3D GameStudio

Atelier sur les particules

par Wolfgang Knecht / Novembre 2001

Cher lecteur

Bienvenue à l'atelier sur les particules! Cet atelier est fait pour vous aider à réaliser vos propres effets de particules. Cet atelier est basé sur le nouveau système de particules qui qui est apparu avec la dernière mise à jour (5.12).

Comme pour les ateliers précédents, cet atelier complète le reste de la documentation mais il ne la remplace pas. Je suppose que vous connaissez les bases de WDL et de WED.

J'espère que vous aimerez cet atelier. Les questions où les commentaires doivent être envoyés à millennium-arts@gmx.at et j'essayerais de vous aider.
Traduit en français par Alain BREGEON.

Wolfgang "Knex|MA" Knecht

Comment travailler avec cet atelier
Concernant les particules tous les dispositifs ne sont pas implémentés dans chaque édition (ex. alpha, flare, bright, ...). Aussi j'ai décidé de faire une différence de code pour l'édition basse et haute. Si un code fonctionne avec une édition supérieure, il sera étiqueté de cette façon: H
Réciproquement un code pour une édition inférieure sera étiqueté de cette façon : L
Et si le code est pour toutes les éditions alors il sera étiqueté comme ceci : A

Une étiquette est valide jusqu'à la prochaine étiquette.

ex.: Si vous possédez l'édition standard suivez ceci L et A Si vous possédez l'édition professionnelle suivez H et A

L'intégralité du code pour les éditions inférieures se trouve dans pfxlow.wdl.
L'intégralité du code pour les éditions inférieures se trouve dans pfxhigh.wdl.

Assurez vous d'avoir la dernière mise à jour de 3D Gamestudio
Il est important d'avoir la dernière mise à jour, parce que depuis la version 5.12 il y a un nouveau système de particule aussi chaque commande se référant aux particules nécessite au moins la version 5.12 ou au dessus.

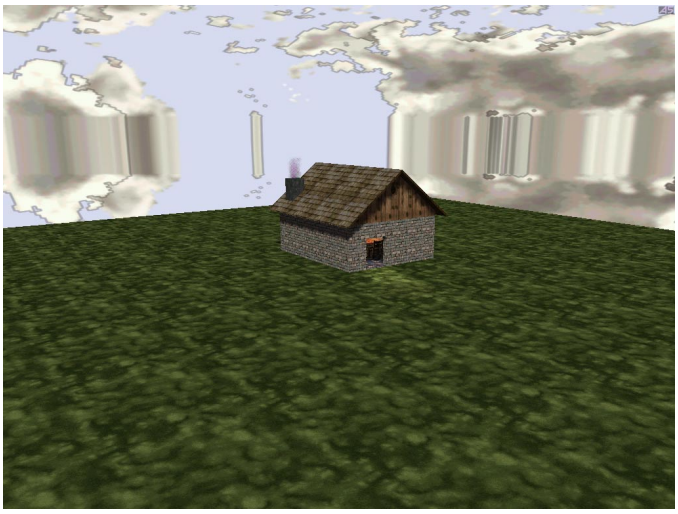
Preparez votre espace de travail
Créez un répertoire nommé "Particle Workshop" dans votre répertoire GStudio. Ce répertoire contiendra tous les fichiers dont vous aurez besoin pour cet atelier.

Décompressez le contenu de PARTICLEFX.ZIP dans ce répertoire. Ce répertoire doit maintenant contenir les fichiers suivants :

fire.pcx
fire.wmb
particle.wmp
pfxhigh.wdl
pfxlow.wdl
rain.pcx
smoke.pcx
smoke.wmb
spell.pcx
warlock.mdl

Créez un niveau

Dans cet atelier nous n'expliquerons pas comment créer un niveau pas à pas, parce que l'architecture du niveau n'est pas importante pour nos effets de particule. Pensez juste à faire quelque chose qui ressemble à un toit dans votre niveau car il nous servira plus tard pour l'effet pluie. Vous devez également avoir une entité joueur dans votre niveau avec une action de mouvement (par exemple. **`player_walk`**). Je pense que l'idéal serait que vous utilisiez particle.wmp qui est fourni avec cet atelier. Créez un nouveau fichier WDL que vous nommerez particle.wdl pour ce niveau (File/Map Properties/New).



Que sont les particules ?

Les particules sont de petites images bitmap 2D qui apparaîssent par millier. Elles sont utilisées pour des effets comme de la fumée, du feu, des tornades, de la pluie et tous les effets imaginables. Peut-être vous pensez pouvoir utiliser des sprites à la place des particules. Dans certains cas c'est vrai, mais les particules sont plus rapides que les sprites. Une autre différence entre les sprites et les particules, les particules n'ont pas de détection de collision.

OK, on démarre

Pour commencer nous allons créer un simple effet de fumée. Puis nous verrons comment faire du feu. Un autre effet intéressant est la pluie. Après ces effets, nous apprendrons à créer de jolis effets de 'sort'

Nous allons concevoir ce code afin qu'il puisse être ajouté à différents projets avec un minimum de modifications. Tout ceci nous amène à faire un fichier WDL séparé qui pourra être inclu dans nos différents projets.

Nous allons démarrer en créant un fichier "pfx.wdl" avec un éditeur dfe texte simple. Ce fichier dans le répertoire de votre projet ou vous pouvez créer un nouveau dossier dans lequel vous rangerz vos propres fichiers template. La première chose à faire est d'ajouter un bloc de commentaires au tout début de notre code afin que si nous revoyons ce code dans quelques mois nous sachions pourquoi ce code est fait.

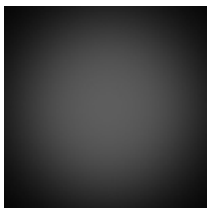
```
////////////////////////////////////////////////
// Fichier: pfx.wdl
// WDL code effets de feu, fumée, pluie et sorts
////////////////////////////////////////////////
```

Fumée

```
/******************
* Effet de fumée *
*****************/
```

```
bmap smoke_parmap=<smoke.pcx>;
```

D'abord nous devons définir notre bitmap, qu'utilisera chaque particule de fumée.



smoke.pcx

Je ne suis pas un bon artiste 2D, aussi j'utilise des images très simples pour mes effets. Si vous voulez, vous pouvez faire de meilleures images pour des résultats meilleurs.

```
function smoke_property();    // prototype
```

```
function smoke_effect();      // prototype
```

Puis nous avons défini les fonctions dont nous aurons besoin pour notre effet.

```
function smoke_effect() {
```

smoke_effect() est la fonction qui place les valeurs de début pour chaque particule. Dans cette fonction nous gérons les propriétés comme la vitesse, la couleur, la transparence etc.

```
my.vel_x=0;
my.vel_y=0;
my.vel_z=random(3);
```

Avec **my.vel_x**, **my.vel_y** et **my.vel_z** nous gérons la vitesse des particules.
Normallement la fumée a la caractéristique de se déplacer vers le haut sur l'axe Z. Aussi nous donnons à **my.vel_z** une valeur aléatoire entre 0 et 3. Si vous voulez vous pouvez changer cette valeur pour que la fumée se déplace plus ou moins vite. Peut-être pensez-vous à multiplier **my.vel_z** par **time**. Ne le faites pas, **vel_x**, **vel_y** et **vel_z** sont automatiquement relatif à **time**.

```
my.move=on;
```

my.vel_x, **my.vel_y** et **my.vel_z** n'ont aucun effet si **my.move** est mis à off.

```
my.size=15;
```

my.size détermine comme son nom l'indique, la taille des particules. Vous pouvez tester différentes valeurs. La valeur par défaut de **my.size** est 4 mais la fumée ne semble pas bonne si vous voyez des petites particules. Aussi une plus grande valeur comme 15 donne de meilleurs résultats.

```
my.bmap=smoke_parmap;
my.flare=on;
my.alpha=10;
```

Vous souvenez-vous de **smoke_parmap** que nous avons défini auparavant? Nous allons à présent l'utiliser. Avec **my.bmap** nous assignons **smoke_parmap** à la particule. **my.flare** mes un effet d'éclat à la bitmap de lae particle. Ce qui fera que les zones sombres seront plus transparentes que les claires. Avec **my.alpha** nous jouons sur l'intensité de la transparence. Si **my.alpha** est mis à 100, la particule n'est généralement pas transparente. Si nous mettons 0 la particule est totalement transparente et nous ne la verrons désormais plus.

```
     my.lifespan=64;
```

my.lifespan donne la durée de vie de la particule en ticks. Ici la particule de fumée durera 4 secondes (64/16 secondes). La valeur par défaut de **my.lifespan** est 80 (5 seconds). my.lifespan décompte continuellement. Si elle arrive à 0 la particle sera enlevée. Si vous voulez une durée de vie plus ou moins longue pour cette particule, il suffit de modifier cette valeur.

```
     my.red=128;
     my.green=128;
     my.blue=128;
     my.transparent=on;
```

my.red, **my.green** et **my.blue** gère la couleur (RBV) de la particule. Si toutes les valeurs sont à 128 la couleur de la particule sera grise comme de la fumée.

```
     my.function=NULL;
}
```

Après avoir mis toutes ces valeurs, nous n'avons plus besoin de rien aussi nous mettons la fonction de la particule à **NULL**. La particule existera jusqu'à ce que my.lifespan soit à 0

```
     my.function=smoke_property;
}
```

Maintenant nous assignons une nouvelle fonction à la particule pour qu'elle ne soit pas née pour rien.
Maintenant nous devons coder les choses qui changent à chaque répétition de la fonction **smoke_property()**.

```
function smoke_property() {
my.alpha-=0.1*time;
     if (my.alpha<=0) {
          my.lifespan=0;
     }
}
```

Nous réduisons **my.alpha** à chaque répétition de la fonction pour disparaître en fondu. Ne pas oublier le facteur **time** pour éviter d'avoir des effets différents sur différents ordinateurs! Si la particule a disparu complètement **my.lifespan** est mis à 0 ce qui aura pour effet de l'enlever.

OK, notre fonction pour la particule est créée. A présent nous voulons une fonction qui créra toutes les particules. L'action **smoke** pourra être ajoutée à n'importe quelle entité.

```
action smoke {
     while (1) {
          temp.x=my.x+random(32)-16;
          temp.y=my.y+random(32)-16;
          temp.z=my.z;
```

Je suppose que vous savez à quoi sert **while (1)**. Si non, elle est nécessaire pour faire une boucle sans fin (la fumée monte tout le temps).
Nous utilisons le tableau **temp** pour la position de départ de chaque particule. La position de départ sera par rapport à l'entité **my** – avec une distance maximale de+/- 16 quants. Si vous voulez changer cette distance sur l'axe des x – ou des y – changez juste les valeurs 32 et16 comme vous voulez(si la distance doit être symmétrique, la première valeur doit être deux fois la deuxième.)

```
          effect(smoke_effect,5*time,temp,temp);
          wait 1;
     }
}
```

Avec **effect** nous créons les particules. Dans l'ancien système de particules nous utilisions l'instruction **emit** pour faire la même chose.

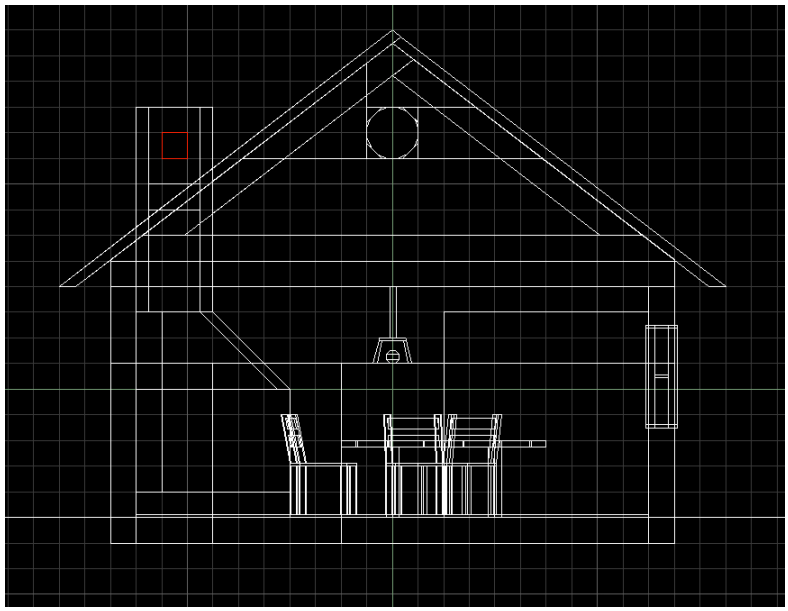
Le premier paramètre (**smoke_effect**) met la fonction pour chaque particule. Normallement la fonction paramètre doit toujours être avant l'instruction effect. Autrement **effect** ne trouvera pas la fonction et il y aura une erreur.
Mais comme au début de notre effet fumée, nous avons déclarés les prototypes de nos fonctions, alors ça fonctionnera également si la fonction particule est après l'instruction effect.
Le deuxième paramètre (5***time**) indique combien de particules doivent être créées. Dans ce cas **5*time** (chaque 1/16 seconde 5 nouvelles particules seront créées) est suffisant. Si vous voulez une fumée plus mince, vous devez mettre une valeur plus petite.
Le troisième paramètre (**temp**) donne les positions de départ des particules. Ce sont les **temp.x**, **temp.y** et **temp.z** que nous avons vu au-dessus.
Le dernier paramètre (**temp**) donne la vitesse de départ. Dans notre effet nous n'avons pas besoin de cette valeur mais il est nécessaire de mettre une valeur sinon erreur) aussi nous mettons la variable **temp**. Nous n'en avons pas besoin car souvenez-vous nous avons assigné une autre valeur de vitesse à notre fonction **smoke_effect**.
`Wait 1` permet une pause pour les autres fonctions.

Comment utiliser la fumée

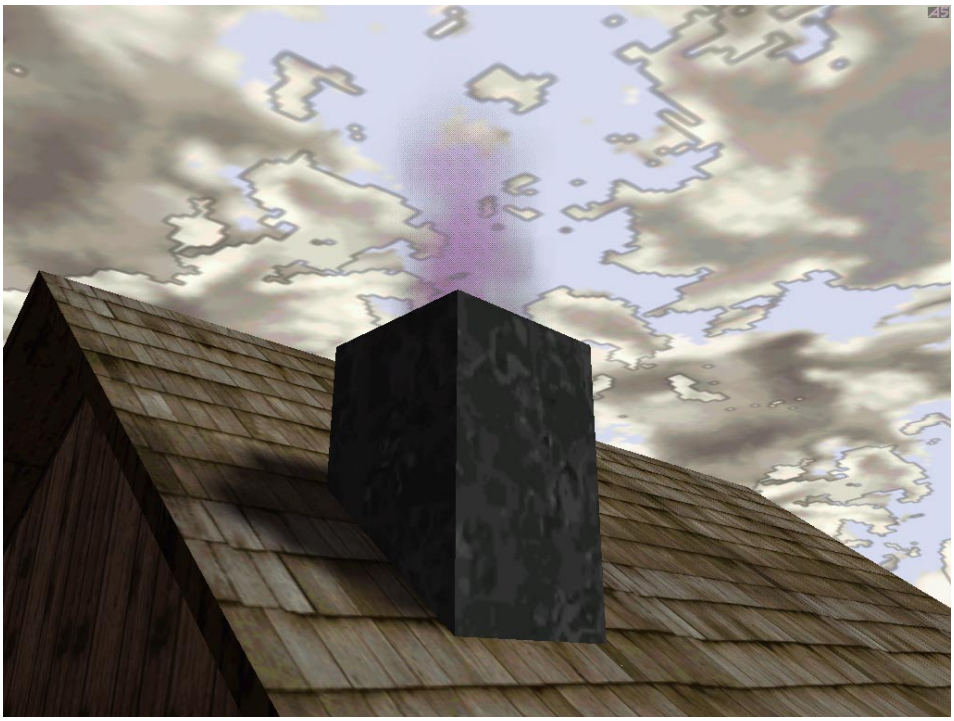
L'utilisation de l'effet fumée est très facile. Vous devez juste inclure pfx.wdl à la suite des autres includes dans le fichier WDL de votre jeu.

```
include <pfx.wdl>;
```

Puis ajoutez une entité dans le niveau ou vous souhaitez voir la fumée. Comme entité vous pouvez utiliser smoke.wmb par exemple.



Rendez cette entité invisible et assignez lui l'action **`smoke`**. Compilez et exécutez votre niveau. Voici le résultat.



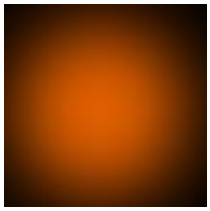
Feu

Si vous utilisez une version inférieure de 3DGS, le feu ne sera pas génial avec les particules. Les sprites sont mieux. Mais pour ceux qui veulent quand même essayer il y a un code alternatif de disponible, bien entendu.

```
/**************
* Fire effect *
**************/
```

```
bmap fire_parmap=<fire.pcx>;
```

Pour commencer nous définissons une image bitmap de feu comme nous l'avons fait fait pour l'effet de fumée



fire.pcx

```
function fire_effect();
function fire_property();
```

Puis nous définissons les deux fonctions que nous utiliserons.

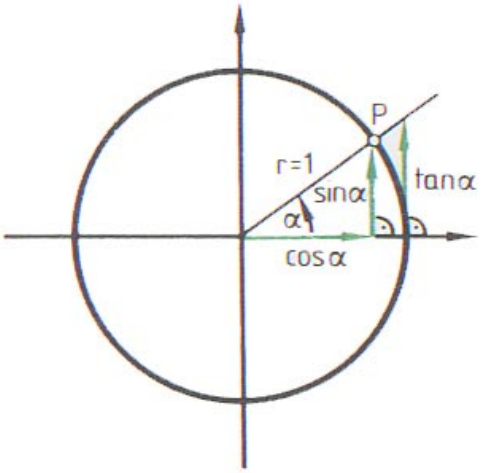
Pour le feu nous allons devoir faire un peu d'algèbre (si vous n'êtes pas bon en algèbre – ne soyez pas désolé, j'ai du interroger mon grand frère aussi :)).

```
function fire_effect() {
     my.x+=random        (30*(sin(total_ticks*200)        *
     cos(total_ticks*50)));

     my.y+=random        (30*(sin(total_ticks*200)        *
     sin(total_ticks*50)));
```

Ce n'est pas aussi compliqué que ça en a l'air. Analysons ceci :
my.x, **my.y** et **my.z** sont la position de départ (au milieu du feu) de chaque particule. Mais le feu n'est pas seulement un point, aussi nous devons calculer des coordonnées aléatoires autour du milieu du feu. Si vous connaissez l'algèbre vous savez que les coordonnées d'un simple cercle sont calculées comme suit :

x = rayon * sin (angle)
y = rayon * cos (angle)



Dans notre cas, angle est **total_ticks** multiplié par 50
total_ticks est le nombre de ticks qui sont passés depuis que nous lancé notre jeu. Aussi **total_ticks** compte continuellement comme nous avons besoin. 50 doit l'accélérer. Maintenant nous avons besoin d'un rayon, qui devient alternativement plus petit ou plus grand. Pour obtenir quelque chose comme cela, **sin** est parfait. **sin** est toujours un nomber entre –1 et 1
Donc notre rayon ressemble à cela :

rayon = 30*sin(total_ticks*200)

Ainsi le rayon sera toujours entre -30 et 30.

Si nous remplaçons la fonction pour les coordonnées du cercle ci-dessus avec la fonction que nous venons de trouver, les coordonnées d'un point qui se déplacerait en spirale serait :

x = 30*sin(total_ticks*200) * sin(total_ticks*50)
y = 30*sin(total_ticks*200) * cos(total_ticks*50)

A la fin nous devons juste ajouter un nombre aléatoire

```
my.size=15;
my.vel_x=0;
my.vel_y=0;
my.vel_z=2+random(2);
my.move=on;
```

Nous devons mettre les valeurs de paramètres, nous connaissons à présent depuis les effets de fumée vus auparavant.

```
my.bmap=fire_parmap;
my.flare=on;
my.bright=on;
my.alpha=25;
```

Nous connaissons tous ces paramètres. **my.bright** est nouveau.
my.bright est très utile pour le feu puisqu'il donne à la particule quelque chose qui ressemble à un rougeoiment. S'il y a des particules qui se chevauchent elles deviennent plus brillante
Regardez un feu, vous noterez qu'il est plus brillant au milieu qu'à l'extérieur.
C'est exactement pour cet effet que nous avons besoin de **my.bright**.

```
     my.lifespan=16;
     my.red=255;
     my.green=255;
     my.blue=0;
     my.transparent=on;
```

Nous connaissons déjà les résultats que donnent ces paramètres. La durée de vie des particules sera de 1 seconde (16/16 secondes). Avec les valeurs de **my.red**, **my.green** et **my.blue,** la couleur obtenue sera jaune. Comme pour la fumée le feu est également transparent.

```
     my.function=fire_property;
}
```

'Juste-né' – la fonction est faite.

```
function fire_property() {
```

```
     my.alpha-=1*time;
     if (my.alpha<=0) {
          my.lifespan=0;
     }
}
```

C'est pratiquement identique à l'effet de fumée. La seule différence est que le fondu est un petit peu plus rapide.

```
     my.red-=5*time;
     my.green-=15*time;
}
```

Si vous observez un feu, vous noterez qu'il est plus intensif au milieu.
Les couleurs changent. Dans le mileir il est jaune (quelques fois presque blanc) et à l'extérieur il devient rouge foncé.
Aussi nous ajustons la couleur des particules à chaque répétiton de la fonction.

Vous pouvez changer ces valeurs (5 et 15) un peu si vous voulez. J'en ai essayé d'autres et j'estime que le feu est plus vrai avec celles que j'ai retenues.
Vous pourriez penser que nous devrions vérifier si **my.red** et **my.green** ont bien une valeur entre 0 et 255. Ce n'est pas faux mais inutile car le moteur le fait automatiquement pour nous.

OK, la première partie de notre effet 'feu' est terminée. La seconde partie n'est pas difficile.

```
action fire {
     While (1) {
```

Nous savons ce qui arrivera avec l'instruction ci dessus. Comme pour l'effet 'fumée' nous utilisons **temp** pour notre vecteur de départ. Bien entendu nous pourrions utiliser une autre variable vecteur, mais dans notre cas, **temp** est suffisant
Une telle fonction pourrait ressembler à ceci :

```
          temp.x=my.x;
          temp.y=my.y;
          temp.z=my.z;
```

Nous mettons **temp.x**, **temp.y** et **temp.z** à la position de départ.

```
          effect(fire_effect,20*time,temp,temp);
          Wait 1;
     }
}
```

L'instruction effect n'est pas nouvelle pour nous. Maintenant nous créons tous les 1/16 de seconde 20 particules. Si vous voulez faire un feu plus ou moins intense, vous devez changer cette valeur.

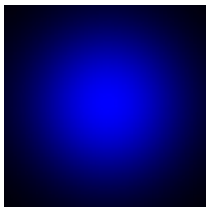
Comment utiliser le feu ?
Le feu s'utilise de la même façon que la fumée. Vous devez juste inclure le fichier pfx.wdl dans le fichier wdl de votre jeu, ajouter une entité à l'emplacement du feu et lui donner l'actio 'fire'. Comme entité vous pouvez utiliser du bois (ex. fire.wmb) ou ce que vous voulez.

Pluie

```
/**************
* Effet de pluie *
**************/
```

Avec cet effet pluie il est facile d'ajuster l'effet comme vous voulez. Vous pouvez agir sur l'intensité de la pluie et la direction / vitesse du vent.

```
bmap rain_parmap=<rain.pcx>;
```



rain.pcx

```
function rain_effect();
function rain_property();

var raining=0;
```

Avec cet effet pluie nous pouvons basculer la pluie en oui / non, aussi nous avons besoin d'une variable pour nous dire s'il pleut. Si **raining** est à 0 il ne pleut pas. Si **raining** est à 1 il pleut.

```
function rain_effect() {
     my.vel_z=(-12.5-random(5));
my.x=camera.x+random(1500)-750;
     my.y=camera.y+random(1500)-750;
```

Maintenant nous donnons la position de départ de la particule. Nous ne voulons pas créer de particules sur tout le niveau. La performance en souffrirait. Aussi nous allons créer de lapluie uniquement autour de la camera (+/- 750 quants dans les axes x et y relatifs à la position de la caméra). Le joueur ne le saura pas.
Si vous voulez, vous pouvez régler l'étendue de la pluie selon vos besoins.

```
     my.skill_x=my.x;
     my.skill_y=my.y;
     my.skill_z=player.z+player.min_z;
```

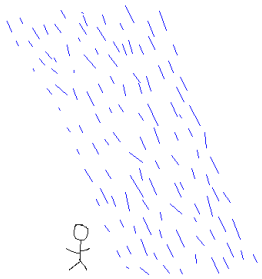
Nous mettons **my.skill_x**, **my.skill_y** et **my.skill_z** à la position où nous voulons que la particule de pluie disparaîsse si elle n'a pas touché un bloc auparavant. Nous aurons besoin de ces variables pour une fonction **trace** bientôt.

```
my.z=camera.z+500;
my.move=on;
```

Il commencera à pleuvoir 500 quants au dessus de notre tête. Si votre niveau contient une construction haute (par exemple un hall), il faudra faire démarrer la pluie à plus de à plus de 500 quants au dessus de la caméra parce que si le hall est plus haut que 500 quants alors il pleuvra dans le hall. Mais pour un niveau normal, 500 quants est largement suffisant.

```
my.x-=33*my.vel_x;
my.y-=33*my.vel_y;
```

Si le vent est très fort, il se pourrait qu'il ne pleuve pas à la position de la caméra mais qu'il pleuve à côté de vous. Aussi nous corrigeons cela en mettant la position de départ en fonction de la vitesse du vent.



La pluie passe le joueur

C'est beaucoup mieux

```
trace_mode = ignore_me;
trace(my.x,my.skill_x);
my.skill_z=target.z;
```

C'est la fonction **trace** dont on a parlé plus haut. Ici nous vérifions s'il y a quelque chose entre la position de départ de chaque particule et le sol. **My.skill_z** donne la coordonnée Z où la particule frappera quelque chose.

```
my.bmap=rain_parmap;
my.flare=on;
```

Vous devez savoir ce que cela signifie. Sinon reportez-vous à notre premier effet ('fumée').

```
     my.size=2;
     my.red=0;
     my.green=0;
     my.blue=128;
```

Nous mettons la taille et la couleur.

```
     my.function=rain_property;
}
```

Une autre fonction 'juste-né' est faite.

```
function rain_property() {
     if (my.z<=my.skill_z) {
          my.lifespan=0;
     }
}
```

Les particules sont enlevées si elles touchent quelque chose. Comme cela il ne peut pas pleuvoir dans une maison, mais nous pouvons voir la pluie par la fenêtre.

OK, nous avons terminé la fonction particule , continuons avec la fonction qui crée les particules.

```
function rain(vel_x,vel_y,intensity) {
```

Si la fonction pluie est appelée, elle obtient 3 paramètres. Avec les 2 premiers nous gérons la vitesse du vent, avec le troisième nous ajustons l'intensité de la pluie.

```
     if (intensity==0) {intensity=20;}
```

Si **intensity** n'est pas mis, il ne pleuvra pas. Aussi nous le mettons à 20 qui est une valeur raisonnable.

```
     if (raining==1) {
          raining=0;
     } else {
          raining=1;
     }
```

Vérifions s'il pleut ou pas. S'il pleut, cela s'arrête, sinon ça démarre.

```
     while (raining==1) {
          temp.x=vel_x;
          temp.y=vel_y;
```

Si **raining** est à 1, il pleut, aussi nous mettons **temp.x** et **temp.y** à la direction/vitesse du vent. nous en avons besoin pour le remettre à la fonction particule(vous vous rapellez l'instruction dans la fonction 'juste né'?)

```
            effect(rain_effect, intensity*time, temp, temp);
            wait 1;
      }
}
```

effect crée nos particules. Comme vous savez, le premier paramètre met la fonction particule, le second le nombre de particules qui seront créées,
le troisième, leur position de départ (nous n'avons pas besoin de ce paramètre dans notr cas car nous gérons la position de départ à l'intérieur de la fonction particule) et le donne la vitesse de départ (dans notre cas utilisé pour le vent).

Comment utiliser la pluie ?
Juste inclure le fichier pfx.wdl dans le fichier WDL de votre jeu (particle.wdl) et d'appeler la fonction pluie si vous voulez de la pluie.

```
rain();
```

Une pluie simple qui tombe tout droit .

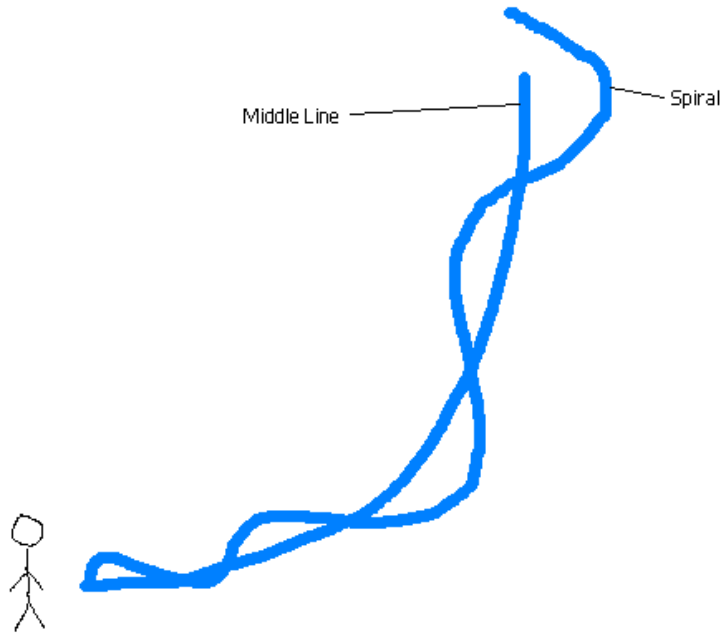
```
rain(20,10,50);
```

Pluie lourde de côté.

Si vous voulez stopper la pluie, il faut juste appeler **`rain()`** une autre fois.

Sort ou charme

Avant tout, si nous voulons créer un effet de charme nous devons savoir à quoi il doit ressembler.



Ne vous ai-je pas dit que j'étais un très mauvais artiste ?

L'effet doit partir de la position du joueur et se déplacer vers le haut vers le ciel. Il doit y avoir une ligne médiane et autour de cette ligne, il doit y avoir une spirale. Les particules doivent disparaître en fondu et tomber par terre. Le rayon de la spirale doit augmenter continuellement.
C'est embrouillant n'est-ce pas ? Si vous ne comprenez pas ce que je dit, continuons et et vous comprendrez.

```
/***************
* Effet charme *
***************/

var spell_angle=0;
var spell_radius=0;
var spell_time=0;
var spell_existing=0;
```

Nous avons besoin de quelques variables. **`spell_angle`** donne la direction du charme. Si le joueur regarde vers le sud, le charme doit apparaître au sud. Si le joueur regarde à l'est, le charme doit apparaître à l'est.

spell_radius met le rayon de la spirale autour de la ligne médiane du charme. **spell_radius** augmente continuellement.

spell_time met l'âge ode l'effet charme en ticks. L'effect dure 50 ticks.

Et nous avons besoin de **spell_existing** si un effet de charme existe déjà à ce moment. Si **spell_existing** est à 1 (un autre charme existe), nous ne pourrons démarrer notre charme tant que **spell_existing** ne sera pas à 0.

```
bmap spell_parmap=<spell.pcx>;
```

spell.pcx

```
function spell_effect();
function spell_property();

function spell_effect() {
     my.size=10;
my.vel_x=0;
     my.vel_y=0;
     my.vel_z=0;
     my.move=on;

     my.bmap=spell_parmap;
     my.flare=on;
     my.bright=on;
     my.alpha=25;

     my.lifespan=16;
     my.red=0;
     my.green=150;
     my.blue=255;
     my.transparent=on;

     my.function=spell_property;
}
```

Nous devons aussi mettre les paramètres de début dans cet effet. Si vous n'êtes pas sûrs d'une instruction ci-dessus, regardez les effets précédents.

```
function spell_property() {
```

```
     my.alpha-=1*time;
     if (my.alpha<=0) {
          my.lifespan=0;
     }
```

Disparition en fondu sur la particule. Elle est enlevée dès qu'elle n'est plus visible.

```
     my.gravity=2;
}
```

my.gravity met la gravité (la même que **my.vel_z * (-1)**). Chaque partiucle se déplacera de haut en bas pendant sa vie entière.

Ca n'était pas trop difficile. Maintenant voyons une partie plus difficile.

```
function spell() {
     if (spell_existing==0) {
          spell_existing=1;
          spell_time=0;
          spell_angle=0;
          spell_radius=0;
          spell_angle=player.pan;
```

Pour commencer nous vérifions s'il n'existe pas un autre charme. S'il n'y en a pas (`spell_existing=0`), nous en créons un nouveau. Puis nous mettons nos variables à leurs valeurs de départ. Nous devons conna^tre la direction dans laquelle regarde le joueur, pour être sûr que le charme démarre dans cette direction.

```
          while (spell_time<50) {
               temp.x=player.x;
               temp.y=player.y;
               temp.z=player.z;
```

La durée de vie du charme est de 50 ticks (environ 3 secondes). La position de départ est la position du joueur.

```
               temp.x+=sin(90 - spell_angle) * cos (min
                    (360,spell_time*2+270))*500;
               temp.y+=cos(90 - spell_angle) * cos (min
                    (360,spell_time*2+270))*500;
```

```
temp.z+=sin (min (360, spell_time * 2
     +270))*500+500;
```

Le vecteur **temp** est à présent un point le long d'un quart de cercle relatif à la durée de vie du charme -> **spell_time**.

```
effect(spell_effect,1*time,temp, temp);
```

Maintenant nous créons des particules sur ces points le long de la courbe.
Ca donne ceci :

Cela forme la ligne médiane du charme. Autour de cette ligne nous allons créer des particules le long d'une spirale.

```
temp.x+=sin(- spell_angle) * cos
     (spell_time*50)*spell_radius;
temp.y+=cos (- spell_angle) * cos
     (spell_time*50)*spell_radius;

temp.z+=cos(- spell_time * 2) * sin
     (spell_time*50)*spell_radius;
temp.x+=sin(90-spell_angle)*sin(-spell_time
     *2)*sin(spell_time*50)*spell_radius;
temp.y+=cos(90-spell_angle)*sin(-spell_time
     *2)*sin(spell_time*50)*spell_radius;
```

Nous devons tenir compte que la spirale doit être normale à la ligne médiane.
Aussi le calcul de la position de chaque particule le long de la spirale peu paraître un peu embrouillé. Si vous ne comprenez pas les instructions sinus et cosinus je vous suggère de consulter un livre de mathématiques. Ces instructions sont quelques fois très utiles lorsque nous créons des jeux d'ordinateur. Vous pouvez produire de jolis effets avec cette connaissance.

```
effect(spell_effect,1*time, temp, temp);
```

effect crée les particules le long de la spirale.

```
                spell_radius+=1*time;
                spell_time+=time;
                wait 1;
        }
```

spell_radius et **spell_time** augmentent continuellement.

```
        rain(0,0,0);
        spell_existing=0;
    }
}
```

Après que l'effet est fini, nous pouvons mettre une action/function pour définir ce qui arrivera si l'effet charme est appelé. Dans notre effet nous appelons la fonction **rain**. Donc nous pouvons faire pleuvoir (ou arrêter la pluie)) en appelant ce charme.
La variable **spell_existing** est mise à 0. Ce qui signifie qu'il n'y a pas de charme à ce moment.

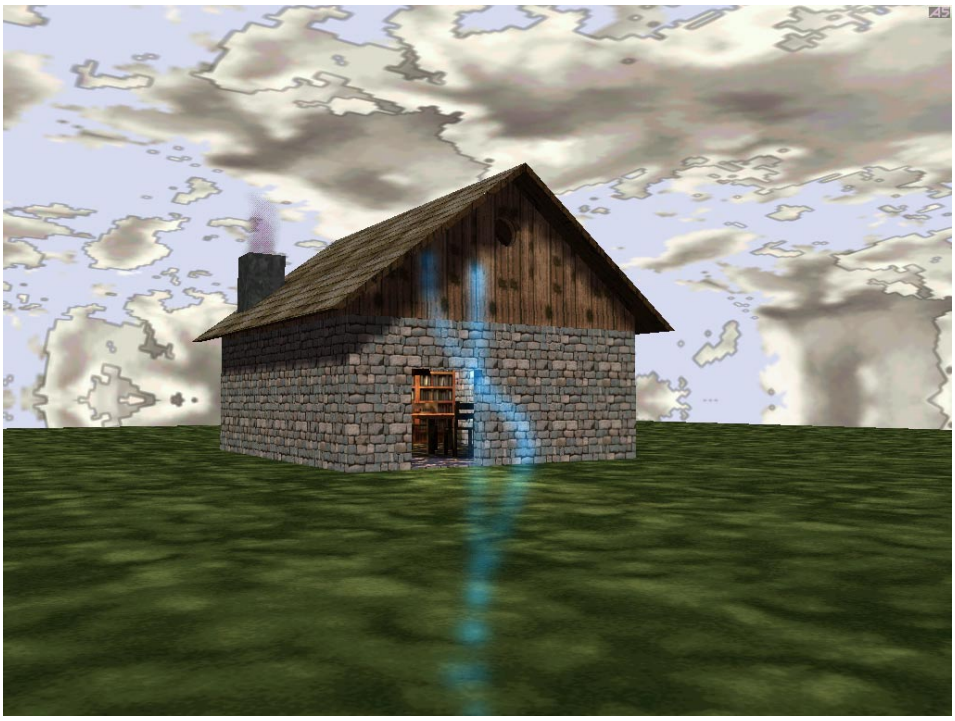
Comment utilser le charme ?
L'utilisation du charme est un peu différent des autres. Vous devez inclure pfx.wdl APRÈS movement.wdl dans le fichier WDL de votre jeu (particle.wdl).
La raison de ceci est que nous utilisons le pointeur **`player,`** qui est défini dans movement.wdl, dans notre fonction. Aussi si vous utilisez votre propre action de mouvement vous devez créer un pointeur **`player.`**

Pour assigner une touche à la fonction spell, utilisez simplement l'instruction **`key_set`** dans votre fonction Main (ou à l'endroit où vous souhaitez assigner une nouvelle touche).

ex.:

```
key_set(key_for_str("S"),spell);
```

Si vous pressez [S], le charme démarre.



Quelques 'trucs' pour terminer

- N'oubliez jamais de multiplier les valeurs qui changent continuellement dans vos effets de particules par **`time`** pour éviter d'avoir des résultats différents sur différents ordinateurs.

- Ne jamais utiliser d'instructions **`wait`**/**`waitt`** dans une fonction de particule. Cela conduit à un crash du moteur

- Utilisez le 4ème paramètre de l'instruction d'effet pour remettre des valeurs à la fonction de particules. Quant une particule est née, **`my.vel_x`**, **`my.vel_y`** et **`my.vel_z`** ont ces valeurs, qui ont été remises par l'instruction effect iSi vous assignez **`my.vel_x`** par exemple à **`my.skill_x`**, vous pouvez utiliser ces valeurs pour tout autre chose que la vitesse.