Partikel Tutorial

Geschrieben für A6
V.1

Inhalt

Vorwort

Hallo! Willkommen bei meinem 2 Tutorial! Es soll ihnen helfen tolle Partikeleffekte zu programmieren. Dieses Tutorial basiert auf dem Partikelsystem der A6 Engine, ich habe die Effekte nicht auf der A5 getestet, aber ich glaube, dass sie alle funktionieren werden.

Einführung

Was sind eigentlich Partikel? Partikel sind kleine Grafiken, die in einem Spiel zu Tausenden vorkommen, man benutzt sie für viele Effekte wie z.B: Feuer, Rauch, Explosionen, etc. Man kann sehr leicht Partikeleffekte programmieren, doch ein sehr guter Effekt braucht viel Wissen von mathematischen Funktionen, aber keine Angst, am Ende dieses Tutorial werden Sie auch coole Effekte programmieren können. Ein Partikel kann so aussehen:



Partikel kann man mit jedem beliebigen Zeichenprogramm erstellt werden, ich benutze dazu das Freeware Programm the Gimp, dies ist sehr leistungsfähig. Ein Partikel muss als grösse eine 2er Potenz besitzen, z.B: 32x32. Alles was bei einem Partikel schwarz ist, kann mit einem bestimmten C-Script Befehl transparent gemacht werden, aber dazu später mehr.

Aber nun legen wir los! Zuerst sollten Sie ein kleines Level erstellen in dem wir dann die Effekte testen werden, ich benutze dafür ein Haus. Ich schlage Ihnen vor, dass Sie auch ein Haus bauen, da wir z.B: Eine Fackel machen werden.

Mein „Haus“

Erster Partikeleffekt eine Fackel

Nun wollen Sie sicher Ihren ersten Partikeleffekt programmieren. Ich habe mich dazu entschlossen, dass wir zuerst einen Partikeleffekt für eine Fackel programmiere. Ich glaube Sie wissen was eine Fackel ist und ich glaube, dass eine gut gesetzte Fackel viel zu einer guten Atmosphäre beiträgt. Wie soll den der Partikeleffekt überhaupt aussehen? Nun, die Partikel sollen aufsteigen und sie sollten ein wenig Licht erzeugen. Als Partikel verwenden wir dieses:

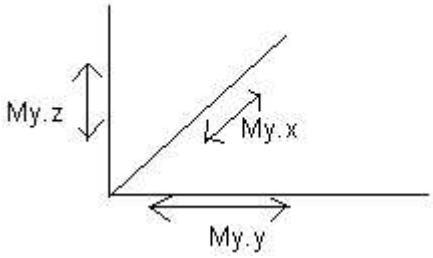
Ich finde, dieses lässt sich sehr gut für einen Feuereffekt benutzen. Nun beginnen wir mit dem Code, erstellen Sie eine neue WDL-Datei.

```
bmap fire_p = <fire.bmp>;//Die Grafik für das Bild

Funktion ausblenden()
{
	my.alpha -= 7* time; //Verringere Alpha

	if(my.alpha <= 0) //ist Alpha kleine gleich 0
	{
	 my.lifespan= 0; //Vernichte Partikel
	}
}

function fackel_feuer()
{

	  my.vel_x = (random(1)-0.5);//Geschwindigkeit der Bewegung an der X-Achse
	  my.vel_y = (random(1)-0.5);//Geschwindigkeit der Bewegung an der Y-Achse
	  my.vel_z = random(10);//Geschwindigkeit der Bewegung an der Z-Achse

	  my.flare = on;//dunkle Stellen sind transparenter
	  my.bright = on;//Partikel werden beleuchtet
	  my.bmap = fire_p;//Grafik des Effekts
	  my.size = 7;//Grösse des Partikel
	  my.move = on;//Partikel darf sich bewegen

	  my.function = ausblenden;//Funktion die ein Partikel hat
}
```

Erklärung

Nun, dass ist wohl für Sie ein ziemlich grosser Broken, aber keine Angst, ich erkläre den Code jetzt Zeile für Zeile.

Bmap ist ein Variabeltyp für Grafiken (Bilder). Die Syntax sieht so aus:
bmap name = <bild-name.bildtype>; Dies sollte wohl selbst erklärend sein, mit dem Namen, kann man nun auf das Bild zugreifen. Was macht die mysteriöse Funktion namens Ausblenden? Diese ist dafür da, dass ein Partikel nicht für immer existiert, dass Partikel wird langsam ausgeblendet und dann vernichtet. Ich glaube, Sie sollten diese Funktion anhand der grünen Kommentare verstehen. Der Alpha-Wert eines Partikels ist am Anfang 100 und wird dann schrittweise herunter gezählt. Die Funktion namens fackel_feuer ist der eigentliche Effekt, von hier aus werden die Parameter der Partikel gesteuert, darum erkläre ich diese jetzt genauer. Vel_x, Vel_y und Vel_z geben die Geschwindigkeit in die bezüglichen Vektoren an. Diese Grafik zeigt Ihnen in welche Richtung die verschiedenen Richtungsvektoren sich orientieren.



Random(x) erstellt eine Zufallszahl zwischen der angegeben Zahl, hier als x dargestellt (Random(1)-0.5) erstellt eine Zufall zwischen 0.5 und -0.5. Flare bewirkt, dass dunkle Teile eines Partikels transparent sind, dadurch lassen sich auch runde, viereckige, ovale, etc. Partikel erstellen. Bright beleuchtet die Partikel, es sieht dann so aus als würden sie glühen. My.Bmap = name; diese Funktion, weisst dem Effekt eine bestimmte Grafik zu, diese dann verwendet wird, man sieht, es gibt verschiedene Funktionen von bmap. Size definiert die grösse des Partikels, meist sehen kleine Partikel besser aus als grosse, darum ist ein Wert von 15 meisten angebracht und liefert gute Ergebnisse. Move erlaubt dem Partikel sich zu bewegen oder auch nicht, meistens sollte dieser Wert auf on gestellt sein. My.Function dieser Befehl weisst einem Partikel eine Funktion zu, z.B: Das Ausblenden, man kann den Wert aber auch auf Null stellen, dann führt das Partikel keine Funktion aus. Nun erstellen wir die Aktion für die Fackel:

```
action fackel
{
	my.light = on;//Die Fackel besitzt eine eigene Lichtquelle
	my.green = 0;//Grünwert des Lichts
	my.blue = 100;//Blauwerts des Lichts
	my.red = 100;//Rotwert des Lichts/
	my.lightrange = 300;//Reichweite des Lichts in Quants
	my.visible = off;//Mache den Entity unsichtbar

	while(1)//Solange wiederholen wie das Spiel läuft
	{
		effect(fackel_feuer, 10*time, my.x, normal);//Partikeleffekt erstellen
		wait(1);//Einen Framezyklus warten
	}
}
```

Erklärung

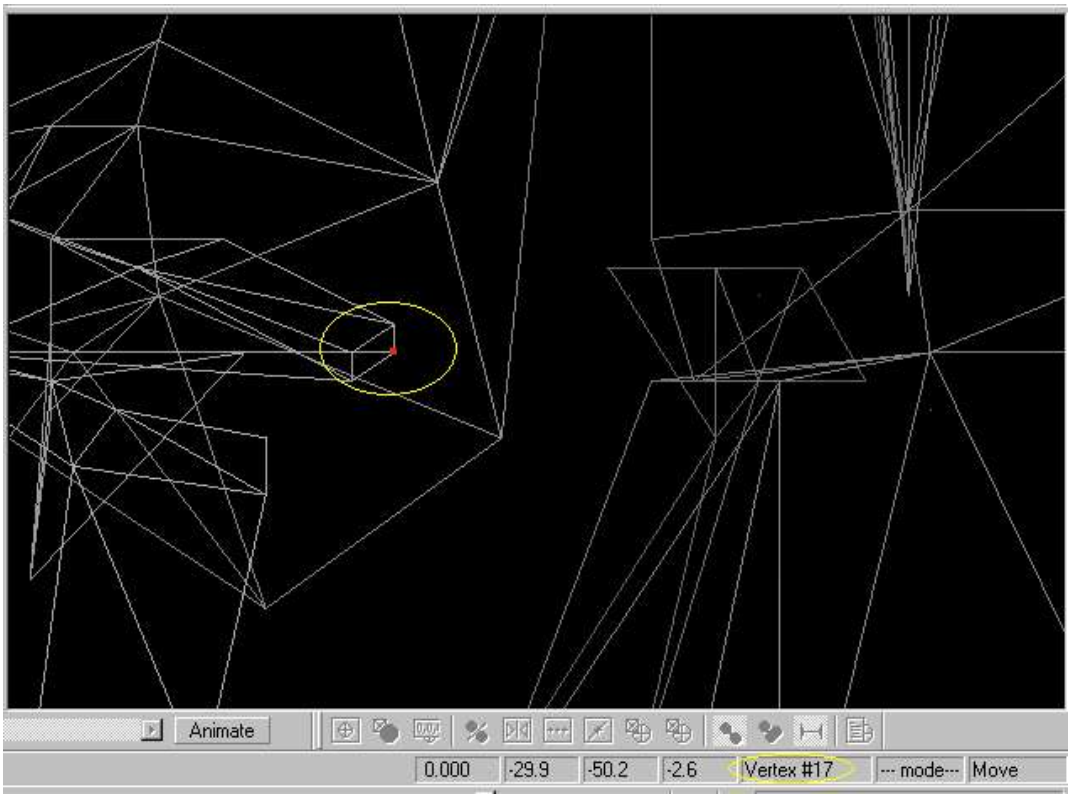
my.light weisst dem Entity eine eigene Lichtquelle zu. my.green, my.blue und my.red sind RGB-Werte deren Wertbereich bis und mit 255 beträgt, sie geben dem Licht die Farbe. My.lightrange bestimmt die Reichweite des Lichtes, alles was ausserhalb liegt, ist total schwarz. Nun zu dem mysteriösem Befehl Effect. Wie Sie wohl vermuten, ist dieser Befehl für die Erstellung der Partikel verantwortlich. Die Syntax lautet wie folgend: effect(funktion, anzahl, vektor, geschwindikeitsvektor); Funktion ist die Funktion die die Partikel steuert, Anzahl bestimmt wie viele Partikel erstellt werden (Achtung! Bei zu vielen Partikeln bricht die Framerate ein!), vektor gibt die Position an, ein Vektor ist nichts anderes als die Werte X, Y und Z, der Geschwindigkeitsvektor bestimmt die Geschwindigkeit der Partikel am Anfang, meistens genügt aber normal. Das Multiplizieren mit Time in der Effectanweisung, ist dafür da, dass auf jedem PC der Effekt in etwa gleich aussieht.

Testen des Effekts

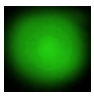
Fügen Sie nun eine Fackel oder desgleichen in Ihr Level ein, fügen Sie auch noch Block.mdl den Sie im Hauptverzeichnis dieses Tutorials finden an der Position der Fackel ein und geben ihm die Aktion Fackel. Nun Compilern Sie den Level bitte und starten ihn dann. Wenn alles so aussieht wie hier, haben Sie alles richtig gemacht!

Partikeleffekt einem Vertex zuordnen

Sie fragen sich vielleicht, wie man bei einem Auto den Auspuff zum rauchen bringt, oder wie man einen Finger so zum leuchten bringt wie der von ET. Das letztere werden wir nun machen. Aber zuerst brauchen wir die Vertexnummer an dem der Effekt erstellt werden soll. Starten Sie also MED und laden Warlock.mdl.



Wir werden den Effekt dem Vertex nr. 17 zuordnen. Dafür gib es einen nützliche Befehl namens Vec_For_Vertex die Syntax lautet wie folgend
Vec_for_vertex(vektor, entity, nummer); Vektor gibt die Variabel an in den die Koordinaten des Vertex kopiert werden sollen, Entity ist der Name des Entitys an dem das Vertex platziert ist und die Nummer gibt die Nummer des Vertex an. Als Partikel verwenden wir folgende Grafik:



Nun setzen Sie bitte warlock.mdl ins Level ein und geben ihm die Aktion ET und Compilern bitte den Level. Öffnen Sie nun bitte wieder Ihre C-Script Datei und schreiben folgenden Code:

```
bmap et_partikel = <grun.bmp>;//Grafik

function et_finger()
{
	my.flare = on;//Flare einschalten
	my.alpha = 70;//Transparents
	my.bmap = et_partikel;//Bild des Partikels
	my.lifespan = 1;//Partikel lebt nicht sehr lange
	my.size = 9;//Partikelgrösse 9
	my.move = off;//Partikel bewegt sich nicht

	my.function = null;//Partikel besitzt keine Funktion
}

string box = <box.mdl>;/*String, dafür man nachher diesen Entity mit ent_create erstellen kann*/
entity* finger;//Ein Synonym mit dem man auf den Entity zugreifen kann

action licht_finger
{
	finger= me;//Mein Name ist Finger
	my.light = on;//Licht einschalten
	my.green=255;//Hellgrün leuchten
	my.blue = 0;//Kein Blauanteil
	my.red = 0;//Keinen Rotanteil
	my.lightrange = 20;//Licht reicht 20 Quants weit
	my.invisible = On;//Entity nicht sichtbar
}

action et
{
	vec_For_vertex(temp,my, 17);//Koordinaten des Vertex #17 in Temp speichern

	ent_create(box, temp, licht_finger);//Einen Entity an dieser Position erstellen

	while(1)//Ausführen solange Spiel läuft
	{
		//Entity animieren
		if(my.skill1 > 100) { my.skill1 = 0; }//Wenn skill1 grösser ist als 100 dann 0
		my.skill1 += 2 *time;//Skill1 erhöhen um 2 mal time
		ent_cycle("attack", my.skill1);//Animation abspielen
		vec_For_vertex(temp,my, 17);//Wieder Position des Vertex #17 holen
		vec_set(finger.x, temp);//Kopiere temp in die Position von Finger(Der Box)
		effect(et_finger, 1, temp, normal);//Partikeleffekt erstellen
		wait(1);//einen Framezyklus warten
	}
}
```

Erklärung
Die erste Funktion sollten Sie nun verstehen, da ist nicht viel neues, ausser das die Partikel nicht bewegt werden. String ist eigentlich nur ein Textobjekt und die Syntax lautet wie folgend String name = „Text“; man kann in einen Text auch einfach nur einen Text speichern, aber man kann auch Entitys speichern und diese dann per ent_create erstellen. Entity* nannte man früher Synonym aber Synonym ist veraltet und nennt sich nun eben Entity*. Man nennt diesen Typ auch Zeiger, weil er einen Verweis auf ein Objekt besitzt, man kann wen man den Zeiger einmal einem Objekt zugewiesen hat, auf alle Eigenschaften zugreifen und diese auch verändern. Die Aktion Finger sollten Sie eigentlich dank der Kommentare verstehen, da die Aktion nur aus Lichtbefehlen und Invisible besteht, Invisible bestimmt einfach ob ein Entity sichtbar ist. Nun kommen wir zu ent_create. Mit dieser Funktion lässt sich ein Entity an einer bestimmten Position erstellen, die Syntax lautet wie folgend
ent_create(entity_string, Position, Aktion);
Nun für was brauchen wir eigentlich einen Entity an der Position des Fingers? Nun, da wir keinen Lichteffekt einem Vertex zuordnen können, erstellen wir einen Entity und weisen dem einen Lichteffekt zu und bewegen ihn mit dem Model mit, dies geschieht mit vec_set diese Funktion kopiert einen Vektor in einen anderen. die Syntax lautet wie folgend:
vec_set(Zielvektor, Startvektor);
So ist der Entity immer an der Position des Vertex.

```
f(my.skill1 > 100) { my.skill1 = 0; }//Wenn skill1 grösser ist als 100 dann 0
my.skill1 += 2 *time;//Skill1 erhöhen um 2 mal time
ent_cycle("attack", my.skill1);//Animation abspielen
```

Diese Zeilen Code animieren das Model, ich erkläre hier nur ent_cycle. Ent_Cycle führt eine Animation aus, hier „attack“ wobei my.skill1 sagt, bei welchem Prozent die Animation ist, wollen Sie mehr über diesen Befehl wissen, so schauen Sie doch bitte im Manual nach.

Gedanken
Ich finde diesen Effekt eigentlich nicht aussergewöhnlich gut, aber ich glaube, mit diesem Effekt lernt man viel darüber, wie man Vertex basierte Partikeleffekte erstellen kann.

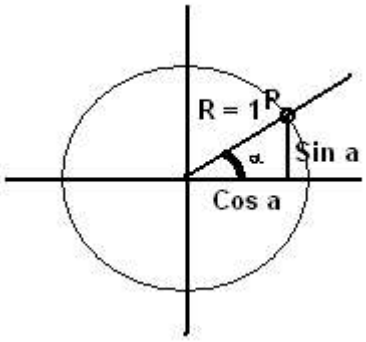

Der fertige Effekt

Partikelkreise

Ich glaube, Sie möchten jetzt wissen, wie man mit Partikel kreise macht. Nun, dafür benötigt man das Wissen über Cosinus und Sininus, beide sind in C-Script unter Cos und Sin vorhanden. Bevor wir loslegen können, entwickeln wir noch eine Formel um einen Punkt auf einem Kreis zu berechnen.

X = radius *sin(Winkel)
Y = radius *cos(Winkel)

Nun grafisch dargestellt:



Das nun in C-Script:

```
temp.x = my.x + radius * cos(winkel);
temp.y  = my.y + radius* sin(winkel);
```

So nun haben Sie genug Wissen um damit einen Effekt zu erstellen. Der Warlock soll sich aus Partikeln ein Kraftfeld um sich herum aufbauen. Dazu verwenden wir folgendes Partikel:

Die Partikel werden Erdanziehung haben, dazu verwenden wir jedoch nicht Gravity sondern programmieren selber eine. Die Partikel werden auf Kopfhöhe erstellt, für die Grösse eines Entity stehen folgende Funktionen zur Verfügung:

```
my.max_z
my.min_z
```

Die Kopfhöhe berechnet man wie folgend:
my.z + my.max_z

Nun der Code der den Partikel die Parameter gibt:

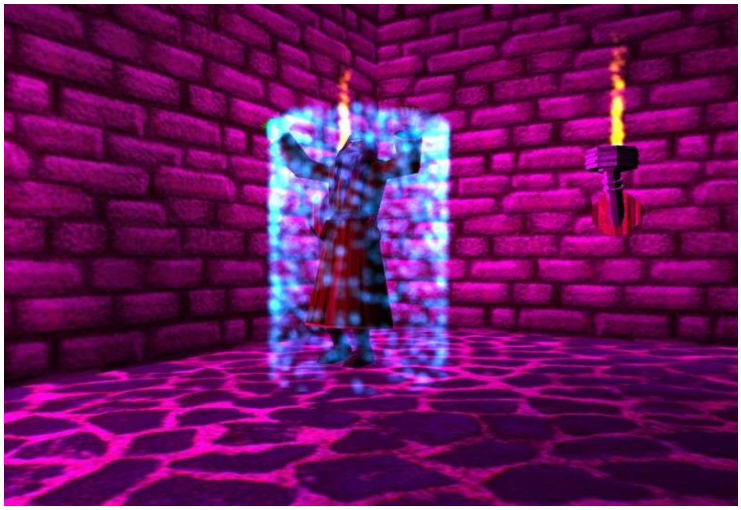
```
bmap schutz_schild = <tropfen2.bmp>;

function ausblenden(ausblend)
{
	my.alpha -= 0.1* TIME; //Verringere Alpha
	IF(my.alpha <= 0) //ist Alpha kleine gleich 0
	{
	 my.lifespan= 0; //Vernichte Partikel
	}
}


function schutz_schild_partikel()
{
	my.vel_x = 0;//Keine Bewegung an der X-Achse
	my.vel_y = 0;//Keine Bewegung an der Y-Achse
	my.vel_z = -random(2);//Leichte Erdanziehung
	my.flare = on;//Flare an
	my.alpha = 70;//Transparents
	my.bmap = schutz_schild;//Partikelbild
	my.size = 9;//Partikelgrösse
	my.move = on;//Partikel dürfen sich bewegen
	my.bright = on;//Partikel werden Beleuchtet
	my.function = ausblenden(.01);

}
```

Den Code sollten Sie nun eigentlich verstehen, neu ist nur, dass wir nun einen Parameter an die Funktion ausblenden(ausblend) übergeben, so lässt sich eine Funktion für alle Partikeleffekte nutzen. Nun schreiben wir die Aktion Warlock2.

```
action warlock2
{
Var winkel;
	while(1)
	{
		//Animation
		if(my.skill1 > 100) { my.skill1 = 0; }
		my.skill1 += 2 *time;
		ent_cycle("attack", my.skill1);


		winkel += 20 * time;//Winkel um 20 mal Time erhöhen
		//Cos und Sin Berechnungen
		temp.x = my.x + 40 * sin(winkel);
		temp.y = my.y + 40 * cos(winkel);

		temp.z = my.z+my.max_z;//Die Kopfhöhe

		effect(schutz_Schild_partikel, 20*time, temp, normal);//Effekt erstellen
		wait(1);
	}
}
```

Erklärung
Ich glaube, dass dieser Code keine Erklärung braucht, da alles Oben schon besprochen wurde. Nur, dass der Radius des Kreises 100 beträgt, was man alles damit machen kann, erfahren Sie im nächsten Kapitel. Die Variabel Winkel ist dafür da, dass die Partikel nicht nur an einer Position erstellt werden, die Partikel, werden auf dem Radius auf verschiedenen Positionen erstellt.

Testen
Geben Sie nun dem Warlock den Befehl Warlock2 und Compilern Sie das Level und starten es dann.



Der Lohn für Ihre Mühe!

Gedanken
Ich denke, dass das jetzt ein sehr schöner Effekt ist, versuchen Sie einmal ein wenig mit den Parameter zu spielen! Man kann oder sollte alle Zahlen durch Variablen ersetzen, so lässt sich ein Effekt für viele Entiys nutzen, ich hab das hier der Einfachheit zu liebe nicht gemacht.

Spiralen

Wissen Sie was eine Spirale ist? Ich glaube Sie wissen das und wen Sie es nicht wissen, werten Sie es bald lernen. Zuerst überlegen wir mal aus was eine Spirale besteht. Natürlich aus einem Kreis, der einen stetig ansteigenden Radius hat, diese Eigenschaft mit dem oberen Kapitel kombiniert, sollte also eine Spirale geben! Dafür brauchen wir jedoch eine weitere Variabel, diese nennen wir passend „radius“ diese Variabel soll bei jedem Aufruf um 1 hochgezählt werden, wobei sie noch mit time multipliziert wird. Eigentlich ist das schon alles für eine Spirale. Wir ersetzen einfach die Zahl die für den Radius steht durch die Variabel.

```
action warlock3
{
Var radius;
Var winkel;

	while(1)
	{
		//Animation
		if(my.skill1 > 100) { my.skill1 = 0; }
		my.skill1 += 2 *time;
		ent_cycle("attack", my.skill1);

		if(radius < 360)
		{
			radius += 1 * time;//Radius langsam vergrössern
			winkel += 20 * time;//Winkel um 20 mal Time erhöhen

			//Cos und Sin Berechnungen
			temp.x = my.x + radius * sin(winkel);
			temp.y = my.y + radius * cos(winkel);

			temp.z = my.z+my.max_z;//Die Fusshöhe

			effect(spirale, 20*time, temp, normal);//Effekt erstellen
		}
	  wait(1);
	}
}
```

(Alle Erneuerungen sind orange markiert.)

Erklärung
Wir haben eine neue Variabel namens Radius genommen, diese wird bei jedem Aufruf hochgezählt und wie Sie vielleicht schon vermutet haben, bestimmt diese Variabel den Radius. Ist der Radius grösser als 150, dann ist der Effekt fertig und wird nicht mehr aufgerufen. Die Spirale bewegt sich an dem Vektor Z hoch, wobei sie bei den Füssen anfängt, auch Radius wird zu Z hinzugezählt, worauf die Spirale sich in die Lüfte erhebt.

Die Spiralengrafik
Für diesen Effekt verwenden wir folgendes Partikel:

Und der Code:

```
bmap partikel_spirale = <grun.bmp>;

function spirale()
{
	my.vel_x = 0;//Keine Bewegung an der X-Achse
	my.vel_y = 0;//Keine Bewegung an der Y-Achse
	my.vel_z = 0;//Keine Bewegung an der Z-Achse
	my.move = off;//Keine Bewegung
	my.flare = on;//Flare eingeschaltet
	my.bright = on//Bright einschalten
	my.alpha = 70;//70% Transparent
	my.bmap =grun;//Bitmap des Partikels
	my.function = ausblenden3(2);//Funktion
}
```

Man sollte beachten, dass dieser Code vor der Action Warlock3 geschrieben werden sollte, sonst gibt es einen Error! Nun noch die Aktion einem Entity zuweisen und testen.

Der fertige Effekt, na ja, ich gebe zu, ich habe gemogelt, diese Partikel werden in meiner Funktion nicht ausgeblendet :-), damit man es besser sieht.

3D-Partikelkreise

Jetzt werden Sie lernen wie man Kreise erstellt, die 3 Dimensional wirken. Die Partikel werden den Entity umschwirren. Der Effekt wird etwas so aussehen:

Das Problem

Wir wissen bereits, wie man mit Partikel einen Kreis erstellt, jetzt müssen wir versuchen, dein Kreis um 90° zu drehen. Das heisst, wir müssen dem Richtungsvektor Z entweder die Berechnung von dem Vektor Y oder X zuweisen. Der Vektor der Ersetzt wird erhält einfach seine Position, also my.x oder my.y.

```
temp.x = my.x + radius * sin(winkel);//Bleibt normal
temp.y = my.y;//Y enthält jetzt die Position des Players an der Y-Achse
temp.z = my.z+ radius * cos(winkel);/*Die Berechnung von Y einfach auf die Z-Achse bezogen*/
```

Führt zu:

Daraus können wir schlissen, dass

```
temp.x = my.x;//Enthält die Position des Players an der X-Achse
temp.y = my.y  + radius * cos(winkel);//Wieder normal
temp.z = my.z + radius * sin(winkel);/*Die Berechnung von X einfach auf die Z-Achse bezogen*/
```

Zu dem führt:

Wen wir nun das kombinieren, sollte es so aussehen:

Und nun Die Aktion Warlock4, wir werden die gleiche Partikelgrafik wie oben verwenden, werden aber eine eigene Funktion dafür schreiben. Wenn Sie wollen, dass es geschlossene Kreise sind, verwenden Sie die Funktion von der Spirale, vielleicht müssen Sie auch noch einige Parameter anpassen.

```
bmap partikel_spirale = <grun.bmp>;
function schild2()
{
	my.vel_x = 0;
	my.vel_y = 0;
	my.vel_z = 0;
	my.flare = on;
	my.alpha = 100;

	my.bmap = partikel_spirale;
	my.size = 19;

	my.function = ausblenden3(2);
	my.lifespan = 100;
}

action warlock4
{
	radius = 70;
	Var winkel;

	while(1)
	{
		//Animation
		if(my.skill1 > 100) { my.skill1 = 0; }
		my.skill1 += 2 *time;
		ent_cycle("attack", my.skill1);


		winkel += 20 * time;//Winkel um 20 mal Time erhöhen

		//Cos und Sin Berechnungen für den ersten Kreis
		temp.x = my.x + radius * sin(winkel);
		temp.y = my.y;
		temp.z = my.z+ radius * cos(winkel);

		effect(schild2,  20*time, temp, normal);//Partikel erstellen

		//Anderer Kreis
		temp.x = my.x;
		temp.y = my.y+ radius * cos(winkel);
		temp.z = my.z+ radius  * sin(winkel);

		effect(schild2, 20*time, temp, normal);//Partikel erstellen

		wait(1);
	}
}
```

Erklärung

Alles wurde eigentlich schon oben erklärt, wir rufen 2mal effect auf, einfach mit anderen Parameter.

Gedanken
Damit Sie den Entity auch noch bewegen können, müssen Sie lifespan anpassen.

Referenz

Dieses Kapitel erklärt alle Befehle die es für Partikel gibt, wobei darauf geachtet werden sollte, dass es einige Optionen nicht für alle Editionen gibt.

Vel_X, Vel_Y und Vel_z **Editionen: Alle**
Diese Variabeln geben die Geschwindigkeit an den bestimmten Achsen an, funktioniert nur wen Move aktiviert ist!

Move **Editionen: Alle**
Gibt an, ob sich das Partikel bewegen kann. Default: Off

Lifespan **Editionen:Alle**
Bestimmt wie lange ein Partikel existiert (in Ticks) 80 ist voreingestellt.

Size **Editionen: Alle**
Bestimmt die Grösse eines Partikels. Voreingestellt: 4

Alpha **Editionen: Alle**
Bestimmt die Transparents: 0 unsichtbar 100 total sichtbar. Default: 50

Bright **Editionen: Alle**
In Kombination mit flare oder transparent, illuminiert bright den Partikel oder die Entity vor dem Hintergrund, anstatt beides zu mischen. Beleuchtet das Partikel. Default: Off

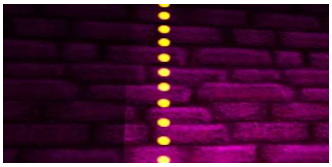
Flare **Editionen: Alle**
Macht das Partikel transparent, alle schwarzen Flächen wen das Partikel schon einen Alphawert besitzt ist Flare automatisch eingestellt.

Bmap **Editionen: Alle**
Die Bitmap des Partikels, wen eine vorhanden ist.

RED, BLUE, GREEN **Editionen: Alle**
RGB-Farbwerte, ist keine Bitmap gesetzt, werden diese Farben benutzt, existiert eine, werden die Farbwerte der Bitmap hinzugezählt.

Beam **Edtitionen: Ab Com**
Wird dieser Flag gesetzt, wird das Partikel verschmiert, damit lassen sich Lasereffekte, etc. erzeugen, sollte aber sparsam eingesetzt werden, da es die Framerate negativ beeinflusst.

Gesetzter Beamflag



Das Gleiche ohne gesetzten Beamflag

Streak **Editionen: Ab Com**
Das Partikel wird entlang einer Linie gestreckt. Default: OFF

Gesetzter Streakflag

Function **Editionen: Alle**
Wen das Partikel eine Funktion wie z.B: Das sanfte ausblenden besitzen soll, wird dieser Befehl benutzt. Default: Null

Gravity **Editionen: Alle**
Ist dieser Flag gesetzt, haben die Partikel erdanziehung. Default: OFF

Tipps

Verwenden Sie nie zu viele Partikel, denn das könnte das Spiel unspielbar machen. Experimentieren Sie mit den verschiedenen Einstellungen, denn mir ist noch nie von Anfang an ein Effekt richtig gut gelungen. Lesen Sie die Scripts von anderen Leuten und versuchen Sie, deren Effekte nachzubauen und zu Verstehen.

Worte zum Schluss

Ich hoffe, dass Ihnen dieses Tutorial gefallen hat und Sie nun mehr über Partikel wissen. Sollten Sie einen Fehler gefunden haben, schreiben Sie mir, haben Sie noch eine Ergänzung so schreiben Sie mir doch! Meine Email-Adresse: burnner@gmx.ch ich freue mich auf Post :-). Die Partikel die Sie im Tutorialordner finden können, dürfen Sie verwenden ohne irgendwelche Credits oder so auch wen das nett wäre!