DAC V3.0 Bonus Funktionalität - von MadRussian (übersetzt von MemphisBelle)

Beschreibung der 6 Bonus-Funktionen

Dieser Teil der “Liesmich” beschreibt die Verwendung des Bonusmaterials von MadRussian.
Dazu gehören 2 extra Demo-Missionen, die Du innerhalb der 33-teiligen DAC-Demo kennen lernst.

Folgendes AddOn wird benötgt: DAC_Source (benötigt MAP_EU_V1.00)

DAC_fInsideOfZones

Diese neue Funktion ermöglicht es Dir mit großer Genauigkeit einen Wegpunkt in einer Zone zu bestimmen.

Bitte bedenke, dass sobald DAC 3.0 gestartet wird, nun eine "Location" unter jeden neuen DAC-Zone erzeugt wird. Mit dieser neuen Funktion ist es möglich entweder die Liste der Zonen oder der Wegpunkte abzurufen, die sich innerhalb einer Zone befinden.

Diese Funktion arbeitet mit Rechtecken und Ellipsen, allerdings nicht mit Polygon-Zonen.
Um dieses Feature zu nutzen, muss DAC bereits initialisiert sein. Hier ist der Scriptaufruf:

```
_zones = [position, info_requested] call DAC_fInsideOfZones;
```

1. position (array)
2**,** info_requested (möglich sind die Werte "Zones" oder "ZoneLocs")

Beispiele: _zones = [position player, "Zones"] call **DAC_fInsideOfZones**;
_locations = [position player, "ZoneLocs"] call **DAC_fInsideOfZones**;

Zusätzlich kann man über eine „Location“ die dazugehörige Zone erfahren, bzw. über die „Zone“ die jeweilige Location ermitteln.

Beispiele: _zoneLoc = z1 getVariable "DAC_ZoneLoc";
_zone = _zoneLoc getVariable "DAC_Zone";

DAC_fReadMarkers

Mit dieser Funktion ist es möglich auf schnelle Art und Weise eine Vielzahl von eigenen Positionen zu erstellen und diese dann in einen Array abzuspeichern, und das ganze mit nur einem einzigen Befehl.

Speziell diese Funktion kann innerhalb und außerhalb von DAC genutzt werden.

Zuerst erstelle im Editor einen Marker auf der Karte (In diesem Fall ist ein kleines Icon wie ein Punkt zu empfehlen). Nachdem dieser Marker einen Namen zugewiesen bekommen hat, kann dieser so oft wie nötig kopiert und wieder eingefügt werden (STRG+C / STRG+V), um so viele eigene Positionen wie benötigt zu erstellen.

Wenn Dein Set von Positionen fertig ist, nutze folgenden Scriptaufruf:

```
_points = [base_string, info_requested, blank_limit, delete] call DAC_fReadMarkers;
```

1. base_string (string) - Der Name des ersten Makers in diesem Set
2. info requested (string) – Einer der folgenden Werte ist möglich:
   "Pos", "Pos&Dir", "Pos&Dir&Txt", oder "Pos&Dir&Txt&Name"
3. blank limit (numeric) – Die Anzahl der Marker "Lücken" die man Ignorieren kann.
   Der Wert 1000 wird hier als Beispiel vorgeschlagen.
4. delete (boolean) - Option zur Auswahl, ob die Marker gelöscht werden sollen, oder nicht, nachdem sie von DAC eingelesen wurden

Beispiele:

```
_positions = ["WayPt", "Pos", 1000, true] call DAC_fReadMarkers;
```
Gibt ein Array von Positionen zurück, die durch den Basis-Marker “WayPt” und seine Kopien erstellt wurde (die Kopien heißen dann “WayPt_1”,“WayPt_2”,“WayPt_3”,“WayPt_4”…..).
Die Marker werden nach Aufruf der Funktion wieder gelöscht.

```
_posDirPairs = ["StartingVehics", "Pos&Dir", 1000, false] call DAC_fReadMarkers;
```
Gibt ein Array mit Position und Ausrichtung zurück, die durch den Basis-Marker “StartingVehics” und seine Kopien erstellt wurde (die Kopien heißen dann:
“ StartingVehics_1”,“ StartingVehics_2”,“ StartingVehics_3”,“ StartingVehics_4”…..).
Die Marker werden nach Aufruf der Funktion **nicht** gelöscht.

Hinweise:

Du hast jederzeit die Freiheit jeden x-beliebigen Marker innerhalb eines Sets zu löschen (oder auch später noch welche hinzuzufügen), ohne Angst haben zu müssen, dass DAC damit nicht zurecht kommt bzw. Positionen vermisst. Dafür ist das "blank_limit".

Ein Beispiel: Mal angenommen die Marker, die sich auf der Karte befinden heißen …

"WayPt", "WayPt_1", "WayPt_2", "WayPt_4", und "WayPt_7"

Ein blank_limit von 2 würde die Daten von allen 5 Marker speichern und zurückgeben.
Bei einem genutzten blank_limit von 1 würden nur die ersten 4 Marker entsprechende Informationen zurückgeben. Daher nutzt den angegeben Wert von 1000 und vergesst den Rest!

Plaziert den Basis Marker am besten an einer Stelle auf der Karte, die man sich gut merken kann, oder ihr geht das Risiko ein, diesen in einem Meer von Punkten zu verlieren

Um im Briefing die Karte „sauber“ zu halten, nutzt Ihr am besten die “init.sqf” (möglichst vor irgendwelchen Sleep-Befehlen). Das wird zur Folge haben, dass alle eigenen Marker bereits gelöscht sein werden, bevor der Spieler das Briefing überhaupt zu sehen bekommt.

Am schnellsten erstellt Ihr ein Vielzahl von benutzerdefinierten Markerpositionen mit Hilfe der Kopieren/Einfügen-Funktion (immer abgeleitet von dem Basis-Marker).

DAC_fGetZoneWaypoints

Hiermit könnt Ihr direkt die Wegpunkte innerhalb einer jeden DAC Zone abfragen.
Um dieses Feature nutzen zu können, muss DAC dafür allerdings bereits initialisiert sein.
Hier ist der Scriptaufruf:

```
_positions = [zone, zone_type] call DAC_fGetZoneWaypoints;
```

1. Zone
2. Art der Zone, abhängig davon, was genereirt warden soll ("S", "V", "T", "A", oder "C")
   Erstellt jeweils Wegpunkte für: "**S**" Für Infanterie
   "**V**" Für Fahrzeuge
   "**T**" Für Kettenfahrzeuge
   "**A**" Für Flugzeuge
   "**C**" Für Camps

Beispiel:
```
_positions = [z1, "S"] call DAC_fGetZoneWaypoints;
_positions = [z1, "T"] call DAC_fGetZoneWaypoints;
```

DAC_fReplaceZoneWaypoints

Außerdem ist es möglich, die Wegpunkte in einer jeden DAC Zone jederzeit neu zu platzieren.

Einheiten, die sich innerhalb der Zone befinden, beginnen innerhalb der ersten Sekunden des Funktionsaufrufes die neuen Wegpunkte zu nutzen.

Um dieses Feature nutzen zu können, muss DAC dafür allerdings bereits initialisiert sein.
Hier ist der Scriptaufruf:

```
_result = [zone, type, new_positions, max_replace] call DAC_fReplaceZoneWaypoints;
```

1. **zone,** Name der Zone

2. **type** (string) - Art der Zone ("S", "V", "T", "A", oder "C") – Erklärung siehe oben

3. **new_positions** (array) - Ein Set von benutzerdefinierten Positionen

4. **max_replace** (numeric) -Maximale Anzahl von Wegpunkten, die ersetzt werden können.
   Wenn dieser Parameter genutzt wird, wird diese Funktion entsprechend viele Wegpunkte per Zufall aus dem Set benutzen.
   Wird dieses Feld freigelassen wird das komplette Set genutzt.

Folgende Werte werden zurückgegeben:
[replacement_report, actual_replaced, of_total, _type, applied_positions]:

replacement_report (string)	=	Rückmeldung über den Status
actual_replaced (numeric)	=	Anzahl der Wegpunkte, die wirklich ersetzt wurden
of_total (numeric)	=	Volle Anzahl der Zonen Wegpunkte (des jeweilige Wegpunkt Typen)
type (string)	=	Art der zurückgegebenen Wegpunkte
replaced_positions (array)	=	Array der eigentlich ersetzten Wegpunkte

Beispiel:

[z1, "V", _customVehPoints] call **DAC_fReplaceZoneWaypoints**;
Ersetzt Wegpunkte in Zone z1, und zwar von der Kategorie „V“ (Vehicle).
Es werden soviel Wegpunkte ersetzt, wie das Array „_customVehPoints“ enthält (wenn möglich).

[z1, "S", _customCampPoints, 10] call **DAC_fReplaceZoneWaypoints**;
Ersetzt Wegpunkte in Zone z1, und zwar von der Kategorie „S“ (Soldiers).
Es werden 10 zufällige Wegpunkte aus dem Array „_customCampPoints“ ersetzt.

DAC_fReadAllCustomPoints

Diese Funktion ermöglicht das Einlesen aller selbst erstellten bzw. ersetzten Wegpunkte.
Sie ist außerdem für die Verwendung der Funktion DAC_fApplyPointsWithinZone entstanden.

Um dieses Feature nutzen zu können, muss DAC dafür bereits initialisiert sein.
Hier ist der Scriptaufruf:

```
_result = [base_S, base_V, base_T, base_B, base_A, base_C, delete, blank_limit] call
          DAC_fReadAllCustomPoints;
```

1. base_S (string) - Bezeichnung für Infanterie Wegpunkte
2. base_V (string) - Bezeichnung für Radfahrzeug Wegpunkte
3. base_T (string) - Bezeichnung für Kettenfahrzeug Wegpunkte
4. base_B (string) - Bezeichnung für Radfahrzeug und Kettenfahrzeug Wegpunkte (siehe **Hinweis**)
5. base_A (string) - Bezeichnung für Luftfahrzeug Wegpunkte
6. base_C (string) - Bezeichnung für Camp Wegpunkte
7. blank_limit (numeric) - Die Anzahl der zu ignorienrenden Marker "Lücken" pro Type (s. oben)
8. delete (boolean) - Ob ein Marker gelöscht wird oder nicht, nachdem er eingelesen wurde

Die Rückgabe der Funktion besteht aus einem Meta Array in dem die Locations angegeben sind.
(wichtig: diese Locations sind elementar wichtig für den DAC, und sollten von den Zonen Locations, von denen weiter oben unter DAC_fInsideOfZones die Rede war, getrennt werden. Sobald diese ihre Verwendung hatten, werden sie zurückgegeben.

Beispiele:

["ptS", "ptV", "ptT", "ptB", "ptA", "PtC", 1000, true] call **DAC_fReadAllCustomPoints**;
Liest alle Typen der selbst erstellten Wegpunkte ein und löscht alle Marker

["ptS", "ptV", "ptT", "", "", "", 1000, false] call **DAC_fReadAllCustomPoints**;
Liest nur die ersten 3 Typen der selbst erstellten Wegunkte ein, und löscht **keinen** Marker.

Hinweis:

Wie oben beschrieben, so haben diese beiden Typen spezielle Eigenschaften. Ihr werdet ggf feststellen, dass Kettenfahrzeuge so ziemlich überall dort hinfahren können, wo auch Radfahrzeuge hinfahren können. Das ist der Moment indem "beide" Wegpunkte ins Spiel kommen. Wenn ein "Both" Marker eingelesen wurde, wird er zu einem Radfahrzeug- oder einem Kettenfahrzeug-wegpunkt. Die Absicht ist es, ein sorgenfreies Plazieren von eigenen Wegpunkten von Land-fahrzeugen zu ermöglichen, während gleichzeitig die Übersichtlichkeit im Editor gewahrt bleibt.

Wenn eine Insel mit eigenen Wegpunkten erstellt wird, so kann diese später als Template abgespeichert werden, um sie mit einer späteren DAC Mission zu vermischen.

Ich verweise hier auf die in der Demo enthaltenen Missionen "**DAC_quick_waypoints**",
um dieses dynamische Wegpunktsystem mal in Aktion zu sehen.

DAC_fApplyPointsWithinZone

Mit dieser Funktion kann man die Massenersetzung von speziellen Wegpunktypen gleichzeitig auf der Ziel-Zone anwenden. Sie ist spezielle für die Verwendung mit DAC_fReadAllCustomPoints erstellt worden. Um dieses Feature nutzen zu können, muss DAC dafür allerdings bereits initialisiert sein.

WICHTIG:
Damit diese Funktion auch funktioniert, muss DAC_fReadAllCustomPoints zuerst gestartet sein

Hier ist der Scriptaufruf:

_result = [**zone**, **para_S, para_V, para_T, para_A, para_C**] call **DAC_fApplyPointsWithinZone**

1. zone
2. [usage, cap] Versorgung mit weiteren Infateriewegpunkten
3. [usage, cap] Versorgung mit weiteren Fahrzeug Wegpunkten
4. [usage, cap] Versorgung mit weiteren Panzer Wegpunkten
5. [usage, cap] Versorgung mit weiteren Lufteinheiten Wegpunkten
6. [usage, cap] Versorgung mit weiteren Camp Wegpunkten

usage (numeric, 0..1) = Die Anzahl der eigenen Wegpunkte, die in der Zone vorkommen
cap (numeric) = Die Maximale Anzahl der Zonen Wegpunkte, die ersetzt werden können

Returned _result besteht aus:
[application_report, applied_positions] where:
application _report (string) = Meldung der Operation
applied_positions (array) = Meta Array von allen ersetzten Wegpunkten

Beispiele:

_result = [z1, [0.4, 10], [0.4, 10], [0.4, 10], [0.4, 10], [0.4, 10]] call **DAC_fApplyPointsWithinZone**;
Würde alle Arten von Wegpunkten aus der Zone Z1 ersetzen, unter der Verwendung von 40% der eigenen Wegpunkte, die in Z1 entdeckt werden und bedeckt jede Art mit einem Maximum von 10000 Wegpunkten (genauer gesagt, es gibt kein Limit)

_result = [z1, [], [], [], [1, 10000], [1, 10000]] call **DAC_fApplyPointsWithinZone**;
Würde Luftfahrzeug und Camp Wegpunkte ersetzen, die in Z1 liegen, nutzt 100% der eigenen Wegpunkte aus den Luftfahrzeug und Camp bezogenen entdeckten Wegpunkten und bedeckt jede Art mit einem Maximum von 10000 Wegpunkten (genauer gesagt, es gibt kein Limit)

Anmerkung:

Schau Dir die in der Demo enthaltenen Missionen "DAC_quick_waypoints" an,
um eine dynamisches "echte Welt" Beispiel dieses eigenen Wegpunktesystems zu sehen.