4. Sound

Grundsätzliches

Das Thema Sounds in Arma ist etwas knifflig und muss absolut fehlerfrei bearbeitet werden.

Die häufigsten Fehlerquellen liegen an folgenden Stellen:

1. Alle Sounds müssen als .ogg vorliegen. Geht zum Beispiel per Audacity. Auf die Lautstärke der Datei in Relation zum übrigen Sound achten. Hier mit dem Bearbeitungsprogramm justieren, NICHT mit dem Lautstärkeparameter in der description.ext.

2. Alle Pfade müssen genau stimmen, wo sie verlangt werden. Dort, wo Classnames abgerufen werden, darf KEIN Pfad angegeben werden.

3. Die description.ext muss einwandfrei erstellt werden. Jedes Semikolon gesetzt, alle Klammern geschlossen?

4. ACHTUNG: *say3D, playSound* und etliche andere Befehle rund um Sounds sind Befehle mit LOKALEM Effekt. Sie müssen also ggf. per remoteExec ausgeführt werden, soll der Sound für alle Spieler hörbar sein. Hier ist der JIP-Parameter besonders zu beachten, denn nicht jeder Sound soll für nachjoinende Spieler nachträglich abgespielt werden!

Say3D mit Loop

Erzeugt einen kontinuierlichen 3d-Sound, ausgehend von einem Trigger oder einem Editor-Objekt. **Decription.ext** muss dafür eingerichtet sein! Exakte Schreibweise und Pfad beachten! Die Sounddatei wird immer wieder abgespielt (Loop), dabei die Länge der Datei beachten! Zeitpunkt der Wiederholung wird definiert mit "sleep" (in Sekunden). Referenz nur auf .wav (sehr groß!) oder besser auf .ogg. Lässt sich mit Audacity erstellen.

Funktioniert super, da die Init-Boxen bei jedem Spieler und bei jedem JIP initiiert werden!

a) An ein Objekt geheftet in die Init:

nul = [this] spawn {while {true} do {(_this select 0) say3D ["milchatter", Reichweite, 1]; sleep 196;};};

b) An einen Trigger geheftet: Bedingungen festlegen, dann On Act.:

nul = [thisTrigger] spawn {while {true} do {(_this select 0) say3D ["Sounddatei", Reichweite, 1]; sleep 45};};

Erläuterung: Reichweite in Metern, die "1" hinter Reichweite definiert den Pitch.

ACHTUNG: Ein Sound kann in Arma 3 nur beendet werden, indem man die Quelle des Sounds löscht!

Say3D ohne Loop

Verweist auf description.ext! **Exakte** Schreibweise und Pfad beachten! Die Datei wird nur einmal abgespielt.

1. Ohne remoteExec (funktioniert, wenn alle beim Serverstart dabei sind):

a) Als einfache Objekt-Init:

```
this say3D ["Sounddatei", Reichweite, 1];
```

b) mit addAction (siehe auch Einträge zu addAction):

```
Objektname addAction [ "MUSIC ON", {Objektname this say3D ["Sounddatei", Reichweite, Pitch];}];
```

2. Mit remoteExec:

```
[smartphone, ["song", 15, 1, false]] remoteExec ["say3D", 0, true];
```

Die **Syntax** ist:

```
[Objekt, ["Datei", Reichweite, Pitch, Radioeffekt]] remoteExec ["say3D", targets, JIP];
```

Erläuterung zu targets: (0=global, -2=alle außer Server), JIP (true/false)

ACHTUNG: Ein Sound kann in Arma 3 nur beendet werden, indem man die Quelle des Sounds löscht!

Will man also einen Sound beenden, ohne das Objekt zu zerstören, muss man es löschen und an exakt der gleichen Position wieder erstellen - autoradios sind so nur möglich, wenn man an die Fahrzeuge ein Objekt anhängt (attach) und es als Emitter nutzt. Habe ich aber noch die gemacht.

Im Beispiel unten spielt das "Smartphone" den Song per Aktionsmenü ab, sofort werden alle bisherigen Aktionsmenü-Einträge gelöscht ("removeAllActions"), damit der Sound durch die Clients kein weiteres Mal abgespielt werden kann (keine Überlagerungen!).

Der neu erzeugte addAction-Eintrag "Radio aus" löst bei Aktivierung global (remoteExec) folgendes aus: Er findet die genaue Position (getPosATL) des Smartphones und dessen Ausrichtung (getDir),

löscht das Objekt und spawnt es genau dort wieder (ohne die Aktionsmenü-Einträge!).

Das sieht dann so aus:

```
    [smartphone, ["sameintheend", 15, 1, false]] remoteExec ["say3D", 0, true];
    smartphone remoteExec ["removeAllActions"];

[smartphone, ["Radio aus",
        {
            _pos = getPosATL (_this select 0);
            _dir = getDir (_this select 0);
            deleteVehicle (_this select 0);

            smartphone = "Land_MobilePhone_smart_F" createVehicle [0, 0, 0];
            smartphone setPosATL [_pos select 0, _pos select 1, _pos select 2];
            smartphone setDir _dir;

        }]] remoteExec ["addAction"];
```

Man könnte nun mit diesem laufenden Script die alten Einträge wieder hinzufügen und folgendes einfach anhängen. Fertig wäre das fertige Radio zum Ein- und Ausschalten:

```
    [smartphone, ["Sublime", {execVM "scripts\playRadio1.sqf"}, [], 6, false, true, "", "", 2]] remoteExec ["addAction"];
    [smartphone, ["Dylan", {execVM "scripts\playRadio2.sqf"}, [], 5, false, true, "", "", 2]] remoteExec ["addAction"];
    [smartphone, ["Ataris", {execVM "scripts\playRadio3.sqf"}, [], 5, false, true, "", "", 2]] remoteExec ["addAction"];
    [smartphone, ["CCR", {execVM "scripts\playRadio4.sqf"}, [], 5, false, true, "", "", 2]] remoteExec ["addAction"];
```

Description.ext für CfgSounds

Muss im Hauptverzeichnis der Mission liegen. Die Datei defiiert die Sounds, damit Scripts darauf zugreifen können. Exakte Syntax und Pfad beachten!! Siehe auch den Eintrag "Wichtige Dateitypen beim Missionsbau" hier im Wiki.

https://community.bistudio.com/wiki/Description.ext

Hier sind beispielhaft drei Sounds aus einer früheren Mission definiert.

```
class CfgSounds

{
sounds[] = {};
class bridgetalk
{
name = "bridgetalk";
sound[] = {"\sounds\bridgetalk.ogg", +2, 1, 5}; //
titles[] = {};
};
class captain
{
name = "captain";
sound[] = {"\sounds\captain.ogg", 1, 1, 15}; //
titles[] = {};
};
class alarm1
{
name = "alarm1";
sound[] = {"\sounds\alarm1.ogg", 0.6, 1, 2}; //
titles[] = {};
};
};
```

Syntax ist: Dateiname.endung, Lautstärke, Pitch, Reichweite (Reichweite scheint aber hier nicht zu funktionieren. besser in Obj.-Init! einstellen)

Beispiel: Getriggerter Alarm

Soll bei der Zerstörung eines ganz bestimmten Objekts (oder mehrerer) ein Dauer-Alarm ausgelöst werden, muss dies per Trigger geschehen. Die Bedingung ist, dass das Objekt nicht mehr "am Leben" ist (!alive "varName";). Es bietet sich an, mit Say3D zu arbeiten, da der Default-Alarm nur in 2D abgespielt wird.

Trigger OnAct.:

```
nul = [thisTrigger] spawn {while {true} do {(_this select 0) say3D ["Sounddatei", Reichweite, 1]; sleep 45};};
```

Der Trigger fungiert dann als 3D-Emitter des Sounds, es kann aber auch ein Lautsprecher daneben gestellt werden. Möglich ist auch eine Ausführung per Script. Abhängig von Description.ext!!!