wert muss zwischen 0 und 1
| liegen.

_setFlighth | Dieses Array benötigt wieder 2 Einträge.
| 1. Eintrag = **Minimalflughöhe** für Helis in dieser Zone
| 2. Eintrag = **Maximalflughöhe** für Helis in dieser Zone

_setPause | Dieses Array benötigt genau **5** Einträge:
| 1. Eintrag = Maximale Pause an den **Wegpunkten** für **Infanterie**
| 2. Eintrag = Maximale Pause an den **Wegpunkten** für **unbewaffnete Fahrzeuge**
| 3. Eintrag = Maximale Pause an den **Wegpunkten** für **bewaffnete Fahrzeuge**
| 4. Eintrag = Maximale Pause an den **Start-/Landezonen** für **Helis**
| 5. Eintrag = Maximale Pause für die am **Spawn-Camp** stationierte Gruppe

_setSay | In diesem Array kannst Du Standard-Say-Sounds, oder auch eigene Sounds ablegen,
| die dann jeweils der Leader einer Gruppe „ausspricht“, wenn eine bestimmte Situation
| eintrifft.
|
| DAC hält dafür 5 verschiedene Situationen bereit und für jede Situation kannst Du mehrere
| Say-Sounds ablegen, aus denen dann per Zufall ein Sound ausgewählt wird.
|
| **Diese 5 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 keinen Feindkontakt mehr**
|
| Möchtest Du dieses Merkmal nutzen, müssen in dem Array **_setSay** 5 weitere Arrays
| vorhanden sein, auch wenn Du nur 1 oder 2 Situationen mit Sounds belegen möchtest.
|
| Hier ein Beispiel :
|
| **_setSay = [["rus3","rus4"],["rus6","rus10"],["rus2"],[],["rus11","rus14","rus19"]]**
|
| Der Leader der Gruppe ruft „**rus3**“ oder „**rus4**“ wenn er einen seiner Wegpunkte erreicht hat.
| Der Leader der Gruppe ruft „**rus6**“ oder „**rus10**“ wenn seine Gruppe Feindeinheiten entdeckt.

| Der Leader der Gruppe ruft „**rus2**“ wenn seine Gruppe unter Beschuss steht.
| Der Leader der Gruppe ruft **nichts** wenn seine Gruppe Hilfe anfordert (**das Array ist leer**).
| Der Leader der Gruppe ruft „**rus11**“, „**rus14**“, oder „**rus19**“ wenn seine Gruppe keinen
| Feindkontakt mehr hat.
|
| Ist das KI-Addon deaktiviert, kommt nur Situation 1 zum tragen, also wenn die Gruppe einen
| ihrer Wegpunkte erreicht hat.

_setSupport | Dieser Wert bestimmt, ob die Einheiten aus dieser Zone einem Hilferuf nachkommen, also
| als Verstärkung ausrücken, um eigene Einheiten zu unterstützen.
|
| Damit kann man erzwingen, dass die Gruppen ihre Zonen bzw. ihre Routen zwischen mehreren
| Zonen, zwecks Unterstützung, nicht velassen und somit die Zonen nicht „leer laufen“.
| Achtung, diese Einstellung greift nur, wenn das DAC-KI-Addon aktiviert wurde.

DAC_Config_Waypoints.sqs

In dieser **Config_Waypoints.sqs** liegen die Parameter für die Wegpunktgenerierung. Die Einstellungen gelten Global für alle Zonen. Es ist aber geplant, auch diese Config erweiterbar zu machen, also so wie die Configs für **Units** und **Bevaviour**, so, dass jede Zone mit individuellen Parametern gespeist werden kann.

Für den Fall, dass Du diese Einstellungen mal anpassen musst, z.B. für eine besondere Insel, dann setze zur Kontrolle auf jeden Fall die **DAC_Marker** ein. Sie zeigen Dir dann, wie DAC deine Einstellungen umsetzt.

Gut, in diesem Script gibt es 5 „Blöcke“ und jeder Block steht für eine Einheiten-Kategorie. Die Parameter erkläre ich mal anhand von dem Block „Tank“, also bewaffnete Fahrzeuge (ich weiß, ist der falsche Ausdruck)

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#WP_Typ_3
;----------->   | Tank |
;////////////////////////////////
_CheckRadius    = 40
_checkAreaH     = 35
_checkMaxh      = 1000
_checkMinH      = 1
_checkNear      = 0
_checkObjH      = 14
_checkCount     = 200
```

_CheckRadius = 40

Das ist der Radius, in dem an einer zufälligen Position nach Objekten gescannt wird. Je größer dieser Radius ist, desto mehr Zeit wird für den Scannvorgang an einer Position benötigt. Bei Infanterie ist diese Radius-Angabe mit **15** relativ klein, deshalb sind Wegpunkte für Infanterie auch wesentlich schneller gefunden. Das liegt daran, dass die Wegpunkte für Infanterie auch in Wäldern liegen können.

Für Fahrzeuge muss man aber einen größeren Wert einstellen, damit keine Wegpunkte in bewaldeten Gebieten generiert werden. Das hat folgenden Grund: Diese Wälder, die Du z.B. von Everon kennst, bestehen aus zusammengesetzten Segmenten (ein Segment enthält wiederum mehrere Bäume). Diese Segmente sind ganz normale OFP-Objekte.

Da diese Segmente relativ groß sind, liegen sie entsprechend weit auseinander. Damit der DAC-WP-Scanner auch wirklich jedes Segment erfasst, muss eben der Radius groß genug sein. Ist der Radius zu klein bemessen, kann es eben passieren, dass Wegpunkte mitten im Wald generiert werden.

Der Wert bei Helis und Spawn-Camps ist sogar noch größer. Damit wird ausgeschlossen, dass in diesem Radius keine störenden Objekte im Weg sind…besonders wichtig bei Helis, die brauchen Platz ;-)

Bei Infanterie kannst Du versuchen, den Wert noch zu reduzieren. Je kleiner der Radius, desto wahrscheinlicher sind generierte Wegpunkte in Ortschaften.

Achtung, diesen Wert bitte nur im 5m Schritten ändern.
Mögliche Werte sind also…5,10,15,20,25,30 usw.

_checkAreaH = 35

Dieser Wert bestimmt den maximalen Höhenunterschied innerhalb des Scannbereichs. Du kannst also bestimmen, wie uneben das Gelände für einen Wegpunkt sein darf. Für Helis und Camps muss der Bereich für eine gültige Position ziemlich flach sein. Klar, sonst würde ein Heli unter Umständen an einem Berghang generiert. Das gleiche gilt für Spawn-Camps. Auch da muss das Gelände ziemlich flach sein.

_checkMaxh = 1000

Das ist die maximale Höhenangabe, für Wegpunkte. Oberhalb von diesem Wert werden keine Wegpunkte mehr generiert. Damit kann man ausschließen, dass Wegpunkte auf unerreichbaren Höhen generiert werden.

_checkMinH = 1

Das ist die minimale Höhenangabe, für Wegpunkte. Unterhalb von diesem Wert werden keine Wegpunkte mehr generiert. Damit kann man z.B. erreichen, dass keine Wegpunkte direkt am Strand oder Wasser generiert werden und somit auch keine Einheiten dort erstellt werden.

_checkNear = 0

Dieser Wert “erzwingt” einen Mindestabstand von Wegpunkten untereinander. Das ist besonders wichtig im Zusammenhang mit Helis und Spawn-Camps. Damit erreichst Du, dass z.B. die Spawn-Camps nicht direkt nebeneinander liegen, sondern einen gewissen Abstand zueinander haben.

Achtung, durch diesen Wert (wenn > 0), dauert die Wegpunktgenerierung unter Umständen wesentlich länger, da alle Wegpunkte miteinander verglichen werden.

_checkObjH = 14

Dieser Wert bestimmt, wie „hoch“ ein Objekt höchstens sein darf, damit der Scanner es noch zulässt. Befinden sich in dem Scanbereich also nur Objekte, die kleine/niedriger sind, als der angegebene Wert, wird die Position als gültiger Wegpunkt erkannt. Befindet sich in dem Scanbereich dagegen nur ein Objekt, dessen Höhenwert über dem angegebenen Wert liegt, wird der Scanvorgang an dieser Position sofort abgebrochen und es wird eine neue Position gesucht.

Bei der Kategorie Heli z.B., ist der Wert mit **4** relativ klein. Das muss in diesem Fall auch sein, damit in seinem Bereich keine größeren Objekte auftauchen, die beim Starten/Landen stören könnten. Also Büsche und so’n Zeugs, werden durch diesen Wert gerade noch akzeptiert ;-)

_checkCount = 200

Dieser Wert dient dazu, die Anzahl der Versuche einen gültigen Wegpunkt zu finden, einzugrenzen. So wirst Du (durch eine Fehlermeldung) frühzeitig von DAC darüber informiert, das es in einer Zone Probleme mit der Wegpunktgenerierug gibt.

Sorg’ dann dafür, dass Du z.B. die entsprechende Zone an eine andere/günstigere Position versetzt, oder evtl. die Abmessungen der Zone vergrößerst, um somit DAC die Möglichkeit gibst, auch die Wegpunkte generieren zu können.

Multiplayerbetrieb

Möchtest Du DAC in einer Multiplayermission einsetzen, ist es zwingend erforderlich, das eine Logic-Einheit mit dem Namen „Server“ auf der Map platziert wird.

Ansonsten gibt es im Multiplayerbetrieb noch folgende Einschränkungen:

Die Fortschrittsanzeigen (Zonen vermessen, Wegpunkte generieren, Einheiten erzeugen) werden während der Initialisierung von DAC **nicht** angezeigt. Die Clients bekommen lediglich angezeigt, dass DAC gestartet wurde, und wenn DAC die Initialisierung erfolgreich abgeschlossen hat.

Die Funksprüche und DAC- Systemmeldungen werden **nicht** angezeigt.

Die Sounds für die 5 verschiedenen KI-Situationen werden **nicht** zu hören sein.

Möchtest Du ein Script / mehrere Scripte auf Server- Seite bzw. auf Client- Seite starten, nachdem DAC mit der Initialisierung fertig ist, kannst Du das am Besten an folgender Stelle tun:

Im Script **\DAC\DAC_Config_Creator.sqs** gibt es dafür je eine Stelle, wo die entsprechenden Scriptaufrufe eintragen werden können:

#StartScriptOnClient
← dort den Scriptaufruf für die **Clients** eintragen
exit

#StartScriptOnServer
← dort den Scriptaufruf für den **Server** eintragen
exit

Tipps und Tricks

Bei den Inseln „Malden“ und „Tonal“ musst Du in der **DAC_Config_Waypoints.sqs** eine kleine Anpassung vornehmen, damit die Wegpunkte für Fahrzeuge korrekt generiert wird. Folgenden Wert musst Du abändern:

_checkObjH = 8 (Standard = _checkObjH = 14)

und zwar bei **#WP_Typ_2** und **#WP_Typ_3**

Mit DAC neigt man schnell dazu, zu viele Einheiten zu generieren… ist ja auch schön einfach. Aber lieber große Zonen mit weniger Einheiten, als kleine Zonen mit zu vielen Einheiten generieren. Auch wenn DAC in der Lage ist, entfernte Gruppen zu reduzieren, wirst Du irgendwann auch die entfernten Gruppen erreichen und das kann dann, wenn es zu viele sind, unspielbar werden.

Denk bitte daran, das DAC nur eine gewisse Anzahl an Wegpunkten per **DAC_marker** anzeigen kann. Ist das Kontingent an Markern aufgebraucht, wird halt auch nichts mehr angezeigt. Das führt schon mal dazu, dass man meint, DAC generiert nicht alle angegebenen Wegpunkte und Einheiten

---

Du kannst das „Vermessen der Zonen“ umgehen und somit etwas Zeit bei der Initialisierung sparen, wenn Du in jeder Zone die Angabe über ihre Größe angibst. Dafür hält DAC zwei optionale Parameter bereit:

Standard : [z1,[1,0,0],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Erweitert: [z1,[1,0,0,500,800],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Größe X-Achse der Zone

Größe Y-Achse der Zone

Achtung, dass macht nur Sinn, wenn Du alle Zonen mit diesem optionalen Parametern ausführst.

Der Nachteil ist, dass bei jeder Größenänderung der Zone, auch unbedingt diese Parameter angepasst werden müssen. Ansonsten „übersieht“ DAC die Größenänderung. Das fällt auch nicht sofort auf.

Wenn Du DAC in eine bestehende Mission einbinden willst, musst Du evtl. die Bedingung in den Zonen anpassen. Die Bedingung lautet normalerweise „**time > 1**“, also wenn die Mission älter als 1 Sekunde ist, beginnt DAC mit der Initialisierung.

Je nach dem was in Deiner Mission alles noch initialisiert werden muss, oder auch wenn Du zu Beginn direkt eine Cut-Szene startest, oder wie auch immer, ist es sinnvoll mit DAC zu warten, bis diese ganzen Sachen „erledigt“ wurden.

Das heißt, die Bedingung könnte dann z.B. „time > 120“ oder „IntroFertig“ heißen, je nach dem wie lange es dauert, bzw. welche Variable den optimalen Zeitpunkt wiedergibt. Die Bedingung muss dann auf jeden Fall in allen Zonen gleich sein.

Es ist auch möglich, die Scriptaufrufe aus den Zonen, in ein Script auszulagern. Dieses Script könnte dann zum Beispiel so aussehen:

```
;[]exec „CallZones.sqs“
@time > 1
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,15,5],[ ],[2,1,25,12],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,0,0],[ ],[ ],[3,1,5,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,0,0],[ ],[ ],[25],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z5,[3,1,0],[ ],[ ],[3,1,20,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z6,[4,1,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
.....
exit
```

In den Zonen selbst, kann dann der Scriptaufruf und die Bedingung (time > 1) rausgenommen werden. Wichtig ist nur, dass die Zonennamen in diesem Script und die Namen der Auslöser übereinstimmen.

Der Vorteil dieser Variante ist zum einen, dass man Änderungen zentral vornehmen kann, und das man eine Variable sozusagen „vorschalten“ kann, die dann für alle Scriptaufrufe gilt. Hier ein Beispiel:

```
;[]exec „CallZones.sqs“
_KiLevel        = 4
_KiType         = 33
@time > 1
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,15,5],[ ],[2,1,25,12],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,0,0],[ ],[ ],[3,1,5,8],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,0,0],[ ],[ ],[25],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z5,[3,1,0],[ ],[ ],[3,1,20,8],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z6,[4,1,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
.....
exit
```

Die beiden Variablen _KiLevel + _KiType könnten natürlich auch global sein und z.B. aus einem Dialog oder aus den Multiplayer-Settings kommen ;-)

DAC kommt auch mit stationären Einheiten klar ☺ Am Besten ist, Du legst dafür eine extra **Unit_Config** an. Unten mal ein Beispiel: Die stationäre Einheit unter _Unit_Pool_T eintragen. Dann noch den Ersten Eintrag unter _Unit_Pool_S mit einer Einheit Deiner Wahl belegen, die dann als „Besatzung“ dient. Normalerweise steht ja an dieser Stelle eine reguläre „Crew-Einheit“ um damit bewaffnete Fahrzeuge zu bemannen.

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_501
;//////////////////////////

_Unit_Pool_S        = ["Soldiereb "," Soldiereb "," Soldiereb ","Soldiereb"]
_Unit_Pool_V        = [ ]
_Unit_Pool_T        = ["tpillboxeast"]
_Unit_Pool_A        = [ ]
_Unit_Pool_C        = [ ]

goto "SetPool"
```

Danksagung

In Erster Linie möchte ich ganz klar **BIS** danken ☺ Was wären wir ohne diese Jungs ;-)

Danke natürlich auch an das komplette Mapfact-Team für die gute Kooperation und den Freiraum, den mir das Team für die Entwicklung von DAC gegeben hat.

Besonderer Dank geht außerdem noch an :

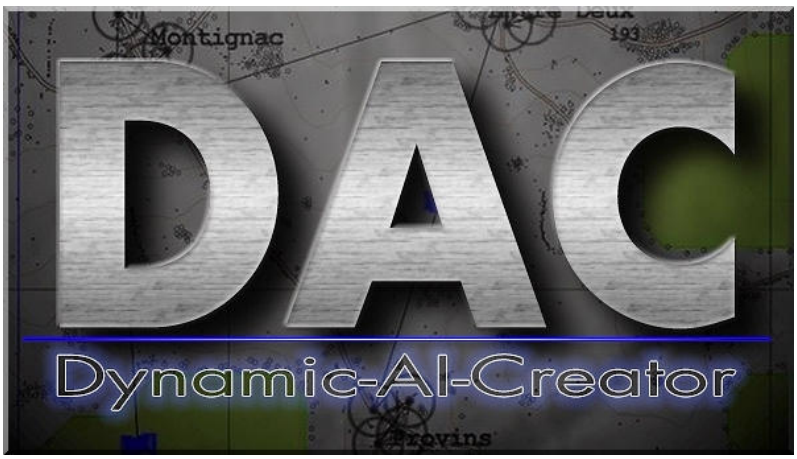
MCPXXL	DACWriter, Unit_Config, Tester
BadAss	Übersetzer, Tester
Mr.Murray	Tester
SnipingJack	DAC-Logo
Simba	Übersetzer, Tester
Aron	Tester
Honchoblack	Tester
Sanctuary	Tester
Tacrod	Tester
Dr.Strangelove	Tester

Wenn ich noch jemanden vergessen habe, bitte ich hiermit um Verzeihung.

Vielen Dank !!!

DAC ist inoffiziell, also bitte keine Klagen an BIS oder Codemasters.
Macht mich nicht für eventuelle Rechnerprobleme verantwortlich.
Die Installation und Benutzung erfolgt auf eigene Gefahr!

Du siehst, DAC bietet sehr viele Möglichkeiten. Ok, diese Liesmich.pdf ist echt krass und wer bis hierhin gelesen hat, muss wirklich ein großes Interesse gehabt haben, aber ich denke es lohnt sich auch ☺



Gruß
Silola

silola@mapfact.net
silola@freenet.de

ICQ: 287-388-751

Hier noch ein paar Anhaltspunkte, was ich in der nächsten Version von DAC gerne umsetzen würde:

 # bewegliche Zonen
 # Wegpunkteinstellungen für jede Zone
 # Objektgenerierung in Zonen
 # GroupLink II v. KeyCat