

Titel: FX Grundlagen und Einbau

Schwierigkeit: Erweiterte Grundkenntnisse

Typ: FX-Coding

Programm: Final BIG

Alien aka Infiltrator

Willkommen zum (hilfreichen?) Tutorial

Dieses Tutorial dient als Grundlegender Einstieg, ich werde nacheinander die verschiedenen Verwendungsbereiche der FX anschneiden, allerdings nur deren Einbau, bzw. wie man sie in verschiedenen Situationen auslöst.

Für die Erstellung von FX lest bitte meine beiden anderen Tutorials.

Inhaltsverzeichnis

Einführung und Inhaltsverzeichnis	Seite 1
Einbindung einer FX	Seite 2 - 4
Die FXList und ihre Parameter	Seite 5 - 11
- ParticleSystem-Block	Seite 5 - 6
- DynamicDecal-Block	Seite 7
- ViewShake-Block	Seite 8
- FXListAtBonePos-Block	Seite 8
- AttachedModel-Block	Seite 8
- CameraShakerVolume-Block	Seite 9
- Der ParticleSysBone	Seite 9
- EvaEvent-Block	Seite 9
- Sound-Block	Seite 10
- Buff-Block	Seite 10
- TerrainScorch-Block	Seite 11
Schlusswort und Hinweise	Seite 11

Zuerst etwas Theorie, und zwar wo und wie man eine FXList (oder auch eine FXParticlesystem) angibt. Danach behandeln wir den genauen Aufbau und die Möglichkeiten für die FXList selbst.

Wo kann man eine FXList überall angeben?

Eine FXList findet meistens in Weapons und in Objects Anwendung.

Außerdem können Sie aus einer FXParticlesystem heraus aufgerufen werden.

Ob eine FXList oder ein FXParticlesystem von einer Zeile aufgerufen wird erkennt man an der Benennungsregel, die EA Games scheinbar verwendet hat, denn nahezu alle Namen von FXLists beginnen mit „FX_“ während Particlesystems diese Präfix nicht im Namen tragen.

Hier kann man FX einbinden:

- FireFX in Weapons
- An eine Animation gecoded
- In Buffs/Attributemodifier
- Die UpgradeFX in Upgrades
- LevelUpFx der ExperienceLevels Ini
- In einem ModelConditionState
- Als Death Event

FX während Attacken

In Weapons gibt es die Zeile FireFX, diese Zeile gibt an welche FX beim Abfeuern der Weapon gezeigt wird. Gemeint ist eine FXList kein FXParticleSystem.

FireFX = FX_ArvelegBlast

Die PreAttackFX-Zeile die man in manchen Weapons findet ist nicht funktionsfähig. Will man eine FX vor einer Weapon ausführen, so muss man dies über die Animation in der Einheiten INI regeln.

An Animationen „geheftete“ FX

Dies ist eine FX die an einem bestimmten Zeitpunkt in einer Animation ausgelöst wird.

```

;;; WORD OF POWER ANIMS MOUNTED ;;;
AnimationState = SPECIAL_WEAPON_ONE MOUNTED
    StateName = Attacking
    Animation
        AnimationName = GUGdffHrs_SKL.GUGdffHrs_SPLA
        AnimationMode = ONCE
        AnimationBlendTime = 4
    End
    FXEvent = Frame:5 Name:FX_GandalfPreAttackBlast
End

```

Sooooo dies ist ein Beispiel aus der Gandalf.ini, zu sehen ist ohne jeglichen Zweifel die Animation die während dem Wort der Macht abgespielt wird.

Von Belang ist für uns hier die FXEvent-Zeile...

Frame:*Zahl*

Gibt dem Spiel die Information, wann genau die FX innerhalb der Animation ausgelöst wird. Dazu muss man wissen, das in einer Sekunde 30 Frames abgespielt werden.

Name:*FXList Name*

Hier fügen wir den Namen der FXList ein...

Eigentlich eine sehr simple Einbindung.... eigentlich ist das angeben eine FX nie schwer oder aufwändig^^

Attributemodifier / Buffs

Unsere Modifier und Buffs sind wohl die FX die wir am häufigsten sehen, sei es ein gelbes Leuchten oder sonstiger Glanz, die Einbindung des Fxs ist wieder mal ein Einzeiler.

```
FX = FX_GenericBuff
```

Diese simple Angabe bindet eine FXList ein...

Das wars auch schon^^.

FX in Upgrades

Die sogenannte Upgrade FX wird ebenfalls so eingebunden:

```
UpgradeFX = FX_PorterDeliverSiegeHammer
```

Es handelt sich auch hier um eine FXList. Diese wird dann an jedem Object ausgelöst das diese Upgrade erhält. Findet z.B. Anwendung wenn einer Einheit Forged Blades gekauft wird, dann leuchtet diese kurz auf. Für Upgrades die das aufleveln betreffen sollte man dies nicht tun, dafür gibt es speziell in der Expieriencelevels.ini einen Befehl (dazu braucht es natürlich ein Expieriencelevel-Eintrag für das Object).

Experiencelevels und FX

In einem ExperienceLevel-Block gibt man einen FX so an:

```
LevelUpFx    = FX:ElfBarracksUpgrade
```

Dabei ist allerdings sehr wichtig, das der Name der FX nur der Teil NACH „FX:“ ist, das „FX:“ muss als Anweisung davor gesetzt werden.

Außerdem handelt es sich hier NICHT um eine FXList sondern um ein FXParticleSystem.

FX in Objects als Teil des ModelConditionState

Es ist möglich eine FXParticleSystem in einer ModelConditionState permanent zu verankern, somit lassen sich Auren oder ähnliche permanente Effekte leicht verwirklichen. Hier ein Beispiel aus Galadriels Ini:

```
DefaultModelConditionState
    Model          = EUGaldriGd_SKN
    ParticleSysBone = B_PELVIS GaladAura FollowBone:YES
    ParticleSysBone = B_PELVIS GaladAura02 FollowBone:YES
End
```

Die ParticleSysBone-Zeile bindet ein FXParticleSystem an einen Bone der Einheit. Der Bone ist in diesem Fall „B_PELVIS“, danach folgt der FX-Name und dann der Standard-Syntax „FollowBone:YES“.

FX in einem SlowDeathBehavior

Zu guter Letzt gibt es noch die Möglichkeit eine FXList an den Tod einer temporären Einheit zu heften.

```
Behavior = SlowDeathBehavior ModuleTag_05
    DeathTypes = ALL -FADED
    DestructionDelay = 2500
    FX = FINAL FX_BalrogFinalDeath
    DeathFlags = DEATH_1
End
```

Soooo, ja...

FX = „Zeitpunkt“ „FXList-Name“

Für Zeitpunkt steht zur Auswahl: INITIAL und FINAL

Wobei ersteres den Anfang und Final das Ende des „Sterbens“ beschreibt.

Die FXList

Die sog. FX List besteht aus einem oder mehreren verschiedenen Blöcken, wird eine FXList aufgerufen, werden alle Aktionen die in den Blöcken angegeben werden ausgelöst. Natürlich an der Stelle an der die FXList ausgelöst wird (z.B. von einer Einheit im Nahkampf).

Ich werde jetzt nacheinander die einzelnen Blöcke aufzählen und erklären, beginnen wir mit meinem Favoriten, dem ParticleSystem-Block.

Hier ein Code-Beispiel:

```
ParticleSystem
  Name = MenFortressGround
  Offset = X:0.0 Y:0.0 Z:2.0
End
```

Ein denkbar einfaches Beispiel, wie es am öfteren Verwendung findet.

„ParticleSystem“ eröffnet unseren Block.

„Name“ gibt den Namen der FX an, die aus der FXParticleSystem-Ini benutzt wird.

Außerdem kann der Name Zeile ein „FollowBone:yes“ angehängt werden, dazu später mehr.

„Offset“ (X Y Z) gibt an, wie die FX räumlich versetzt erzeugt wird, wird die Zeile weggelassen, gelten die Werte X:0 Y:0 Z:0. Das Verhalten des ParticleSystems wird dabei nicht beeinflusst.

Es gibt allerdings noch andere Parameter für den ParticleSystem-Block:

Count = 1

Hier wird angegeben wie oft dieses ParticleSystem an dieser Stelle erzeugt werden soll, sehr sinnvoll ist dieser Parameter allerdings nicht, da ein und die selbe FX gleichzeitig an einem Ort nur dann Sinn ergibt wenn die FX große Zufallswerte hat...

CreateAtGroundHeight = Yes

Die FX wird auf Bodenhöhe gespawnt...

InitialDelay = 1000 1000 UNIFORM

Hier wird definiert mit welcher Zeitverzögerung das ParticleSystem ausgelöst werden soll, bei einer FXList mit mehreren verschiedenen FX durchaus eine nützliche Funktion, da diverse Effekte zeitlich versetzt voneinander ausgelöst werden können, ohne diese Verzögerung in das System schreiben zu müssen. (Es gibt ebenfalls eine InitialDelay Zeile innerhalb des ParticleSystems)

Die Zeitangabe erfolgt in Millisekunden also im oberen Beispiel eine Sekunde. Das UNIFORM gibt die Zeiteinheit an, und sollte so belassen werden.

ObjectFilter = NONE +STRUCTURE

Je nachdem welchen Typ das Ziel der FX hat, wird durch den Filter entschieden ob er gezeigt wird oder nicht. In diesem Beispiel wird der FX nur gezeigt wenn Gebäude angegriffen werden.

OnlyIfOnLand = Yes

OnlyIfOnWater = Yes

Für beide Zeilen gilt beim weggelassen werden automatisch „No“, die Bedeutung sollte klar sein.

OrientToObject = Yes

Bei weglassen gilt „No“, dabei wird der FX so gespawnt, das seine Ausrichtung die des FX auslösenden Element entspricht. FX sind normalerweise immer gleich, egal ob die Einheit nach Norden oder Süden blickt. Mit diesem Parameter dreht sich die FX immer mit, dies ist vor allem für unsymmetrische FX sehr wichtig. (beim Word der macht wird kein Unterschied festzustellen sein...)

Ricochet = Yes

„No“ bei weglassen. Ricochet bedeutet „Querschläger“ oder „abprallen“, ich habe diesen Parameter nie verwendet und kann daher keine Aussage machen, vermute aber das er entweder einen gewissen Zufallsfaktor in die FX bringt, oder ihre Particle von anderen Modellen/Elementen abprallen lässt.

SetTargetMatrix = Yes

Selbiges wie bei Ricochet, dieser Parameter findet nur sehr selten Verwendung und zwar nur bei Lichtstrahlen-Effekten. Ich vermute das das Typische Verteil-Verhalten der LightRays von diesem Parameter abhängt.

SystemLifetime = 60

Dieser Parameter wird von EA Games nur ein einziges mal verwendet, ist allerdings recht Praktische wenn man eine FX für verschiedene Zwecke benutzen möchte. Dieser Parameter überschreibt nämlich die SystemLifetime die in dem ParticleSystem angegeben ist.

Kombiniert man eine FX mit viele Zufallszahlen mit Count und dieser Zeile, lassen sich für verschiedene LVL einer Attacke einfach FX erstellen, da man dann einfach die Lifetime und den Count in der List höher setzt, statt für jede LVL der Attacke eine neue FX zu schreiben.

CreateBoneOverride = BAT_HHEAD

und

TargetBoneOverride = B_SWORDBONE

Um diese Parameter zu verwenden zu können müsst ihr bei Name ein FollowBone: Yes hinzufügen, hier ein Beispiel:

Name = LightningStrike FollowBone:Yes

OrientToObject = Yes

CreateBoneOverride = B_SWORDBONE

TargetBoneOverride = B_SWORDBONE

Damit wird die FX zwischen dem angegebenen Bone am Weapon-Nutzer (FX-Auslöser) und dem genannten Bone des Weapon-Ziels „gespannt“. Das ParticleSystem sollte folgende Zeile beinhalten damit diese Parameter Funktionieren:

Type = STREAK

Praktische Anwendung hat das bisher bei dem LightningStrike von Gandalf gefunden.
(Blitzschwert)

UseTargetOffset = Yes

und

TargetOffset = X:0.0 Y:0.0 Z:300

TargetOffset kann nur verwendet werden wenn UseTargetOffset = Yes angegeben wird. Beide Parameter werden nur in Verbindung mit den beiden BoneOverride Parametern verwendet.

Weather = SNOWY

Für diesen Parameter gibt es SNOWY und NORMAL, solltet ihr FX machen die Wasser darstellen ist es ein besonders netter Gedanke auf Schneemaps entsprechende gefrorene FX zu machen, bzw Schnee... Dies lässt sich mit dem Weather Parameter verwirklichen da er die FX nur dann zeigt wenn das genannte Wetter zutrifft...

Der Allseits beliebte DynamicDecal-Block

Dieser Block hats in sich, er bietet die Möglichkeit die Textur des Bodens mit einer Bilddatei zu überlagern. Zu einer gelungenen FX gehört meiner Meinung nach (z.B. bei größeren Attacken oder Spawns) immer auch eine nachhaltige Veränderung der Spielumgebung, dafür bietet sich das verbrennen von Erde oder eben der DynamicDecal an.

DecalName = EXSnowBlastPrint

Der Name der Textur/Bilddatei die gezeigt wird.

Color = R:10 G:20 B:20

Die Textur wird zusätzlich eingefärbt (0 0 0 wäre original Farbton)

Lifetime = 22000

Die Gesamtzeit in der das Decal sichtbar ist, eigentlich ein überflüssiger Parameter aber leider ein Notwendiges übel, es gibt eine einfache Regel für diesen parameter:

StartingDelay + OpacityFadeTimeOne + OpacityPeakTime + OpacityFadeTimeTwo = Lifetime

Offset = X:0.0 Y:-20.0 Z:0.0

Das Offset regelt wo (von der Source) aus das Decal erzeugt wird.

Size = 325

Gibt an wie groß die Textur auf dem Boden erstellt wird
(im Beispiel wäre die Textur also 325x325 groß)

Shader = ADDITIVE

Der Decal wird anders auf den Boden gezeichnet als es normalerweise der Fall ist, das Verhalten gleicht dem, das wir aus den FXParticleSystem bei den Shadern kennen.

StartingDelay = 1000

Diese Zeit (wie die anderen Zeitangaben ebenfalls in Millisekunden) verstreicht bevor der Decal-Effekt eintritt.

OpacityStart = 0

Die Sichtbarkeit zu Beginn des Effekts.

OpacityFadeTimeOne = 2000

Die Zeit die das Decal benötigt um von OpacityStart zu OpacityPeak über zu blenden.

OpacityPeak = 100

Die Sichtbarkeit des Decals in der Mitte, diese ist die wichtigste, da man für diese als einzige eine Zeit angeben kann in der sich die Sichtbarkeit nicht ändert.

OpacityPeakTime = 30000

Die Zeit in der der Wert von OpacityPeak beibehalten wird.

OpacityFadeTimeTwo = 30000

Die Zeit die das Decal benötigt um von OpacityPeak zu OpacityEnd über zu blenden. (hier empfiehlt sich in den meisten Fällen eine sehr große Zeit, denn dann wird der „Abgang“ schön weich und fällt dem Betrachter meist gar nicht auf)

OpacityEnd = 0

Die Sichtbarkeit des Decals zum Schluss.

Der ViewShake-Block

Dieser Block ist für das Wackeln der „Kamera“ also der Ansicht des Spielers zuständig. Er hat nur zwei Parameter, wobei eigentlich nur einer davon Anwendung findet.

Type = SUBTLE

Der Type-Parameter gibt die Stärke bzw. Art des „Wackelns“ an, möglich sind hierbei:

SUBTLE //Schwächster

NORMAL

STRONG

SEVERE

CINE_EXTREME

CINE_INSANE //Stärkster

Bei Attacken die bei steigendem LVL Stärker werden den Type zu wechseln ist ein recht nettes Detail das man in seine FX einbauen kann.

ObjectFilter = NONE +STRUCTURE

Die Funktionsweise dieses Parameters ist selbige wie im ParticleSystem Block, er dient dazu, den Effekt vom Typ des Target-Objects abhängig zu machen.

Der FXListAtBonePos-Block

Der Name lässt es vermuten: Dieser Block spawnt eine FXList an einem Bone, hier kurz die Erklärung:

FXListAtBonePos

BoneName = B_HEAD

FX = FX_FurnaceSteam

Weather = SNOWY

End

Eigentlich sind die Begriffe selbsterklärend, aber hier nochmal zur Sicherheit:

Bei BoneName gebt ihr den Bone an, an dem die FXList gespawnt werden soll.

FX gibt die FXList an die gespawnt/erzeugt wird.

Weather erlaubt erneut eine Wetterabhängigkeit zu erzeugen.

Der AttachedModel-Block

Dieser Block heftet ein Model an den FX-Owner.

Modelname = g_arrow

Dies ist der einzige Parameter für diesen Block, er gibt einfach nur den Modelnamen des anzuheftenden Models an.

EA Games verwendet diesen Block nur für g_arrow, also wenn ein Pfeil in einer Einheit stecken bleibt, theoretisch könnte man aber auch andere schöne Dinge damit verwirklichen....

Der CameraShakerVolume-Block

Dies ist die komplizierte Version des ViewShake Blocks und erlaubt das schütteln der Kamera besser zu kontrollieren.

Ich verweise darauf das er eigentlich unnötig ist und ich mich nicht mit ihm befasst habe, allerdings hier mal eine vermutetet Funktionsweise:

CameraShakerVolume

Radius = 500

Duration_Seconds = 1.5

Amplitude_Degrees = 0.5

End

Radius

Stellt euch vor die Kamera wäre in einem Kreis eingeschlossen und würde darin hin und herspringen. Der Radius begrenzt die Größe des Wackelns im gesamten.

Duration_Seconds

Die Zeit die der Effekt andauert (ich würde behaupten, eigentlich der einzige Grund mich dazu zu kriegen das zu verwenden, denn eine Zeit kann ich beim ViewShake nicht angeben...)

Diese Zeitangabe erfolgt NICHT in Millisekunden sondern in Sekunden (z.B: 2.5)

Amplitude_Degrees

Die Größe der einzelnen „Sprünge“ innerhalb des Kreises...

EvaEvent-Block

Hiermit lässt sich ein Eva-Event auslösen, benutzt diesen Effekt mit Bedacht, es macht z.B. nicht wirklich Sinn jedes mal eine Meldung Auszulösen wenn Gandalf mit dem Stab zuschlägt, EvaEvents sollte man nur für wirklich wichtige Dinge verwenden.

EvaEventAlly = None

Der Name des EvaEvents (aus der EvaEvent.ini) das Verbündeten angezeigt wird...

EvaEventOwner = WormtongueCreated

Der Name des EvaEvents das dem Besitzer der FXSource angezeigt wird.

RequiredSourceModelConditions = ONE_RING

Die ModelCondition die das auslösende Object haben muss um dieses Event auszulösen, sollt diese Bedingung nicht erfüllt werden wird das EvaEvent einfach übersprungen. Wie man sieht findet dies von EA Games nur Verwendung um das verlieren des Ringes anzukündigen.

Eine denkbare Anwendung wäre auch z.B. bei zusammengehörigen Einheiten wie bei den Elben-Zwillingen, ein andere Todesnachricht zu geben wenn einer der Beiden stirbt und der Andere lebt, als wenn einer ohne die Anwesenheit seines Bruders stirbt. (Einfach über AuraEffekt + ModelconditionState zu regeln)

Der ParticleSysBone

Diesen Parameter kennen wir ja bereits aus dem Angeben in einem ModelConditionState, ebenso können wir diesen in die FXList schreiben, und zwar ohne Block, soll heißen einfach ohne Eröffnung und abschließendes „End“, sondern so wie wir es von dem einfügen in einen ModelConditionState gewohnt sind ;-)

Der Sound-Block

Spielt einen Sound aus der Sound.ini

ExcludedSourceModelConditions = DISGUISED

Spielt den Sound nicht wenn genannter Status zutrifft.

RequiredSourceModelConditions = CREATE_A_HERO_00

Spielt den Sound nur wenn die Bedingung erfüllt ist.

Name = AragornVoiceRespawn

Der Name des Sounds in der Sound.ini

ObjectFilter = ANY +ORC +HORDE

Objecte an denen der Sound gespielt werden kann.

SourceObjectFilter = NONE +DwarvenAxeThrower +DwarvenBanner

Nur von diesen Objecten kann der Sound ausgelöst werden.

StopIfNuggetPlayed = Yes

Der Sound stoppt wenn die Aktion vorzeitig beendet wird.

Verhindert das von einer Figur zwei Sounds gleichzeitig gespielt werden...

Für die drei zuletzt genannten Parameter empfehle ich FX_TrollGrabInitiate als Beispiel.

Dort wird ein Unterschiedlicher Sound gespielt wenn er einen Ork oder einen Baum aufnehmen soll, welcher dann abgebrochen wird sollte er noch anhalten wenn das Ziel aufgehoben wird.

Der BuffNugget-Block

BuffCavalryTemplate = DominateBuff

BuffInfantryTemplate

BuffMonsterTemplate

BuffOrcTemplate

BuffShipTemplate

BuffThingTemplate

BuffTrollTemplate

Diese Zeilen geben an welche Art von Objects den genannten Buff erhält, eine sehr unglückliche Art die Buffs zu integrieren wenn ihr mich fragt...

BuffLifeTime = 100

Die Dauer des Buffs, angegeben in Frames (Angegebene Zahl /30 = Sekunden)

BuffType

Es gibt folgende Typen:

Buff

Cursed

Debuff

Dominate

GloriousCharge //laut Kommentar von EA Games, nicht von mir getestet...

Healing

LeaderShip

Poison

IsComplexBuff = No

Um ehrlich zu sein weiß ich nicht wirklich etwas damit anzufangen... wenn nicht angegeben „Yes“.

TerrainScorch-Block

Dieser Block erzeugt verbrennt die Erde im Zielgebiet.

Type = RANDOM

Von seitens EA Games wird nur RANDOM verwendet, es gibt allerdings in den Kommentaren folgende Angabe:

SCORCH_1, SCORCH_2, SCORCH_3, SCORCH_4, SHADOW_SCORCH, RANDOM

Ist allerdings ungetestet...

Weather = SNOWY

Altbekannt und keine erneute Erklärung wert^^

RandomRange = X:4 Y:6

Damit könnt ihr den Random-Wert eingrenzen und die Stärke des Verbrennens eingrenzen.

Radius = 30

Die Größe des Gebiets das verbrannt wird.

Ende (auch Önde genannt)

Es gibt noch folgende Blöcke, deren Funktion ich euch nicht sagen kann, bzw. getestet habe und keinerlei Änderung aufgetreten ist:

LightPulse

Wird verwendet, allerdings kann ich keine Funktion erkennen.

RayEffect

Tracer

FXParticleSysBoneNugget

Laser

Diese 4 sind in den Comments genannt finden jedoch keine Verwendung und ich habe sie bisher nicht getestet, lasst es mich wissen wenn ihr sie erfolgreich eingesetzt habt, danke.

Solltet ihr gewisse Punkte anders sehen wie ich, oder es einfach besser wissen, meldet euch bei mir und ich werde das ganze dementsprechend updaten, überprüfen und verbessern ;-)

Joar, das war jetzt sehr Basic, dient ja auch nur als Trittbrett für die beiden anderen Tutorials.

Greez

Alien aka Infiltrator

(ACHTUNG: ES BESTEHT KEINERLEI GARANTIE AUF OBRIGEN INHALT, DER VERFASSER STAND ZUM ZEITPUNKT DES VERFASSEN UNTER PERMANENTER HARD-ROCK-BESCHALLUNG)