Atelier sur les effets spéciaux
rev:1.31

Avant propos

Bienvenue à mon atelier sur les effets spéciaux!

Avec cet atelier vous allez apprendre à créer de nombreux effets et lorsque vous le quitterez vous aurez la connaissance des effets WDL et des nouvelles particularités du moteur A5.1 pour créer vos propres effets étonnants!

Cet atelier est fait pour le moteur A5. Une version A4 peut être trouvée sur la section de téléchargement d'Interactive.
(http://acknexinteractive.fws1.com/download.htm)

En écrivant cet atelier je vous envoie un petit mot d'encouragement pour que vous alliez le plus loin possible et que vous puissiez créer de grands nouveaux effets sans ennuis.
Salutations,

Nizzy

Utilisation de time

Avant de commencer,je voudrais expliquer l'utilisation de ***time***. Un codage sans cette variable est entièrement dépendant de votre taux de rafraîchissement. Comme:

```
while(1)
{
	my.alpha -= 5;
	wait(1);
}
```

Pour contrer ceci, nous allons utiliser la variable système ***time.*** Elle représente le temps entre le changement de frame. La plupart du temps > 1 sauf sur les machines lentes. Donc, en multipliant le nombre par ***time*** vous aurez des effets plus lisses indépendants du taux d'affichage.

```
while(1)
{
```

```
	my.alpha -= 5 * time;
	wait(1);
}
```

Le rougeoiment alpha

Maintenant, nous allons inclure une gemme rayonnante qui nous redonnera des points de vie. Mais nous ne voulons pas que cela émette de la lumière, nous le ferons donc *transparent* et nous augmenterons et diminuerons la clarté pour obtenir un effet rayonnant.
Pour ce faire nous emploierons le drapeau ***transparent*** et la valeur ***alpha***. Le drapeau ***transparent*** fait que l'entité sera semi-opaque. Et la valeur ***alpha*** dit de combien elle est opaque, 100 étant opaque et 0 étant invisible. Regardons ceci:

```
action gem
{
	if (my.skill5 == 0) {my.skill = 25; }
	my.event = medi_pickup;
	item_pickup();
}
```

C'est le même code que celui qui est employé dans le medi packs. Maintenant nous pouvons ajouter notre code rayonnant comme ceci.

```
action gem
{
	if (my.skill5 == 0) {my.skill5 = 25; } // par défaut +25 santé
	if (my.skill3 == 0) {my.skill3 = 75; } // par défaut 75 max opaque
	if (my.skill2 == 0) {my.skill2 = 25; } // par défaut 25 max transparence
	my.event = medi_pickup;
	my.transparent; le rend sensible à la valeur alpha
	item_pickup();
	while (1)
	{
		// Nous voulons que cela rougeoie en permanence et non juste une fois.
		while (my.alpha <= my.skill3)
		{
			my.alpha += time; // le rend plus opaque
			wait(1);
		}
		while (my.alpha >= my.skill2)
		{
			my.alpha -= time; // le rend plus transparent
			wait(1);
		}
	}
}
```

Ce n'est pas plus sympa comme ça ? Essayons à présent de faire un effet d'éclat.
Un éclat est attaché à une lumière et devient plus grand lorsque vous êtes loin de lui et plus petit lorsque vous êtes tout près. Pour notre éclat nous utiliserons un éclat circulaire simple utilisant le drapeau transparent ***flare***.

```
var flare_var = 0;
action light_flare
{
	my.flare = on;
	my.red = your.red;
	my.green = your.green;
	my.blue = your.blue;
```

```
	my.light = on;
	my.lightrange = 0;
	my.passable = on;
	my.facing = on;
	my.near = on;
	while (1)
	{
		flare_var = vec_dist(player.x,my.x);
		IF (flare_var > 2000)
		{
			flare_var = 2000;
		}
		my.scale_x = (flare_var / 900 + 0.5);
		my.scale_y = (flare_var / 900 + 0.5);
		if (flare_var < 30)
		{
			my.invisible =on;
		}
		else
		{
			my.invisible = off;
		}
		wait(1);
	}
}
```

Bien, c'est plutôt pas mal! Il est coupé s'il est partiellement bloqué par une autre surface. Aussi nous lui ajoutons un ***my.near*** à présent. Voici le code définitif:

```
var flare_var = 0;
action light_flare
{
	my.flare = on;
	my.red = your.red;
	my.green = your.green;
	my.blue = your.blue;
	my.light = on;
	my.lightrange = 0;
	my.passable = on;
	my.oriented = on; // fera à présent face en utilisant les angles pan, tilt et roll
	pour obtenir un meilleur effet de face en utilisant me_on_player()
	my.near = on; // réduit les effets de coupure
	while (1)
	{
		me_on_player();
		flare_var = vec_dist(player.x, my.x);
		flare_var = min(flare_var,2000); //optimisé pour ne jamais être < à 2000
		my.scale_x = (flare_var / 900 + 0.5);
		my.scale_y = my.scale_x; // petite optimisation (scale_x = scale_y toujours)
		if (flare_var < 30)
		{
			my.invisible = on;
		}
		else
		{
			my.invisible = off;
		}
		wait(1);
	}
}

function me_on_player() {
```

```
		// pour être toujours face au joueur
		vec_set(temp,player.x);
		vec_sub(temp,my.x);
		vec_to_angle(my.pan,temp);
}
```

A présent c'est exactement ce que nous souhaitions! Nous allons à présent faire des effets de verres! Pour clore cette section de l'atelier nous allons créer un bris de vitre en utilisant la transparence ***alpha***. Notre bris de fenêtre agira comme un vrai bris de verre. Il ne doit pas se casser du premier coup à chaque fois. Pour ce faire nous utiliserons la variable ***_health*** utlisée pour les ennemis dans war.wdl. Commençons par regarder notre ***action***:

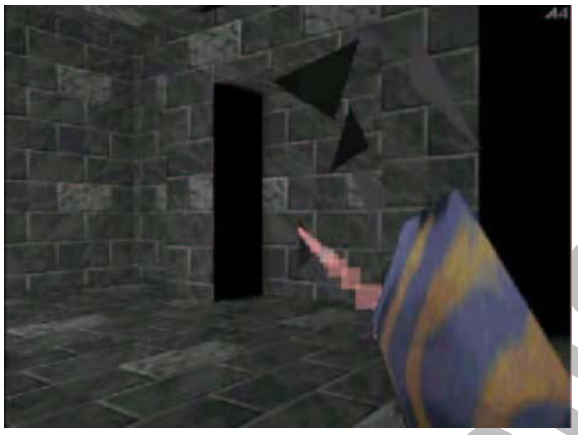
```
action window
{
		my.transparent = on;
		my._health = 50;
		my.event = glass_hit;
		my.enable_shoot = on; //sensible au balles
		my.enable_scan = on; //sensible aux explosions
		my.push = 10; //passe partout avec une poussée <10
		while (1)
		{
			my.alpha = 100 - my._health; //opacité proportionnelle aux dégâts.
		}
}
```

Cela met la variable health de la fenêtre à 50 et nous avons une transparence de 50% au départ qui deviendra plus opaque à chaque dégât pris. A présent regardons la fonction évènement et la fonction particule qui seront placées avant l'action:

```
function glass_hit();
{
		if (event_type == event_scan){ if (indicator != _explode) && (indicator !=
		_gunfire) {end;}} // quitte si ce n'est pas une explosion d'un coup de feu
		my._health -= damage; //soustrait les dommages causés par l'explosion ou la balle.
		if (my._health <= 0)
		{
			glass_gib(10);
			remove(me);
		}
}

function glass_gib(numberOfParts)
{
		temp = 0;
		while(temp < numberOfParts)
		{
			create(<glass.mdl>, my.pos, _gib_action); // utilise gib_action dans war.wdl
			pour animer les éclats de verre, les faire toumoyer et rebondir
			temp += 1;
		}
}
```

A présent la fenêtre éclatera en 10 morceaux qui disparaîtront en fondu après sa destruction.



Voici le code définitif:

```
function glass_hit();
{
	if (event_type == event_scan){ if (indicator != _explode) && (indicator !=
	_gunfire) {end;}} // quitte si ce n'est pas une explosion d'un coup de feu
	my._health -= damage; // soustrait les dommages causés par l'explosion ou la balle.
	if (my._health <= 0)
	{
		glass_gib(10);
		remove(me);
	}
}

function glass_gib(numberOfParts)
{
	temp = 0;
	while(temp < numberOfParts)
	{
		create(<glass.mdl>, my.pos, _gib_action); // utilise gib_action dans war.wdl
		pour animer les éclats de verre, les faire tournoyer et rebondir
		temp += 1;
	}
}

action window
{
	my.transparent = on;
	my._health = 50;
	my.event = glass_hit;
	my.enable_shoot = on; //sensible aux balles
	my.enable_scan = on; //sensible aux explosions
	my.push = 10; //passe partout avec une poussée <10
	while (1)
	{
		my.alpha = 100 - my._health; //opacité proportionnelle aux dégâts.
	}
}
```

Lumières

La première chose à déterminer dans l'éclairage est ce que vous souhaitez que la lumière fasse. Dans cet atelier nous représenterons l'éclairage d'un feu et le vacillement d'une lampe cassée. Nous commençons avec le feu.
Bien, notre éclairage utilisera sa variable d'entité ***skill4*** pour la distance de la lumière et nous ajouterons une valeur aléatoire avec sa variable ***skill5*** pour l'intensité. Les couleurs de la lumière seront données par les variables ***skill1-3*** et devront s'exécuter pour toujours. Nous utiliserons donc une boucle ***while***. Allons-y tout simplement:

```
action fire
{
	my.red = my.skill1;
	my.green = my.skill2;
	my.blue = my.skill3;
	while (1)
	{
		my.lightrange = my.skill4 + random(skill5);
		wait(1);
	}
}
```

Les ***rouge***, ***vert***, ***bleu*** sont donnés par les variables ***skill1-3*** comme indiqué précédemment. Et ***lightrange*** donne l'étendue de lumière émise par l'entité. Sa valeur minimale est donnée par ***skill4*** et nous l'ajoutons à ***skill5*** pour créer un éclat aléatoire. Nous attendons ensuite une frame avec ***wait*** et nous recommençons.
A présent j'ai créé un nouveau niveau avec un grand bloc creux et la texture par défaut, j'ai ajouté le modèle torch.mdl (inclu dans cet atelier) je lui ai assigné l'action fire et il ne se passe RIEN. Mais pourquoi tant de haine?!
A votre avis ? Tout simlement parceque je n'ai pas renseigné les variables d'entité ***skill1-5***! Pour contourner ceci, nous pouvons avoir à faire à des utilisateurs distraits, nous allons donner des valeurs par défaut à ces variables qui seront utilisées si l'utilisateur ne les renseigne pas. Comment faire cela? Essayons ceci.

```
action fire
{
	if (my.skill1 == 0) { my.skill1 = 155; }
	if (my.skill2 == 0) { my.skill2 = 55; }
	if (my.skill3 == 0) { my.skill3 = 0; } //orange light
	if (my.skill4 == 0) { my.skill4 = 300; }
	if (my.skill5 == 0) { my.skill5 = 100; }
	my.red = my.skill1;
	my.green = my.skill2;
	my.blue = my.skill3;
	while (1)
	{
		my.lightrange = my.skill4 + random(skill5);
		wait(1);
	}
}
```

Maintenant avant toute chose nous vérifions si les variables sont à 0, si c'est le cas alors nous mettrons nos valeurs par défaut! Aprésent lorsque j'exécute mon niveau tout simplement j'obtiens bien un feu avec une lumière rouge vacillante comme un vrai! Mais il manque une chose. La variation. Essayons de changer un peu la couleur à chaque fois, plus sombre en réduisant la valeur de ***lightrange***, plus clair en augmentant la valeur de ***lightrange***.

```
action fire
{
```

```
	if (my.skill1 == 0) { my.skill1 = 225; }
	if (my.skill2 == 0) { my.skill2 = 140; }
	if (my.skill3 == 0) { my.skill3 = 55; } //lumière rouge
	if (my.skill4 == 0) { my.skill4 = 70; }
	if (my.skill5 == 0) { my.skill5 = 100; }
	my.red = my.skill1;
	my.green = my.skill2;
	my.blue = my.skill3;
	while (1)
	{
		my.lightrange = my.skill4 + random(my.skill5);
		my.red += random(my.skill6)-(my.skill6 / 2);
		my.green += random(my.skill7)-(my.skill7 / 2);
		my.blue += random(my.skill8)-(my.skill8 / 2);
		wait(1);
	}
}
```

Là, cela semble parfait. A présent nous allons ajouter ou soustraire jusqu'à la moitié de ***skill6-8*** des valeurs RVB.

Maintenant à notre lumière cassée vacillante. Nous devons l'éteindre pendant un temps aléatoire puis la faire vaciller ensuite pendant un court temps aléatoire.

```
var light_var;
action light_flicker
{
	if (my.skill1 == 0) { my.skill1 = 155; }
	if (my.skill2 == 0) { my.skill2 = 55; }
	if (my.skill3 == 0) { my.skill3 = 55; } //lumière rouge
	while (1)
	{
		// lumière allumée
		my.red = my.skill1;
		my.blue = my.skill2;
		my.green = my.skill3;
		my.ambient = 100;
		my.lightrange = 150;
		light_var = random(8);
		wait(light_var);
		// lumière éteinte
		my.ambient = 0;
		my.lightrange = 0;
		light_var = random(16);
		wait(light_var);
		wait(1);
	}
}
```

light_var est mis à une valeur aléatoire qui est utilisée par l'instruction ***wait*** 16 fois pour le temps de l'extinction et 8 fois pour le temps de l'allumage. Je pense que ça donnera un effet sympa!

Particules

Avant de commencer cette section, je voudrais donner un grand merci à Gaehh. Son tutorial m'a permis d'apprendre que l'on pouvait faire de grandes choses avec les particules, je ne connaissais rien du drapeau ***bright*** ou de my_pos_x etc. Je pensais qu'il y avait uniquement my_speed_x etc. le my_pos_x etc. Sera donc utilisé dans les prochains effets et tutoriaux. ***MERCI GAEHH***.

Ok, commençons par quelque chose de simple, le saignement.
Notre fonction de particule sera presque la même que la dispertion de particule et emploiera beaucoup des grandes nouvelles particularités du moteurs de système de particule. Dans cette atelier j'expliquerai aussi tout ce que je peux de chaque drapeau pour vous aider à créer de meilleurs effets!

my.bright-
Les particules seront plus brillantes dans les secteurs qui ont des particules se chevauchant pour créer l'effet de feu ou un rougeoiement.

my.transparent-
Les particules assumeront la transparence alpha et emploieront la valeur de ***my.alpha*** pour mettre le % de transparence, si nul, les particules seront 50 % transparentes.

my.lifespan-
Les particules vivront jusqu'à ***my.lifespan*** = 0**, lifespan** diminuera la durée de vie des particules automatiquement. Mettre à 0 pour tuer la particule.

my.flare-
Les particules seront transparentes comme avec ***my_transparent*** mais les secteurs plus sombres seront plus transparents tandis que les secteurs plus claires de la particule seront plus opaques.

my.alpha-
Met le % de transparence des particules, 100 % est opaque et 0 % est clair.

my.function-
Mis à nul pour continuer la vie des particules jusqu'à ***my.lifespan*** = 0;.

Voilà, démarrons l'écriture du script!

Commençons avec ***action***:

```
var firevar;
action fire1
{
```

```
		if (my.skill1 == 0) { my.skill1 = 8; }
		while (1)
		{
			firevar = my.skill1; // mets le nombre de particules qui doivent ëtre libérées.
			effect(particle_fire,firevar * time,my.x,nullvector); // émet les particules à partir de my.x en
			utilisant la nouvelle commande d'effet (A5)
			wait(1);
		}
}
```

Cette ***action*** est comme une autre ***action*** à part le fait qu'elle utilise des variables ***skills***
Qui inter-agissent avec la partie d'***effect***. Pour ce faire vous devez copiez la variable ***skill*** dans une variable ***var***. et utiliser *var* dans la commande ***effect***.

Maintenant nous pouvons créer la fonction:

```
bmap fire_map1 = <light.bmp>;
var firevar2,1;
function vec_fire(&vec)
{
	vec[0] = random(30) – 15;
	vec[1] = random(30) – 15;
	vec[2] = random(10);
}

function particle_fire()
{
	vec_fire(temp);
	vec_set(my.vel_x,temp);
	my.lifespan = 15;
	my.flare = on; //transparence d'éclat
	my.bright = on; //effet de rougeoiment
	my.move = on; //se déplaceront selon vel_x, vel_y, et vel_z
	my.streak = on; //nouvel effet de bande
	my.bmap = fire_map1;
	my.alpha = 20;
	my.size = 500;
	my.function = null; //rien pour l'instant!
}
```

Wow, cela ne semble pas MAUVAIS? Cependant, c'est seulement une "ossature" pour un scénario de particules beaucoup plus avancé pour une vraie physique de feux réels.
Comment obtenir éventuellement une jolie courbe agréable.

```
function fire_func()
{
	my.vel_x == my.vel_x * 0.5;
	my.vel_y == my.vel_y * 0.5;
}
//also change the "my.function = null;" in particle_fire to "my.function = fire_func;"
```

Maintenant nous divisons la vitesse x et y par 2 à chaque frame, ce qui fait qu'elle n'atteindra jamais 0, mais ça ne joue pas sur la performance.(il aura la valeur 1 puis .5,.25,.125,.0625 et ainsi de suite).

A présent le but que nous nous sommes fixés : comment obtenir cette impression d'un vrai feu. Plutôt que fabriquer toutes sortes de formules plus compliquées les unes que les autres, nous allons utiliser une idée de chaleur. Si une particule est trop éloignée du centre du feu elle meurt parce que c'est plus froid, elle se déplacera également moins vite parce qu'il y a moins de chaleur pour monter. Aussi ce que nous devons faire c'est de modifier ***my.vel_z***, comme vous verrez.

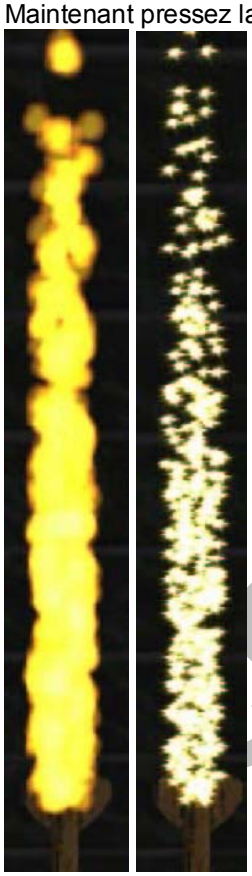
```
function vec_fire(&vec)
{
	vec[0] = random(30) – 15;
	vec[1] = random(30) – 15;
	vec[2] = random(10) – (abs(my.vel_x) + abs(my.vel_y)) * 0.3 + random(5);
}
```

Finalement nous voulons évaluer quel graphisme présente le mieux.

```
function fire_func()
{
	my.vel_x = my.vel_x * 0.5;
	my.vel_y = my.vel_y * 0.5;
	my.alpha -= 2 * time; //disparait en fondu graduellement
	if(my.alpha < 0) { my.lifespan = 0; } // est tué lorsqu'alpha < 0
	if (firevar2 == 1) { my.bmap = firemap1; }
	if (firevar2 == 2) { my.bmap = firemap2; }
	if (firevar2 == 3) { my.bmap = firemap3; }
	if (firevar2 == 4) { my.bmap = firemap4; }
}

function firechange()
{
	firevar2 += 1;
	if (firevar2 == 5) { firevar2 -= 4; }
	wait(1);
}
on_t firechange;
```

Maintenant pressez la touche T pour changer le graphisme!



Pour finir, nous ferons une fontaine de particule comme dans la démonstration Adeptus, commençant d'abord avec la fonction:

```
bmap fountain_map = <sparkle.bmp>;
function vec_fountain(&vec)
{
	vec[0] = random(6) – 3;
	vec[1] = random(6) – 3;
	vec[2] = random(10) + 20;
}

function particle_alpha_fade()
{
	my.alpha -= time * 2; // s'adapte au taux d'affichage
	if(my.alpha < 0) { my.lifespan = 0;}
}

function particle_fountain()
{
	vec_fountain(temp);
	vec_set(my.vel_x,temp); //met la vitesse
	my.lifespan = 50;
	my.flare = on; // transparence d'éclat
	my.bright = on; // effet rougeoyant
	my.move = on; // se déplacent selon vel_x, vel_y, et vel_z
	my.streak = on; //nouvel effet de bande
	my.bmap = fountain_map;
	my.gravity = 3;
	my.alpha = 50;
	my.size = 10;
	my.function = particle_alpha_fade; //disparaissent en fondu près de la fin de vie
}
```

C'est presque la même fonction (seulement adaptée aux particules A5!!), mais regardons ce qui est changé. J'ai enlevé toutes les vieilles instructions SET et les ai remplacées avec de simples définitions de variable et la fonction alpha-fade. Aussi, les particules disparaîtront en fondu plutôt que de simplement disparaître. Maintenant nous voulons voir comment tout cela fonctionne. ***my.vel_x*** et ***my.vel_y*** auront un maximum de 10 dans n'importe quelle direction. ***my.vel_z*** montera à plus de 20, mais moins de 30. ***my.size*** définit la taille des particules. ***my.alpha*** est mis en opaque (100) et disparaîtra en fondu comme sa vie continue. ***my.flare*** rend les particules moins opaques dans les zones sombres. Et pour finir, ***my.lifespan = 50*** tuent les particules agées de plus de 50.

Explosions

Les explosions sont ordinaires et pourtant si différentes, mais je vais vous donner un début pour cela. Nos explosions seront des animations de sprites mais nous y ajouterons quelques effets. Nous utiliserons le sprite d'explosion de MISSION2 comme suit:

Nous démarrons avec l'animation du sprite:

```
action explode1
```

```
{
	my.facing = on; // face à la caméra
	my.near = on;
	my.flare = on;
	my.passable = on; // on ne pousse pas le joueur à travers les murs
	my.frame = 1;
	my._dieframes = 16;
	wait(1);
	play_entsound(my,explo_wham,1000);
	my.red = 210;
	my.green = 100;
	my.blue = 100;
	my.ambient = 100;
	my.lightrange = 110;
	// utilise la nouvelle animation des sprites
	while(my.frame < my._dieframes)
	{
		wait(1);
		my.lightrange += 15;
		my.red += 20 * time; // vire au rouge // s'adapte au taux d'affichage
		my.blue -= 20 * time; // s'adapte au taux d'affichage
		my.frame += time; // s'adapte au taux d'affichage
	}
	wait(1);
	ent_remove(my);
}
```

Essayons de mettre des particules:

```
bmap explo_map = <expl.bmp>;
function vec_explode(&vec)
{
	vec[0] = random(6) – 3;
	vec[1] = random(6) – 3;
	vec[2] = random(10) + 20;
}

function explosion_func()
{
	if(my.lifespan > 30) { my.alpha += 3 * time; } // s'adapte au taux d'affichage
	if(my.lifespan <= 30) { my.alpha -= time; }
}

function particle_explo();
{
	vec_explode(temp);
	vec_set(my.vec_x,temp);
	my.flare = on;
	my.bright = on;
	my.move = on;
	my.streak = on;
	my.bmap = explo_map;
	my.gravity = 6;
	my.alpha = 30;
	my.size = random(50) + 50; // 50-100
	my.lifespan = 50;
	my.function = explosion_func;
}
```

Et nous ajoutons ***effect*** à explode1.

```
action explode1
{
	my.facing = on; // face à la caméra
	my.near = on;
	my.flare = on;
	my.passable = on; // ne pas pousser le joueur à travers les murs
	my.frame = 1;
	wait(1);
	play_entsound(my,explo_wham,1000);
	effect(particle_explo,50 * time,my.x,nullskill);
	my.red = 210;
	my.green = 100;
	my.blue = 100;
	my.ambient = 100;
	my.lightrange = 110;
	// utilise la nouvelle animation des sprites
	while(my.frame < my._dieframes)
	{
	wait(1);
		my.lightrange += 15;
		my.red += 20 * time; // vire au rouge // s'adapte au taux d'affichage
		my.blue -= 20 * time; // s'adapte au taux d'affichage
		my.frame += time; // s'adapte au taux d'affichage
	}
	wait(1);
	ent_remove(my);
}
```

Maintenant on le teste!

Sphères de Ciel

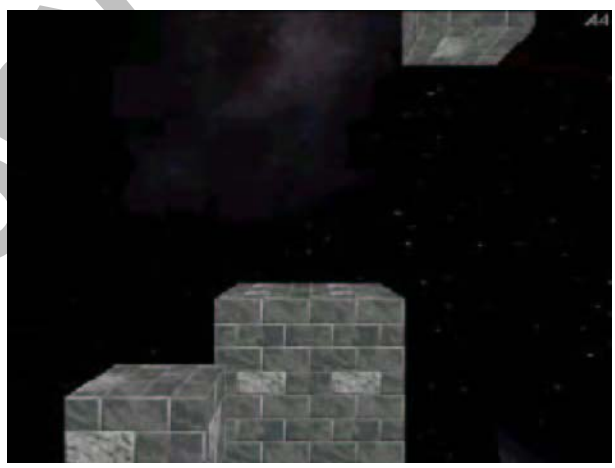
Cela vous présente l'utilisation des sphères de ciel semblables à celles employées dans *l' atelier du vol dans l'espace*, mais pour jeux de tir, arènes et autres de ce genre. Nous commencerons par la partie graphique.

1. Ouvrir MED.
2. Créer une large sphère, vous pouvez en faire une mieux avec milkshape et l'importer, elle sera mieux en high-poly, MED ne **divise pas assez bien.**.
3. Maintenant une texture avec des étoiles ou avec ce à quoi vous voulez que ressemble votre ciel.
4. Faîtes le assez grand pour qu'il englobe votre niveau dans sa totalité (qui est réellement grand)
5. Maintenant vous serez **à l'intérieur**, aussi la texture doit être à l'intérieur, pour cela basculer en mode triangle et sélectionnez tous les triangles.
6. Cliquez sur le bouton **flip normals** et sauvez le comme **stars.mdl** ou le nom que vous voulez.

Maintenant nous avons un script tout simple à faire:

```
action starsphere
{
	while(1)
	{
		my.pan += 0.5 * time;
		my.tilt += 0.5 * time;
		wait(1);
	}
}
```

Si vous voulez, vous pouvez le laisser sans action. Cette action se contente de le faire tourner très lentement.



Fumée

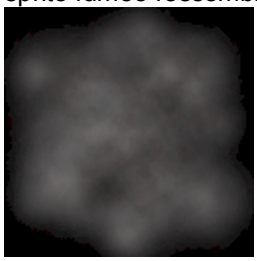
Que ce soit la fumée au lancement d'une fusée ou celle d'un feu, vous allez employer la fumée dans votre jeu et je suis sûr que vous pensez que les calibres particle_smoke ne sont pas exactement ce que vous attendez d'une "qualité commerciale". Aussi nous allons faire le notre mais commençons par regarder celui qui existe :

```
function vec_smoke(&vec)
{
	vec[0] = random(1) – 0.5;
	vec[1] = random(1) – 0.5;
	vec[2] = random(1) + 1;
}

function smoke_func()
{
	if(my.size < 800) { my.size += time; }
	my.vel_x = my.vel_x / 2;
	my.vel_y = my.vel_y / 2;
}

function particle_smoke()
{
	vec_smoke(temp);
	vec_set(my.vel_x,temp);
	my.size = random(5)+5; // taille de 5 à 10
	my.bmap = smoke_map2;
	my.flare = on;
	my.lifespan = 100;
	my.alpha = 50;
	my.move = on;
	my.function = smoke_func;
}
```

Ok, après tout il n'est pas si mauvais que cela. Il a juste besoin d'être un peu amélioré. Tout d'abord le sprite fumée ressemble a de la m… Essayons ce lui de MISSION2.



Cela semble un peu sombre, mais il semble assez clair quand le noir est enlevé et que la transparence d'éclat (***flare***) est mise. Maintenant notre fumée semble presque réelle, voyez combien un graphisme peut changez un effet entier ? Mais elle flotte dans l'air puis BANG elle disparaît brusquement, essayons de mettre un fondu pour sa disparition.

```
function smoke_func()
{
	if(my.size < 800) { my.size += time * 0.1; }
	my.alpha -= 2.1 * time;
	if(my.alpha < 0) { my.lifespan = 0; }
	my.vel_x = my.vel_x / 1.2;
	my.vel_y = my.vel_y / 1.2;
}
```

Maintenant nous soustrayons 2.1 fois alpha à chaque frame et le faisons faire disparaître en fondu lorsqu'il a atteint sa Taille maximale. N'est-ce pas étonnant ? Essayez maintenant de coder votre propre effet de fumée à partir de zéro! Si vous avez besoin d'aide regardez la section de particule.

Gibbing!!

NDT : je n'ai pas trouvé de traduction satisfaisante pour ce mot. , cela pourrait être à priori éparpillement (scatter), éclaboussure (splash), mille morceaux etc…

Oh yay! Le merveilleux script de GIBBING!!! Il discute des modificationsque j'ai apporté au script de base qui se trouve dans war.wdl.
Si vous pensez qu'il est "grand", vous ne vous attendiez probablement pas à ce que je n'ai modifié que quelques lignes de code pour le faire. Mais il a fallu que je le fasse à la perfection pour obtenir le résultat souhaité.
La première chose que nous allons faire, ce sont quelques particules de sang.

```
bmap blood_map = <blood.bmp>;
function fade_func()
{
	my.alpha -= my.skill_x * time;
	if(my.alpha < 0) { my.lifespan = 0; }
}

function vec_blood(&vec)
{
	vec[0] = random(20)-10;
	vec[1] = random(20)-10;
	vec[2] = random(20)+10;
}

function particle_blood
{
	vec_blood(temp);
	vec_set(my.vel_x,temp);
	my.move = on;
	my.streak = on;
	my.flare = on;
	my.size = 10;
	my.alpha = 65;
	my.bmap = blood_map;
```

```
	my.lifespan = 35;
	my.skill_x = 2;
	my.gravity = 6;
	my.function = fade_func;
}
```

Hmmm... cela vous parait familier? Peut-être que scatter_speed vous donnerait un indice! C'EST LE CODE PARTICLE_SCATTER! J'ai juste changé quelques choses mineures.
my.bmap est à présent blood_map.
my.flare est maintenant valide.
Maintenant j'ai besoin d'utiliser cet effet pour sortir les particules du body_gib_action.

```
function _body_gib_action()
{
	// mets les particules à la même échelle que actor_scale
	if (gibnow >= gibmax) { ent_remove(my); end; }
	gibnow += 1;
	vec_scale(my.SCALE_x,actor_scale);
	my.enable_block = on;
	// Initialise la particule gibt
	my._speed_x = 15 * (RANDOM(10) - 5); // -125 -> +125
	my._speed_y = 15 * (RANDOM(10) - 5); // -125 -> +125
	my._speed_z = RANDOM(35) + 15; // 15 -> 50
	my._aspeed_pan = RANDOM(35) + 5; // 35 -> 70
	my._aspeed_tilt = RANDOM(35) + 5; // 35 -> 70
	my._aspeed_roll = RANDOM(35) + 5; // 35 -> 70
	my._force = 0;
	my.roll = RANDOM(180); // démarre avec une orientation aléatoire
	my.pan = RANDOM(180);
	my._force = -2;
	my.push = -1; // permettent à l'utilisateur/ennemis de passer à travers
	abspeed[0] = 5 * my._speed_x;
	abspeed[1] = 5 * my._speed_y;
	abspeed[2] = my._speed_z * time * 5;
	move(me,nullskill,abspeed); // les déplacent loin du point de Gib pour ne pas avoir
	d'éclaboussure de sang flottant
	// Anime la particule gibt
	my.skill9 = 50;
	while(my.skill9 > -75)
	{
		abspeed[0] = my._speed_x * time;
		abspeed[1] = my._speed_y * time;
		abspeed[2] = my._speed_z * time;
		my.pan += my._aspeed_pan * time;
		my.tilt += my._aspeed_tilt * time;
		my.roll += my._aspeed_roll * time;
		vec_scale(absdist,movement_scale); // met à l'échelle la distance absolue par
		movement_scale
		move(me,nullskill,abspeed);
		effect(particle_blood,bloodlevel * time,my.x,nullskill);
		if(bounce.z)
		{
			create(<blooda.bmp>,my.x,blood_spat);
			my._speed_z = -(my._speed_z/2);
			if(my._speed_z < 0.25)
			{
				my._speed_x = 0;
				my._speed_y = 0;
				my._speed_z = 0;
				my._aspeed_pan = 0;
				my._aspeed_tilt = 0;
```

```
                                        my._aspeed_roll = 0;
                                }
                                gibvar = bloodlevel * splatlevel;
                                effect(particle_blood,gibvar * time,my.x,nullskill);
                                // my.skill9 -= 100;
                        }
                        my._speed_z -= 2;
                        my.skill9 -= 1;
                        wait(1);
                }
                my.passable = on;
                // Fondu
                my.transparent = on;
                my.alpha = 100;
                while(1)
                {
                        my.alpha -= 5*time;
                        if(my.alpha <=0)
                        {
                                // remove
                                gibnow -= 1;
                                ent_remove(my);
                                return;
                        }
                wait(1);
                }
}
```

J'ai ajouté un ***effect*** dans la boucle while pour que le sang goutte constamment. Il y a également unn ***effect*** pour émettre plus de sang lorsqu'il rebondit (***bounces)***. Aussi "create(<blooda.bmp>,my.x,blood_spat);" a été ajouté pour laisser des décalcomanies sur le mur en utilisant la fonction blood_spat.

```
function blood_spat()
{
        scan_sector.pan = 360;
        scan_sector.roll = 360;
        scan_sector.tilt = 360;
        my_angle.pan = my.pan;
        my_angle.tilt = my.tilt;
        scan(my.x,my_angle,scan_sector);

        if(bullet_hole_counter <= kmaxbullethole && you != player)
        {
                bullet_hole_counter += 1; // incrémente le compteur de trou de balle
                my.transparent = on;
                my.passable = on;
                my.oriented = on;
                vec_to_angle(my.pan,normal); // tourne vers la cible normal
                waitt(160); // temps que reste le sang avant de disparaître, attend le temps fixé par
                waitt(160), plutôt que 160 frames avec wait(160)
                while(my.alpha > 0) // fondu
                {
                        my.alpha -= time;
                        waitt(1);
                }
                bullet_hole_counter -= 1; // décrémente le compteur de trou de balle
                ent_remove(my);
        }
        else
        {
                ent_remove(my);
```

```
        }
    }
```

En utilisant la variable ***kmaxbullethole*** pour le maximum de balle dans le niveau j'ai également inclus les éclaboussures de sang comme trou de balle. C'est très similaire à la fonction "trou de balle"It is very similar to the bullet hole function. Mais plutôt que d'utiliser wait(160) j'utilise waitt(160). De cette façon les éclaboussures restent 160 ticks, un temps fixe, qui est mieux que 160 frames, un temps variable.

Avancé

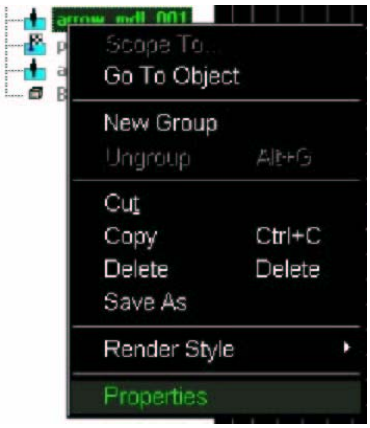
Maintenant que vous avez examiné l'atelier dans son entier vous pouvez vouloir des choses plus avancées. C'est ce qu'il vous reste à faire en attendant mon prochain atelier. J'espère que vous avez appris beaucoup de cet atelier et que vous êtes maintenant capable de faire vos propres scénarios. Je verrai votre contributions sur le Forum des utilisateur!
A bientôt,
Nizzy

Annexe A

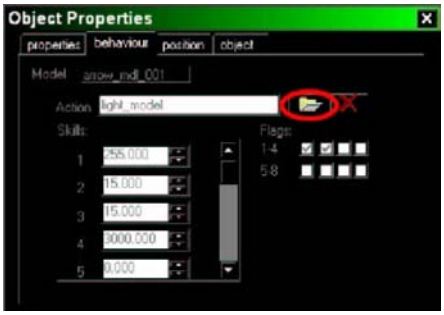
Pour utiliser ces effets, incluez (***include)*** effects.wdl dans le script principal de votre jeu, assignez les actions correspondantes aux entités désirées dans WED, et mettez les variables et les drapeaux pour personnaliser les effets.

Inclure effects.wdl
Créer un nouveau niveau. Puis File > Map Properties. Cliquez sur le bouton new button à droite du champ script pour créer un nouveau script. Puis ouvrez le fichier dans wordpad ou n'importe quel éditeur WDL. Ajoutez “include <effects.wdl>;” au début de la liste des include. Maintenant les effets se trouvent dans la liste du menu action.

Assigner l'action
Ouvrez WED et créez un nouveau niveau ou ouvrez un niveau existant. Ajoutez une entité, clic droit sur cette entité dans l'explorateur d'objets et sélectionnez Properties.



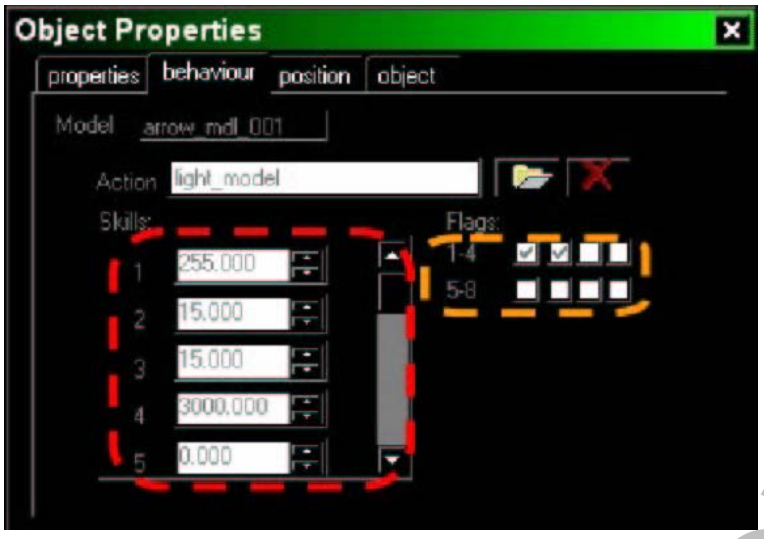
Cela ouvre la fenêtre des propriétés, cliquez sur l'onglet comportement (behavior) et cliquez sur l'icône ouvrir à côté du champ action.



Cela affichera la liste des actions *incluses* dans vos scripts du jeu.

Sélectionnez l'effet que vous désirez et cliquez sur OK.

Mainteant vous pouvez personnaliser l'effet en utilisant les variables *skills* et les drapeaux *flags*.



Notez : le scénario peut différer des effets créés dans cet atelier, mais il est fait pour un but plus général. Les effets de cet atelier ont été personnalisés donc vous pouvez apprendre comment les personnaliser pour adapter votre jeu.

Light (lumière)
Flag1 = flame-light
Skill1-3 = couleur de la lumière rouge – vert - bleu (155,55,55)
Skill4 = distance de la lumière(300)
Skill5 = intensité de la flamme (100)
Flag2 = lumière cassée
Skill1-3 = couleur de la lumière rouge – vert - bleu (155,55,55)

Alpha_glow (rougeoiment alpha)
Skill1 = min. alpha transparence (25)
Skill2 = max. alpha transparence (75)
Skill3 = vitesse de rougeoiment (5)

Test_gib
Aucune variable ou drapeau n'est utilisé pour test_gib

Fire1
Skill1 = intensité (8)

Starsphere
Aucune variable ou drapeau n'est utilisé pour la sphère de ciel

Window (fenêtre)
Aucune variable ou drapeau, attaché à un sprite de fenêtre et il volera en éclat quand il sera touché par un tir dans le jeu .

Gem (Gemme)
Skill1 = valeur alpha (100)

Smoker
Aucune variable ou drapeau, utilisé pour court panache de fumée.

Appendix B
Crédits-

Traduction française : Alain Brégeon
Game engine design- Conitec
WDL Script- Dan Niezgocki
Graphics- Dan Niezgocki
Graphics- Conitec
Particle help etc.- Gaehh
Special thanx- All those who gave ideas on the user forum, you are who made this possible!
Special thanx- Doug Poston, great support and help.
Very Special thanx- JCL, for your time, patience and everything else.