GameStudio

C-Script / WDL

Tutorial & Manuel

pour le moteur A5 version 5.15

Johann C. Lotter / Conitec Janvier 2002

Partially translated by Guenther Mulder

WED level editor	Matthew Ayres, Paul Hsu, Wladimir Stolipin
MED model editor	Wladimir Stolipin
Map compiler	Gennadi Wart, Marco Grubert
WDL compiler	Volker Kleipa
A5 engine	Johann Christian Lotter
Example games	Czeslav Gorski, Harald Schmidt, Doug Poston
Script prefabs	Doug Poston

GameStudio est publié en Amérique du Nord par Abracadata, Inc. (http://www.abracadata.com), MacMillan Software (http://www.macmillansoftware.com), et Conitec, Inc. (http://www.conitec.net). GameStudio est publié en Allemagne par Sybex GmbH (http://www.sybex.de) et Conitec GmbH (http://www.conitec.de).

Les dernières nouvelles, les démonstrations, les mises à jour et les outils, aussi bien que le Magazine des Utilisateurs, le Forum des Utilisateurs et le concours annuel sont disponibles à la page principale GameStudio http://www.3dgamestudio.com/

Si vous possédez l'édition commerciale ou professionnelle, vous avez droit à 3 ou 12 mois de support technique via émail. Expédiez s'il vous plaît par e-mail votre question à acknex@conitec.net et donnez votre édition, la date d'achat et le numéro de série ou le numéro de client (sur le disque clef ou sur votre facture). Essayez de trouver la réponse dans le manuel d'abord! On ne peut répondre aux questions techniques que si elles sont envoyés émail à cette adresse. Normalement on répond à chaque question en un jour ouvrable.

Avant propos à propos de la traduction française de ce manuel

La traduction est un art difficile. D'autant plus qu'il m'a fallu apprendre le moteur au fur et à mesure de la traduction. Malgré tous mes efforts il est clair que tout n'est pas parfait. Je vous invite donc lorsque vous avez le moindre doute, un problème de compréhension voire un mauvais fonctionnement par rapport à ce que vous avez lu, à consulter le manuel en Anglais ou en Allemand, tous deux étant téléchargeables sur le site de Conitec et à me faire part de vos remarques. Et je réclame par avance votre indulgence.

Je vous souhaite le plus grand plaisir à lire ces lignes et à mettre en pratique ce que vous apprendrez.

Alain Brégeon
mailto:alainbregeon@hotmail.com

Table des Matières

Félicitations : En tant que l'heureux propriétaire de 3D GameStudio vous pouvez créer maintenant des applications interactives en temps réel - particulièrement, mais également des jeux électroniques 2D et 3D - rapidement, facilement et sans connaissance de programmation précédente.

Ce manuel traite principalement du langage de programmation de GameStudio le C-Script. Avant de lire ceci, vous devez démarrer l'éditeur de niveau WED et commencer avec le tutorial WED pour avoir une vue d'ensemble de la façon dont travaille 3D GameStudio. En atteignant la dernière leçon de WED, vous serez prêts pour écrire des scénarios. Ce manuel est composé de plusieurs parties :

La première partie, le tutorial de script est une introduction à la programmation des jeux.

Dans la partie suivante, vous trouverez la référence du script, la syntaxe et les objets du langage de programmation y sont décrits.

Si vous ne voulez pas écrire vos propres scripts mais que vous préférez employer les préfabriqués, lisez la troisième partie pour apprendre comment définir votre jeu en mettant les drapeaux et les paramètres des fonctions prédeterminées.

Note : Si vous avez travaillé auparavant avec une version de GameStudio, vous avez probablement connu le langage de programmation sous le nom de WDL. Pas d'inquiétude, nous n'allons pas vous faire apprendre un nouveau langage à présent. Les structures et les commandes spécifiques à WDL comme SKILL, SYNONYM, CALL, ne sont plus documentés dans ce manuel mais sont toujours supportés par le compilateur C-Script. Aussi pour l'instant il n'y a pas de différence entre C-Script et WDL. Nous recommandons, cependant, de ne pas utiliser les mots-clés abandonnés et la syntaxe de WDL pour de nouveaux projets. Les prochaines versions de GameStudio distingueront entre les structures de type C-Script type et celles de type WDL. Pour terminer la vieille syntaxe ne sera plus soutenue pour des raisons d'efficacité de compilateur.

Tutorial: Apprenez vous-même la programmation des jeux en 6 jours.

Commençons par quelques suppositions simples : vous avez fouillé dans les Tutoriaux WED et avez créé quelques niveaux sympa. Maintenant vous voulez coller vos niveaux ensemble pour en faire un grand jeu 3D. Vous ne voulez pas employer les calibres (template) de jeu prédéfinis - vous voulez insérer des monstres, des effets et les énigmes de votre propre imagination. Pour accomplir cela, vous estimez que vous devrez écrire quelques scénarios WDL.

Après la fin de ce tutorial, vous saurez comment programmer un jeu. Bien qu'il soit simple de travailler avec, C-Script a les particularités de base d'un langage moderne de programmation orienté objet. La syntaxe C-Script est basée sur Javascript, le langage des pages Web. Aussi en apprenant le langage de script C-SCRIPT, vous apprendrez également l'essentiel de la programmation de site Web ou d'ordinateur. Si vous voulez vous tourner vers d'autres langages de programmation modernes, comme C, C ++ ou Java, C-SCRIPT est également une base d'introduction. Et une des meilleures choses de C-SCRIPT est que vous pouvez faire beaucoup avec très peu de programmation.

Assez parlé de C-SCRIPT. Parlons maintenant de ce tutorial.

Aujourd'hui, nous exécuterons la première partie d'un tutorial de six jours qui vise à vous faciliter la programmation de jeu en écrivant des scénarios C-SCRIPT utiles immédiatement. Voici un bref aperçu de ce que vous apprendrez chaque jour.

Dimanche :	Votre premier programme (leçon basée sur le Tutorial Java script de Thau).
Lundi :	Entités, portes, ascenseurs, clefs.
Mardi :	Mouvement du joueur, physique de jeu.
Mercredi :	Intelligence artificielle, ennemis, combat.
Jeudi :	Interface utilisateur, panneaux, menus.
Vendredi :	Programmation avancée, sauvegarde, chargement, jeux à plusieurs niveaux.

Avant de nous lancer, voici quelques points importants à noter à propos de C-SCRIPT et de ce tutorial.

Premièrement : n'oubliez pas que ce tutorial ne remplace pas du Manuel de Référence de C-SCRIPT. Bien que vous appreniez la majeure partie de la grammaire de C-SCRIPT ici, vous n'apprendrez pas toutes les fonctions et effets disponibles. Mais vous apprendrez assez pour continuer tout seul.

Deuxièmement : regardez les sources des scripts existants! La meilleure façon d'apprendre C-SCRIPT est de regarder les scénarios dans le répertoire des calibres (template) et les scénarios que les autres personnes ont écrits. Faites-le fréquemment!

Troisièmement : expérimentez librement et souvent. À plusieurs endroits de ce tutorial, on vous donnera l'occasion d'essayer des choses. N'ayez pas peur de vous étendre au-delà de l'exercice pour essayer de nouvelles choses.

Ça va, assez avec la morale. Au travail. Écrivons quelques scénarios.

Dimanche après-midi : la syntaxe C-SCRIPT

Si vous êtes déjà familiers avec la programmation et les langages de script comme Java script ou même C++, vous pouvez choisir de sauter ce chapitre et commencer juste par la leçon de lundi à propos des portes et des clefs. Sinon : démarrez WED ouvrez et exécutez le niveau office (lancer le avec le bouton [run], mais n'oubliez pas de le compiler [build] auparavant). Tandis que vous regardez le ciel, appuyez sur la touche [Tab]. Le jeu se gèle et un curseur clignotant apparaît sur l'écran. Maintenant tapez

```
sky_scale = 0.3;
```

Et pressez [Entrée]. Si vous observez maintenant un changement de la texture du ciel, vous avez simplement été témoin de l'effet de votre toute première instruction de script! Changez le 0.3 par une autre valeur en utilisant les touches de direction et [Suppr], puis pressez [Entrée] de nouveau. Amusez vous à mettre d'autres valeurs et observez l'effet dans le jeu. Vous pouvez quitter ce mode d'entrée direct par [Echap].

Ce que vous avez fait c'est de changer la valeur d'une variable nommée sky_scale. Il ne vous aura pas échappé que cette variable est responsable de la graduation de la texture du ciel. Les déclarations C-SCRIPT sont juste du texte qui peut être tapé dans un éditeur de texte ou – en utilisant la touche [Tab] - dans le jeu lui-même. La première méthode est la normale, la deuxième est uniquement pour tester en cours du jeu. À ce propos, la fonction qui permet d'entrer une ligne à partir de la touche [Tab] a été écrite dans C-SCRIPT.

Quittez le jeu et ouvrez maintenant le fichier office.wdl dans le répertoire de TRAVAIL (work) en utilisant un éditeur de texte ordinaire - le Bloc-notes™ de Windows par exemple. Une meilleure solution est un éditeur de syntaxe C-SCRIPT spécial - vous pouvez en télécharger un sur le Site Web Conitec. N'utilisez jamais un traitement de texte comme Word™ ou WordPerfect™ pour vos scénarios - ils saccageront les fichiers avec leurs codes de formatage!

Vous devez maintenant voir le commencement du fichier principal office.wdl, qui ressemble à ceci:

```
////////////////////////////////////////////////////////////////////////
// Office test level
////////////////////////////////////////////////////////////////////////
path "..\\template"; // chemin depuis le répertoire WORK vers le répertoire Template
include <movement.wdl>; // librairies de fonctions C-SCRIPT, localisées dans TEMPLATE
include <messageswdl>;
include <doors.wdl>;
…

////////////////////////////////////////////////////////////////////////
// After engine start, the MAIN function is executed, so that the 2-D engine
…
function main()
        {
        ....
```

C'est un scénario de base, comme il en est automatiquement créé après que vous ayez cliqué le bouton [new] des Propriétés de Carte (Map Properties).

Tout ce qui se trouve entre // et la fin de la ligne est un commentaire et sera ignoré par le moteur. Une règle de base d'un bon style de script consiste en ce que vous devez toujours penser à la personne suivante qui doit regarder votre scénario. Cela pourrait être un ami, un collaborateur, un employeur ou vous dans trois mois. La façon la plus facile de vous assurer que vous comprendrez votre propre scénario dans trois mois est de faire des commentaires librement et souvent. Si vous voulez faire un énorme commentaire, vous pouvez le mettre entre /* et */ comme cela :

```
/* Ceci est un bloc
de texte qui me
sert de commentaires */
```

Au début du fichier de script on donne un chemin pour les répertoires et beaucoup de nouveaux fichiers script sont inclus à partir de ce chemin. Include les traite comme si leur contenu avait été directement tapé dans le scénario. Ces

fichiers calibrés ('template') C-SCRIPT contiennent des objets prédéfinis et des fonctions disponibles pour le jeu. Ils sont décrits dans d'autres chapitres du manuel et peuvent être employés pour faire un jeu simple, comme un jeu de tir, sans ou avec très peu de programmation. Mais si vous avez l'intention de créer un jeu qui ne ressemble à aucun autre, vous pouvez décider de rejeter toute la substance prédéfinie et réécrire tout le scénario à partir de zéro. Vous commencerez par un fichier C-SCRIPT vide dans ce cas. La fonction principale (function main()) – au dessous est exécutée directement au début de moteur et contient toutes les choses à faire au début du jeu - qui est, dans notre exemple, de charger le niveau office.wmb.

Cependant, commençons maintenant à apprendre le langage - comment C-SCRIPT stocke l'information, comment il prend des décisions basées sur cette information et comment changer l'information basée sur l'interaction d'utilisateur. Prêt ? Il est temps d'apprendre les principes de base de programmation. Premier arrêt : les variables.

Variables et chaînes de caractères

Si vous avez appris l'algèbre, vous avez déjà vu des variables. Si vous n'avez pas appris l'algèbre, ne vous inquiétez pas, les variables sont simplement le moyen pour n'importe quelle langage de programmation, de stocker l'information numérique. Par exemple, si vous écrivez "x = 2", "x" est une variable qui contient la valeur "2." Si vous dites ensuite "y = x+3", "y" aura la valeur "5."

Voici un exemple de script qui crée des variables :

```
// definit quelques variables
var secs_per_min = 60;
var mins_per_hour = 60;
var hours_per_day = 24;
var days_per_year = 365.25;
var secs_per_day;
var secs_per_year;
```

Comme vous voyez, chaque déclaration ou instruction doit finir avec un point-virgule. Si vous l'oubliez, vous pouvez être sur de recevoir un message d'erreur du moteur. La première ligne commençant par // est un commentaire qui énonce une évidence. Le groupe de lignes suivantes est des définitions de variables. Il y a quelques choses à noter au sujet de ces lignes :

Avant que vous ne puissiez employer une variable, vous devez la définir avec le mot "var", suivi par son nom, ensuite par une valeur initiale facultative après un "=". Quelques variables (comme notre sky_scale) n'ont pas besoin d'avoir été définies - elles sont déjà prédéfinies intérieurement par le moteur. À la différence de son frère' C++, C-Script connaît seulement un type de variables. C++ a des douzaines de types variables pour différents buts, comme int, float, double, short, long .. en C-Script vous pouvez juste employer var.

Les noms de variables doivent commencer par une lettre ou le caractère souligné (_). Après le premier caractère, les variables peuvent avoir des nombres. Ainsi harry_23 est un nom acceptable pour une variable.

Les noms variables ne sont pas sensibles à la casse dans C-SCRIPT. Cela signifie que les variables Boucle, boucle et BOUCLE seront considérées comme identiques. Généralement, c'est une bonne idée de choisir une convention de nom et de s'y tenir. J'aime avoir toutes mes variables en minuscule, avec le caractère souligné séparateur de mots. D'autres préfèrent employer la capitalisation interne, comme SecsPerMin.

Le nom des variables doit décrire ce qu'elles sont. Des noms de variables comme a, b ou un nombre, ne seront pas très explicites pour une personne qui essaye de comprendre votre scénario. Ne donnez pas à vos variables des noms si longs qu'il faille beaucoup de temps pour les taper, mais suffisamment long pour être explicite.

Vous pouvez donner à une variable une valeur initiale quand vous la définissez. Dans l'exemple, on a donné à certaines des variables une valeur la première fois qu'elles ont été définies. Vous ne devez pas faire cela et nous verrons des exemples où c'est ok pour définir une variable bien que nous ne connaissions pas sa valeur tout de suite. Une variable peut représenter plus qu'un nombre - nous apprendrons plus tard les vecteurs qui contiennent trois nombres et les tableaux qui contiennent autant de nombres que vous le souhaitez.

Les déclarations se terminent par un point-virgule. Les déclarations sont les phrases en langage de programmation et les points-virgules sont les signes de ponctuation de fin. Les espaces et les sauts de ligne sont normalement ignorés, donc la disposition du scénario sert seulement à être plus lisible pour les gens. Cet exemple pourrait avoir été écrit en une ligne vraiment longue si vous enlevez les commentaires. Mais ce serait dur de la lire.

Fonctions

Prêt à faire quelque chose avec nos variables ? Un objet qui fait quelque chose avec quelque chose est une fonction, constituée d'instructions :

```
// faisons quelques calculs
function some_calculations()
{
	secs_per_day = secs_per_min * mins_per_hour * hours_per_day;
	secs_per_year = secs_per_day * days_per_year;
}
```

Les règles de nom des fonctions sont les mêmes que pour les variables. Le premier caractère doit être une lettre ou un caractère souligné. Le reste des caractères peut être des nombres . Aussi, vous devez vous assurer que vous ne nommez pas une fonction avec le même nom qu'une variable!

Après le nom viennent deux parenthèses, entre lesquelles des paramètres pourraient être transférés à la fonction - mais nous n'employons pas cela ici, donc les parenthèses sont vides. Vient ensuite le corps de la fonction entre accolades. C'est le jeu d'instructions que vous voulez exécuter quand la fonction est exécutée. Dans notre exemple nous voyons deux instructions avec quelques maths de base. L'expression à droite du "=" est calculée et le résultat est placé dans la variable qui est à gauche du "=". Ainsi après que le moteur ait exécuté la fonction de C-SCRIPT, la variable secs_per_year contiendra ce que vous obtenez quand vous vous multipliez 60, 60, 24 et 365.25. A partir du moment ou la fonction a été exécutée, chaque fois que C-SCRIPT voit la variable secs_per_year, il la remplacera par ce nombre énorme.

Vous pouvez apprendre une deuxième chose de cette fonction : Dans la plupart des cas, l'ordre des instructions importe. Les instructions sont exécutées du commencement de la fonction à la fin. Si nous avions échangé les deux instructions, le résultat donnerait n'importe quoi, parce que la deuxième instruction a besoin de secs_per_day déjà calculé!

Après avoir vu les variables, voyons à présent une deuxième façon de stocker de l'information, nommé chaîne de caractères . Les chaînes de caractères sont quelque peu semblables aux variables, mais elles contiennent des caractères au lieu de nombres. N'importe quel groupe de caractères, donnés entre guillemets, est un contenu valable d'une chaîne de caractères . Donc il est correct de définir :

```
// a string definition
string hello = "Salut le monde!";
```

Cette déclaration dans un fichier de script définit la chaîne de caractères hello et la remplit des caractères donnés. Cependant il y a quelques différences subtiles entre des chaînes de caractères et des variables. Vous ne pouvez pas faire de mathématiques avec une chaîne de caractères. Vous pouvez changer le contenu d'une chaîne de caractères comme vous le faîtes avec une variable.

Assez de théorie, écrivons à présent notre premier programme.

Mon Premier Programme Informatique

Démarrez WED. Créez un nouveau projet vide (File ➔ New), nommez le "tutorial" en utilisant File ➔ Save AS puis créez un nouveau fichier script en ouvrant File ➔ Map Properties et en cliquant sur le bouton [New]. Le nouveau script tutorial.wdl est à présent affiché dans le champ script.

A présent ouvrez tutorial.wdl dans votre répertoire de travail en utilisant bloc-notes™ ou tout autre éditeur de texte simple. Ce script contient déjà un certain nombre de choses, similaires à office.wdl que nous avons vu auparavant. Effacez tout, nous allons pas faire un jeu entier mais juste quelques exercices. Tapez ce qui suit dans votre fichier script qui est vide pour l'instant :

```
////////////////////////////////////////////////////
// quelques exercices C-SCRIPT

string hello = "hello world!";

text screen_txt // l'objet texte pour afficher notre chaîne de caractère
{
	font = _a4font; // fonte standard
	pos_x = 5; // le texte commence à la position de l'écran (5,40) en pixels
	pos_y = 40;
}

function print(str) // affiche un paramètre string à l'écran
{
	screen_txt.string = str;
	screen_txt.visible = on;
}

function main()
{
	screen_color.blue = 128; // met un fond bleu foncé
	print(hello); // affiche la chaîne
}
```

Sauvegardez à présent votre travail, cliquez sur le bouton [RUN] dans WED, cochez la case [Window] puis [Go!]. S'il ne démarre pas maintenant, mais donne un message d'erreur, vous pouvez être sûrs que vous avez oublié un petit point virgule ou quelque chose comme ça ou que vous avez mal tapé quelque chose. La ligne en question est montrée dans le message d'erreur, donc vous pouvez facilement voir et corriger votre erreur. Contrôlez soigneusement jusqu'à ce que le moteur fonctionne enfin. Notre premier programme C-SCRIPT est en cours d'éxécution!

Nous avons programmé une application qui dit juste "hello world" dans une fenêtre de fond bleue. Bien, nous ne pouvons pas faire beaucoup plus dans cette application sauf de pressez la touche [echap] ou [F10] pour sortir.

Néanmoins nous pouvons apprendre beaucoup de ce programme. Nous avons employé un nouvel objet , un **texte**. Les textes sont pour l'affichage des chaînes de caractères sur l'écran. Ils sont des objets plus compliqués que les variables ou les chaînes de caractères. Ils se composent d'un corps entre des accolades, comme des fonctions. Dans le corps quelques valeurs de propriétés de texte peuvent être écrites par défaut. Les propriétés de texte que nous utilisons ici sont **screen_txt** pour la position xy sur l'écran et la fonte de caractère (**_a4font** est une fonte de défaut employé intérieurement par le moteur).

Ensuite nous avons défini la fonction **print**. À la différence de la fonction **some_calculations** dont nous avons discuté auparavant, celle-ci prend un paramètre. Les paramètres de fonction sont des nombres, des chaînes de caractères ou d'autre item remis à la fonction en l'exécutant. Le paramètre est connu de la fonction sous le nom que nous avons donné entre parantheses dans la définition de fonction. Dans notre cas, nous employons le paramètre comme une chaîne de caractères. Notre fonction d'impression (**print**) montre la chaîne de caractères sur l'écran en mettant la propriété de l'objet texte de la chaîne de caractères à cela et ensuite en mettant la propriété visible de l'objet de texte propriété sur **on** (1). Notez le point après **my_text**. Il indique que l'article suivant, string ou visible, appartient à l'objet My_text. Vous pouvez lire dans la partie de référence de ce manuel toutes les propriétés qu'un texte peut avoir (Allez au chapitre "Interface Utilisateur (User interface)"). La propriété string d'un texte donne juste les caractères à afficher. Visible est un drapeau - semblable à une variable, mais il peut seulement prendre deux valeurs, on (1) ou off (0). Donc il est employé comme un commutateur "Marche/Arrêt", dans notre

cas pour la visibilité de l'objet de texte. Les chaînes de caractères (**string**)) et les drapeaux sont mis avec le signe " =" comme pour les variables.

Comme nous l'avons dit précédemment, la fonction **Main** est éxécutée dès le lancement du moteur. Au départ elle met le fond à la couleur bleue en utilisant une variable prédéfinie **screen_color**. Souvenez-vous de notre première déclaration avec sky_scale ? Ces deux **screen_color** et **sky_scale** sont des varaiables prédéfinies qui changent certaines propriétés de notre jeu. C-SCRIPT contient une quantité de variables prédéfinies comme celles-ci pour contrôler les caractéristiques du moteur. **Screen_color** est un **vecteur couleur** avec les valeurs rouge, vert et bleu qui sont à 0 par défaut. Chaque couleur peut prendre une valeur entre 0 et 255 et les 3 ensembles définissent une couleur pour le fond de notre écran. Lorsque nous mettons 128 à bleu et que nous avons rouge et vert qui sont à 0, nous obtenons du bleu foncé.

A la ligne suivante la fonction **Main** exécute la fonction '**print**' qui reçoit la chaine **hello** comme paramètre. Exécuter (ou appeler – **call**) une fonction à partir d'une autre fonction est fait lorsque vous tapez le nom de la fonction suivie par deux parenthèses et des paramètres à l'intérieur (s'il y en a). Ici vous voyez les avantages des paramètres de fonctions. A chaque fois qu'une chaine de caractère sera passée à la fonction **print**, elle sera affichée, aussi nous pouvons utiliser **print** à partir de maintenat comme une fonction d'affichage à but général. Notez que la chaine **hello** est connu à l'intérieur de la fonction **print** sous le 'nom local' **str** – **Hello** et **str** sont la même chaine de caractères. A ce propos, il n'est pas nécessaire de définir séparément les chaines de caractères. Comme pour les variables, elles peuvent être données directement dans les instructions comme ceci :

```
function main()
{
	screen_color.blue = 128; // met un bleu foncé pour le fond
	print("Hello world!"); // affiche la chaine directement
}
```

Maintenant que vous avez fait ce travail, il est temps d'apprendre les clauses if-else.

L'embranchement if (si)

L'embranchement qu'implique "**if**" permet à votre programme de se comporter très différemment selon des conditions, comme des entrées d'utilisateur. Par exemple, vous pourriez écrire un scénario qui agirait d'une façon pour vous et autrement d'une autre façon vers quelqu'un d'autre. Voici la forme de base d'une instruction « if »:

```
if (some condition is true) // si (quelque condition est vraie)
{
	do something; // faites quelque chose;
	do something;
	do something;
}
```

Les parties importantes de cette structure sont :
- Elle commence par le mot "**if**" (ou en majuscule si vous préférez "IF") .
- Il y a une condition numérique entre parenthèses qui est ou vraie ou fausse.
- Il y a un jeu d'instructions qui doit être exécuté si la condition est vraie. Ces instructions sont entre accolades.

Souvenez-vous, l'espacement doit seulement rendre votre scénario plus lisible. Je préfère mettre en retrait les instructions entre accolades. Mais vous pouvez mettre toute l'instruction sur une seule ligne si vous voulez. Il y a une autre instruction, **else** (sinon), qui est tout le contraire de **if**. Si vous placez un **else** directement après la parenthèse finale d'une l'instruction **if**, les instructions entre les accolades de l'**else** sont exécutées seulement si celles de l'instruction if ne sont pas exécutées. Voici un exemple d'instructions **if** et **else** en action :

```
function which_key()
{
	if (key_space == 1) { // barre [espace] de pressée?
		print("Pressez la barre espace!");
	} else {
```

```
            print("Pressez une autre touche!");
        }
}
```

Il y a des messages différents d'affichés selon que key_space vaut 1 ou pas. Key_space est une variable prédéfinie qui est à 1 aussi longtemps que la touche [espace] est appuyée, sinon elle vaut 0. La condition "==" entre parenthèses est vraie si la valeur de key_space est égale à 1.

Entrez cette fonction au-dessus de la fonction Main de notre tutorial.wdl et entrez une déclaration complémentaire dans la fonction Main.

```
function main()
{
       screen_color.blue = 128; // set a dark blue background
       print("Hello world!"); // display string directly
       on_anykey = which_key; // assign the which_key function to the "any key" event
}
```

On_anykey appelle la fonction assignée chaque fois qu'une touche est frappée. Comme vous voyez, il y a deux façons d'exécuter une fonction – oul'appeler de l'intérieur d'une autre fonction, ou son assignation à quelque événement (event) qui la fera commencer. Le fait de la taper dans un fichier script n'est pas suffisant pour qu'une fonction soit exécutée! Seulement la fonction Main est automatiquement assignée à l'événement " début du jeu". Notez qu'en assignant une fonction à un événement, au lieu de l'appeler directement, on ne passe aucun paramètre et aucune parantheses ne sont données.

Maintenant démarrez le programme et appuyez sur n'importe quelle touche. Vous aurez des messges différents selon que vous pressiez la barre [espace] ou pas.

Notez aussi que la condition est deux signes « égal ». C'est une confusion que beaucoup font au début. Si vous mettez un signe égal au lieu de deux, il ne travaillera pas. D'autres conditions typiques sont :

(var_1 > var_2)	est vrai si var_1 est plus grand que var_2
(var_1 < var_2)	est vrai si var_1 est plus petit que var_2
(var_1 < = var_2)	est vrai si var_1 est inférieur ou égal à var_2
(var_1 > = var_2)	est vrai si var_1 est supérieur ou égal à var_2
(var_1! = var_2)	est vrai si var_1 est différent de var_2

Deux façons de faire connaissance avec les conditions. Si vous voulez que 2 choses soient vraies avant d'exécuter les instructions entre accolades, vous pouvez faire cela :

```
if ((var_1 > 18) && (var_1 < 21))
{
    do something;
}
```

Remarquez les deux esperluettes (&). C'est la façon de dire "et" dans C-SCRIPT. Remarquez aussi que la clause entière, y compris les deux sous-parties et les esperluettes, doit être incluse dans des parenthèses.

Si vous voulez qu'au moins l'une ou l'autre des 2 choses soit vraie pour exécuter les instructions entre accolades, faites ceci :

```
if ((var_1 == 7) || (var_2 == 13))
{
    do something;
}
```

O.K., il est temps maintenant de faire une petite révision des choses que nous avons couvertes jusqu'ici. Si vous pensez ne pas avoir appris une de ces choses, n'hésitez pas à revenir dessus :

- // et /* */ sont employés pour des commentaires. Commentez vos scénarios fréquemment
- Des variables peuvent contenir des nombres. Il y a quelques restrictions et règles à garder à l'esprit en nommant des variables.
- Les chaînes de caractères peuvent contenir des séquences de caractères.
- Les fonctions sont pour faire ou pour changer quelque chose et sont constituées d'instructions.
- Les Instructions et les déclarations se terminent par un point virgule.
- Les fonctions peuvent être appelées soit directement, soit être assignée à des événements.
- Des paramètres peuvent être passés lorsque des fonctions appellent directement d'autres fonctions.
- Utilisez les clauses if-else (si-sinon) pour faire que vos fonctions C-SCRIPT se comportent différemment selon des conditions.

Félicitations si vous avez réussi à digérer toute cette substance. Ça faisait beaucoup à apprendre. Nous avons examiné les variables, les chaînes de caractères, les fonctions et les clauses if-else, que l'on retrouve dans des formes équivalentes dans tous les langages de programmation. Maintenant il est temps d'apprendre le reste de la syntaxe C-SCRIPT. Il y a seulement un aspect principal de syntaxe C-SCRIPT que nous devons encore couvrir : les boucles. Commençons par des fonctions C-SCRIPT un peu plus compliquées et présentons les boucles.

Les boucles

Parfois vous voulez faire la même chose plus d'une fois. Disons, par exemple, que vous voulez obtenir un code secret de quelqu'un et vous voulez continuer à lui demander tant qu'il ne vous a pas donné le code juste. Si vous aviez juste voulu lui donner deux essais, vous auriez fait quelque chose comme cela :

```
var the_code = 12345; // code secret
var entered_number = 0;
string entry_line[80]; // juste une longue chaîne vide

function check_code()
{
	print ( “entrez un nombre s'il vous plait...”); // Affiche ceci à l'écran
	waitt(16); // attend 16 ticks (1 seconde)

	print( entry_line);
	inkey(entry_line); // interroge le clavier pour entrer une chaîne
	entered_number = str_to_num(entry_line); // convertit la chaîne entrée en un nombre

	if (entered_number != the_code)
	{
		print("mauvais – entrez le code de nouveau…");
		waitt(16);

		print( entry_line);
		inkey(entry_line); // de nouveau l'entrée de l'utilisateur
		entered_number = str_to_num(entry_line); // convertit la chaîne entrée en un nombre

		 if (entered_number != the_code)
		{
			print( “encore mauvais!”);
			return; // deux essais, on termine la fonction
		 } else {
		print( “wow! Vous avez réussi!”); // bonne réponse au deuxième essai
		return;
		}
	} else {
	print( “wow! Vous avez réussi!”); //bonne réponse au premier essai
	return;
	}
}
```

```
function main()
{
	screen_color.blue = 128; // set a dark blue background
	check_code(); // start the check_code function
}
```

C'est une fonction longue et quelque peu laide ! Cependant, vous avez appris quelques nouvelles instructions ici. Si vous donnez un nombre entre crochets [] dans la définition d'une chaîne, vous créez une chaine vide composées du nombre d'espaces. Waitt() attend juste le nombre donné en 1/16 de secondes et continue ensuite la fonction. Inkey() produit un curseur clignotant et entre les caractères tapés par l'utilisateur dans la chaîne donnée jusqu'à ce qu'il presse [Entrée]. Str_to_num convertit un contenu numérique d'une chaîne - "123", par exemple – en valeur numérique correspondante. Nous devons faire cela parce que nous ne pouvons pas comparer une chaîne avec un nombre dans la condition if - cela ressemblerait à la comparaison de pommes avec des oranges. return termine juste la fonction.

Comme vous voyez, cette fonction ne travaillera pas si vous voulez continuer à demander jusqu'à obtenir le bon code. Et c'est déjà assez laid - imaginez si vous aviez voulu demander quatre fois au lieu de deux ! Vous auriez quatre niveaux d'if-else ce qui n'est jamais une bonne chose.

La meilleure façon de faire des choses semblables plus d'une fois c'est d'employer une boucle. Dans notre cas, vous pouvez employer une boucle pour continuer à demander le code secret jusqu'à ce que la personne renonce. Voici un exemple de boucle d'attente en action.

```
function check_code()
{
	while (entered_number != the_code)
	{
		print("Please enter correct number..."); // display this on screen
		waitt(16); // wait 16 ticks (1 second)
		print(entry_line);
		inkey(entry_line); // requests user input into a string
		entered_number = str_to_num(entry_line); // converts input from a string to a
		number
	}
	print("Wow! You've guessed it!"); // display "right" message
}
```

C'est la boucle while, qui simplifie vraiment notre fonction. Les boucles while entrent dans cette forme générale :

```
while (some test is true) //tant que (quelque chose est vrai)
{
	do the stuff inside the curly braces // éxécute ce qui se trouve entre accolades
}
```

Donc ces lignes disent, "Tant que la réponse n'est pas égale au code, continuer à obtenir l'entrée de l'utilisateur." La boucle continuera à exécuter les instructions à l'intérieur des accolades jusqu'à ce que test soit faux. Dans ce cas, test sera seulement faux quand les caractères entrés par l'utilisateur seront les mêmes que le code (c'est-à-dire "12345"). Test est vrai si code est faux – ce qui s'écrit "!=" !

Parce que l'on donne une valeur à entered_number par l'instruction str_to_num à l'intérieur de la boucle while, il n'aura aucune valeur la première fois que nous exécutons la boucle. Ainsi nous l'avons défini avant avec une valeur initiale de 0, qui n'est pas certainement le bon code. Bien que l'utilisation des boucles infinies soit souvent utile, les boucles sont plus généralement employées pour exécuter un jeu d'instructions un nombre spécifique de fois. Voici un autre exemple de boucle while qui montre comment faire cela.

Nous voulons écrire un programme qui vous fait entrer un nombre et imprime ensuite autant de x à l'écran. Revoyons cela. D'abord, définissez quelques nouvelles chaînes et variables :

```
var counter = 0;
var entered_number = 0;
string entry_line[80];// une chaîne avec rien du tout

function count_x()
{
	print("entrez un nombre svp..."); // affiche"entrez un nombre svp..." à l'écran
```

```
        waitt(16); // attends 1 seconde, puis continue

        str_cpy(entry_line,empty_string); // remets à 0 la ligne d'entrée en prévision d'une saisie
        print(entry_line);
        inkey(entry_line); // demande à l'utilisateur d'entrer une ligne
        entered_number = str_to_num(entry_line); // convertit la chaîne en un nombre

        str_cpy(result_string,empty_string); // remets à 0 la chaîne résultat
        while (counter < entered_number) {
        str_cat(result_string,"x"); // ajoute "x" à la chaîne résultat
        counter = counter + 1; // incrémente la boucle
        }
}
function main()
{
        screen_color.blue = 128; // set a dark blue background
        on_p = count_x; // assigne la fonction count_x à la touche [P]
}
```

La déclaration on_p est similaire à on_anykey que nous connaissons déjà mais il assigne la fonction count_x à la touche [P]. Comme vous le savez probablement on peut assigner n'importe quelle fonction à n'importe quelle touche on utilisant on_a, on_b etc pour le faire.

Count_x dit : "tant que la variable counter est inférieure au nombre demandé de x, ajouter un autre x à la ligne et ajoute un à la valeur de counter. " str_cpy copie une deuxième chaîne dans la première, tandis que str_cat ajoute une deuxième chaîne à la fin de la première. Cette boucle continuera à ajouter un "x" au result_string et ajoutera un à la valeur counter tant que counter est inférieur au nombre demandé.

L'augmentation d'une variable de boucle est si commune que les programmeurs ont développé un raccourci. En employant le raccourci, la boucle while pourrait avoir été écrite comme cela :

```
while (counter < entered_number) {
        str_cat(entry_line,"x");  // ajoute "x" à la chaine
        counter += 1;                          équivaut à  counter = counter + 1;
}
```

La troisième ligne, counter + = 1, dit «ajoute 1 à moi. » Si vous avez a_number = 5, et si vous écrivez a_number += 3, c'est comme si vous écriviez a_number = a_number + 3. Vous pouvez utiliser également - =, * = et / = de la même façon. Les programmeurs sont paresseux ; ils inventent toujours des raccourcis comme cela.

Essayez à présent la fonction. Elle semble bien fonctionner - mais seulement la première fois. Si nous pressons la touche [P] une deuxième fois, il s'affiche moins de 'x' que ce nous avons entré - ou aucun 'x' du tout. Mais que se passe-t-il ici ?

Mise au point

Si vous avez écrit une fonction, vous remarquerez parfois qu'elle ne travaille pas ou se comporte différemment de ce que vous attendiez. Et souvent vous ne savez pas même pourquoi. Vous pouvez seulement penser à deux raisons :

1) Parfois la langue C-SCRIPT ne travaille pas correctement.
2) Je suis trop stupide pour la programmation de jeu.

Le compilateur C-SCRIPT est parfaitement testé de la part de GameStudio, aussi vous pouvez normalement être confiant que le 1 n'est pas le cas. Et vous n'êtes pas trop stupide, je vous rassure - les bogues arrivent aux programmeurs les plus expérimentés tout le temps. En fait, il est tout à fait normal qu'une fonction ne travaille pas du premier coup. Et c'est aussi tout à fait normal que même un programmeur chevronné n'en trouve pas la raison en regardant simplement fixement le code de C-SCRIPT.

Les professionnels mettent au point leur code. La mise au point n'est pas un secret. Cela signifie juste ici : exécuter le code non dans son ensemble, mais ligne par ligne et examiner en détail ce qui arrive, pour trouver des bogues

possibles. C-SCRIPT a un programme de mise au point incorporé. Vous pouvez l'activer en ajoutant juste l'instruction suivante au commencement de la fonction pour l'examiner :

```
function count_x()
{
breakpoint; // démarre la mise au point ici
print("entrez un nombre SVP...");
;...
```

Si nous démarrons maintenant le niveau et pressons [P], le jeu se gèle, nous entendons un son court et voyons la ligne suivante sur l'écran :

```
<= print(@2)
```

C'est l'instruction qui suit le point de contrôle.Le @.. est la représentation interne d'une chaine de carctère entrée directement. Maintenant pressez [Espace]. La ligne défile vers le haut et nous voyons quelque chose dans le genre de

```
0.000 <= print(@2)
       <= screen_txt.string = str
```

Nous avons maintenant exécuté une instruction. Elle a appelée la fonction print et nous sommes à présent à l'intérieur de cette fonction et nous voyons sa première ligne. Str est notre chaine « entrez un nombre svp » qui est à présent assignée à screen_txt.string. Suivant avec la touche [Espace] :

```
15135.457 <= screen_txt.string = str
       <= screen_txt.visible = on
```

Le nombre étrange (sur votre PC il doit différer de 15135.457) est le code de résultat de l'expression – dans ce cas un pointeur interne pour assigner la chaine dont il nous importe peu pour l'instant. Pressez [Espace]

```
16.000 <= screen_txt.visible = on
 <= waitt(16)
```

Nous avons à présent quitté la fonction print et sommes de nouveau dans notre fonction count_x. Vous devez voir d'affiché à l'écran notre texte 'tapez un nombre svp', puisque nous l'avons rendu visible par la dernière instruction de la fonction print. Pressez [espace] plusieurs fois jusqu'à obtenir :

```
0.000 <= inkey(entry_line)
       <= entered_number = str_to_num(entry_line)
```

Maintenant, avec le curseur clignotant dessous, entrez un "2" et pressez [Entrée]. L' [Espace] suivant donne

```
2.000 < = entered_number = str_to_num (entry_line)
      < = str_cpy (entry_line,@4)
```

Comme nous pouvons voir, l'instruction **str_to_num** a converti notre chaîne de caractères en un chiffre 2, comme attendu. Pour des expressions numériques et pour quelques instructions, le code de résultat montre simplement le nombre résultant. Si nous appuyons sur [Espace] encore deux fois, nous pouvons voir un autre code de résultat significatif :

```
1.000 < = (counter < entered_number)
      < = str_cat (entry_line, @6)
```

Le code de résultat pour une condition **if (si)** ou **while (tant que)** (le **if** ou le **while** lui-même n'est pas montré) est 1.000 pour vrai et 0.000 pour faux. Donc nous pouvons voir que la condition while a ici été accomplie et nous sommes à présent dans la boucle.

Ainsi en allant instruction par instruction dans notre fonction, nous avons une ligne de temps parfaite de ce qui arrive quand une personne choisit deux x au prompt :

Premier passage
- entry_line = "" (parce que nous y avons copié "" par str_cpy)
- counter=0 (parce que nous l'avons défini avec la valeur initiale 0)
- entered_number = 2 (parce que c'est ce que l'utilisateur a demandé)
- 0 est inférieur à 2 donc
- "x" est ajouté à entry_line, et maintenant entry_line = "x"
- counter + = 1, et maintenant counter = 1

Retour arrière dans la boucle : deuxième passage
- counter = 1
- entered_number = 2
- entry_line = "x"
- 1 est inférieur à 2 donc
- "x" est ajouté à entry_line, et maintenant entry_line = "xx"
- counter + = 1, et maintenant counter = 2

Retour arrière dans la boucle : troisième passage
- counter = 2
- entered_number = 2
- entry_line = "xx"
- 2 n'est PAS inférieur à 2 donc
- nous sortons de la boucle et nous passons à la suite

O.K. - jusqu'ici tout est bon. Nous avons maintenant fini de mettre au point notre fonction. Nous quittons maintenant le mode de mise au point par [Ctrl-Espace] et nous retournons au mode de fonctionnement normal.

La fonction s'est comportée comme attendu, mais nous savons que le problème surgit seulement dans la deuxième tentative. Aussi pressons [P] de nouveau pour exécuter en pas à pas une deuxième fois. A l'entrée du nombre, nous entrons maintenant "3". Nous nous attendons donc à avoir trois boucles. Mais regardez ce qui se passe à la première boucle :

```
3.000       < = counter + = 1;
            < =}
```

Quoi ? - l'augmentation de counter donne 3 déjà dans la première boucle! Bien entendu, comme une boucle est exécutée un seul 'x' est produit. Et maintenant nous sommes capables de trouver la raison : la variable counter a toujours sa valeur 2 de la dernière exécution de la fonction. Maintenant comme nous avons finalement trouvé le bogue, la correction est facile en ajoutant

```
counter = 0;
```

au début de la fonction. N'oubliez pas de tels détails en programmant une boucle!

Employez le programme de mise au point aussi souvent que possible. C'est une bonne façon d'apprendre comment vos fonctions travaillent et surtout comment C-SCRIPT travaille. Les professionnels mettent d'habitude au point leur code même s'il semble travailler – c'est de cette façon qu'ils peuvent détecter les bogues cachées avant même qu'ils n'arrivent dans certaines circonstances. Si vous êtes vraiment coincé avec un problème dans votre code, vous trouverez en annexe un récapitulatif des pièges les plus communs de C-SCRIPT.

Maintenant que vous avez appris la mise au point, vous pouvez vous considérer comme un programmeur cadet. Dans les jours suivant, nous allons laisser le monde un peu sec de la syntaxe C-SCRIPT et entrer dans celui plus concret des jeux. Nous apprendrons comment vous pouvez employer des scénarios C-SCRIPT pour définir le comportement du joueur, d'acteurs ou de l'interface utilisateur de façons puissantes et intéressantes

Lundi: Les portes et les clés

Il est toujours impressionnant d'avoir des choses qui se déplacent comme une partie de notre monde - comme des rouages d'horloge en mouvement, l'ouverture d'entrées secrète ou avoir le joueur trouvant soudainement de l'eau qui coule – au simple contact d'un bouton ! Donc dans ce chapitre de notre tutorial nous allons examiner comment mettre en mouvement les parties de notre monde de différentes façons. L'exemple le plus facile et le plus fréquent de mouvement dans notre monde est une ouverture de porte.

Nous savons grâce au tutorial de WED comment le faire : nous devons juste attacher l'action de porte (door) prédéfinie à une entité de carte. Alors, si le joueur presse la touche [Espace] près de la porte, elle tournera horizontalement autour de son centre. Mais comment cela est-il réalisé ? Regardons de plus prêt les actions d'entité.

Actions d'entité

Les entités sont les objets qui se déplacent dans notre monde de jeu - comme les portes, les acteurs, les ennemis ou le joueur lui-même. Leur mouvement est fait par des fonctions spéciales- nommées des actions – qu'on leur assigne. Et ces actions consistent normalement en une ou plusieurs boucles plus énormes avec l'instruction wait(1) à la fin.

Voici un exemple pour une boucle qui fait juste tourner une entité sur elle-même. Peut-être avez-vous rencontré un exemple semblable dans le tutorial WED :

```
action entity_rotate
{
	while (1) {
	my.pan+= 3;
	wait(1);
	}
}
```

C'est une description pour "fait moi tourner en permanence avec une vitesse de 3 degrés par affichage". Nous pouvons assigner cette action à une entité WED. La seule différence entre une fonction et une action est que cette dernière est définie par le mot action, apparaît dans la liste d'action de WED et n'a pas de parenthèses après le nom. L'action de chaque entité est lancée une fois au début de jeu. Nous avons ici simplement 1 comme condition pour while (ce qui se traduit par «tant que 1 fait ceci »). 1 est considéré comme étant toujours vrai. Aussi while(1) {...} est une instruction qui se répète toujours. On appelle cela une boucle infinie.

À l'intérieur de cette boucle, la valeur 3 est ajoutée à my.pan. My est un objet au même titre que l'objet msg avec lequel nous sommes déjà familiers. Cependant, my n'est pas un objet texte, c'est un objet d'entité. Plus précisément, my est un synonyme pour l'entité à laquelle l'action a été assignée. Un synonyme est un nom provisoire pour quelque chose. Comme dans notre langue parlée "moi" ou "ma" ou "mon" se réfèrent à la personne qui parle, dans C-SCRIPT my se réfère à l'entité qui exécute cette action. Si vous aviez assigné cette action non pas à une entité, mais, par 'exemple, à une touche comme on_p, vous recevriez un message d'erreur ("empty pointer" (pointeur vide)) - parce que my est exclusivement pour des entités et key n'est pas une entité!

Pan est l'angle horizontal de l'entité, en degrés. Cet angle augmentera de manière permanente de 3. Pour parler clairement : l'entité tournera. Ça n'a pas d'importance si la valeur pan excède 360 degrés - le moteur gère cela.

La dernière instruction, wait(1), suspend l'action pour un cycle d'affichage, avant la répétition de la boucle. Ainsi nous avons une rotation de trois degrés par cycle d'affichage. Nous avons déjà appris waitt () (avec deux t) dont l'unité de temps est le 1/16 de secondes, tandis que l'unité de temps de wait() et un cycle d'affichage. Sur un PC bas de gamme, qui tourne tout juste à 16 fps (frame par seconde), les deux sont presque les mêmes. Tenez compte qu'à l'extérieur de la fonction – le rendu, le changement de variable globale et l'exécution d'autres fonctions - seront exécutés pendant cette petite pause wait() ! Donc chaque boucle infinie doit toujours contenir un wait() ou un waitt(). Autrement la boucle ne permettra pas à des éléments extérieurs à elles de s'exécuter et votre PC se figera (où produira le message d'erreur «infinite loop» (boucle infinie)).

Le Premier Mouvement

Démarrez WED ouvrez un de vos niveaux et créez un nouveau fichier de scénario (Map Properties -> Script -> New). Mais cette fois nous n'allons pas nous servir des fonctions prédéfinies. Cette fois nous allons écrire nos propres fonctions. Fermez WED de nouveau. Utilisez un éditeur de texte pour ouvrir votre fichier de scénario name.wdl, sachant que ***name*** est en fait le nom de votre carte. Placez-vous maintenant à la fin du fichier de scénario et saisissez votre première action d'entité :

```
action door1
{
	while (1)
	{
		play_entsound(my,open_snd,66);
		while (my.pan < 90) {
			my.pan += 3*time; // tourne dans le sens contraire des aiguilles d'une montre
			wait(1);
		}
		waitt(16);
		play_entsound(my,close_snd,66);
		while (my.pan > 0) {
			my.pan -= 3*time; // tourne dans le sens des aiguilles d'une montre
			wait(1);
		}
		waitt(16);
	}
}
```

O.K., voyons ce que nous avons écrit. Nous avons défini une action nommée door1 - une action qui pourra donc être attachée à une entité de porte, ce que nous allons faire maintenant. Pour créer une entité de porte vous-même ouvrez une nouvelle carte nommée mydoor.wmp, placez un cube simple, taillez-le à la forme d'une porte, donnez-lui une texture bois, déplacez-la pour quelle soit debout bien droite et l'origine de la carte placée à sa position de charnière et pour finir build (compilez-le) avec l'option Entity Map de cochée. Si vous ne voulez pas faire tout cela, employez simplement le prédéfini Porta.wmb du niveau office.

Vous avez maintenant une entité de carte que vous pouvez placer quelque part dans votre niveau de test pour WED. Faites-le. Puis, avec l'entité de porte dans WED toujours sélectionnée ouvrez Entity Properties (Propriétés d'Entités) et cliquez sur le champ action. Si vous avez tout fait correctement, vous pouvez maintenant choisir votre nouvelle action de door1 dans la liste. Recompilez le niveau en cochant Update Entites - qui compile beaucoup plus rapidement dans le cas où vous n'auriez changé rien d'autres que des entités. Exécutez-le maintenant par Run level. Si vous n'avez pas encore de joueur dans votre niveau, Vous pouvez basculer en mode de mouvement de caméra direct en appuyant la touche [0]. Si vous avez tout fait correctement, vous pouvez maintenant observer une porte s'ouvrant et se fermant de manière permanente, comme par la magie !

Comme vous vous souvenez, my est le pointeur pour l'entité à laquelle l'action est attachée - notre porte. La première instruction est while(1), ce qui signifie : Répétez pour toujours toutes les instructions entre les 2 accolades suivantes. Nous devons répéter les instructions parce que nous voulons que cette action soit exécutée de manière permanente pendant le jeu. Autrement, elle serait seulement exécutée une fois au début du jeu et serait terminée ensuite. Et la porte se déplacerait d'un petit pas et s'arrêterait ensuite pour toujours.

Dans la boucle infinie, nous faisons jouer un son en ouvrant la porte par l'instruction Play_entsound() et continuons avec une autre *boucle intérieure* (nous pouvons imbriquer des boucles comme vous le voyez). Cette boucle intérieure augmentera l'angle de la porte tant qu'elle est au-dessous de 90 degrés. La quantité pour augmenter l'angle est multipliée par time, la durée d'un cycle d'affichage, pour arriver à un incrément angulaire de la vitesse d'ouverture, indépendant du taux d'affichage.

Puis la porte s'arrêtera pendant une seconde. Nous employons ici l'instruction waitt(), qui attend un certain temps, à la différence de wait(), qui attend un certain nombre de cycles d'affichage. Après la pause nous faisons tourner la porte en arrière tant que son angle n'a pas atteint zéro et ensuite, après une autre pause, tout recommence.

Nous remarquons quelques défauts de notre action. La porte tourne toujours entre 0 et 90 degrés, c'est-à-dire Est et Nord, indépendamment de son orientation initiale. Et elle dépassera sa position finale de quelques degrés, en raison du fait, que le dernier incrément peut donner une valeur supérieure à 90 ou inférieure à zéro. Nous devons corriger cela :

```
action door1
{
	my.skill25 = 0; // compteur de degrés indépendant de l'angle initial
	while (1)
	{
		play_entsound(my,open_snd,66);
		while (my.skill25 < 90) {
		my.pan -= 3*time; // tourne dans le sens des aiguilles d'une montre
		my.skill25 += 3*time;
		wait(1);
	}
	my.pan += my.skill25-90; // correct the overshoot
	my.skill25 = 90;
	waitt(16);
	play_entsound(my,close_snd,66);
	while (my.skill25 > 0) {
	my.pan += 3*time; // tourne dans le sens contraire des aiguilles d'une montre
	my.skill25 -= 3*time;
	wait(1);
	}
		my.pan += my.skill25; // correct the overshoot
		my.skill25 = 0;
		waitt(16);
	}
}
```

La porte s'ouvre maintenant dans le sens des aiguilles d'une montre et se ferme en sens inverse des aiguilles d'une montre. Une variable d'entité, skill25, est employée pour compter l'angle d'ouverture, si la porte s'ouvre maintenant indépendamment de sa valeur pan initiale. Les entités ont comme cela 48 variables internes qui peuvent être employées dans des actions. Alors me direz-vous, pourquoi ne pouvons-nous pas définir juste une variable par une déclaration var pour le compteur ? Parce que nous pouvons avoir plus d'une porte dans notre niveau. Les variables que nous définissons sont généralement partagées par toutes les actions et les fonctions.

O.K., la porte se déplace maintenant. Mais comment pouvons-nous faire maintenant pour qu'elle réagisse avec nous, l'utilisateur ?

Interaction d'utilisateur

La porte à côté de la vue de caméra s'ouvrira si nous appuyons une touche sur le clavier. L'instruction scan (balayage) est faite pour cela. Scan déclenchera une fonction d'entités event (événement), s'il est dans le cône du balayage. Les fonctions event (d'événement) contrôlent une réaction des entités sur quelque chose se produisant. D'abord écrivons l'action qui exécute un scan (balayage) et l'assigne à une touche :

```
var indicator = 0;
var my_pos[3];
var my_angle[3];

function scan_me()
{
	my_pos.x = camera.x;
	my_pos.y = camera.y;.
	my_pos.z = camera.z;
	my_angle.pan = camera.pan;
	my_angle.tilt = camera.tilt;
	temp.pan = 120;
	temp.tilt = 180;
	temp.z = 200; // scan range - 200 quants
	indicator = 1; // this is for opening
	scan(my_pos,my_angle,temp);
}
```

La fonction entière met juste les vecteurs qui sont nécessaires comme les trois paramètres pour l'instruction scan. Vous devez lire la description de scan dans le manuel de référence pour comprendre ce que nous faisons. Nous avons défini 2 vecteurs pour stocker une position XYZ et un angle tridimensionnel. Un vecteur est juste une variable qui ne contient pas un mais trois nombres, indiqués par [3] après le nom de la variable. Les trois nombres sont accessibles par .X, .Y, .Z ou alternativement comme .pan, .tilt et.roll, les paramètres du vecteur. Temp est un vecteur prédéfini que nous employons pour des résultats intermédiaires - ici pour la largeur et le rayon d'action du cône de balayage. Nous avons mis le centre du cône de balayage à la position de caméra, sa direction à la direction de caméra, son rayon d'action à 200 quants et sa largeur à pratiquement un demi-cercle.

La variable indicator est mise à 1 comme un signal pour l'entité dont la fonction d'événement peut être déclenchée par l'instruction scan. Si indicator a été mis à 1 à ce moment là, l'entité sait que ce balayage était destiné à faire fonctionner une porte. Les instructions de balayage peuvent être employées pour beaucoup d'autres buts comme, pour des explosions par exemple et nous devons distinguer ici, pour empêcher qu'une grenade détonante ouvre soudainement toutes les portes dans le niveau.

Comment notre porte peut-elle détecter l'instruction scan (balayage) ?

```
define _counter skill25; // nous utilisons un nom significatif pour  SKILL25

function door_event()
{
	if (indicator != 1) { return; } // must be right kind of scan
	if (my._counter <= 0) // si la porte est fermée, on l'ouvre
	{
		play_entsound(my,open_snd,66);
		while (my._counter < 90)
		{
			my.pan -= 3*time; // tourne  dans le sens des aiguilles d'une montre
			my._counter += 3*time;
			wait(1);
		}
		my.pan += my._counter-90; // corrige le dépassement
		my._counter = 90;
	} else // autrement on la ferme
	{
		play_entsound(my,close_snd,66);
		while (my._counter > 0)
		{
			my.pan += 3*time; // tourne  dans le sens contraire des aiguilles d'une montre
			my._counter -= 3*time;
			wait(1);
		}
		my.pan += my._counter; // corrige le dépassement
		my._counter = 0;
	}
}


action door1
{
	my.event = door_event;
	my.enable_scan = on; // rend la porte sensible au balayage
	my._counter = 0; // compteur en degré, independant de l'angle initial
}
```

D'abord, nous avons employé la déclaration define pour donner le nom _counter à skill25 qui décrit pour quoi il est employé. Les scénarios de calibre (template) font une grande utilisation de cette façon de nommer des variables d'entité.

L'action de porte est maintenant scindée en deux, une action initialisante et une fonction d'événement. La porte est rendue sensible aux instructions de scan (balayage) et la fonction d'événement est validée. Cette action vérifie d'abord s'il s'agit d'un scan ouvrant ou autre chose. Si c'était le cas, il vérifiera si c'était déjà fermé ou ouvert et ouvrira ou

fermera respectivement ensuite. Tout ce que nous devons faire maintenant est d'attacher la fonction scan_me à une touche :

```
on_space = scan_me;
```

Maintenant nous devons marcher à moins de 200 quants de la porte et appuyer ensuite la touche [Espace].

Prêt pour plus ? Il s'agit de la clef qui nous permettra d'ouvrir la porte ? Faisons en sorte que la clef mette à 1 une certaine variable lorsque nous l'avons et laissons la porte vérifier cette variable avant l'ouverture. D'abord nous devons définir une action pour une entité modèle qui sera un article clef :

```
sound key_fetch = <beamer.wav>;
var key1 = 0;


function key_pickup()
{
	if (indicator != 1) { return; } // must be right kind of scan
	key1 = 1;
	play_sound(key_fetch,50);
	ent_remove(my);
}

action key
{
	my.event = key_pickup;
	my.enable_scan = on; // pick up by pressing SPACE..
}
```

Aussitôt que nous sommes près de la clef et appuyons la touche [Espace], elle disparaîtra, le son sera joué et la variable key1 sera mise à 1. De cette façon les articles peuvent être pris et influencer le jeu. La porte doit maintenant vérifier la variable key1 sur l'ouverture :

```
function door_event()
{
	if (indicator != 1) { return; } // must be right kind of scan
	if (key1 != 1) { return; } // key must have been picked up already
	... // and so on...
```

À propos, l'entité clef pourrait bien sûr avoir la forme d'un bouton ou d'un levier et nous pourrions modifier ses paramètres d'animation ou de peau pour montrer son activation, au lieu de la faire disparaître par ent_move. Bien entendu nous pouvons utiliser plusieurs variables identiques à key1, afin de manipuler des clefs différentes qui ouvrent des portes différentes.

Et comment pouvons-nous faire pour que la porte s'ouvre si une entité s'en approche tout simplement ? C'est du gâteau : Chaque entité de joueur pourrait exécuter une instruction de scan répétée (une fois par seconde – pas plus, car le scan est lent). Ou nous pourrions employer event_trigger au lieu de ou en plus de l'événement scan. Nous le laisserons comme exercice au lecteur - jusqu'à mardi.

Mardi : Physique de Jeu

Pour vous donner un exemple de la physique simple employée pour des objets dans un jeu, nous allons maintenant créer une des fonctions les plus communes : le mouvement de l'entité de joueur.

Pour sûr, vous n'avez pas besoin de le faire, parce que vous pouvez employer l'action prédéfinie player_move avec sa quantité d'options. Mais si vous voulez devenir un maître en C-SCRIPT, il arrivera un moment ou vous devrez écrire vos propres fonctions de mouvement pour les entités spéciales de votre jeu - et pour ce faire vous devez connaître la physique de jeu. Ça va devenir un peu mathématique, mais n'ayez pas peur : les quatre principes de base de l'arithmétique seront suffisants. Nous ne présentons pas la physique réelle ici. Nous employons une forme simplifiée de physique, particulièrement bienvenue pour un jeu temps réel où des centaines d'acteurs exécutent des mouvements arithmétiques qui doivent être aussi rapide et simple que possible. Vers la fin de la journée vous serez capables d'infuser des entités avec n'importe quelle sorte de comportement de mouvement à travers vos propres fonctions. Et même mieux, vous n'avez pas besoin attendre jusqu'à mardi pour commencer cette leçon…

Ouvrez votre fichier de niveau C-SCRIPT. Cette fois nous n'allons pas nous servir des fonctions de mouvement de joueur prédéfinies dans movement.wdl. Cette fois nous allons écrire la notre propre.

```
action move_me
{
	while (1)
	{
		my.x += 10 * key_force.x;
		my.y += 10 * key_force.y;
		move_view();
		wait(1);
	}
}
```

Les deux premières instructions dans la boucle while assigne des nouvelles valeurs à my.x et my.y. My est l'entité à laquelle nous assignons l'action. My.x et my.y sont les coordonnées X et Y de notre position, utilisant le même système de coordonnées que nous avons employé en construisant la carte dans WED. Changer ces coordonnées génère un mouvement ou plus précisément, la téléportation de l'entité. Nous changeons la position en ajoutant une expression arithmétique; l'addition est faite par le + = et l'expression est 10*key_force.x et 10*key_force.y. Autrement dit, "multipliez key_force par 10 et ajoutez le résultat à la position des entités ".

Les coordonnées X et Y de la variable vectorielle prédéfinie key_force contiennent les valeurs numériques correspondant à notre entrée par les touches de curseur. Tant que vous appuyez la touche [Haut] (la flèche qui monte), Key_force.y prend la valeur 1 et tant que vous appuyez la touche [Bas] (la flèche qui descend), la valeur est -1. Ainsi en appuyant une des touches de direction, autant de fois la valeur sera ajoutée ou soustraite de la position des entités, déplaçant ainsi l'entité le long de la direction X ou Y du système de coordonnée.

L'instruction suivante exécutera une fonction prédéfinie séparée - move_view - pour déplacer la vue de la caméra à la position des entités. Nous n'analyserons pas la fonction move_view ici, mais nous sommes néanmoins reconnaissant qu'elle existe - elle est placée dans le fichier de calibre (template) movement.wdl. La dernière instruction, wait(1), suspend cette action jusqu'au cycle d'affichage suivant, où toutes les instructions seront répétées de nouveau en raison du while(1). Si vous aviez oublié l'instruction wait(), l'action ne sera jamais Interrompue et ainsi le cycle d'affichage ne finira jamais. Ce serait mauvais.

Assez de théorie. Essayons ce que nous avons écrit. Si WED est déjà ouvert, choisissez Map Properties, et la re-sélectionner le même fichier de scénario - nous devons le faire parce que nous avons ajouté une nouvelle action d'entité. Placez une entité de la taille d'un joueur - guard.mdl par exemple - dans le niveau ouvrez Entity Properties et cliquez le champ action. Si vous avez tout fait correctement, vous pouvez maintenant choisir votre nouvelle action move_me dans la liste. Recompilez le niveau en utilisant Update Entities - qui compile beaucoup plus rapidement dans

le cas où vous n'auriez rien changé sauf des entités. Choisissez maintenant Run Level, et essayez les touches de direction pour marcher.

Hmm. Bien, mais admettez que votre première action de mouvement ne se comporte pas aussi bien que l'action prédéfinie. Ce qui arrive est que le mouvement ne suit pas la ligne de la vue de joueur- Il suit directement les axes des X et des Y du système de coordonnée. En fait, vous ne pouvez pas faire tourner le joueur du tout. Pas étonnant, vous n'avez pas fourni d'instruction pour cela. Et même plus mauvais encore, le joueur ne sera pas arrêté par les murs, il passera directement à travers.

Réglons cela en premier. Pour obtenir la détection de collision dans votre mouvement d'entité, vous ne devez pas changez les coordonnées d'entité directement - qui téléporte l'entité à sa nouvelle position - mais utiliser l'instruction de mouvement pour un mouvement continu sur une distance donnée :

```
var dist[3];

action move_me
{
	while (1)
	{
		dist.x = 10 * key_force.x;
		dist.y = 10 * key_force.y;
		dist.z = 0;
		move(my,nullvector,dist);
		move_view();
		wait(1);
	}
}
```

Nous avons défini un vecteur, dist, qui stocke la distance à couvrir. Un vecteur est juste une variable qui contient trois nombres, qui sont employés ici pour les coordonnées X, Y et Z. Nous mettons sa coordonnée Z à zéro pour empêcher le joueur de voler soudainement dans l'air. L'instruction move my…change la position du joueur de la même façon que si nous avions écrit :

```
my.x += dist.x;
my.y += dist.y;
my.z += dist.z;
```

La seule différence est que par move, l'entité my - notre joueur - exécute une détection de collision sur la distance donnée par dist. Move qui nécessite deux distances vectorielles, nous avons mis la première à zéro en utilisant nullvector prédéfini. La première distance est tournée d'après les angles des entités, La deuxième (employée ici) ne l'est pas.

L'utilisation de la première distance tournée, au lieu de la seconde qui ne tourne pas, nous donnerait l'occasion de faire tourner la direction de mouvement dans la ligne de vue du joueur et d'utiliser key_force.x pour changer son angle :

```
action move_me
{
	while (1)
	{
		my.pan -= 10 * key_force.x;
		dist.x = 10 * key_force.y;
		dist.y = 0;
		dist.z = 0;
		move(my,dist,nullvector);
		move_view();
		wait(1);
	}.
}
```

Notez s'il vous plaît que nous employons maintenant une distance tournée move, tandis que la distance non tournée est mise à zéro en employant le nullvector. dist.x - qui donne la direction en avant – et qui est maintenant changé par Key_force.y.

Maintenant nous sommes capables de faire tourner le joueur et de le déplacer dans chaque direction. Mais le mouvement semble être toujours saccadé et artificiel. Et notre vitesse dépend du taux d'affichage - sur des PC plus rapides L'instruction move est exécutée plus souvent, donc le joueur couvre une plus grande distance dans le même Temps. Cela n'est pas viable. Nous allons essayer maintenant de représenter une formule qui réalise des mouvements lisses et naturels.

Accélération, Inertie, Friction

Supposons que notre joueur se déplace tout droit à une vitesse constante. La distance qu'il parcourt augmente, selon sa vitesse, avec le temps :

ds = v * dt

> ***v*** = Vitesse en quants par tick
> ***ds*** = Distance en quants
> ***dt*** = Temps en ticks

Dans notre monde virtuel - comme vous vous souvenez - les distances sont mesurées en quants (environ 1 pouce) et le temps est mesuré en tick (environ 1/16de seconde). C'est pourquoi l'unité de mesure pour la vitesse est quants par tick.

Mais qu'arrive-t-il si nous voulons changer la vitesse ? Nous devrons causer une force qui fera impact sur le joueur. Plus grande est la force, plus grand est l'impact qu'il a sur la vitesse par unité de temps. Encore, chaque corps montre une certaine résistance contre de tels changements. Cette résistance, appelée inertie, est le résultat de la masse du corps. Plus la masse du corps est importante, plus petit le changement de vitesse sera – en supposant une force constante. Nous pouvons exprimer cela dans des formules :

dv = a * dt
a = F / m

dv = Change de vitesse
a = Change de vitesse par tick (accélération)
F = Force
m = Masse

Il y a trois types de base de forces avec lesquelles nous devrons compter dans notre monde virtuel. Pour commencer il y a la force de propulsion. C'est une variation de vitesse volontaire qui peut par exemple être incitée par le clavier et le mouvement de levier de commande. Dans notre environnement de jeu simplifié cette force sera proportionnelle à la masse du joueur : Plus grand est un type, plus sera la force qu'il s'appliquera pour accélérer.

P = p * m

Nous appellera la deuxième force la dérive. Cela peut être un courant qui porte le joueur dans une certaine direction ou la gravité, le tirant vers le bas. Les forces de dérive peuvent varier d'un endroit à l'autre. Dans notre monde de jeu, La quantité de la dérive est aussi proportionnelle à la masse du joueur. C'est évident pour la gravité, mais peut aussi être rapproché pour les dérives normales qui appliquent plus de force à un objet s'il a plus de volume.

D = d * m

La troisième force ayant un effet sur le joueur est la friction. Cette force essaye continuellement de le ralentir. À la différence de la physique réaliste, où la force de ralentissement consiste en une part de friction et en une part d'atténuation, nous employons une force artificielle qui augmente avec la masse du joueur (la pression sur le sol) et la vitesse :

R = - f * m * v

R = force de friction
f = facteur de friction
v = vitesse
m = masse

Le coefficient de friction f dépend de la nature de la surface sur laquelle le joueur se déplace : la glace ayant un coefficient plus petit de friction que la pierre. S'il est aéroporté, la friction aérienne est presque à zéro. L'indicateur négatif est supposé montrer que cette force résiste à la vitesse. La masse m est aussi une partie de l'équation, comme dans notre monde de jeu le joueur rencontrera une friction plus grande sur la surface s'il pèse plus. Ces trois forces - la force F de propulsion, la dérive D, et la décélération R – s'ajoutent pour changer la vitesse du joueur:

dv = a * dt = (P + D + R) / m * dt = (p + d - f * v) * dt

Ainsi dv doit être ajouté à la vitesse à chaque frame. p, d, et f sont ici les facteurs de la propulsion, la dérive et la friction. Dt est le temps pendant lequel l'accélération a changé la vitesse - c'est ici le temps entre deux cycles d'affichage, parce que nous calculons une nouvelle vitesse à chaque cycle d'affichage. Comme vous pouvez voir, en mettant toutes nos forces à être la masse proportionnelle, la masse est enlevée de l'équation, n'ayant aucune influence sur le mouvement dans la physique d'acteur. Nous pourrions traduire la dernière formule dans une fonction correspondante (il n'y a aucune dérive encore) :

```
var force[3];
var dist[3];

action move_me
{
	while (1)
	{
		force.pan = -10 * key_force.x; // calcule la force de rotation
		my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill14;//vitesse de rotation
		my.pan += time * my.skill14; // tourne le joueur

		force.x = 10 * key_force.y; // calculela force de translation
		my.skill11 = time*force.x + max(1-time*0.7,0)*my.skill11; // calcule la vitesse
		dist.x = time * my.skill11; // distance à couvrir
		dist.y = 0;
		dist.z = 0;
		move(my,dist,nullvector); // déplace le joueur

		move_view(); // déplace la camera
		wait(1);
	}
}
```

Nos facteurs de force sont 10 ici, nos facteurs de friction 0.7 et pour la rotation et pour la translation. Nous devons utiliser des variables d'entité pour les vitesses, parce qu'elles doivent être préservées ensemble avec l'entité; la vitesse précédente est nécessaire pour notre formule. Skill14 stocke la vitesse angulaire et skill11 la vitesse avant.

Time est le dt de la formule, le temps du dernier cycle d'affichage. Nous employons maintenant la formule mathématiquement correcte pour calculer la distance – à laquelle le joueur se déplace maintenant avec la même vitesse sur chaque PC, presque indépendant du taux d'affichage ! Notez que nous avons converti notre formule originale théorique qui donne le changement de vitesse par cycle d'affichage.

v -> v + dv
v -> v + (p - f * v) * dt

qui peut s'écrire dans C-SCRIPT

```
my.skill11 = my.skill11 + (force.x - 0.7 * my.skill11) * time;
```

et transformée à travers les étapes suivantes

```
my.skill11 = my.skill11 + time * force.x - time * 0.7 * my.skill11;

my.skill11 = time * force.x + (1 - time * 0.7) * my.skill11;
```

et pour terminer quelque chose d'un peu plus compliqué

```
my.skill11 = time * force.x + max(1 - time * 0.7,0) * my.skill11;
```

Pourquoi cela et qu'est-ce que max ? Max (a, b) donne le plus grand nombre de a ou b. A travers max(1- Time*0.7,0) nous limitons le résultat à des valeurs positives. Nous devons le faire, parce qu'autrement sur de très bas taux d'affichage - et des hautes valeurs times - time*0.7 peut être plus grand que 1, donnant des résultats négatifs. Cela changerait complètement la vitesse - le joueur se déplacerait en arrière ! Des différences si subtiles entre l'ordinateur simulé et la réalité réelle doivent toujours être considérées. (Normalement, même un programmeur de jeu expérimenté ne remarquerait pas un tel effet avant qu'une action ne se comporte différemment de celle attendue).

Nous nous déplaçons maintenant de façon beaucoup plus fluide et naturelle grâce à la nouvelle action de mouvement. Nous sommes capables d'accélérer doucement et de ralentir de la même façon. Mais qu'arrive-t-il si le joueur rencontre un escalier ou un précipice ?

Chute

Tandis que nous déplaçons notre joueur horizontalement en pressant des touches, son mouvement vertical sera causé par son environnement. S'il est debout sur la terre ferme, il ne doit exécuter aucun mouvement vertical du tout. Mais s'il flotte soudainement en l'air - cela peut être le cas s'il devait marcher sur le bord d'un abîme - il doit tomber. Pour cela nous devons déterminer sa hauteur par rapport au sol. Nous employons la l'instruction trace() pour cela.

```
action move_me
{
	while (1)
	{
		force.pan = -10 * key_force.x; // calcule la force de rotation
		my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill14;// calcule la vitesse de rotation
		my.pan += time * my.skill14; // tourne le joueur
		my.skill11 = time*force.x + max(1-time*0.7,0)*my.skill11; // calcule la vitesse avant
		dist.x = time * my.skill11; // distance avant
		dist.y = 0;

		vec_set(temp,my.x);
		temp.z -= 4000; // calcule une position 4000 quants sous le joueur
		//Mettre un mode trace pour employer la coque du joueur et détecter les entités de carte et les surfaces de niveau seulement
		trace_mode = ignore_me+ignore_sprites+ignore_models+use_box;
		dist.z -= trace(my.x,temp); // soustrait la distance verticale par rapport au sol

		move(my,dist,nullvector); // déplace le joueur
		move_view(); // déplace la camera
		wait(1);
	}
}
```

trace() retourne la distance à l'obstacle suivant le long d'un rayon entre deux positions. Nous employons le centre du joueur (my.x) comme la première position et mettons le vecteur temp à une deuxième position quelque mille quants plus bas. Par l'intermédiaire du mode trace use_box nous donnons au rayon de trace une ''épaisseur' à la coque du modèle de joueur – qui est normalement 32 quants de diamètre - et retournons aussi la distance non pas par rapport au centre du joueur, mais par rapport à sa boîte de limitation. Ce sont ses pieds dans ce cas, parce que nous traçons verticalement en bas. Donc trace() calcule la distance entre les pieds du joueur et le premier niveau ou surface de carte en dessous. Si ses pieds sont au-dessous du niveau du plancher, trace() rend logiquement une valeur négative.

Nous avons pris la voie facile en déplaçant simplement le joueur verticalement de la distance exacte entre ses pieds et la terre. Ainsi dist.z est mis à la distance verticale à couvrir. Nous sommes maintenant capables de marcher dans des régions de différentes hauteurs avec notre joueur, par exemple des escaliers. S'il vous plaît notez que par simplicité

nous avons employé la distance relative pour se déplacer dans la direction Z. Normalement nous aurions dû employer la distance absolue ici, mais tant que le joueur n'est pas incliné, cela n'a aucune importance.

Malheureusement le joueur s'adapte maintenant aux hauteurs différentes aussi anormalement et saccadé qu'avec notre tentative initiale de mouvements en avant. Pour atteindre des mouvements plus doux et plus naturels de nouveau nous devrons jeter un coup d'œil aux forces qui ont un impact aux mouvements verticaux du joueur.

- Tant que le joueur est aéroporté - c'est-à-dire que trace() retourne une valeur au-dessus de 0 - seules la gravitation et la friction aérienne ont un impact sur lui. La gravitation est une dérive que nous avons déjà rencontrée : il donne au joueur une accélération vers le bas...

- -. .. avant que le joueur touche le sol ou s'enfonce dedans - c'est-à-dire que trace() retourne une valeur inférieure ou égale à 0. Dans ce cas une force de *résistance additionnelle* entre en vigueur. C'est une nouvelle sorte de force qui devient plus forte lorsque le joueur s'enfonce plus profond dans la terre. Il l'incite avec une accélération ascendante. En outre la friction augmente significativement quand la terre est pénétrée.

Selon le centre des entités de joueur, les pieds du joueur sont même capables de pénétrer dans les surfaces de roche dure, impénétrables! Dans le monde réel cette sorte de surface ne donnerait jamais bien sûr de voie, mais les articulations du genou du joueur, peuvent produire les mêmes effets. Ainsi la force de résistance résultant de l'élasticité de ses joints ou - si le joueur est motorisé - la suspension de son véhicule. Nous pouvons également expliquer une friction de la terre accrue par la friction de ses muscles ou des chocs absorbants du véhicule.

```
var friction;

action move_me

{
	while (1)
	{
		force.pan = -10 * key_force.x; // calcule la force de rotation
		my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill14;// vitesse de rotation
		my.pan += time * my.skill14; // tourne le joueur
		vec_set(temp,my.x);
		temp.z -= 4000; // calcule une position 4000 quants sous le joueur
		//Mettre un mode trace pour employer la coque du joueur et détecter les entités de carte et les surfaces de niveau seulement
		trace_mode = ignore_me+ignore_sprites+ignore_models+use_box;
		result = trace(my.x,temp);// soustrait la distance verticale par rapport au sol
		if (result > 5) // en l'air?
		{
			force.x = 0; // pas de force poussante
			force.y = 0;
			force.z = -5; // force de gravité
			friction = 0.1; // friction de l'air
		} else // sur ou sous le sol
		{
			force.x = 10 * key_force.y; // force de translation
			force.y = 0;
			force.z = -0.5 * result; // elasticité du sol
			friction = 0.7; // friction du sol
		}
		my.skill11 = time*force.x + max(1-time*friction,0)*my.skill11; // calcule la vitesse avant
		my.skill13 = time*force.z + max(1-time*friction,0)*my.skill13; // calcule la vitesse verticale
		dist.x = time * my.skill11; // distance avant
		dist.y = 0;
		dist.z = time * my.skill13; // distance bas

		move(my,dist,nullvector); // déplace le joueur
		move_view(); // déplace la caméra
		wait(1);
	}
}
```

Par l'instruction **if** nous sommes capables de distinguer si le joueur est aéroporté ou 5 quants sous terre. Dans le premier cas il ne sera pas poussé par les touches de direction, mais tiré vers le bas par une force de gravité constante.

Dans le dernier cas une force résistante, proportionnelle (en négatif) à la profondeur de l'immersion (result de trace ()), essaye de le sortir de la terre. Nous employons une variable d'entité, skill13, pour stocker la vitesse verticale.

Bien - maintenant nous pourrions ajouter des particularités plus sympa à notre action. Nous pourrions employer plus de touches pour le laisser regardez en haut et en bas ou sautez ou se baissez. Notre scénario deviendrait de plus en plus grand, à atteindre la taille de l'action Player_move prédéfinie dans movement.wdl. Nous nous quittons maintenant. Au moins jusqu'à mercredi.

Mercredi : Intelligence artificielle

La chose la plus stimulante dans le développement d'un jeu est comment définir le comportement intelligent pour les adversaires. C'est ainsi, qu'avant que nous en ayons terminé vraiment avec ce tutorial, nous nous serons occupé nous-mêmes un peu de la création de vie artificielle intelligente.

Pour peupler notre monde avec des créatures vivantes, nous nous servons normalement d'entités modèles. Les entités modèles sont ces objets qui sont capables d'être animés et sont normalement employés pour des acteurs, des adversaires ou des monstres. Des monstres voulant être pris au sérieux ont besoin de se comporter d'une façon 'réaliste'. Ils doivent réagir aux joueurs de façon intelligentes, lui échapper tant qu'il est le plus fort et le pourchasser dés qu'il montre des signes de faiblesse. Une sorte d'intelligence artificielle est nécessaire pour cela.

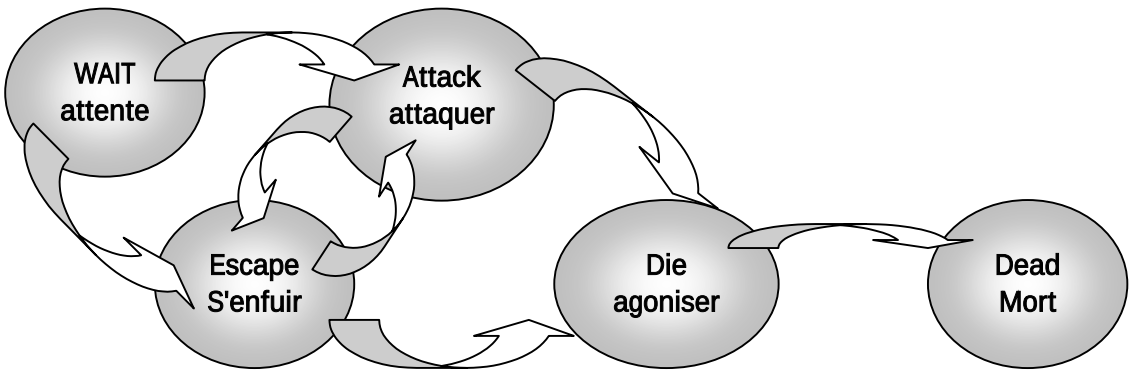
Dans ce chapitre du tutorial vous apprendrez à donner à vos créatures une personnalité électronique. Nous ferons une excursion à la théorie de machines d'état par conséquent. À la fin du chapitre vous serez capable de créer des êtres électroniques qui dans leur complexité sont comparable aux organismes simples.

La théorie de boîtes noires

Le terme d'Intelligence artificielle est synonyme des méthodes qui donnent aux machines la capacité de prendre des décisions sensibles. Si notre but est atteint peu importe si l'intelligence de ces machines reste primaire. Le but est de les faire se comporter aussi intelligemment que possible. L'intelligence des machines a ouvert la voie à de nombreuses discussions, et bien que cela soit sans doute très intéressant cette discussion n'a pas la moindre influence sur les scénarios C-SCRIPT.

Comme la structure interne de la machine n'a aucun intérêt pour nous pour l'instant, nous la considérerons comme une boîte noire qui est définie seulement par un comportement observable, c'est à dire des actions et des réactions. Si le comportement de notre machine est caractérisé par une certaine simplicité il peut être décrit comme une machine d'état. Le comportement d'une machine d'état peut se décomposer en un certain nombre de comportements distincts. Chacun de ces modèles comportementaux égale un état intérieur. Ainsi la machine d'état a un nombre limité d'états disponibles; elle est toujours dans un de ces états. Les états dans ce cas constituent 'la vie intérieure ' de la boîte noire. Pour chaque état il existe des circonstances, Par exemple un stimulus externe, qui fera que la machine passe dans un état différent.

Par exemple, un acteur dans un jeu de tir pourrait avoir les états comportementaux suivants :
Wait**i**ng (Attente), Attack**i**ng (Attaque), Escap(e)ing (Fuite), D (ie) ying (agonisant) et Dead (Mort).



Les flèches indiquent les transitions entre les états. Ces transitions peuvent être déclenchées par des événements dans les scénarios C-SCRIPT. Voici une liste des événements qui peuvent causer la réponse d'un acteur à un changement d'états :

Evénement	Fait par
L'acteur découvre le joueur	Fonction avec l'instruction trace() périodique et évaluation du résultat (result). Soyez prudent, trace() est lent !
Le joueur vient près de l'acteur (ou vice versa).	Event_trigger
Joueur et acteur en contact	Event_entity, event_impact
Le joueur s'est éloigné de l'acteur au-delà d'une certaine distance	Fonction qui calcule périodiquement la distance à l'entité de joueur
L'acteur est près d'un objet qui explose	Event_scan
Le joueur frappe l'acteur avec l'instruction trace().	Event_shoot
Un cycle d'animation du modèle acteur a passé	La fonction qui compare périodiquement le paramètre d'affichage (frame)
Une variable qui est changée par une autre fonction a obtenu une certaine valeur	La fonction qui compare périodiquement cette variable
Un certain temps a passé	La fonction qui exécute un waitt () ou déclenche le compte à rebours d'une variable d'entité
Evénement aléatoire	La fonction qui compare périodiquement une expression contenant random ().

Bien entendu chacun de ces événements peut être combiné avec n'importe quel autre. Ou l'action principale des entités ou sa fonction event est responsable de la transition entre des états.

A part des états nous sommes aussi capables de définir des variables intérieures pour la machine qui peut acquérir des valeurs différentes et influencer le comportement. Nos variables d'entité (skill) sont des variables intérieures. Leurs valeurs peuvent être utilisées dans une fonction et décider par exemple finalement de basculer dans un autre état.

La table d'états

La division du comportement de notre machine d'états et de transitions nous permet de tenir l'action d'acteur quelque peu transparente. Pour donner un exemple nous définirons maintenant notre premier état d'acteur de machine. Pour cela nous allons nous servir d'un des robots du jeu «Mission».

Avant l'écriture de la première ligne du scénario C-SCRIPT nous allons créer un concept théorique. Quelle sorte d'états notre robot peut avoir, comment allons nous distinguer ces états les uns des autres, quelles variables et quelles transitions avons-nous besoin ?

Au départ du jeu le robot est dans un état d'attente (wait). Cela signifie qu'il se cachera dans un coin sombre attendant le joueur. Si le joueur s'approche, il passera à l'état attaque Attack (). Si le joueur lui tire dessus avec son arme à feu, la transition suivante dépendra de la santé restante du robot, c'est-à-dire le nombre de coups qu'il avait déjà été forcé de prendre : l'état suivant sera Attaquer, s'Enfuir ou Agoniser (Attack, Escape or Die).

Etat	Joueur près de	Santé > 30	Santé < = 30	Santé < = 0	Agonie terminée
Attendre	Attaquer	Attaquer	S'enfuir	Agoniser	Mort

Pour tenir la trace de la santé nous emploierons une des variables d'entité du robot. Pour une meilleure transparence nous résumons les états dans une table d'états. Nous avons défini cinq événements pour le robot qui peuvent changer son état. Trois d'entre eux dépendent d'une variable de santé interne. Nous pouvons maintenant ajouter le maintien d'états à la table.

Etat	Joueur près de	Santé > 30	Santé < = 30	Santé < = 0	Agonie terminée
Attendre	Attaquer	Attaquer	S'enfuir	Agoniser	Mort
Attaquer	Attaquer	Attaquer	S'enfuir	Agoniser	Mort
S'enfuir	S'enfuir	S'enfuir	S'enfuir	Agoniser	Mort
Agoniser	Agoniser	Agoniser	Agoniser	Agoniser	Mort
Mort	Mort	Mort	Mort	Mort	Mort

Notre robot est une machine assez simple : si un comportement plus complexe est exigé notre table contiendra beaucoup plus de colonnes et de lignes. En y regardant de plus près vous pourriez remarquer que quelques-unes des transitions inutiles ont été définies dans la table. Par exemple la transition entre Attendre et Mort ne sera jamais faite parce que le robot passe toujours par la phase d'agonie. Mais parce que sur des machines à états beaucoup plus complexe des cas spécifiques comme celui-ci ne sont pas toujours si faciles à voir, il est important de définir la table aussi complètement que possible.

Une fois que la table d'états est finie démarrons la programmation de C-SCRIPT. Nous pourrions écrire tout ensemble dans une grande action ou définir des fonctions différentes pour chaque état. Nous choisissons la dernière approche - elle offre l'avantage que chaque fonction peut être réutilisée pour des machines différentes.

```
action my_robot
{
	my._walkframes = 1;
	my._entforce = 0.7;
	my._armor = 100;
	my._health = 100;

	my.enable_scan = on;		// sensible aux explosions
	my.enable_shoot = on;		// sensible aux coups de feux
	my.event = myfight_event;	// manipule les coups

	mystate_wait();			// First state: watch for the player
}
```

Dorénavant nous ferons une utilisation lourde des fonctions d'entité prédéfinies et des variables de Movement.wdl et actors.wdl. D'abord nous donnons le nombre de frame pour la marche à pied, la course, l'attaque et les cycles d'agonie pour notre robot, enemy1.mdl. Tous les paramètres commençant par un caractère souligné (_) sont des variables d'entité, que l'on donne aux noms définis (define) par mots clés dans movement.wdl.

Maintenant nous devons définir la fonction event, qui doit gérer les coups et les fonctions mystate_wait, mystate_attack, mystate_escape, et mystate_die, qui représentent les états différents de la machine. Pour l'état mort nous n'avons pas besoin d'une fonction : on suppose qu'un ennemi mort n'a plus de mouvement désormais.

Commençons par la fonction agonie. C'est la plus simple :

```
function mystate_die()
{
	my._animdist = 0;	// le temps d'animation d'entité remise à zéro
	while (my._animdist < 100)
	{
		ent_frame("death",my._animdist); // set the frame from the percentage
		my._animdist += 5 * time; // calcule le pourcentage suivant pour la mort dans ~1.25 secondes
		wait(1);
	}
	my.enable_scan = off; // devient insensible
	my.enable_shoot = off;
	my.event = null; // et ne réagit plus jamais paix à son âme
```

```
}
```

C'est juste l'animation. My_animdist donne le pourcentage de l'animation d'agonie du modèle. Il est d'abord mis à 0, puis augmenté de manière permanente dans la boucle d'animation avant qu'il n'ait atteint 100 %. L'instruction ent_frame bascule par les frame d'animation qui commencent par le nom "death". Pour tester l'action, ajoutons un simple mystate_wait et une fonction event qui ne fait rien, mais reçoit des coups:

```
function myfight_event()
{
	if (((event_type == event_scan) && (indicator == _explode))
		|| ((event_type == event_shoot) && (indicator == _gunfire)))
	{
		if (my._armor <= 0) { my._health -= damage; }
		else { my._armor -= damage; }
	}
}

function mystate_wait()
{
	while (1)
	{
		if (my._health <= 0) { mystate_die(); return; }
		wait(1);
	}
}
```

Faites attention à bien écrire cette fonction avant l'action my_robot, parce que cette dernière a besoin de la première. La fonction event décrémentera la valeur de la variable d'entité armure si une explosion ou un coup de feu touche notre acteur. Les variables indicator et damage sont mises par la fonction d'arme correspondante. Comme vous l'avez appris hier, indicator est employé pour distinguer l'événement balayage ou tir d'autres balayages qui ne sont pas employés pour des explosions, mais pour d'autres buts. Et damage représente la force de l'arme. S'il n'y a aucune armure restante, la santé sera réduite par les nouveaux coups jusqu'à ce que la fonction mystate_wait n'amorce l'agonie si la santé a atteint le zéro.

Placez maintenant le robot et une arme dans votre niveau. Vous pouvez construire une nouvelle arme ou employer Sparkgun ou flashgun de weapons.wdl. Assignez my_robot à l'acteur, marchez dans le niveau, prenez l'arme à feu et commencez à tirer sur le robot impuissant. Si vous avez tout fait correctement, après environ 10 coups de Sparkgun il mordra la poussière.

Bien, il n'est pas juste de tuer un acteur qui ne peut pas riposter. Pour cela il doit nous détecter d'abord. Nous emploierons l'instruction de balayage (scan) pour observation. Le joueur a mis enable_scan si l'action de player_move est employée, Donc le robot peut employer event_detect pour voir s'il a balayé le joueur.

```
function myfight_event()
{
	if (((event_type == event_scan) && (indicator == _explode))
		|| ((event_type == event_shoot) && (indicator == _gunfire)))
	{
		my._signal = 1; // en tirant le joueur donne également sa position
		if (my._armor <= 0) { my._health -= damage; }
		else { my._armor -= damage; }
	}
	if ((event_type == event_detect) && (you == player)) {
	my._signal = 1; }
}


function mystate_wait()
{
	while (1)
	{
		if (my._health <= 0) { mystate_die(); return; }
		if (my._health < 30) { mystate_escape(); return; }

		// scan for the player
```

```
			temp.pan = 180;
			temp.tilt = 180;
			temp.z = 1000;
			indicator = _watch;
			scan(my.x,my.pan,temp);
			if (my._signal == 1) { mystate_attack(); return; }
			waitt(8);
		}
}


action my_robot
{
		my._walkframes = 1;
		my._entforce = 0.7;
		my._armor = 100;
		my._health = 100;
		my._signal = 0; // joueur non encore détecté

		my.enable_scan = on; // sensible of explosions
		my.enable_shoot = on; // sensible of gunshots
		my.enable_detect = on; // sensible of the player
		my.event = myfight_event; // handle hits & detection

		mystate_wait(); // first state: watch for the player
}
```

Ici nous nous servons d'une variable d'entité nous avons nommé my._signal (_signal est défini dans le Movement.wdl) pour permettre aux différentes actions d'entité d'échanger des messages simples - dans ce cas, que l'entité a détecté le joueur. La fonction mystate_wait parcourt pour le joueur dans un rayon de 1000 quants devant le robot. Nous avons donné une valeur à la variable indicator pour être certain que notre balayage ne soit pas pris pour une explosion ou une ouverture d'une porte.

En changeant wait(1) par waitt (8) nous avons réduit le taux de boucle dans la fonction mystate_wait à seulement deux instructions de balayage par seconde. Le balayage (scan) est lent et des balayages simultanés de centaines de robots peuvent réduire considérablement le taux d'affichage. Le robot aura maintenant besoin de la moitié d'une seconde pour détecter le joueur. C'est son temps de réaction.

Notez s'il vous plaît qu'en utilisant le balayage (scan) le joueur peu détecter à travers les murs. Pour empêcher cela, une instruction trace() doit suivre pour vérifier si le joueur est vraiment visible (nous l'avons omis ici, mais vous le trouverez dans le WAR.wdl). Chaque entité trouvée avec enable_scan donnera un événement detect, mais seul le joueur mettra my._signal à 1, qui est le signal pour l'attaque :

```
function mystate_attack()
{
		my._animdist = 0;
		while (1)
		{
			if (my._health <= 0) { mystate_die(); return; }
			if (my._health < 30) { mystate_escape(); return; }.

		// turn towards player
			temp.x = player.x - my.x;
			temp.y = player.y - my.y;
			temp.z = player.z - my.z;
			vec_to_angle(my_angle,temp);
			force = 4; // set rotation speed
			actor_turn(); // rotate towards my_angle

			ent_cycle("attack", my._animdist);
			my._animdist += 5 * time;
			if (my._animdist > 100) {
				my._animdist -= 100;
		// fire at player at end of animation cycle
			shot_speed.x = 100;
			shot_speed.y = 0;
```

```
                shot_speed.z = 0;
                my_angle.pan = my.pan; // tilt is already set from vec_to_angle
                vec_rotate(shot_speed,my_angle);
        // now shot_speed points ahead, towards the player
                damage = 20;
                fire_mode = 303.2; // orange fireball, with smoke
                create(fireball,my.x,bullet_shot);
                }
                wait(1);
        }
}
```

Cette fonction contient quelques calculs 3-D. Quand le robot a détecté le joueur, il doit se tourner vers lui et commencer à lui tirer dessus. D'abord la direction du robot au joueur est calculée. Nous employons la variable temp comme vecteur de direction -qui est la distance directe entre deux points dans l'espace. Comme vous voyez, un vecteur de direction peut être calculé simplement en soustrayant les trois composants vectoriels de la position de départ de la position cible. Nous ne ferons pas de cours sur l'arithmétique vectorielle ici - si vous ne me croyez pas, vous êtes libres d'acheter un livre d'algèbre linéaire.

L'étape suivante convertit cette direction en angles, employant l'instruction vec_to_angle (). La valeur pan du vecteur my_angle devient l'angle horizontal entre le robot et le joueur et la valeur tilt devient l'angle vertical entre eux. La fonction prédéfinie actor_turn (définie dans actors.wdl) fait tourner une entité horizontalement vers un angle cible avec une vitesse donnée par la variable de force. Nous pourrions directement mettre l'angle pan de notre robot à my_angle.pan, mais alors il se tournerait anormalement vite et ainsi ferait toujours face au joueur.

Ensuite il joue l'animation attaquante. En atteignant le dernier encadrement, il doit nous envoyer une balle. Nous employons la fonction de bullet_shot ici. Cette fonction a été écrite pour notre arme flashgun, mais est un bon exemple pour vous montrer que l'on peut utiliser pratiquement chaque fonction pour des buts différents. Bullet_shot exige que le vecteur shot_speed soit mis à la bonne vitesse et la bonne direction. Nous le faisons en le mettant à 100 quants vers la direction X, le faisant ensuite tourner dans la direction horizontale notre robot fait face, en réutilisant la valeur de my_angle.tilt précédemment calculée à la direction de la balle qui pointe verticalement vers le joueur. Vec_rotate () fait le travail.

À ce point une nouvelle énigme pour le lecteur : les deux toutes premières lignes dans la boucle d'attaque, qui commencent toutes les deux par if (my.health ..., doivent se trouver exactement dans cet ordre. Pourquoi ?

Le dernier état, la fuite (escape), c'est du gâteau maintenant :

```
function mystate_escape()
{
        while (1)
        {
                if (my._health <= 0) { mystate_die(); return; }
                // turn away from player
                temp.x = my.x - player.x;.
                temp.y = my.y - player.y;
                temp.z = my.z - player.z;
                vec_to_angle(my_angle,temp);
                force = 4;
                actor_turn();
                // walk away
                actor_move();

                wait(1);
        }
}
```

La seule nouvelle fonction que nous appelons ici est actor_move, qui est définie dans le script movement.wdl et laisse l'acteur continuer sa promenade.

Machines D'état Avancées

Le comportement intelligent et personnalisé des adversaires donne une satisfaction plus grande particulièrement avec des jeux d'action ou des jeux de rôle. Une fois que vous avez joué à un jeu pendant un certain temps vous commencerez à remarquer certaines des caractéristiques de vos adversaires - des faiblesses ou des particularités - dont vous pouvez vous servir. Vous êtes capables de développer une stratégie. Si de telles caractéristiques manquent, telle que la stratégie : le jeu sera un simple jeu de tir primitif et de tels jeux deviennent rapidement ennuyeux.

Comment pourrions-nous améliorer notre robot ? Le plus évident sont les possibilités d'élargissement d'état, qui donnent à l'acteur un plus de complexité, plus de répertoire 'personnel' de comportement. Il pourrait pousser un cri de colère quand il voit le joueur; il pourrait rebondir quand il frappe ou commencer à filer. Sur l'attaque, il pourrait s'avancer vers le joueur, esquivant de côté pour éviter un coup direct. Certaines de ces possibilités ont été réalisées dans le fichier war.wdl.

Les améliorations du comportement stratégique de l'acteur sont beaucoup plus compliquées à réaliser et on n'en trouve pas souvent dans des jeux d'action. On va clairement dans la direction de développer un degré plus haut d'intelligence artificielle qui peut seulement être compris en laissant les acteurs communiquer entre eux (par des instructions de balayage et des variables) et / ou en les faisant se comporter en ce qui concerne la position du joueur quant à la topographie du niveau. Quelques suggestions pour cela aussi :

- Aussitôt qu'un acteur ('le gardien') aperçoit le joueur il alarme un ami acteur à une certaine distance. Convenu pour donner l'alarme on trouve des objets comme des détecteurs, des caméras, des barrières lumineuses etc. Il est important pour entretenir le suspense que le joueur soit conscient de l'alarme!

- Si plusieurs acteurs attaquent le joueur, ils essayent de l'entourer et de l'encercler.

- Les acteurs sont à l'affût du joueur, se cachant autour des coins, en s'avançant vers le joueur juste au point où il ne peut pas encore les voir.

- Il peut y avoir des canyons et des pièces dans le niveau, qui fonctionne comme 'des pièges' : ici, les acteurs peuvent attaquer le joueur de tous les côtés. Les acteurs essayent de taquiner le joueur ou le pousser dans tels cas.

- Les acteurs peuvent livrer un pronostic sur le résultat du combat en comparant la force du joueur à leur propre force. Si le joueur est toujours tout à fait fort, ils appellent du renfort, si non, ils attaquent directement. S'ils voient le joueur déjà attaqué, ils rejoignent l'attaque pour combiner leurs forces.

  Mais nous ne devons jamais exagérer cela. Ce ne sont pas les échecs. Le joueur doit toujours être conscient qu'il y a le danger en avant. Les acteurs qui se cachent complètement et le mensonge dans l'embuscade pour le joueur assez longtemps pour soudainement lui sauter dessus de tous les côtés pour l'achever ne présentent pas beaucoup d'amusement pour votre jeu –

Jeudi : l'Interface utilisateur

Jusqu'ici tout a concerné exclusivement la construction d'un monde virtuel et ses habitants. Dans ce chapitre nous allons parler des fonctions telles que, bien qu'étant à l'extérieur du monde, fassent toujours partie du jeu. Ce sont les fonctions d'interface utilisateur qui rendent possible la communication entre l'utilisateur, étant assis confortablement devant son écran d'ordinateur et son homologue imaginaire le joueur, qui est l'entité qui doit passer en courant dans des labyrinthes souterrains et faire le sale travail.

Textes

Si le joueur a quelque chose à dire à l'utilisateur, il peut le faire simplement en écrivant un texte. Les textes sont employés pour faisant un rapport sur l'écran, mais peuvent aussi être employés pour un dialogue entre le joueur et des acteurs dans un jeu d'aventure. Comme avec les images bitmaps, les textures et les objets il y a une hiérarchie de textes, qui va du texte brut non raffiné au rapport coloré et fantaisiste sur écran.

Comme nous nous souvenons, le texte brut est défini comme une suite de caractères avec le mot clé string :

```
string my_string = "c'est un texte!!";
```

La chaîne de caractères peut même consister en plusieurs lignes, chacune correspondant à plusieurs lignes dans le fichier C-SCRIPT ou être séparée l'une de l'autre par la chaîne de caractères "\n".

```
string my_string_3 =
"C'est la première ligne,
c'est la deuxième, \n et ceci, finalement, est la troisième ligne. "
```

Bien entendu le "\n" entre la deuxième et troisième ligne n'apparaîtra pas sur l'écran. Pour que le texte soit visible à l'écran nous devons définir en premier un jeu de caractères (font). Une fonte est simplement composée de petites images pour chaque lettre. Les images apparaissent dans un certain ordre dans une image bitmap. Cet ordre correspond aux 128 premiers ou 256 caractères du jeu de caractère du PC (en code ASCII).

En somme le bitmap doit laisser la place pour 128 ou 256 lettres, nombres et symboles, chaque caractère prenant exactement la même quantité d'espace. La première rangée restes vide, la troisième rangée contient les majuscules, la quatrième contient les lettres minuscules. La place entre les caractères peut être mise à la couleur transparente 0. Il va de soi que chaque jeu imaginable de caractères - par exemple des symboles grecs, des caractères extraterrestres ou ceux issus de votre propre imagination - peuvent être représentés. Dans notre exemple les caractères accentués français prennent la place des caractères spéciaux '[', '', ']' et '{', '\', ''. Dans le répertoire de calibre(template) une image bitmap simple (ackfont.pcx) a été ajoutée qui peut servir comme un exemple pour le groupement de votre propre jeu de caractères.

Le mot clé font est employé pour définir le jeu de caractères comme tel :

```
font standard_font = <panfont.pcx>, 8,10;
```

Les deux paramètres finaux donnent la largeur et la hauteur en pixels d'un simple caractère dans le jeu de caractères de l'image bitmap (incluant les l'espace vide). Une image bitmap qui a 128 caractères doit inclure exactement 8*10*128 = 10240 pixels. La fonte de l'image bitmaps de 11, 128 ou 256 caractères est possible. Une fois que nous avons la fonte disponible nous pouvons faire un texte de notre chaîne de caractères :

```
text my_text {
        pos_x = 20;
        pos_y = 40;
        font = standard_font;
        string = my_string;
}
```

Pos_x et pos_y retournent la position du texte sur l'écran en pixels, en référence au coin supérieur gauche. Mais nous devrons le rendre d'abord visible, parce que le texte défini ci-dessus ne se montrera pas toujours sur l'écran. La fonction suivante fait apparaître notre texte sur l'écran :

```
function show_my_text () {my_text.visible = on;}
```

Cette méthode nous permet de montrer plusieurs textes sur écran simultanément. Comme nous pouvons changer les positions du texte pos_x et pos_y pendant le jeu, le texte peut aussi être fait pour défiler à travers l'écran verticalement :

```
function roll_my_text()
{
        my_text.pos_y = 480; // déplace le texte sous l'écran
        my_text.visible = on; // le rendret visible
        while (my_text.pos_y > 0) { // aussi longtemps que le texte tient sur l'écran
        my_text.pos_y -= 1; // défile vers le haut une ligne de pixel
        wait(1); // puis on attend une frame
        }
        waitt(16); // temps suffisant pour lire la ligne
        my_text.visible = off; // cache le texte
}

on_p = roll_my_text;
```

La fonction est introduite en pressant la touche [P]. Nous avons encastré une boucle **while**; le déplacement du texte et les instructions **wait()** sont répétés à chaque cycle d'affichage. Seulement si la valeur de pos_y après beaucoup de soustraction de 1 arrive finalement à 0, c'est-à-dire s'il touche la fin supérieure de l'écran, la boucle **while** ne sera plus exécutée désormais. La fonction fait alors disparaître le texte et se termine.

Panneaux

Beaucoup de jeux emploient des affichages montrant des nombres ou une barre pour donner de l'information sur l'état des joueurs ou des variables du jeu. Un exemple classique serait le jeu de rôle où la santé, la force de combat et des douzaines d'autres caractéristiques de joueur changent constamment et influence le jeu.

Un panneau sur l'écran représente de tels affichages. Pour donner un exemple de panneau nous représenterons maintenant les valeurs de certaines variables sur l'écran sous la forme de nombres. Il est très utile d'avoir ces variables présentes sous vos yeux lorsque vous mettez au point une fonction écrite par vous même.

Les panneaux sont définis de la même façon que les textes. Au lieu d'une chaîne de caractères cette fois il y a des définitions pour des affichages numériques :

```
font standard_font = <panfont.pcx>,8,10;

panel debug_panel {.
        pos_x = 2;
        pos_y = 2;
        flags = visible,refresh; // le rend visible et affichable sur la vue
        digits 0,0,3,standard_font,16,time_fac; // frames par seconde
        digits 40,0,3,standard_font,1,camera.pan;
        digits 80,0,3,standard_font,1,camera.tilt;
        digits 120,0,4,standard_font,1,camera.x;
        digits 160,0,4,standard_font,1,camera.y;
}
```

Nous avons maintenant quelques affichages de nombres dans la partie haute de l'écran qui nous informe de l'état du joueur ou du jeu. Chaque ligne de chiffres dans la définition de panneau affiche un paramètre variable ou numérique. **Digit** contient 8 paramètres, qui déterminent la position X et Y, dans le panneau d'affichage, le nombre de chiffres, la fonte, un facteur pour la multiplication et la variable ou le paramètre lui-même. Les valeurs de position donnent la distance en pixels du coin supérieur gauche de l'affichage numérique au coin supérieur gauche du panneau. Le drapeau **refresh** fait que le panneau est reconstruit constamment. C'est la seule façon que nous avons pour le montrer au-dessus de la vue de caméra, qui est aussi produite à nouveau avec chaque cycle d'affichage. Avec les textes nous n'avons pas à mentionner le drapeau **refresh** parce qu'il est mis automatiquement.

Dans le panneau nous pouvons maintenant voir - de la gauche vers la droite - le taux d'affichage actuel en frame par seconde et encore quatre paramètres donnant la position de la vue de caméra. Si nous voulons suivre l'état d'autres variables pendant le jeu nous pouvons les montrer ici à la place de celles-ci.

Il y a une chose minuscule incorrecte avec l'exposition du taux d'affichage pour lequel nous avons employé la variable **time_fac**. Cette variable retourne la valeur 1, si le taux d'affichage est exactement de 16 images par seconde et change proportionnellement. Cela signifie que les changements d'affichage sont plutôt rapides et durs à lire. Un petit truc le fera ralentir un peu :

```
var fps;

function show_panel()
{
	while (1)
	{ // repeat forever
	fps = 0.8*fps + 0.2*time_fac;
	wait(1);
	}
}
```

Notre fonction **show_panel** qui s'exécute éternellement est responsable 'pour traiter' les variables d'affichage. Elle doit être commencée au début de jeu par une instruction **show_panel ()** dans la fonction principale et être exécutée pour toujours. Le **debug_panel** doit maintenant montrer le **fps** au lieu de la variable **time_fac**. L'expression arithmétique s'assure qu'un changement spontané de **time_fac** a seulement un impact de 20 % sur l'affichage, qui fait que la valeur change cinq fois plus lentement.

Un nouvel exemple pour un affichage est une image bitmap qui peut être décalée sous une fenêtre selon les valeurs de variables données. Nous voudrions nous servir d'un affichage **windows** (fenêtre) pour représenter une boussole sur l'écran. Nous commençons en peignant une image bitmap pour la barre de boussole (remplaçons le 'O' de Ost par un 'E' pour Est et le W de West par le O de Ouest pour l'adapter à la langue française) :

```
bmap compass_map = <compass.pcx>; // 160x20 pixels
var compass_pos[2] = 0,0;

panel compass_pan {
	pos_x = 220;
	pos_y = 2;.
	flags = visible,refresh;
	window 0,0, 40,20,compass_map,compass_pos.x,compass_pos.y;
}
```

La boussole nous donnera toujours la ligne de la caméra de vue, 0° équivalent à Est. La ligne de **window** est structurée de la même façon que **digits**; ici de nouveau on donne d'abord la position, puis la largeur et la hauteur de la fenêtre en pixels, puis l'image bitmap à décaler et finalement les deux paramètres qui donnent le changement horizontal et vertical en pixels. Nous employons seulement un changement horizontal ici, donc **compass_pos.y** est toujours à 0. Nous calculons **compass_pos.x** à partir de l'angle de caméra :

```
function show_panel()
{
	while (1)
	{ // repeat forever
		fps = 0.8*fps + 0.2*time_fac;
		compass_pos.x = 120 - (camera.pan % 360)/3;
		wait(1);
	}
}
```

Pour un défilement circulaire nous avons fait une image bitmap de boussole de 160 pixels de large qui se répète après 120 pixels. Donc notre fenêtre de 40 pixels est toujours remplie de l'image bitmap boussole (120+40 = 160). Pour une pleine rotation de 360 degrés l'image bitmap doit changer de 120 pixels, soit un tiers - donc nous divisons l'angle par trois. Nous devons limiter la gamme angulaire de 0 à 360 par la fonction modulo, parce que camera.pan peut être à l'extérieur de cette gamme. Et finalement nous avons modifié complètement la direction de changement en soustrayant la quantité de changement de sa valeur maximale de 120 pixels.

Vendredi : contrôle du Jeu

Habituellement un jeu consiste en plusieurs niveaux que le joueur doit franchir les uns après les. Cela nous confronte à un nouveau problème. Nous devons créer un passage entre ces niveaux. Si le joueur, sur un niveau inférieur, a rassemblé des particularités ou des armes on doit lui permettre de les prendre avec lui aux niveaux plus hauts. A part cela vous devez être capables d'arrêter de jouer à tout moment et de reprendre le jeu plus tard à l'endroit ou vous l'aviez quitté. Cela exige de nous la capacité de sauvegarder ou complètement ou partiellement la situation actuelle dans le jeu.

Changement de Niveaux

Il y a une façon facile de changer de niveaux. Le joueur peut marcher par une certaine porte, qui – pouvant être déclenchée par event_trigger - exécute la fonction suivante :

```
function moveto_level2()
{
	msg_show("changement de niveau, patientez svp,5");
	wait(2); // temps pendant lequel le message reste visible
	level_load("level_2.wmb");
}
```

La fonction de msg_show peut être trouvée dans le message.wdl et montrera un message sur l'écran pendant quelques secondes. Bien sûr, vous pouvez alternativement montrer un panneau en plein écran ou jouer un film AVI.

Le chargement d'un niveau enlèvera toutes les entités, y compris le joueur et son modèle d'arme, purifiera tous les synonymes d'entité et terminera les actions de toutes les entités - qui inclut la fonction ci-dessus, si elle a été déclenchée par une entité de porte. Toutes les variables de non entités gardent cependant leurs valeurs et toutes les fonctions de non entité continuent de fonctionner. Aussi après le chargement du niveau, la chose suivante à faire est de recréer les entités d'arme, qui ont dû été stockées auparavant en utilisant des variables (un synonyme ne peut pas le faire parce que les synonymes d'entité sont purifiés). Et la fonction qui fait cela doit être une fonction globale. Il y a un truc facile pour faire une action d'entité globale :

```
action moveto_level2()
{
	msg_show("changement de niveau, patientez svp,5");
	wait(2); // temps pendant lequel le message reste visible
	my = null; // maintenant cette action devient une fonction globale et continue de fonctionner
	level_load ("level_2.wmb");
	wait(1); // temps pour que le nouveau niveau soit chargé
	weapon_restore(); // fonction qui recrée les modèles d'armes
}
```

Nous avons dû insérer une instruction wait() après level_load parce que le niveau n'est pas chargé immédiatement, mais après la fin du cycle d'affichage actuel. Pour recréer toutes les armes que le joueur a prises pendant le jeu, nous devons définir une fonction individuelle weapon_restore dépendant des armes que nous employons dans le niveau. Si nous employons weapons.wdl prédéfini, nous savons que le fait de prendre une arme met la variable correspondante weapon1 .. weapon7 à non zéro. Cette fonction pourrait ressembler à ceci (gun1 etc. étant les fonctions initialisant les armes à feu et les rendant invisibles) :

```
function weapon_restore()
{
	if (weapon1 != 0) { create(<smallgun.mdl>,nullvector, gun1); }
	if (weapon2 != 0) { create(<biggun.mdl>,nullvector, gun2); }
	// – and so on
	gun_select();// weapons.wdl; re-selects the current weapon
}
```

Sauvegarde et Chargement de Jeux

Avec l'instruction de sauvegarde (save) l'état complet du jeu est écrit sur disque dur. Le score du jeu sauvegardé est crypté et comprimé pour qu'il n'utilise pas trop de place sur le disque dur. Avec l'instruction charge (load) un score de jeu sauvegardé est décrypté et rechargé. Toutes les fonctions qui étaient opérationnelles au moment de la sauvegarde vont bien évidemment fonctionner de nouveau après le chargement.. Si le joueur est dans un niveau différent au moment du chargement, le niveau sera changé automatiquement comme avec une instruction level_load. Son propre chargement - comme l'instruction level_load - ne sera pas exécutée immédiatement dans la fonction, l'instruction correspondante pouvant être trouvée, mais seulement après la fin du cycle d'affichage.

Nous définirons une fonction pour la sauvegarde et le chargement du score. Nous voulons sauvegarder un score avec la touche [F2] - et le recharger avec la touche [F3] - . Cela paraît assez simple, n'est-ce pas ? Pour rendre les choses plus agréables, nous voulons être capables de choisir parmi les 3 dernières sauvegardes en appuyant à plusieurs reprises sur la touche [F3]. Un message d'écran correspondant donnera le numéro de la sauvegarde concerné.

```
var_info slot= 1; // nombre de score (1..3)
var load_which = -1; // le numéro que nous voulons charger

savedir "c:\\a5games";

string save_yesno = "sauve jeu (y/n)?";
string load_yesno1 = "charge dernière sauvegarde (y/n/f3)?";
string load_yesno2 = "charge avant dernière sauvegarde (y/n/f3)?";
string load_yesno3 = " charge avant avant dernière sauvegarde (y/n/f3)?";

function qsave_game()
{ // sauve score, avec question
	yesno_txt.string = save_yesno;
	yesno_do = qsave_game1;
	yesno_show();
}

function qsave_game1() { // sauve score directement
	load_status; // charge variable emplacement dernière sauvegarde
	slot += 1; // et incrémente de 1
	if (slot>3) {slot -= 3; } // si >3 alors revient à 1
	load_which = -1; // prépare chargement de la dernière
	save_status();
	save("qgame",slot); // sauve jeu
	if (result < 0)
	{ // erreur de sauvegarde?
			msg.string = save_error;
			} else {
			msg.string = ok_str;
	}
	show_message();
}

function qload_game()
{
	load_which += 1;
	if (load_which > 2) { load_which = 0; }.
	if (load_which == 0) { yesno_txt.string = load_yesno1; }
	if (load_which == 1) { yesno_txt.string = load_yesno2; }
	if (load_which == 2) { yesno_txt.string = load_yesno3; }
	yesno_do = qload_game1;
	yesno_show();
}

function qload_game1
{
	msg.string = wait_str;
	show_message();
	wait(1); // pour montrer le message avant le chargement
	load_status(); // charge la variable emplacement dernière sauvegarde
	slot -= load_which;
	if (slot < 1) { slot += 3; }
	load("qgame",slot);
	wait(1);
```

```
        msg.string = load_error; ça s'est mal passé!
        show_message();
}

on_f2 = qsave_game;
on_f3 = qload_game;
```

Nous allons voir que le problème est de se rappeler le numéro du dernier score sauvegardé après le chargement de quelque autre nouveau score. Parce que le chargement d'un jeu change chaque variable, nous devons sauvegarder le numéro du dernier jeu sauvegardé séparément sur le disque dur. La seule façon de le faire c'est d'employer une variable de renseignements, que nous avons appelée slot (emplacement) dans ce cas et exécutons save_info.

Nous emploierons quelques fonctions de message.wdl et des fonctions de menu.wdl pour afficher les messages à l'écran ou attendre une réponse du clavier - yes/no (oui/non). La fonction qsave_game est exécutée par la touche [F2]. En appuyant sur la touche [Y] ou [Entrée] après la question, nous commençons alors la sauvegarde proprement dite avec la fonction qsave_game1. Avec d'autres variables de renseignements que nous aurions pu définir ailleurs, la variable slot est lue à partir du fichier INFO0. SAV. Cette variable contient le numéro de la dernière sauvegarde. Il est augmenté de 1 et remis à 1 dans le cas où sa valeur dépasserait 3 puis est sauvegardée ensuite de nouveau dans le fichier info0.sav. Ensuite la sauvegarde proprement dite du jeu sous le nom de qgame0.sav, qgame1.sav ou qgame2.sav - selon la valeur actuelle de slot. Si une erreur arrive pendant la sauvegarde, la variable prédéfinie result a automatiquement une valeur différente de 0. Dans ce cas nous nous affichons un message d'erreur ou bien "OK", en utilisant la fonction show_message.

Avec le chargement par la touche [F3] il y a aussi une première question à l'utilisateur. Dans ce cas vous pouvez répondre non seulement par [Y] ou [N], mais aussi en appuyant de nouveau sur [F3]. Chaque pression de [F3] affiche un message différent à l'écran, change le numéro de la variable et assigne cycliquement une des trois fonctions qload_game1, qload_game2 ou qload_game3 à la touche [F3].

Répondre [Y] exécute la fonction qload_game. Cette fonction est censée afficher le message "attendez s'il vous plaît ...." à l'écran pendant le chargement du jeu, qui peut prendre un certain temps dans le cas où un changement de niveaux est nécessaire. L'affichage d'un texte sur l'écran commence, mais avec le cycle d'affichage suivant, nous ne pourrions jamais voir le message s'il n'y avait pas l'instruction wait() , parce que l'instruction de chargement (load) arrête le rafraîchissement de l'écran tant que le niveau sauvegardé n'a pas été chargé complètement.

Pour déterminer le numéro du jeu qui doit être chargé, slot est de nouveau lu à partir de notre fichier de renseignements et le numéro de la variable en est soustrait. Le résultat donne pour le dernier (0), avant-dernier (1) ou avant avant dernier (2) sauvegarde qui doit être chargée. À ce point - enfin - le chargement lui-même peut se faire.

Et maintenant, pour conclure notre tutorial C-SCRIPT, la dernière petite énigme à résoudre pendant le week-end : cette fois nous n'avons pas besoin de tester la variable result pour déterminer si quelque chose a mal tourné avec le chargement du fichier. Pourquoi à votre avis ?

Référence : Syntaxe Script

Les scénarios C-SCRIPT consistent en définitions et fonctions. Les définitions créent des objets, leur donnent un nom et définissent leurs propriétés initiales; les fonctions déterminent le comportement d'objets en changeant dynamiquement leurs propriétés pendant le jeu, selon certaines conditions. C'est commun à tous les langages de programmation. La syntaxe C-SCRIPT est dérivée du Java, Java script ou C, mais simplifiée. Pour qu'un objet existe, il peut ou être placé dans la carte par WED - comme cela est fait avec des entités - ou il peut être créé d'après une définition de scénario. Le dernier est fait en donnant son type, son nom et une valeur initiale dans le fichier de scénario, comme cela :

type name = value;
ou
type name { ... }

À la deuxième ligne, entre les accolades, des valeurs initiales sont assignées aux paramètres simples de l'objet. Un nom peut consister jusqu'à 30 lettres. La casse (majuscule, minuscule) n'importe pas. Les noms ne doivent pas commencer par des nombres, ni contenir n'importe quels caractères spéciaux sauf le souligné _, on ne vous permet pas non plus deux fois le même nom pour des objets différents. Exemples :

```
bmap sky1 = <stars.pcx>; // définit une image bitmap nommée sky1
var position[3] = 10, 20, 30; // definit un vecteur nommé position
text mytext { font = standard_font; string = "this is a text!"; } // definit un texe formaté
```

Le caractère "=" peut être omis dans de telles définitions, comme ça a été fait souvent dans des fichiers C-SCRIPT plus anciens.

Les objets consistent en un ou plus des éléments de base suivants:

Variable - un nombre à virgule fixe jusqu'à six chiffres, avec jusqu'à trois chiffres après la décimale (par Exemple: 123456.789). Ainsi les variables s'étendent de 999999.999 à -999999.999. Un groupe de trois nombres (qui est souvent employé dans l'arithmétique 3D) est appelé un vecteur.

Drapeau (Flag) - une valeur binaire, comme un commutateur, qui peut être mis à oui (on) (1) ou à non (off) (0).

Chaîne de caractères (string) - une chaîne de caractères, qui doit être écrite entre guillemets; les retours à la ligne doivent être écrits dans la notation C comme "\n", anti slashs comme "\\". Exemple: "il est maintenant temps\npour tous les hommes ..." .

<Nom de fichier> (<filename>) - Le nom de fichier donné est limité à 20 caractères et ne doit pas contenir de chemin. Généralement le type de fichier est reconnaissable à son extension. Des extensions valables sont .pcx, .bmp ou .tga pour des images bitmaps ou des entités sprite, .mdl pour des entités modèles, .wmb pour des entités de carte, .mid pour des chansons, .wav pour le bruitage, .avi pour des films, .wdl pour des scénarios. Tous les noms de fichier donnés entre ces 2 caractères <...> dans un scénario C-Script seront empaquetés en une ressource .wrs par l'emballeur de fichier. Quelques fichiers ne doivent pas être empaquetés - par exemple des fichiers .dll ou des films .avi. Si un fichier avec le même nom est trouvé séparément à l'extérieur de la ressource, le moteur l'emploiera au lieu du fichier empaqueté dans la ressource - à moins que ce ne soit un scénario .wdl. Les scénarios doivent toujours être empaquetés dans la ressource.

Dans le monde virtuel nous avons un système de coordonnée main droite XYZ avec l'axe des abscisses qui se tient debout tout droit. Dans le cas de la 2D, pour positionner quelque chose à l'écran ou dans une image bitmap, nous employons un système de coordonnée de XY en unités de pixel, avec l'Axe des ordonnées pointant en bas et l'origine placée dans le coin gauche supérieur.

L'espace et les unités de temps du monde virtuel sont le quant et le tick. Un quant est équivalent à une unité dans les coordonnées WED et MED et également un pixel de texture à l'échelle 1.0. Combien de pouces à un quant dépendra de la taille relative des modèles. Nous recommandons un pouce par quant pour des jeux avec des personnages (jeux de tirs ou jeux d'aventures) et 4 pouces par quant pour des jeux avec des véhicules (des jeux de guerre ou des simulateurs de vol). De cette façon avec la même taille de niveau vous obtiendrez des mondes beaucoup plus grands. Un tick est équivalent à 1/16 de seconde - le temps moyen entre deux cycles d'affichage sur un PC bas de gamme.

Les angles sont donnés en degrés (0 à 360) et comptés en sens inverse des aiguilles d'une montre. Pour des rotations dans trois dimensions les pseudo angles Euler sont employés : pan est l'angle horizontal (0 .. 360) de l'axe Z, tilt est l'angle vertical (-90 .. +90) autour de l'axe des Y et roll est l'angle (0 .. 360) de rotation et d'inclinaison pour l'axe des X. Pour un angle pan, 0 degré est équivalent à la direction positive de l'axe des X, qui indique l'Est de la carte. 90 degrés est le nord, 180 degrés l'ouest et 270 degrés le sud.

Pour les couleurs on utilise les composantes(R,V,B) rouges, vertes, bleues entre 0 et 255. Toutes les 3 à 0 donnent noir; toute les 3 à 255 donnent blanc. La valeur maximale des couleur est 255 et ne doit pas être dépassée. L'ordre des couleurs dans un vecteur couleur est bleu-vert-rouge.

Les caractères spéciaux suivants sont valables dans vos scripts :

...;	le point virgule termine chaque instruction.
{...}	accolades pour inclure des listes d'instruction ou des assignations.
"..."	On donne le texte entre guillemets.
[...]	des index de Tableau sont donnés entre crochets.
<...>	des Noms de fichier sont donnés entre des parenthèses dirigées.
#..	Commentaire jusqu'à la fin de la ligne.
//...	Commentaire jusqu'à la fin de la ligne.
/*. ..*/	bloc de Commentaire.

Chaque langage de programmation est impitoyable avec les erreurs de syntaxe. Chaque virgule oubliée ou superflue ou point virgule produira certainement un message d'erreur au démarrage. Donc soyez soigneux.

Certaines des particularités décrites ici sont seulement disponibles dans l'édition Extra, Commerciale ou Professionnelle ou supérieure et sont marquées avec E, C ou P

Fonctions - le cerveau du jeu

Les fonctions contrôlent le jeu et le comportement pseudo intelligent des entités. Une fonction consiste en une liste d'instructions, qui seront exécutées l'une après l'autre. Les moteurs de Gamestudio, A4 ou A5, sont des moteurs de multitâches. Ainsi beaucoup de fonctions sont exécutées simultanément, comme dans la vie réelle. Les instructions représentent une sorte de langage simple, qui vous permet d'influencer non seulement les entités, mais aussi le jeu de nombreuses façons.

Dans un environnement multi joueur il est important de savoir quelles fonctions tournent sur quelles machines. Les fonctions attachées à une entité (aussi nommées Actions) sont exécutées uniquement sur le serveur; de cette façon elles influencent le monde de jeu entier. Les fonctions assignées à l'interface utilisateur - qui signifie des fonctions attachées aux frappes du clavier ou aux boutons des panneaux - sont locales et seront exécutées seulement sur le poste client. Dans un environnement de joueur simple - qui est le cas normal - le serveur et le client sont le même PC et vous ne devez pas vous occuper d'où vos fonctions sont exécutées.

Il y a beaucoup de fonctions prédéfinies incluses dans les scénarios C-SCRIPT, qui doivent être suffisantes pour toutes les sortes de jeux simples. Vous trouverez une liste de fonctions prédéfinies en annexe. Cependant, si vous voulez écrire vos propres fonctions C-SCRIPT, vous devez les définir de la façon suivante :

function ***name (paramètre1,paramètre2)*** { .... }

Crée une fonction avec le nom (name) donné. Jusqu'à 4 paramètres facultatifs – variable, vecteurs, pointeurs ou expression – qui permet de passer des informations à la fonction lorsqu'elle est appelée. La fonction peut avoir accès à ces paramètres sous le nom donné entre les parenthèses, comme une variable normale. Exemple:

```
function beep_times(times) {
while (times > 0) { beep; times -= 1; wait(1); }
}
// beep_times(7) émettera 7 signaux sonores, beep_times(10) en émettera 10
```

Si une variable ou un autre objet du même nom existe, toutes les références dans la fonction se réfèrent au paramètre 'local' et non pas à la variable 'globale'. La fonction peut changer n'importe quel paramètre, comme dans l'exemple, mais le paramètre original dans la fonction de l'appel reste inchangé. Les paramètres peuvent être omis, dans ce cas l'espace entre les parenthèses reste vide.

Les fonctions peuvent aussi prendre des pointeurs et des drapeaux oui/non comme paramètres. Un paramètre de pointeur doit être assigné à un pointeur C-Script défini de type connu avant qu'il ne puisse être employé directement pour assignation. Cependant il peut être directement remis comme paramètre à d'autres fonctions. Exemple:

```
// cette fonction utilise un pointeur d'entité comme argument.
function flare_init(ent)
{
my = ent; // necessaire, parce que my.bright = on fonctionne, mais ent.bright = on ne fonctionne pas!
my.bright = on;
my.flare = on;
ent_alphaset(0,10);
}

...
flare_init(you);
flare_init(flare0_ent);
..
```

Notez que des arguments non numériques dans des fonctions gardent leur valeur seulement jusqu'à la première instruction wait().

Les fonctions acceptent aussi des tableaux numériques ou des vecteurs (voir ci-dessous) comme paramètres. Parce que C-SCRIPT ne fait aucune différence entre des variables et des tableaux, nous devons indiquer à la fonction que tel paramètre doit être traité comme un tableau, au lieu d'un nombre simple. Pour cela, le nom du paramètre dans la définition de fonction est préfixé par un " &". Exemple :

```
function vector_add (&sum,&v1,&v2)
{ // calcule la somme de 2 vecteurs
   sum[0] = v1[0] + v2[0];
   sum[1] = v1[1] + v2[1];
   sum[2] = v1[2] + v2[2];
}

var vector1[3] = 1,2,3;
var vector2[3] = 4,5,6;
var vector3[3];

...
vector_add(vector3,vector1,vector2); // vector3 contient à présent 5,7,9
```

À l'intérieur de la fonction, on ne permet pas d'abréviations vectorielles comme **.x**, **.pan** etc pour les paramètres tableau ou vecteur. Vous devez toujours employer des index. Cependant en appel de la fonction, vous pouvez aussi passer les positions d'entité ou angles **.x** au lieu de tableaux numériques. Exemple :

```
function vector_direction(&result,&from,&to)
{ // calculate the direction angles between two positions
   result[0] = to[0] - from[0];
   result[1] = to[1] - from[1];
   result[2] = to[2] - from[2];
   vec_to_angle(result,result);
}


...
vector_direction(temp,my.x,you.x);
// sets temp to the angle pointing from my to you
vector_direction(temp,camera.x,nullvector);
// sets temp to the angle from the camera to the level origin
```

Parce que chaque variable est aussi un tableau au moins de longueur 1, vous pouvez passer une variable aussi bien comme un nombre que comme un paramètre de tableau. Cependant il y a quelques différences subtiles entre les deux types de paramètres. Si vous passez une variable à une fonction qui attend un nombre, c'est son contenu qui est remis. Si vous passez la même variable à une fonction qui attend un paramètre de tableau, c'est le pointeur de la variable qui est remis. Si la fonction modifie le paramètre, la modification d'un nombre est 'localement', c'est-à-dire seulement dans les limites de la fonction. La variable passée elle-même n'est pas changée. Ce n'est pas le cas pour tableau ou paramètres vectoriels : Les éléments d'un tableau passé sont modifiés globalement. Il est important de comprendre cette différence. Quelques exemples :

```
function double_num(num)
{
   num *= 2; //num est doublé localement, c'est à dire seulement à l'intérieur de la
fonction
}

function double_vec(&num)
{
   num[0] *= 2; // num[0] est doublé globalement
}
    ...

   temp = 1;
   double_num(temp); // temp est laissé à 1 ici;
   double_num(1); // C'est ok, 1 est un nombre.
   double_num(temp+1); // c'est ok, temp+1 est un nombre
   double_vec(temp); // maintenant temp est changé globalement à 2;
   double_vec(temp); // maintenant temp vaut 4;
   //double_vec(1); // ceci n'est pas permis, 1 n'est pas un vecteur
   //double_vec(temp+1); // ceci n'est pas permis, temp+1 n'est pas un vecteur
```

La longueur d'un tableau n'est pas testé à l'intérieur d'une fonction. Aussi si une fonction utilise un mauvais index (comme num[3] dans notre exemple) cela ne sera pas indiqué – il y aura simplement un plantage du moteur.

Le contenu local de paramètres numériques, aussi bien que les états locaux des indicateurs you et my à l'intérieur de toutes les fonctions sont sauvés ensemble avec le jeu. Les paramètres de tableau ou les paramètres de pointeur ne sont pas sauvés. Donc ils peuvent devenir invalides dans des fonctions en chargeant un jeu. Comme les instructions de sauvegarde/chargement (**save/load**) peuvent seulement arriver pendant le temps d'un **wait()**, emploiez des paramètres d'indicateur ou le tableau seulement dans des fonctions qui ne contiennent pas une boucle **wait()**. Employez-les autrement seulement avant chacune des instructions **wait/waitt** dans la fonction en étant conscient qu'ils peuvent être invalides après le premier **wait/waitt**.

Comme tout les objets, les fonctions doivent être définies **avant** qu'elles ne soient assignées à un autre objet ou appelées par une autre fonction. Parfois c'est incommode, par exemple si deux fonctions s'appellent l'une l'autre. Dans ce cas, une fonction peut être prédéterminée par un **prototype** (connu des programmeurs C ++). Le prototype consiste en titre de fonction avec les paramètres, suivis par un point-virgule, mais sans l'instruction inscrit dans {...}. Exemple :

```
function beep_sum(beep1,beep2); // prototype de la fonction beep_sum

function use_beep_sum()
{
	beep_sum(3,4); // beep_sum is used here, and needs the prototype defined before
}

function beep_sum(beep1,beep2) // c'est la fonction réelle
{
	beep1 += beep2;
	while (beep1 > 0) {
	beep();
	beep1 -= 1;
	wait(1);
	}
}
```

Une fonction peut s'appeler elle-même – cela s 'appelle une **fonction récursive**. Des fonctions récursives sont utiles pour la résolution de certains problèmes mathématiques ou d'IA. Voici un exemple :

```
function factorial(x)
{
	if (x <= 1) { return(1); }
	if (x >= 10) { return(0); } // number becomes too big...
	return (x*factorial(x-1));
}
```

Notez : les fonctions récursives doivent être employées avec soin et sont pour des utilisateurs avancés seulement. Si vous faites une erreur, comme une fonction qui s'appelle infiniment, vous pouvez produire facilement un débordement de pile stack overflow ' qui effondre l'ordinateur. La taille de pile permet une profondeur de récursion d'environ 10,000 imbrication d'appels de fonction.

action ***name*** { .... }

Les actions sont des fonctions un peu spéciales, elles n'acceptent pas de paramètres, mais apparaîtront dans la liste de menu contextuel d'action dans le panneau de propriété d'entité du WED. Donc les actions peuvent être assignées aux entités et les fonctions sont pour des calculs internes. En plus de cela, il n'y a aucune différence entre des fonctions et des actions. C'est notre exemple habituel pour une action d'entité, qui consiste juste en trois instructions et fait tourner l'entité :

```
action ent_rotate {
	while (1)
```

```
    {
        my.pan += my.skill1*time;
        wait(1);
    }
}
```

À l'intérieur d'une fonction, le type le plus basic d'instruction est l'instruction d'assignation :

Var = expression;
Var (+-*/) = expression;

Assigne une valeur ou le résultat d'une expression arithmétique à l'objet donné ou au paramètre d'objet (les paramètres d'objet sont donnés par le nom du paramètre, préfixé par le nom de l'objet ou le synonyme et un point). L'objet, dont le paramètre doit être changé, peut être une entité placée dans la carte ou quelque autre objet défini dans le scénario avant la fonction. L'expression arithmétique peut être composée de n'importe quels nombres, de nouvelles variables ou des paramètres d'objet, des parenthèses et des opérateurs arithmétiques. Exemples :

```
x = (a + 1) * b / c;
z = 10;
my.tilt = asin(3*x + 0.5);
my.z += 2; // incrémente la position Z de l'entité de 2 quants
my.z += 2*time; // déplace z par une vitesse temps - corrigée
my.event = react_function; // met la fonction d'événement de l'entité
my.invisible = on; //mets le drapeau invisible de l'entité à 1
on_s = save_function; // assigne la fonction de sauvegarde à la touche [s]
```

Dans les expressions, en plus des opérateurs de base + - */, les opérateurs binaires suivants sont supportés :

%	Modulo (reste d'une division)
|	OU binaire
^	OU Exclusif binaire
&	ET binaire
>>	Décalage d'un bit vers la droite
<<	Décalage d'un bit vers la gauche

Exemples:

```
x = x >> 2; // divise x par 4
x = x << 3; // multiplie x par 8
x = frc(x) << 10; // copie la partie décimale de x (10 bits) dans la partie entière
```

Le caractère "=" peut être combiné avec les opérateurs de base :

+ =	ajoute l'expression au paramètre
- =	soustrait l'expression du paramètre
* =	multiplie le paramètre à l'expression
/ =	divise le paramètre par l'expression

On doit tenir compte que l'arithmétique décimale arrondit tous les facteurs ou valeurs intermédiaires en dessous de 0.001 à zéro. Aussi les valeurs maximales et minimales de 999999.999 et -999999.999 ne doivent pas être dépassées. Cela s'applique aussi aux résultats intermédiaires dans une expression - donc un petit soin doit être pris en employant des expressions mathématiques. Des opérations invalides - un résultat excédant les valeurs maximales, une division par zéro, une racine carrée d'un nombre négatif etc. - donnera de mauvais résultats ou générera des messages d'erreur.

Dans la description des instructions suivantes, la plupart des instructions acceptent un ou plusieurs paramètres. Les paramètres peuvent être des variables ou d'autres objets. Quelques instructions prennent leurs paramètres entre parenthèses et retourne une valeur qui peut être directement employée dans des expressions, comme suit :

Fonctions variables

Les fonctions suivantes prennent juste un ou deux nombres comme paramètres et retournent un nombre comme résultat :

random(x)
Nombre fractionnaire aléatoire entre 0 et x. L'ordre aléatoire commence toujours par les mêmes nombres, mais elle peut être 'randomisée' au début de jeu par l'instruction **randomize().**

randomize()
Initialise la commande **random()** avec un nombre aléatoire,

sin(x), cos(x), tan(x)
asin(x), acos(x), atan(x)
Fonctions trigonométriques et fonctions trigonométriques inverses. Les angles sont donnés en degré.

fsin(x), fcos(x), ftan(x)
fasin(x), facos(x), fatan(x)
Fonctions trigonométriques et fonctions trigonométriques inverses, multipliées par le facteur f pour augmenter la précision. Les angles sont donnés en degré.

ang(x)
Change l'angle x dans une l'intervalle -180.. +180 en ajoutant ou soustrayant un multiple de 360 degrés.

log(x)
Logarithme de x en base e.

exp(x)
e puissance x.

pow(x,y);
Calcule x élevé à la puissance y. Génère une erreur si x est égal à 0 et y inférieur ou égal à 0, si x est négatif et que y n'est pas entier. Notez que **x*x** est identique, mais est calculé plus rapidement que **pow(x,2)**.

sqrt(x)
Racine carrée de x.

sign(x)
Retourne -1 si (x<0) c'est à dire négatif, 1 si (x>0) c'est à dire positif, 0 si (x==0) c'est à dire nul.

abs(x)
Valeur absolue de x.

int(x)
Partie entière de X (les chiffres **après** la virgule sont tronqués)

frc(x)
Partie décimale de x (les chiffres **avant** la virgule sont tronqués).

max(x,y), min(x,y)
Retourne le plus grand pour max ou le plus petit pour min des 2 valeurs x et y. Exemple:

```
a = max(0,(min(a,10)); // limite la valeur de a entre 0 et 10
```

Instructions vectorielles

Les instructions suivantes fonctionnent sur des vecteurs, c'est-à-dire des groupes de trois nombres. Les vecteurs peuvent être employés pour des buts différents dans le jeu, comme pour décrire une position xyz ou une direction ou une vitesse ou un angle Euler 3d pan, tilt ou roll. Ou un tableau de longueur 3 ou la première de trois variables consécutives ou les paramètres numériques d'un objet pouvant être employé comme un vecteur pour les instructions suivantes. Par exemple, pour employer les paramètres x,y,z de mon entité my comme vecteur de position, on peut juste écrire my.x. Ainsi, des vecteurs valables pour les instructions suivantes sont

- N'importe quels vecteurs prédéfinis et n'importe quelle variable qui est définie comme var[3];
- N'importe quelle variable, élément de tableau dont l'index est un multiple de 3 et une longueur de tableau d'au moins 3 (comme tableau[0], tableau[3], tableau[6] ...)
- N'importe quelle variable d'entité entre skill1 et skill46;
- Le paramètre x, pan ou blue de n'importe quelle entité;
- Le paramètre x ou pan de n'importe quelle vue

Si on donne autre chose à un vecteur que ce qu'il attend, vous risquez d'avoir pour résultat le crash du moteur – en effet pour des raisons de vitesse la validité des vecteurs n'est pas vérifiée.

vec_set ***(vector***1, ***vector***2);
Copie les trois nombres du deuxième vecteur dans le premier.

vec_add ***(vector***1, ***vector***2);
Ajoute le premier vecteur avec le deuxième et copie les trois sommes dans le premier.

vec_sub ***(vector***1, ***vector***2);
Soustrait le deuxième vecteur du premier et copie les trois différences dans le premier. Exemple:

```
vec_set(v1,nullvector); // met le vecteur v1 à 0,0,0
vec_sub(v1,v2); // maintenant le vecteur v1 et le négatif de v2
```

vec_scale ***(vector, var***);
Multiplie les trois nombres du vecteur par le nombre donné ou l'expression..

vec_dot ***(vector***1, ***vector***2);
Retourne le produit des 2 vecteurs.

vec_dist ***(vector***1, ***vector***2);
vec_length ***(vecto***r);
L'instruction vec_dist() retourne la distance entre 2 positions. L'instruction vec_length() retourne la longueur ou la magnitude d'un vecteur, c'est à dire sa distance par rapport à l'origine du niveau. Exemple:

```
distance = vec_dist(my.x,you.x); // calcule la distance entre les entités my et you
```

C'est mathématiquement équivalent, mais plus rapide que de calculer la distance par :

```
temp.x = my.x - you.x;
temp.y = my.y - you.y;
temp.z = my.z - you.z;
distance = sqrt(temp.x*temp.x + temp.y*temp.y + temp.z*temp.z);
// temp.x, temp.y, temp.z doivent être inférieur à 1000 pour ce travail!
```

vec_normalize ***(vecto***r, ***var***);
Tronque le vecteur à la longueur donnée par la deuxième variable, en gardant sa direction. C'est fait intérieurement en divisant les trois nombres du vecteur par sa longueur et en les multipliant ensuite par la variable.

vec_inverse ***(vecto***r);
Inverse le vecteur en multipliant ses composants par –1.

vec_diff ***(vecto***r, ***vector1,vector2***);
Met vector à la différence entre vector1 et vector2. Après l'instruction, vector donne la direction de vector2 vers vector1.

vec_rotate ***(vectordir***, ***vectorangle***);
Fera tourner le premier vecteur par les angles pan, tilt et roll donnés par le deuxième vecteur. ***Vectordir*** signifie ici une vitesse ou une direction, ***vectorangle*** représente un angle tridimensionnel (angle Euler). Exemple:

```
var direction[3] = 10, 0, 0; // la direction pointe droit vers l'est dans notre système de coordonnées
var angle[3] = 90, 45, 0; // pan 90 degrés, tilt 45 degrés, roll 0 degré
...
vec_rotate(direction,angle);
```

Après l'instruction, le vecteur de direction a les nouvelles valeurs 0, 7.07, 7.07. Il a été tourné par 90 degrés en sens inverse des aiguilles d'une montre et incliné par 45 degrés vers le haut, donc il pointe maintenant au Nord et à mi-chemin vers le haut.

ang_rotate ***(vecto***rAngle1, ***vectorAngle2***);
Fait tourner les angles pan, tilt et roll de vectorAngle1 par les angles pan, tilt et roll de VectorAngle2. Les deux vecteurs ont la signification d'angles tridimensionnels (Angles Euler) ici. Intérieurement une rotation quaternion est employée. Cette instruction peut être employée pour tourner un objet orienté arbitrairement d'un axe arbitraire donné par les angles du premier vecteur. Exemple :

```
// Nous faisons une caméra qui suit un vaisseau spatial et qui tourne avec lui.
// Le vaisseau se trouve en permanence au centre de l'écranew.
var cam_dist[3] = -500,100,100; // xyz position de la camera au vaisseau
var cam_ang[3]; // direction des angles de la camera au vaisseau

function chase_camera()
{
// calcule la direction des angles de la vue camera au vaisseau
    vec_diff(temp,nullvector,cam_dist);
    vec_to_angle(cam_ang,temp);
    cam_ang.roll = 0; // roll est inchangé par vec_to_angle
    // mise à jour permanente de la position et des angles de la caméra

    while (1)
    {
           // place lacamera à la bonne position du vaisseau
           vec_set(camera.x,cam_dist);
           vec_rotate(camera.x,ship.pan);
           vec_add(camera.x,ship.x);
           // mets les angles de la caméra aux angles du vaisseau,
           vec_set(camera.pan,ship.pan);
           // et réalise une rotation quaternion par les angles de direction de vue de
           caméra
           ang_rotate(camera.pan,cam_ang);
           wait(1);
    }
}
```

vec_to_angle ***(vectorangl***e, ***vectordir***);
Calcule l'angle pan et tilt et la direction donnés par le deuxième vecteur, et les place dans les paramètres pan et tilt du premier vecteur. Retourne également la longueur du vecteur de direction. Très utile pour échanger un angle contre une direction, donc calculer les angles à une cible. Exemple pour faire tourner l'entité my vers l'entité you:

```
function turn_towards_target() {
// obtenirla direction de l'entité à l'entité YOU
vec_set(temp,you.x);
vec_sub(temp,my.x);
vec_to_angle(my.pan,temp); // maintenant MY regarde vers YOU
}
```

vec_to_screen ***(vecto**r, ***vie**w);

Convertit les coordonnées xyz du vecteur donné en coordonnées d'écran. Après l'instruction, les coordonnées x et y du vecteur contiennent les coordonnées x et y de la position à l'écran, la coordonnée z contient la distance par rapport au plan de l'écran. view doit être visible pour que cette instruction fonctionne. Si la position retournée est invalide, c'est à dire une valeur en dehors de l'écran, l'instruction retourne 0, sinon elle retourne 1.

Cette instruction peut être utilisée pour attacher des textes ou des panneaux, pour afficher des positions d'entités. Exemple:

```
panel flare_pan { bmap = flare_map; flags = transparent,d3d; }
...
// attache un éclat de lentilles à mon modèle entité MY
vec_set(temp,my.x);
if (vec_to_screen(temp,camera)) // si visible à l'écran
{
flare_pan.pos_x = temp.x; // place le panneau d'éclat de lentilles
flare_pan.pos_y = temp.y;
flare_pan.visible = on;
} else {
flare_pan.visible = off; // sino on le sésactive
}
```

vec_for_screen ***(vecto**r, ***vie**w);

C'est l'opposé de l'instruction vec_to_screen(). Elle convertit des coordonnées d'écran, données par l'intermédiaire de vector, en une position de notre monde donnée par view. Comme on ne peut pas déterminer une position unique 3D à partir d'une position écran en x et y seulement, une profondeur doit être donnée par l'intermédiaire de la coordonnée z du vecteur vector. La vue doit être visible pour que l'instruction fonctionne.

Cette instruction peut être employée pour placer des entités dans un niveau en un clic de souris sur l'écran. Exemple:

```
function spawn_sprite() {
// engendre un sprite à la position du clic de souris, 200 quants derrière l'écran
temp.x = mouse_pos.x;
temp.y = mouse_pos.y;
temp.z = 200;
vec_for_screen(temp,camera);
create(<arrow.pcx>,temp,null);
}
on_mouse_left = spawn_sprite;
```

rel_to_screen ***(vecto**r, ***vie**w);
rel_for_screen ***(vecto**r, ***vie**w);

Comme vec_to_screen() et vec_for_screen(). Cependant elle utilise une position relative par rapport à la vue, et non plus par rapport au monde – l'angle et l'origine de la vue ne sont pas pris en compte. Employées pour calculer les positions d'écran des entités définies qui sont attachées à une certaine vue, comme des effets de lentille. Exemple

```
vec_set(temp,sun_pos);
if (vec_to_screen(temp,camera) != 0) { // si position du soleil visible à l'écran.
lens_flare.visible = on; // defined entity
temp.z = 200; // temp contient la position du soleil, donne 200 quants de profondeur
vec_set(lens_flare.x,temp);
rel_for_screen(lens_flare.x,camera); // maintenant l'éclat de lentille est à la position de caméra de soleil
}
```

Instructions de chaîne de caractères

Les instructions suivantes peuvent être employées pour manipuler et évaluer des suites de caractère (des chaînes de caractères). Si la chaîne de caractères n'est pas changée par l'instruction, on peut la donner directement comme argument :

str_cpy ***(string***1, ***string***2)

Copie le contenu de ***string2*** dans ***string***1. Si ***string1*** est plus court que ***string***2, le résultat sera tronqué.

str_cat ***(string***1, ***string***2)

Ajoute une copie de ***string2*** à la fin de ***string***1. Si la longueur originale de ***string1*** est plus courte le résultat sera tronqué. Exemple:

```
string s[30];

str_cpy(s,"hello"); // now s == "Hello "
str_cat(s,"world"); // now s == "Hello World"
```

str_cmpi ***(string***1, ***string***2)

Compare les deux chaînes de caractères sans s'occuper de la casse (majuscule / minuscule) et retourne la valeur 1 si elles sont identiques sinon retourne 0. Exemple:

```
string s1 = "hello world";
string s2 = "hello universe";

str_cmpi(s1,"hello world"); // returns 1
str_cmpi(s2,"hello world"); // returns 0
```

str_cmpni ***(string***1, ***string***2)

Compare les deux chaînes de caractères sans s'occuper de la casse (majuscule / minuscule) et retourne la valeur 1 si les parties qui se chevauchent sont identiques sinon retourne 0. Exemple:

```
string s1 = "hello world";
string s2 = "hello universe";
...

str_cmpni(s1,"hello"); // returns 1
str_cmpni(s2,"hello"); // returns 1
str_cmpni(s1,s2); // returns 0
```

str_len ***(strin***g)

Retourne le nombre de caractère de la chaîne. Exemple:

```
temp = str_len("hello "); // now temp == 6
```

str_clip ***(strin***g, ***numbe***r)

Coupe le nombre donné de caractères du commencement de la chaîne de caractères . Exemple:

```
string s = "hello world";
..
str_clip(s,6); // now s == "World"
```

str_trunc ***(strin***g, ***numbe***r)

Coupe le nombre donné de caractères de la fin de la chaîne de caractères. Exemple:

```
string s = "hello world";
...
str_trunc(s,6); // now s == "Hello"
```

str_stri ***(string***1, ***string***2)

Retourne la position du caractère (en commençant à 1) de la première occurrence (casse indifférente) de ***string2*** dans ***string***1. S'il n'y a aucune occurrence de trouvée, retourne 0. Exemple:

```
string s1 = "hello world";
...
temp = str_stri(s1,"world"); // now temp == 7
```

str_to_num ***(strin***g)

Retourne un nombre convertit de la chaîne de caractères. Exemple:

```
string s = "3.14";
...
pi = str_to_num(s); // now pi == 3.14
```

str_for_num ***(string***, ***numbe***r)

Transforme un nombre en une chaîne de caractères. Exemple:

```
string s[30];
...
str_for_num(s,sqrt(2)); // now s == "1.414"
```

str_to_asc ***(strin***g)

Retourne la valeur ASCII du premier caractère de la chaîne de caractères. Exemple:

```
string s = "a";
...
temp = str_to_asc(s); // now temp == 65
```

str_for_asc ***(strin***g, ***numbe***r)

Met le premier caractère de la chaîne de caractères en caractère ASCII représenté par le nombre. Exemple:

```
string s = " ";
...
str_for_asc(s,65); // now s == "A "
```

str_for_entname ***(strin***g, ***entit***y);

Met la chaîne de caractères au nom assigné dans WED du synonyme d'entité donné.

str_for_entfile ***(strin***g, ***entit***y);

Met la chaîne de caractères au nom de fichier du synonyme d'entité donné. Exemple:

```
string entname[30];
...
you = syn_for_name("waffe1_mdl_012"); // obtient le synonyme de cette entité
str_for_entfile(entname,you); // récupère le nom du fichier comme par exemple "waffe1.mdl"
you.string1 = entname; // fait apparaître le nom du fichier
you.enable_touch = on; // quand l'entité est touchée avec la souris
you.event = handle_touch;.
```

Instructions de fichier

Les instructions suivantes écrivent dans des fichiers ou lisent des fichiers. Ils peuvent être employés pour l'échange de données avec d'autres programmes ou pour beaucoup d'autres buts comme de stocker le contenu de variable pour la mise au point. Les instructions de fichier sont disponibles dans le moteur A5 seulement.

handle = file_open_read ***(strin***g);

Ouvre un fichier pour la lecture. Le fichier doit exister dans savedir. Le nom du fichier incluant l'extension est donné par ***strin***g, sans le chemin. Cette instruction retourne un identificateur de fichier - qui est un nombre unique pour identifier le fichier ouvert. L'identificateur de fichier est employé par d'autres instructions pour avoir accès à ce fichier. Si le fichier ne peut pas être ouvert - s'il n'existe pas, par exemple - 0 est retourné. Si les identificateurs de fichier ne sont pas définis comme var_nsave, ils deviennent invalides après une instruction de chargement (load), donc les opérations de fichier ne doivent pas être interrompues par des opérations de chargement / sauvegarde.

handle = file_open_game ***(strin***g)

Comme file_open_read(), mais cette instruction prend le fichier dans le répertoire du jeu, au lieu du répertoire de sauvegarde. Ce chemin qui est arbitraire peut être fourni dans le jeu. Le fichier doit réellement exister dans le répertoire du jeu, il ne peut pas être chiffré dans une ressource. Exemple:

```
filehandle = file_open_game("address.txt");
```

handle = file_open_write ***(strin***g);

Ouvre le fichier pour écrire dedans. Si le fichier n'existe pas dans le savedir, il sera créé ; autrement c'est le contenu précédent qui est effacé.

handle = file_open_append ***(strin***g);

Ouvre le fichier pour ajouter du texte à la fin. Si le fichier n'existe pas dans le savedir, il sera créé.

file_close ***(handl***e);

Ferme le fichier avec l'identificateur donné (handle). Un fichier doit être fermé avant que d'autres programmes puisse y avoir accès ou avant qu'il ne puisse être ouvert de nouveau dans un mode différent. Aussitôt que le fichier est fermé, l'identificateur devient invalide.

file_var_write ***(handle, va***r)

Ecrit le nombre ou la variable donné dans le fichier avec l'identificateur donné. Le fichier doit être ouvert en écriture ou en ajout. Le nombre est écrit avec 3 décimales, suivi par un blanc. Exemple:

```
var filehandle;
...
filehandle = file_open_write("myfile.txt");
file_var_write(filehandle,123456.789);
file_var_write(filehandle,sqrt(2));
file_close(filehandle); // le fichier contient à présent "123456.789 1.414 "
```

var = file_var_read ***(handl***e)

Retourne le nombre qui a été écrit dans le fichier avec l'identificateur donné et se positionne à la position suivante dans le fichier. Le fichier doit être ouvert pour la lecture. S'il n'y a aucun nouveaux nombres, 0 est retourné. Exemple (avec le fichier créé au dessus):

```
filehandle = file_open_read("myfile.txt");
temp = file_var_read(filehandle); // now temp == 123456.789.
temp = file_var_read(filehandle); // now temp == 1.414
file_close(filehandle);
```

file_str_write ***(handle, strin***g)

Ecrit la chaîne de caractères donnée dans le fichier avec l'identificateur donné. Le fichier doit être ouvert pour l'écriture ou l'ajout. Si la chaîne de caractères contient un '\n ', un retour à la ligne est inséré; ' \\ ' insère un ' \ '.

file_str_read ***(handle, strin***g)

Lit le texte du fichier avec l'identificateur donné dans la chaîne de caractères donnée, avant qu'un délimiteur ne soit atteint. Le délimiteur est ou l'octet '0' dans le fichier ou un retour à la ligne ou une virgule (','). De cette façon, la virgule séparant les chaînes de caractères d'un fichier de base de données, les données peuvent être lues. Le fichier doit être ouvert pour la lecture. La longueur originale de la chaîne de caractères ne sera pas modifiée.

file_chr_read ***(handle, strin***g)

Lit des caractères du fichier avec l'identificateur donné dans la chaîne de caractères donnée, avant que la chaîne de caractères ne soit pleine. Le fichier doit être ouvert pour la lecture. Le nombre de caractères lus est la longueur originale de la chaîne de caractères.

file_asc_write ***(handle, va***r)

Écrit un octet simple donné par le nombre ou la variable (0.. 255) dans le fichier avec l'identificateur donné. Le fichier doit être ouvert pour l'écriture ou l'ajout

var = file_asc_read ***(handl***e)

Lit un octet du fichier avec l'identificateur donné et continue à l'octet suivant dans le fichier. Le fichier doit être ouvert pour la lecture.

Instructions de contrôle

Les instructions suivantes contrôlent l'exécution de la fonction. Les instructions de contrôle ne peuvent pas être employées par des extensions DLL.

if ***(expressio***n) { ***instructions...*** }
if ***(expressio***n) { ***instructions...*** } else { ***instructions...*** }

Exécutera toutes les instructions entre la première paire d'accolades seulement si l'expression entre parenthèses est vraie (c'est-à-dire valeur non zéro). Autrement, les instructions entre la deuxième paire d'accolades (après **else**) seront exécutées. **Else** et le deuxième jeu d'instructions peuvent être omis.

L'expression peut être une comparaison entre deux sous expressions ou une comparaison de drapeaux. Seuls les paramètres de même type peuvent être comparés. Pour des comparaisons de drapeaux les valeurs mis (**on**) ou non mis (**off**) peuvent être comparées. Pour les comparaisons de paramètres non numériques la valeur **null** pour la non-existence peut être employée. Les opérateurs suivants pour la comparaison entre deux paramètres ou sous expressions sont disponibles :

||	Vrai si l'une **ou** l'autre est vraie (ou)
&&	Vrai si l'une **et** l'autres sont vraies (et)
! =	Vrai si toutes les deux ne sont pas égales
==	Vrai si toutes les deux sont égales
< =	Vrai si la première est inférieure ou égale à la deuxième
> =	Vrai si la première est supérieure ou égale à la deuxième
<	Vrai si la première est inférieure à la deuxième
>	Vrai si la première est supérieure à la deuxième.

Toutes les comparaisons se font avec une valeur de 0 pour faux et 1 pour vrai. Les comparaisons peuvent être combinées en employant des parenthèses. Exemples :

```
if (my.flag1 == off) { // si flag1 n'est pas mis (valeur 0),
        y = -1; // alors on met y et z à -1
        z = -1;
} else {
        y = 1; // sinon on met y et z à 1
        z = 1;
}

if (((x+3)<9) || (y==0)) {
// met z à 10 si x+3 et inférieur à 9 OU si y est égal à 0
        z = 10;
} else {
// met z à 5 dans tous les autres cas
        z = 5;
}
```

Notez s'il vous plaît que la comparaison "égal" est faite avec "==", pour la différencier de l'instruction de nomination avec "=". Employez toujours des parenthèses, comme dans l'exemple, pour déterminer l'ordre d'évaluation dans l'expression. Une instruction **if** peut apparaître dans les parenthèses d'une autre instruction **if**; de cette façon vous pouvez imbriquer plusieurs instructions **if**.

while ***(expressio***n) { ***instructions...*** }
break;
continue;

L'instruction **while** répétera toutes les instructions entre les accolades tant que l'expression entre les parenthèses est vraie (qui équivaut à non zéro). Cette répétition d'instructions est appelée une boucle (**loop**). L'expression

sera évaluée de nouveau au commencement de chaque répétition. Si vous voulez que la boucle fonctionne en permanence, employez simplement la valeur 1 pour l'expression.

While est souvent employé pour modifier une valeur lentement pendant un certain nombre d'encadrements, par exemple ouvrir une porte ou déplacer un ascenseur. Pour cela une instruction wait() (voir ci-dessous) doit être insérée entre les accolades, pour indiquer qu'un temps d'affichage doit passer pour chaque répétition.

L'instruction break; entre les accolades, terminera les répétitions et continuera avec la première instruction après la parenthèse fermée. L'instruction continue; commencera immédiatement à la répétition suivante. Si vous voulez un exemple pour l'utilisation de while regardez la fonction ent_rotate au début de ce chapitre.

goto ***label;***

Saute à une étiquette cible dans la fonction et continue à partir de là avec les instructions suivantes. L'étiquette (label) peut être n'importe quel nom suivi par des deux-points comme marque de cible n'importe ou entre deux instructions dans la fonction.

function ***paramètre1,paramètre2…***);
var = function (***paramètre1,paramètre2…***);

Commence la fonction dont on donne le nom, et passe les paramètres donnés qu'il attend (jusqu'à 4). Les paramètres peuvent être omis, dans ce cas les parenthèses restent vides. Après la fin de la fonction ou s'il y a l'instruction wait() (voir ci-dessous) dans cette fonction, il retourne à la fonction actuelle et continue de là avec l'instruction suivante. Par l'instruction de retour (voir ci-dessous) la fonction appelée peut passer un paramètre de résultat en revenant à la fonction appelante. En cas de la rencontre d'un wait(), la fonction nouvellement commencée aussi bien que la fonction actuelle peuvent s'exécuter toutes les deux simultanément au prochain cycle d'affichage. Le résultat de la fonction, s'il existe, peut être assigné directement à une variable ou à un pointeur. Exemple:

```
function average(a,b)
{
return((a+b)/2);
}
// est appelée de cette façon:
    x = average(123,456);
```

return;
return ***(variable***);

Termine la fonction. Dans le deuxième cas la variable donnée ou l'expression sont retournés à la fonction appelante, où il peut être évalué dans une expression. Exemple:

```
function distance()
{ if (my == null || you == null) { return(99999); } // big distance indicates error
return(vec_dist(my.x,you.x));
}

function beep_if_close() {
if (_distance() < 100) { beep; }
}
```

exclusive_global;
exclusive_entity;

Chaque fonction peut être commencée plusieurs fois, dans ce cas plusieurs instances de la même fonction peuvent fonctionner simultanément. Exclusive_global terminera toutes les instances de la fonction actuelle et empêchera ainsi les fonctions fonctionnant en parallèle de se gêner les unes les autres. Exclusive_entity terminera toutes les autres fonctions commencées par my, c'est-à-dire l'entité courante, à l'instruction wait() suivante.

proc_late;

Place la fonction actuelle tout à la fin de la liste de planificateur de fonction. Nécessaire quand la fonction a besoin de résultats ou la position d'une entité ou de la caméra qui sont calculés par d'autres fonctions. Cela garantit que les autres fonctions seront exécutées plus tôt pendant chaque cycle d'encadrement.

wait ***(numbe***r);
waitt ***(numbe***r);

Fait que la fonction fait une pause pour le nombre donné de cycles d'affichage ou tick. Tout ce qui est à l'extérieur de cette fonction – le rendu, la communication d'utilisateur, le changement de variable globale et l'exécution d'autres fonctions - sera exécuté pendant cette pause. Wait(1) est souvent employé à la fin de la boucle while, pendant laquelle une valeur doit changer sans à-coup pendant un certain temps (porte qui s'ouvre par exemple). Si la fonction avec l'instruction wait() a été démarrée par une autre fonction, cette autre fonction continuera aussi et exécutera ses instructions suivantes pendant la pause. Waitt() fait la même chose que wait(), mais la fonction fait une pause pour un nombre donné de tick au lieu de cycles d'affichage. Cette instruction est capable d'attendre pendant une période fixe de temps. Plus le nombre donné est haut, plus le temps de pause sera précis. Le temps minimal que cette instruction peut attendre est un cycle d'affichage.

Instructions d'entités

Les instructions suivantes sont employées pour contrôler les entités, l'environnement et la détection de collision dans le jeu :

content ***(vecto***r);

Vérifie le matériel bloc à la position donnée par le vecteur. L'instruction retourne une valeur de content_empty (1) si l'espace est vide, content_passable (2) si le bloc est passable ou content_solid (3) si un bloc solide est détecté à la position donnée. Les blocs dans le niveau aussi bien que les entités de carte sont détectés. You est mis à la position de l'entité de carte détectée. Cette instruction ne peut pas encore être employée par des extensions DLL. Exemple:

```
// vérifier si la position de la caméra est sous l'eau dans le niveau actuel
result = content(camera.x);
if (result == content_passable) { // sous l'eau
fog_color = 2; // met la globale fog à 2, fog bleu
} else {
fog_color = 0; // mets la globale fog à off
}
```

Ent_alphaset ***(fram***e, ***percen***t);

Produit une carte de transparence alpha pour l'entité sprite my en multipliant la brillance de l'image bitmap sprite par un facteur en pour cent (-100 .. 100). Un facteur négatif produit une carte alpha inverse. La carte alpha peut être calculée individuellement pour chaque encadrement de sprite et donne de meilleur contrôle que le drapeau flare. Si le paramètre d'encadrement est 0, la carte alpha est calculée pour tous les encadrements du sprite. La valeur utile est dans la gamme de 5 .. 30. Exemple:

```
ent_alphaset(0,-15); // crée une carte de transparence pour de la fumée noire épaisse
ent_alphaset(0,5); crée une carte de transparence pour de la fumée blanche mince
```

Ent_alphaset peut être exécuté seulement une fois par entité, les instructions alphaset qui viennent après sont ignorées. Les entités du même fichier doivent avoir la même carte alpha. Les cartes alpha sont seulement efficaces dans les modes 16 ou 32 bits.

ent_create (***string, vector, action***);

créera une nouvelle entité sur le serveur pendant le jeu. Surtout employé pour la création de balles ou d'explosions ou pour la création de joueurs dans un jeu multi joueur. Le nom d'un fichier wmb, mdl, pcx, bmp ou tga pour créer l'entité carte, modèle ou sprite qui en découle est donné par *string*. Notez que le nom de fichier doit être donné aditionnellement dans des parenthèses angulaires (< >) avec un liant (bind) ou une déclaration de chaine pour inclure le fichier dans la ressource ou le dossier de publication.

On peut donner la position ou l'on souhaite que l'entité apparaisse par vector. L'action donnée sera attachée à l'entité et commencera à fonctionner sur le serveur immédiatement après la création, avec le synonyme my de mis à l'entité créée, et you de mis à l'entité créant (s'il y a). Ent_create() retourne un pointeur pour l'entité créée et travaille seulement dans un niveau qui est déjà chargé. Donc il ne peut pas être employé au début même du jeu ou au début même d'une action d'entité ou sur un client avant qu'il ne soit connecté au serveur. Exemple :

```
string flash_pcx = <flash.pcx>; // la chaine peut être définie en "" ou <>
...
you = ent_create(flash_pcx,temp,flash_prog);
```

ent_morph (***entity,string***);

Cette instruction transforme l'entité my en une autre, même de type différent, par exemple un modèle dans un sprite ou un sprite dans une entité de carte. Le nom d'un fichier wmb, mdl, pcx, bmp ou tga pour créer l'entité carte, modèle ou sprite qui en découle est donné par *string*. Notez que le nom de fichier doit être donné aditionnellement dans des parenthèses angulaires (< >) avec un liant (bind) ou une déclaration de chaine pour inclure le fichier dans la ressource ou le dossier de publication. Exemple:

Ent_morph (my , « explo.pcx »); *// change MY en un sprite explosion*

ent_purge ***(entity***);

Libère la mémoire utilisée par l'entité. Utile pour libérer de la mémoire vidéo lorsque les entités de certains types ne sont désormais plus visibles. Notez que ça ne fonctionne que pour des entités modèles ou les cartes ou les terrains ne peuvent pas être purgés. La mémoire vidéo est automatiquement allouée de nouveau quand on voit l'entité la prochaine fois - donc cette instruction ne fera pas de mal. Cependant, n'importe quelles instructions ent_alphaset doivent être exécutées de nouveau. Ainsi l'instruction de purge, suivie par ent_alphaset, peut aussi être employée pour changer le facteur alpha d'une texture d'entité. Exemple:

ent_purge(player); *// purge l'entité player lorsque nous sommes en mode 1 joueur*

ent_preload ***(entity***);

Préaffecte la mémoire de texture pour l'entité donnée. Si de la mémoire vidéo est préaffectée, il n'y aura pas retard quand l'entité apparaîtra la première fois. De la mémoire vidéo est automatiquement préaffectée pour des entités placées au niveau actuel, mais pas pour les entités qui sont créées pendant le jeu avec ent_create ou ent_morph et pas pour les entités globales.

ent_remove ***(entity***);

Enlève l'entité donnée du monde et termine toutes les fonctions pour lesquelles cette entité est my, sauf celle qui fonctionne actuellement. Aucune fonction ne doit ensuite jamais essayer d'avoir accès à l'entité enlevée (via un synonyme my ou you) de nouveau.

ent_frame ***(string, var***);
ent_cycle ***(string, var***);

Ces instructions mettent la frame de l'entité modèle my et les paramètres next_frame à la frame d'animation dont le nom commence par la chaîne de caractères donnée. Si l'animation consiste en plusieurs frame - par exemple "walk1", "walk2", etc. - le var donne un pourcentage dans le cycle d'animation entier, dont les nombres

d'encadrement sont calculés automatiquement. 0 pour cent donnent le premier encadrement de l'animation et 100 pour cent ou donnent le dernier (ent_frame) ou répéte encore à partir premier(ent_cycle). Exemples:

```
ent_frame("jump",90); // 90 percent of jump animation
ent_cycle("jump",90); // 90 percent within jump cycle
```

Si le modèle a trois encadrements d'animation nommés "jump1", "jump2", "jump3", qui sont les encadrements 10, 11 et 12, l'instruction ent_frame dans l'exemple met frame à 11.8 et remet à 0 next_frame. l'instruction ent_cycle met cependant l'encadrement à 12.7 et next_frame à 10, pour aller et venir entre le dernier et le premier encadrement.

ent_next ***(entity***);

Toutes les entités locales sont triées dans une liste interne. Cette instruction retourne un pointeur de la première entité dans la liste lorsque le paramètre entité est NULL, et un pointeur sur l'entité suivante autrement. Lorsque la dernière entité est donnée comme paramètre, l'instruction retourne NULL. Avec cette instruction quelque chose peut être fait à toutes les entités dans un niveau - ou une certaine entité peut être trouvée. Exemple:

```
function hide_all_ents()
{
   you = ent_next(NULL); // se positionne sur la première entité
   while (you != NULL) { // répète jusqu'à ce qu'il n'y ait plus d'entités
   you.invisible = on; // rend l'entité invisible
   you = ent_next(you); // va à la prochaine entité
   }
}
```

Les paramètres de toutes les entités peuvent être lus, mais le changement de paramètres a de l'effet seulement sur les entités actives (celles qui ont une action attachée).

ptr_for_name ***(strin***g);

Retourne un pointeur pour l'entité avec le nom WED donné. Les noms peuvent être assignés aux entités par [Properties] dans WED. De cette façon, les entités placées dans un niveau peuvent avoir accès aux fonctions de C-SCRIPT même si elles n'ont aucune action elles-mêmes. Exemple:

```
string entname[30];
...
you = ptr_for_name("waffe1_mdl_012"); // obtient le pointeur de cette entité
str_for_entfile(entname,you); // récupère le nom du fichier, ici "waffe1.mdl"
you.string1 = entname; // fait apparaître le nom du fichier
you.enable_touch = on; // lorsque l'entité est touchée par la souris
you.event = handle_touch;
```

Notez que quand une entité est statique – ce qui signifie qu'il n'a aucune action attachée – ses paramètres peuvent être lus, mais ne doivent pas être changés pendant le jeu. Egalement ses variables ne sont pas disponibles.

vec_for_vertex ***(vecto***r, entity,***numbe***r);

Met le vecteur donné à la position courante xyz du vertex avec le nombre donné de l'entité modèle ou de l'entité terrain. Cette instruction est extrêmement utile pour la création d'entités de plusieurs parties. Les armes peuvent être attachées et changées sans devoir changer le modèle entier, les effets lumineux peuvent être attachés à des phares ou des moteurs à réaction ou certaines parties de modèles peuvent émettre des effets de particule. Le nombre de vertex est indiqué par MED en sélectionnant un vertex. Exemple:

```
// prendre modèle d'entité vertex no. 248 qui émet régulièrement des particules
vec_for_vertex(temp,my,248); // obtient la position XYZ du vertex 248
effect(particle_flare,1,temp,nullvector);.
```

vec_for_normal ***(vecto***r, entity,***numbe***r);

Comme pour vec_for_vertex, mais retourne le vertex normal du modèle donné ou du terrain vertex. Le Vertex normal est la moyenne des normales des triangles environnants.

vec_for_mesh ***(vecto**r, entity,**numbe**r);
vec_to_mesh ***(vecto**r, entity,**numbe**r);

Exécute la déformation de maille en temps réel de l'entité d'un terrain ou d'un modèle. Vec_for_mesh met le vecteur donné aux coordonnées de maille du numéro de vertex donné.
Vec_to_mesh met les coordonnées de la maille du numéro de vertex donné au vecteur donné. La maille est déformée non seulement sur cette entité, mais sur tous les modèles ou terrains avec le même fichier. Les déformations de maille sont perdues à la sortie du jeu. Elles ne peuvent pas augmenter indéfiniment la taille du modèle : la déformation extérieure des vertices en atteignant deux fois les frontières modèles. Pour une déformation de maille lisse le modèle doit avoir le drapeau 'high prec' de mis dans MED. Exemple :

```
function dent(ent,num) // make a "dent" into a mesh
{
   vec_for_mesh(temp,ent,num);
   vec_scale(temp,0.9); // move the vertex inwards by 10%
   vec_to_mesh(temp,ent,num);
}
```

vec_for_min ***(vecto**r, entity);
vec_for_max ***(vecto**r, entity);

Place le vecteur aux frontières supérieures et inférieures de l'armature courante de l'entité modèle donnée. Ceci peut être employé pour calculer un cadre de rebondissement des entités avec plus de précision que par ses paramètres pré-établis min_x... max_x.

ent_move ***(vector_reldis**t, ***vector_absdist***);

Déplace mon entité my sur une certaine distance et exécute une détection de collision pendant le mouvement (voir le chapitre entité pour la détection de collision). Le premier vecteur donne une distance relative (dans les coordonnées de rotation de l'entité, c'est-à-dire la direction à laquelle l'entité fait face), le deuxième vecteur donne une distance absolue (dans les coordonnées du monde). Le mouvement résultant est une combinaison des deux distances. Normalement le premier vecteur est employé pour la vitesse de propulsion de l'entité et le deuxième pour des forces externes, comme la gravité et la dérive. Pour mettre à zéro l'un de ces vecteurs on peut donner le vecteur nullvector prédéfini.

Ent_move retourne la quantité de distance couverte. Si l'entité ne peut pas se déplacer du tout en raison d'un blocage par des obstacles, il retourne 0. Le vecteur my_speed prédéfini est mis à la distance absolue résultante. Si l'entité est entrée en collision avec quelque chose le vecteur normal est mis à la normale de la surface entrant en collision - qui est un vecteur de longueur 1, pointant perpendiculairement loin de la surface. Il peut être employé pour déterminer l'orientation superficielle. Par exemple, si normal.z est près de 1, l'entité est entrée en collision avec la terre; si normal.z est près de -1, il est entré en collision avec le plafond; si normal.z est près de 0, il a frappé un mur vertical. La surface d'un sprite ou l'entité modèle est considérée comme un cylindre vertical ici. Le vecteur de rebond (bounce) prédéterminé est mis à la direction dans laquelle l'entité entrante en collision rebondirait de la surface et pourrait ainsi être employée pour mettre en œuvre un comportement rebondissant.

Avant de faire ent_move, le mode de mouvement doit être mis par la variable move_mode prédéfinie auparavant. Les combinaisons de mode de mouvement suivantes sont disponibles :

ignore_you	- ignore l'entité you sur la détection de collision.
ignore_passable	- ignore tous les blocs et entités passables.
ignore_passents	- ignore les entités modèle et sprites passables.
ignore_maps	- ignore toutes les entités de carte.
ignore_models	- ignore tous les modèles.

ignore_sprites - ignore tous les sprites.
ignore_push - se déplace sur toutes les entités avec des valeurs de push inférieures à my.
activate_trigger - permet des événements de déclenchement pendant le mouvement.
glide - glisse le long des murs et des entités en cas d'impact.

Le mode ignore_passents peut toujours détecter des blocs passables et dresser la carte d'entités si le mouvement est terminé à l'intérieur d'un bloc passable, la variable in_passable prédéfinie est mise à 1. Si le mouvement est terminé à l'intérieur d'un bloc solide (ce qui peut seulement arriver si des entités passables sont déplacées), la variable in_solid est mise à 1.

Sur une collision, ent_move peut déclencher des événements de collision sur le déplacement aussi bien que sur d'autres entités. Si glide est activé, ent_move essayera de se déplacer autant que possible en glissant le long des surfaces ou en contournant les obstacles. La variable move_friction (valeur de 0 à 1, par défaut 0.25) détermine la friction pour les glissements le long des surfaces. A 0 il n'y a pas de friction du tout, à 1 l'entité colle à la surface et il n'y a pas de glissement du tout.

Ignorer l'entité you peut être employé pour prévenir qu'une balle est immédiatement coincée dans le baril ou dans l'entité qui l'a tirée. Les modes de mouvement peuvent être combinés par l'addition simple, comme cela :

```
move_mode = ignore_you + ignore_passable + ignore_push + activate_trigger + glide;
result = ent_move(reldist,absdist); // same as move(my,reldist,absdist);
```

trace ***(vectorfro***m, ***vectorto***);

C'est l'instruction générale qui est employée par des entités pour détecter leur environnement. Il envoie un rayon de la position vectorfrom à la position vectorto et vérifie si ce rayon frappe un obstacle en avançant. Dans ce cas le vecteur cible (target) prédéfini est mis à la position où le rayon frappe la surface de l'obstacle (peut-être pour placer une tache de sang là), le vecteur normal prédéfini est mis à la normale de cette surface. L'instruction retourne la distance du point de coup. Si rien n'est frappé entre les deux positions, il retourne 0.

Si l'obstacle était une entité, le synonyme you est mis à cette entité ; autrement il est mis à null. En déclenchant l'événement shoot est activé et le drapeau de l'entité de coup enable_shoot est mis, sa fonction event est déclenchée par event_type qui est mis à event_shoot (les événements sont décrits dans le chapitre d'entité).

Par la variable trace_mode prédéfinie le mode pour le traçage peut être composé en ajoutant plusieurs drapeaux de mode. Trace_mode doit être mis immédiatement avant l'instruction trace. Les drapeaux suivants sont disponibles :

ignore_me - ignore l'entité my sur la détection de collision.
ignore_you - ignore l'entité you sur la détection de collision.
ignore_passable - ignore tous les blocs et entités passables.
ignore_passents - ignore les entités modèle et sprites passables.
ignore_maps - ignore toutes les entités de carte.
ignore_models - ignore tous les modèles.
ignore_sprites - ignore tous les sprites.
ignore_push - ignore toutes les entités avec des valeurs de push inférieures à my.
use_box - emploie la boîte de limitation (min_x, max_x etc.) ou la coque de l'entité my.
activate_shoot - permet le déclenchement d'event_shoot de l'entité de coup.
activate_sonar - permet le déclenchement d'event_sonar de l'entité de coup.
scan_texture - met tex_name et d'autres paramètres de la surface cible.

Ignore_passent ignore le modèle et des entités de sprite avec le drapeau passable de mis à 1 (on), mais détecte toujours des blocs passables ou des entités de carte. Il met les drapeaux prédéfinis in_passable, quand le point de départ est dans un bloc passable et on_passable quand le rayon frappe la surface d'un bloc passable. Il peut être employé pour détecter de l'eau au-dessous d'une entité.

Scan_texture peut être employé pour donner aux acteurs quelque vision, en recouvrant le nom de texture et la brillance de la surface de coup. Après l'instruction, la chaîne de caractères prédéfinie tex_name donne le nom de texture de la surface frappée par le rayon ou le nom de fichier d'entité si un modèle, sprite ou terrain a été frappé. Les drapeaux prédéfinis tex_flag1.. tex_flag8 reflètent les états de la texture Flag1.. Flag8, qui peut être mise par WED. La variable tex_light donne la carte de brillance d'ombre (0 .. 255) à la position de coup dans le niveau, tex_fog donne la valeur de fog/albedo. Le nom de texture peut être employé pour vérifier la sorte de plancher au-dessous d'une entité. Par l'instruction str_cmpi () le nom de texture de plancher peut être décidé et quelque comportement de marche différent peut être mis. Une autre possibilité est de faire devenir un joueur invisible pour ses ennemis tant qu'il se cache dans les ombres. Si rien n'a été frappé, tex_name et les autres paramètres ne sont pas mis.

Use_box ne trace pas une ligne, mais un rayon 'épais' avec la taille de la boîte de limitation de l'entité my pour des collisions avec des sprites ou des entités modèles et sa coque grasse ou étroite pour des collisions avec le niveau ou avec des entités de carte. Le résultat livré donne en arrière la distance du point de coup au plan le plus proche de la boîte de limitation dans ce cas. Pour que la détection de collision fonctionne, la boîte ou la coque doivent être à l'extérieur de la cible. Une trace verticale avec use_box est employée par les scénarios de calibre (template) pour détecter la distance du sol aux pieds des entités sur le plancher. Des petits trous ou des grilles apparaîtront remplis pour use_box. Exemples :

Exemples:

```
// évaluer la texture du plancher
    vec_set(temp, MY.x);
    temp.z -= 500; // trace en bas 500 quants ci-dessous
    trace_mode = ignore_me + ignore_passable + ignore_models + ignore_sprites + scan_texture;
    trace (my.x,temp); // maintenant TEX_NAME est mis au nom du plancher en dessous de mon entité my

// regarder si l'ennemi You peut être tiré à partir de ma position et si oui, l'endommager
    trace_mode = ignore_me + ignore_passable + activate_shoot;
    trace (my.x,you.x); // si l'entité You est visible de l'entité My, son événement SHOOT ezt déclenché
```

scan_entity ***(arrayFro***m, ***vektorWidth***);

Balayera toutes les entités dans un segment sphérique de largeur et de direction données. Le premier paramètre est un tableau de 5 valeurs, dont d'abord trois (x, y, z) donnent l'origine de la sphère. Les deux suivants donnent la direction du balayage, c'est-à-dire la ligne centrale du segment sphérique, comme les angles pan et tilt. Au lieu d'un tableau, le paramètre x d'une entité ou d'une vue peut être donné également. Les valeurs pan et tilt du deuxième vecteur donnent la largeur horizontale et verticale du segment. La valeur Z du troisième vecteur donne la profondeur du balayage. Si les valeurs du secteur de balayage sont mises à 360, le balayage sera une sphère entière, comme une explosion. Des petites valeurs de secteur de balayage donnent un très petit cône, comme une décharge d'un fusil.

Si n'importe quelle entité avec enable_scan de mis est trouvée avec son centre dans le segment de balayage, c'est la fonction event qui est déclenché avec event_type de mis à event_scan, result donne la distance et you est mis comme entité qui balaye, s'il y a bien entendu. Aussi, si l'entité my a le drapeau enable_detect de mis, c'est la fonction event qui est déclenché avec event_type de mis à event_detect, et you est mis comme entité qui balaye. De cette façon pour chaque entité détectée une fonction event_detect est déclenchée.

Scan_entity() retourne la distance de l'entité trouvée la plus proche, si il y a; autrement elle retourne zéro. Pour empêcher les déclenchements de balayage de l'entité elle-même, le balayage ne détectera pas l'entité my. Les instructions balayent à travers les murs - donc vous ne pouvez pas l'employer pour des coups de feu, utilisez trace() pour cela. L'instruction scan, cependant, est très utile pour ouvrir des portes, mettre des commutateurs, détecter ou alerter des ennemis et cetera. Exemple:

```
function operate() { // balaye près des portes ou des commutateurs pour les faire fonctionner
temp.pan = 120;
temp.tilt = 120;
temp.z = 200;
scan_entity (camera.x,temp); }.
```

scan_path ***(arrayfro***m, ***vektorwidth***);

Travaille comme scan_entity(), mais trouve le prochain chemin avec le premier point du parcours dans le cône de balayage et l'attache à l'entité. Si le chemin n'est pas trouvé, retourne 0; Autrement c'est la distance jusqu'au premier point qui est retournée. Le vecteur cible (target) est mis à la valeur du premier point du chemin. Exemple:

```
temp.pan = 360; // sphere entière
temp.tilt = 180;
temp.z = 1000;
scan_path(my.x,temp);
```

ent_path ***(strin***g);

Trouve le chemin avec le nom donné et l'attache à l'entité my. Si le chemin n'est pas trouvé, retourne 0; sinon c'est le nombre de points de chemin qui est retourné. Le vecteur cible (target) est mis à la valeur du premier point du chemin. Exemple:

```
ent_path("path_001");
```

ent_waypoint ***(vecto***r, ***numbe***r);

Mets le vecteur du point de chemin de l'entité my à la valeur donnée. Le nombre total de point du chemin est retourné.

ent_nextpoint ***(vecto***r);
ent_prevpoint ***(vecto***r);

Si le vecteur a été mis au point de chemin de l'entité my, cette instruction le met au point de chemin suivant ou précédent. Autrement, le vecteur est mis au dernier point du chemin. Le nombre de nouveau points est retourné. Le petit exemple suivant laisse un acteur marcher le long du chemin suivant, dont le premier point de chemin est à moins de 1000 quants de la position de l'entité :

```
action walk_path // l'entité se promène selon un chemin circulaire
{
	temp.pan = 360; // trouve et attache un chemin à l'entité
	temp.tilt = 180;
	temp.z = 1000;
	result = scan_path(my.x,temp);
	if (result == 0) { return; } // aucun chemin n'est trouvé
	ent_waypoint(my._target_x,1); // stocke le premier point de chemin

	while (1)
	{
		temp.x = my._target_x - my.x; // trouve la direction du point de chemin
		temp.y = my._target_y - my.y;
		temp.z = 0;
		result = vec_to_angle(my_angle,temp);
	// près de la cible, alors trouve le prochain point de chemin
		if (result < 25) { ent_nextpoint(my._target_x); }
	// tourne et marche vers la cible
		force = my._force;
		actor_turnto(my_angle.pan); // tourne vers le point de chemin
		actor_move(); // marche tout droit
		wait(1);
	}
}
```

effect ***(function, number, vec_pos, vec_vel***);

Cette instruction peut être employée pour créer un essaim de petites particules bitmap qui se déplacent, créer des traînées de fusée, des explosions, des rayons laser, des torpilles, la pluie, des tempêtes de neige, la tornade ou tout ce qui y ressemble. Vec_pos donne la position de départ de l'essaim, number donne le nombre de particules ,

vec_vel donne une vitesse ou une longueur de rayon et function définit le comportement de chaque particule. La fonction doit être définie avant. Elle s'exécutera de manière permanente pendant la durée de vie de chaque particule simple et peut changer la position, la vitesse, la couleur, la taille et l'image bitmap de chaque particule par l'intermédiaire du pointeur my, de la même façon qu'une entité normale. Elle peut même créer des nouvelles particules "enfants' - vous avez la flexibilité pour employer des particules pour les effets de chaque description. Les particules disparaissent au loin à la moitié de clip_range.

Notez que tous les drapeaux d'entités et les paramètres sont valables pour les particules (voir le chapitre sur les entités). Seuls les paramètres suivants peuvent être utilisés : lifespan, x, y, vel_x, vel_y, vel_z, gravity, size, alpha, red, green, blue, bmap, flare, bright, transparent, move, beam, strek, function, flag1…flag8, skill_x, skill_y, skill_z. Parce que vous pouvez avoir beaucoup de milliers de particules, la fonction doit être aussi rapide et aussi courte que possible. Les instructions wait() sont interdites ici. Exemple :

```
// helper function: sets the vector to random direction and length
function vec_randomize(&vec,range)
{
	vec[0] = random(1) - 0.5;
	vec[1] = random(1) - 0.5;
	vec[2] = random(1) - 0.5;
	vec_normalize(vec,random(range));
}

// helper function: fades out a particle
function part_alphafade()
{
	my.alpha -= time+time;
	if (my.alpha <= 0) { my.lifespan = 0; }
}

// particle function: generates a fading explosion into vel direction
function effect_explo()
{
	vec_randomize(temp,10);
	vec_add(my.vel_x,temp);
	my.alpha = 25 + random(25);
	my.bmap = scatter_map;
	my.flare = on;
	my.bright = on;
	my.beam = on;
	my.move = on;
	my.function = part_alphafade; // change to a shorter, faster function
}

...
vec_scale(normal,10); // produce an explosion into the normal direction
effect(effect_explo,1000,my.x,normal);
```

Instructions multimedia

Les instructions suivantes contrôlent le jeu de sons, la musique et des films.

snd_play (***soun***d, ***var,balance***);

Joue un son précédemment défini avec le volume donné par la variable (0 .. 100). Renvoie un identificateur (handle) qui permet davantage de manipulation ou d'arrêter prématurément le bruit. L'identificateur est valide aussi longtemps que le son est joué. Utilisez var_nsave pour enregistrer les identificateurs pour les empêcher d'être écrasés avec des valeurs incorrectes par une instruction de chargement (load). Le paramètre ***balance*** donne la position stéréo du bruit en pourcentage (-100.. +100). Vous pouvez créer un effet d'écho en jouant le même son deux fois, avec un léger retard entre les 2. Jusqu'à 32 sons peut être joués simultanément.

snd_playfile ***(filename***, ***volume,balance***);

Comme snd_play, mais joue un fichier son au format .WAV. Utilisez cette instruction pour jouer des sons très long qui seront joué une seule fois.

snd_loop ***(soun**d, **volume,balance**);

Comme snd_play; mais le son jouera continuellement avant qu'il ne soit arrêté explicitement par une instruction snd_stop.

ent_playsound ***(entity, sound, volume);***

Joue un son 3D, joué à la position de l'entité entity. La valeur (en quants) d'un son d'entité est 10 fois son volume. Le volume peut être mis à plus de 100 pour donner une valeur énorme; le son lui-même, bien sûr, n'est pas joué plus fort qu'avec le volume 100. Le son emploiera les capacités stéréo et 3D du matériel du son et produira un changement de doppler si l'entité se déplaçait vers la caméra pendant que l'instruction est exécutée. Notez s'il vous plaît que l'entité doit déjà exister quand le son est joué – ce qui signifie que si elle a été créée, il faut laisser passer au moins 1 cycle d'encadrement auparavant.

snd_stop ***(handle***);

Arrête le son avec le numéro d'identificateur donné par la variable.
Exemple:

```
sound wave = <wave.wav>;
var_nsave wavehandle;
...

wavehandle = snd_play(wave,50,-75);
waitt(50);
snd_stop(wavehandle);
```

snd_tune ***(handl**e, **volume**, **fréquence,balance**);

Modifie le son avec le numéro d'identificateur donné par handle. Il mettra un nouveau ***volume*** (1 .. 100) et une nouvelle ***fréquence*** (10 .. 1000, en pourcentage de la fréquence originale) et la ***balance*** (-100 à +100). Si un de ces paramètres est à 0, sa propriété reste inchangée.

snd_playing ***(handl**e);

Si le son avec l'identificateur donné joue toujours, cette fonction retourne 1, autrement 0.

Les instructions multimédia qui suivent ne peuvent pas être utilisées par les extensions DLL

play_song ***(music, var);***

commence une nouvelle chanson de fond, qui est répétée jusqu'à la prochaine instruction play_song (). Music est un nom précédemment défini pour un fichier midi, var donne le volume pour cette chanson entre 1 et 100. En donnant un volume zéro l'exécution du fichier midi est arrêtée.

play_song_once ***(music, var);***

Comme play_song, mais joue la chanson une seule fois.

C play_cd ***(varfrom, varto);***

Joue les pistes audio d'un CD. Varfrom indique la première, varto est la dernière piste à être jouée (minimum 1, maximum 99). Si varto a une valeur plus grande que le nombre de pistes sur le CD, le CD sera joué jusqu'à sa fin. Si varfrom est 0, le CD jouant actuellement sera arrêté; si varto est 0, le CD reprendra la où il a été arrêté. Si tous les deux sont 0, rien n'arrivera

Après chaque exécution de play_cd la variable prédéfinie cd_track aura le nombre de pistes actuellement jouées ou 0, si le CD ne joue pas. Entre deux, cd_track ne changera pas; pour jouer la piste actuelle continuellement, faites play_cd 0,0; l'instruction doit être exécutée de façon répétitive, par exemple chaque seconde.

C play_moviefile ***("filename");***

Commence une séquence de film en plein écran. Le nom de fichier est un fichier d'animation AVI. Il ne peut pas être joué depuis un fichier de ressource, mais doit exister comme un fichier externe, donc on doit donner le nom de fichier entre "" au lieu de <>. Pendant que le film joue le rendu et les actions d'entité seront suspendues. La variable prédéfinie movie_frame contient le nombre de frame actuellement montrée ou 0 si le film est arrivé à la fin.

stop_movie;

Arrête le film qui joue actuellement.

Instructions Input / output (entrée/sortie)

Les instructions suivantes peuvent être utilisées pour des actions spéciales entre le jeu et le monde extérieur :

inkey ***(string***);

Attends la saisie d'une chaîne de caractères au clavier pour la mettre dans ***string***. La configuration du clavier local sera activée automatiquement. L'instruction attend alors la fin de l'entrée via la touche [Entrée] et continue alors avec la fonction, identique à wait(). Cependant, les instructions de sauvegarde/chargement (save/load) ne seront pas exécutées pendant une entrée inkey. [Echap], [Haut], [Bas], [PgPréc] ou [PgSuiv] vous permet d'interrompre l'entrée à tout moment. Le contenu précédent de la chaîne de caractères sera alors reconstitué. Le texte à entrer peut être édité en employant les touches [Espace arrière], [Suppr], [Droit], [Gauche], [Début] et [Fin].

Si la chaîne de caractères apparaît dans un texte affiché à l'écran (voir ci-dessous), l'entrée aussi bien que le curseur (caractère clignotant #127 de la fonte correspondante) est visible. Si la fin de la chaîne de caractères est atteinte (résultant de la longueur de la chaîne de caractères initiale et de la définition de la chaîne de caractères), aucune nouvelle entrée au clavier ne sera acceptée. Les espaces à droite seront coupés. Après la fin de l'entrée la variable result aura la valeur 27 si l'entrée a été interrompue avec [Echap], 72 avec [Haut], 73 avec [PgPréc], 80 avec [Bas] ou 81 avec [PgSuiv] et 13 pour une fin normale avec la touche [Entrée] La longueur de la chaîne de caractères peut être évaluée avec la variable prédéfinie str_len.

execute ***(string***);

Exécute le contenu de chaîne de caractères comme instruction. Dans un but de mise au point, elle peut être employée pour taper des instructions C-SCRIPT pendant le jeu, comme sur une console, pour changer des variables ou des paramètres et observer les résultats immédiatement. Exemple:

```
function console() {
	inkey (exec_buffer);.
	if (result==13) { // terminé avec Entrée?
		execute (exec_buffer); } // alors exécute la chaîne de carctère
}
```

exec ***(string1, string2);***

Exécute un programme externe .exe, qui doit être placé dans le chemin path. String1 donne le nom du programme, string2 un paramètre de ligne de commande.

screenshot ***("name", var***);

Capture le contenu de l'écran et le sauve comme fichier .pcx. Le nom de fichier est composé avec la chaîne de caractères (name) donnée (maximums 5 caractères) plus un nombre à 3 chiffres (donné par var) et l'extension ".pcx". Dans le mode 8 bits le fichier sera sauvegardé comme un fichier pcx 8 bits, dans le mode 16 bits comme fichier pcx 24 bits couleur vraie. Les copies d'écran en mode 32 bits ne sont pas possibles.

inport ***(val, port***);

Le ***val*** donné prend la valeur du port d'entrée/sortie dont l'adresse est donnée par la valeur ***por***t. A travers cette instruction vous avez accès directement au matériel du PC pour, par exemple, contrôler directement un périphérique externe ou implémenter un nouveau périphérique. Notez que cette instruction ne fonctionne pas sous NT ou Win2000. Exemple:

```
var input;
inport input,372; // donne à input la valeur du contenu du port #372 en décimale
```

outport ***(val, port);***

L'octet donné par ***val*** est envoyé au port dont l'adresse est donné par ***port***. Notez que cette instruction ne fonctionne pas sous NT ou Win2000. Seules les valeurs entre 0 et 255 sont valides.

handle ***(object***);

Retourne un identificateur d'objet. Un identificateur est un nombre unique qui peut être assigné à une variable, et identifie en interne cet objet. L'objet peut être une entité, une action, une chaine, une image bitmap, un panneau ou un pointeur sur eux.
Un identificateur est similaire à un pointeur mais il est juste un nombre et peut donc être assigné à une variable ou à une variable d'entité (skill). Après une opération de sauvegarde / chargement, vous êtes assuré qu'un identificateur généré par cette instruction fera toujours référence au même objet – ce qui n'est pas garantit pour les paramètres pointeur. Pour employer l'objet auquel l'identificateur fait référence, il peut être converti en pointeur par l'instruction suivante.

ptr_for_handle ***(var***);

Assigne l'objet référencé par identificateur donné à un pointeur. Exemple :

```
my.skill40 = handle(you); // store a handle to the YOU entity
...
you = ptr_for_handle(my.skill40) // get the YOU entity back;
```

De cette façon, des objets arbitraires peuvent être stockés dans des variables d'entités ou des variables tableaux.

P select ***(dataview, field name, string);***
P unselect ***(dataview, field name, string);***
P and_select ***(dataview, field name, string);***
P or_select ***(dataview, field name, string);***

Select sélectionne tous les enregistrements de l'objet dataview donné, si les premières lettres du champ de chaîne de caractères avec le nom donné correspondent à la chaîne de caractères donnée. Si la chaîne de caractères (string) est vide, tous les enregistrements sont sélectionnés. Unselect fait le contraire, il désélectionne les enregistrements trouvés et ne change pas les autres. And_select transforme uniquement les enregistrements déjà sélectionnés et or_select uniquement les enregistrements non sélectionnés de dataview.

Instructions game flow (flux du jeu)

Les instructions suivantes contrôlent le flux de jeu et le changement de niveau. Elles ne peuvent pas être appelées directement par des fonctions de DLL externes.

save ***("name", var);***

Compresse, chiffre et sauvegarde l'état de jeu actuel sous un nom donné plus un nombre dans le dossier de jeu (voir savedir). Le fichier est écrit un cycle d'encadrement après l'instruction. Le nom du fichier sauvegardé est composé des cinq premières lettres de la chaîne de caractères donnée plus un nombre à trois chiffres donné par var plus l'extension .sav. Les éléments suivants du jeu seront sauvegardés :

- La carte courante,
- Toutes les variables,
- Tous les pointeurs,
- Toutes les chaînes de caractères changées,
- Toutes les fonctions touche (key) et boutons (on_f1 etc)
- L'état de tous les fonctions qui tournent actuellement,,
- Tous les objets view, panel, et text,
- Toutes les entités qui ont une action attachée.

load ***("name", var);***

Charge, décompresse et décode le jeu qui a été sauvegardé sous le nom ***name*** plus un nombre à trois chiffres donné par var. Le fichier est lu un cycle après l'instruction.

save_info ***("name", var);***

Comme save; mais sauvegarde seulement les variables de renseignements, toutes les images bitmaps changées par freeze_map et toutes les chaînes de caractères changées jusqu'ici.

load_info ***("name", var);***

Comme load; mais charge uniquement les valeurs sauvegardées avec save_info.

level_load ***(string);***

Charge une nouvelle carte de niveau WMB à partir du fichier dont le nom est donné par string.. La palette commute avec une du nouveau niveau. Toutes les entités actuelles sont enlevées et tous les pointeurs d'entité se référant à elles ne peuvent pas être employés plus longtemps. Toutes les actions assignées ou les fonctions appelées par ces entités – ce qui signifie toutes les fonctions dont le synonyme my n'est pas zéro - sont automatiquement terminées à l'instruction wait() suivante. Cependant le scénario C-SCRIPT lui-même continuera à s'exécuter et ne changera pas, même si un scénario différent a été assigné par WED au nouveau niveau. Un cycle d'encadrement après cette instruction le niveau est chargé, deux cycles d'encadrement après l'instruction les entités de la nouvelle carte sont créées et leurs actions sont commencées.

switch_video ***(varres, varbits, varscreen***);

Commute l'écran à la résolution donnée par varres (1 .. 10), à la profondeur de couleur donnée par varbits (8, 16 ou 32) et entre en mode plein écran (varscreen = 1) ou mode fenêtre (varscreen = 2). 0 signifie aucun changement pour l'un ou l'autre des paramètres. La profondeur maximale de couleur et la résolution dépendent de l'édition du moteur. La profondeur de couleur ne peut pas être changée pendant le jeu (voir video_depth). Les modes 16 et 32 bits emploient le matériel d'accélération 3D via Direct3D 7.0. La profondeur de 8 bits de couleur, d'autre part, emploie un logiciel de rendu rapide et travaillera même avec un vieux PC bas de gamme ou un ordinateur portable. Dans des modes 16 bits et 32 bits, basculer en mode fenêtre est seulement possible si la résolution du bureau de windows est la même (haute ou couleur vraie). On peut donner les résolutions d'écran suivantes :

320x200	-	1	
320x240	-	2	mode par défaut pour fly through dans Wed

320x400 - 3 (n'est pas supporté par toutes les cartes vidéo)
400x300 - 4 (n'est pas supporté par toutes les cartes vidéo)
512x384 - 5 (ne fonctionne pas sur certains ordinateurs portables)
640x480 - 6
800x600 - 7 E
1024x768 - 8 C
1280x960 - 9 P
1600x1200 - 10 P

Si le nouveau mode vidéo n'est pas disponible pour n'importe quelles raisons, rien n'arrivera, mais la variable result est mise à 0 ; autrement, après une commutation couronnée de succès, il est mis à 1. Les variables video_mode, video_depth, video_screen, et screen_size sont mises à leurs nouvelles valeurs après l'instruction.

exit;

Termine le jeu et arrête le moteur.

Instructions de remappage clavier

key_pressed ***(number***);

Cette fonction retourne 1 lorsque la touche ou le bouton avec le scan code donné est pressé, sinon 0. Alternativement la variable key_lastpressed peut être testée, elle contient le scan code de la dernière touche ou dernier bouton pressé.

key_for_str ***(string***);

Cette fonction retourne le scan code de la touche pour la chaine donnée sur un clavier américain. Les lettres "A"…."Z" et les chiffres "0"…"9" peuvent être donnés.

str_for_key ***(string, number***);

Cette fonction copie la lettre ou le chiffre avec le scan code donné sur un clavier américain dans la chaine donnée.

key_set ***(number, function***);

Valide la fonction donnée avec la touche donnée par le scan code. De cette façon le remappage clavier peut être fait très facilement. Exemple pour remapper une touche :

```
print("pressez la touche que vous souhaitez assigner à votre fonction");
while (key_any == 0) { wait(1); } // attend qu'il y ait une touche d'appuyée
str_cpy(my_str,"Vous avez pressé ");
str_for_key(temp_str,key_lastpressed);
str_cat(my_str,temp_str);
print(my_str); // indique quelle touche
while (key_any == 1) { wait(1); } // attend que la touche soit relachée
key_set(key_lastpressed,my_function); // assigne la fonction à la touche
```

Instructions Multi joueur

Les instructions de multi joueur sont pour l'envoi d'informations entre les clients et le serveur dans un système de client/serveur. Elles sont symétriques : exécutées sur un client elles envoient quelque chose au serveur; exécutées sur le serveur elles envoient quelque chose à un certain client ou à tous les clients. Il n'y a aucune instruction spéciale pour synchroniser le niveau et mettre à jour les entités - c'est traité automatiquement.

P C send ***(entity.skill)***;
P C send_vec ***(entity.skill)***;

Par cette instruction un client peut mettre à jour une variable d'entité ou un paramètre sur le serveur ou vice versa. Le paramètre de cette instruction doit être une variable d'entité, préfixée par le synonyme de cette entité. Send_vec envoie trois variables consécutives. Les mêmes variables de la même entité sur le PC récepteur sont mises à la valeur reçue. Cette instruction est normalement employée pour envoyer des frappes de clavier et des forces à l'entité de joueur d'un client sur le serveur ou envoyer des événements - comme avoir pris un article - du serveur à un client. Exemple:

```
send(my.skill17);
```

P C send_string ***(string);***

Par cette instruction la chaîne de caractères donnée est envoyée d'un client au serveur ou du serveur à tous les clients. Les fonctions d'événement on_server ou on_client sont déclenchées sur chaque PC qui a reçu la chaîne de caractères, le contenu de cette chaîne de caractères est remplacé par la chaîne de caractères reçue. Cette instruction peut être employée pour l'échange de messages, mais aussi pour l'exécution d'instructions sur tous les clients simultanément. Exemple:

```
inkey message_str; // laisse l'utilisateur taper un message
if (result == 13) { // si [ENTREE] de pressée
send_string(message_str); } // envoie la chaîne à tous les joueurs
// Après cela, message_str sur tous les postes contient la chaîne entrée au clavier
```

P C send_var ***(var)***

Envoie une variable ou un tableau d'un client au serveur ou d'un serveur à tous les clients. Les fonctions d'événement on_server ou on_client sont déclenchées sur chaque PC qui a reçu la variable, le contenu de la variable ou le tableau est remplacé par les nouvelles valeurs. Seules des variables définies par l'utilisateur peuvent être envoyées; les variables système prédéfinies ou des variables de l'ancien style ne peuvent pas être envoyés. Si la variable est un vecteur ou un tableau, tous les éléments sont envoyés.

P session_connect ***(sessionname,hostname)***

Bascule sous d'autres sessions sur d'autres serveurs. Le premièr paramètre chaîne contient le nom de la nouvelle session et le second paramètre donne le nom, le domaine ou l'adresse IP du nouveau serveur. De cette façon le changement de niveau dans un jeu multi-player est possible :

```
session_connect("level2","169.254.73.29");
level_load("level2.wmb");
```

Si une chaîne vide est donnée pour le serveur, une boîte de dialogue apparaîtra pour que vous entriez le nom du domaine ou l'adresse IP manuellement. Si le serveur ou la session n'existe pas il y aura un message d'erreur.

Bitmap instructions

Les instructions suivantes manipulent des images bitmaps.

bmap_for_screen ***(bmap, fac, offset)***;

Capture l'écran en une image bitmap précédemment définie, réduit sa taille. Bmap est soit le nom réel de l'image soit son pointeur. ***Fac*** indique le facteur de réduction (1 .. 16), ***offset*** la position du pixel horizontal de l'image bitmap où l'écran rétréci sera copié. L'image bitmap peut alors être montrée comme une fenêtre dans un panneau, pour indiquer les copies d'écran d'un ou plusieurs jeux sauvegardés. L'image bitmap modifiée sera sauvegardée par save_info, mais est disponible un cycle d'encadrement après l'instruction bmap_to_screen au plus tôt (insérer wait(1) entre freezing et saving) !)

bmap_width ***(bma**p);*
bmap_height ***(bma**p);*

Retourne la largeur et la hauteur de l'image bitmap données en pixels. Exemple:

```
bmap splashmap = <logodark.bmp>; // le logo par défaut du moteur dans le répertoire templates
panel splashscreen { bmap = splashmap; flags = refresh, d3d; }
...
// center the splash screen for non-640x480 resolutions
splashscreen.pos_x = (screen_size.x - bmap_width(splashmap))/2;
splashscreen.pos_y = (screen_size.y - bmap_height(splashmap))/2
```

bmap_purge ***(bma**p);*

Libère la mémoire de texture employée par l'image bitmap donnée. Utile pour libérer la mémoire vidéo quand on ne verra plus certain panneau désormais. La mémoire vidéo est automatiquement allouée de nouveau quand on revoit l'image bitmap une prochaine fois, donc l'instruction ne fera pas de mal. Exemple:

```
bmap_purge(splashmap); // libère l'écran d'accueil après le début du jeu
```

bmap_preload ***(bma**p);*

Préaffecte la mémoire de texture pour la bitmap donnée même si elle n'est pas encore visible à l'écran. Utile pour accéder à la texture de la bitmap depuis une DLL.

Instructions de connexion DLL

Les instructions suivantes sont pour l'accès aux connexions DLL et aux extensions de moteur (DLL = Dynamic Link Library, la Bibliothèque de Liaison Dynamique). La connexion DLL peut être employée par toutes les versions GameStudio. Elle peut être faite avec un outil de développement comme Visual C++™, Borland™ ou Delphi™ et le SDK (le Kit de Développement Source) qui est livré avec l'édition professionnelle ou que vous pouvez acquérir séparément. Les détails de la création de la connexion DLL peuvent être trouvés dans le Manuel du Programmeur GameStudio que vous pouvez trouver sur le site.

Avant qu'une fonction DLL puisse être utilisée, la variable **dll_handle** doit être mis à l'identificateur de la DLL ouverte, et une fonction prototype de la DLL doit être déclarée. Cela peut être fait exactement comme avec une fonction normale, mais avec le mot-clé **dllfunction** au lieu de **function**. Comme une fonction normale, une fonction de DLL peut recevoir jusqu'à 4 paramètres et retourner une valeur. Exemple :

```
dllfunction vector_add(&vec1,&vec2,&vec3); // dll function declaration
```

Après la déclaration, **vector_add** peut être appelé comme une fonction normale si elle existe à l'intérieur de la DLL ouverte :

```
vector_add(temp,my.x,you.x);
```

Pour ouvrir ou fermer une DLL les fonctions suivantes sont valables :

handle = dll_open ***(string)***

Ouvre et active une connexion DLL dont la chaîne de caractères donne le nom et rend un pointeur de DLL. La variable prédéterminée **dll_handle** est aussi mise à ce pointeur. Les pointeurs DLL sont seulement nécessaires si plus d'une connexion DLL est employée en même temps. Le fichier DLL doit vraiment exister dans le répertoire de jeu, il ne peut pas être chiffré dans un fichier de ressource. Si le fichier DLL n'est pas trouvé, 0 est retourné. Exemple:

```
var effects_handle;
```

```
...
effects_handle = dll_open("effects.dll");
```

dll_close ***(handl**e)*

Ferme le fichier DLL avec le pointeur donné. Le pointeur devient invalide après cette instruction. Tous les fichiers DLL ouverts doivent être fermés avant de sortir.

Instructions de mise au point

Les instructions suivantes sont à des fins de test :

beep;

Signal sonore; Joue un son court. Utile pendant le développement d'une fonction, car il permet de découvrir rapidement si une certaine instruction a été atteinte ou une certaine fonction a été exécutée entièrement. On peut donner le son par le nom de beep_sound, par exemple:

```
sound beep_sound = <beep.wav>;
```

breakpoint;

Arrête l'exécution de la fonction et active le programme de mise au point (debugger). Le jeu est gelé. Sur l'écran l'instruction actuelle ou l'expression qui suit le point de contrôle est montrée sous sa forme abrégée. En appuyant [Espace] l'instruction est exécutée et le résultat (s'il y en a un) est affiché au début de la ligne. La ligne monte d'une ligne vers le haut et la ligne suivante de code de C-SCRIPT devient visible. Par appuis successifs sur [Espace] la fonction est exécutée ligne par la ligne et les résultats de chaque expression peuvent être examinés. La sortie de la mise au point est affichée à l'écran par l'objet texte prédéfini _debug_txt, qui peut être placé n'importe ou sur l'écran.

En appuyant [Ctrl-Espace] le programme de mise au point est désactivé et l'exécution normale de la fonction continuera jusqu'à ce qu'un autre ou le même point de contrôle soit atteint de nouveau. En appuyant [Echap] la mise au point est terminée pour la session actuelle et tous les nouveaux points de contrôle sont ignorés. Les points de contrôle sont aussi ignorés dans le module d'éxécution. Notez qu'en plaçant des points de contrôle dans une fonction d'événement ou dans la première partie d'une action d'entité avant le premier wait(), des choses drôles peuvent arriver avec d'autres entités. Elles ne sont pas mises à jour pendant la mise au point et peuvent ainsi apparaître à de fausses places (et rester là jusqu'à ce qu'elles se déplacent de nouveau). Ce n'est pas un problème pour la mise au point.
Des points de contrôle conditionnels peuvent être placés en entrant juste une condition if, comme :

```
if (key_b != 0) { breakpoint; } // point d'arrêt seulement si [B] est pressée
if (my == null) { breakpoint; } // point d'arrêt seulement si MY est un synonyme vide
```

Pour l'examen de variables lorsque l'on met au point certaines lignes dans le scénario, on peut utiliser soit un panneau de mise au point (debug panel) (voir le tutorial, un exemple debug.wdl peut être trouvé dans le dossier de travail work), soit insérer des expressions factices comme :

```
temp = my.pan;
temp = vector.x;
```

Attachement de fonctions ou d'actions

Les fonctions ou les actions peuvent être attachées ou aux entités (par WED) ou à une touche ou à la souris ou à d'autres événements (à travers un script). La syntaxe générale pour assigner une fonction à un événement est :

event = ***function;***

L'erreur classique d'un débutant est d'écrire event = ***function*** **();** au lieu de cela - c'est une syntaxe valable, mais signifie quelque chose de totalement différent et mène inévitablement à un plantage du moteur. Au lieu d'assigner la fonction à l'événement, il appelle la fonction et assigne le numéro rendu.

Les lignes de scénario suivantes peuvent être employées pour assigner des fonctions d'événement aux touches ou à d'autres événements :

on_close

La fonction donnée est exécutée quand la fenêtre windows est fermée en appuyant sutr son icône de fermeture (soit le X, soit alt+F4)

```
function quit() {exit();}
on_close = quit;
```

on_mouse_left, on_mouse_middle, on_mouse_right

La fonction donnée est exécutée quand le bouton gauche, milieu ou droit de la souris est appuyé. Dépendant de la clef, le numéro 280 (gauche), 282 (milieu) et 281 (droit) est passé comme paramètre à la fonction.

on_click

La fonction donnée est exécutée en cliquant gauche avec le pointeur la de souris quelque part dans une vue, sans toucher aucun objet ou panneau.

on_mouse_stop

Cette fonction est déclenchée quand l'indicateur de souris est actif et que la souris a été tenue immobile pendant ½ seconde. La variable mouse_calm peut être employée pour spécifier une distance maximale en pixels pour considérer l'immobilité (3 par défaut). En évaluant la variable mouse_moving vous pouvez aussi découvrir si la souris est tenue stationnaire (0) ou se déplace (1).

on_joy1

La fonction donnée est exécutée quand le premier bouton de levier de commande (joystick) est appuyé. Les noms on_joy2, à on_joy10 peuvent être employés de la même façon pour les autres boutons et pour le second levier de commande (joystick). Dépend du nombre de bouton, les nombres 256 (bouton 1 du joystick 1) à 275 (bouton 10 du joystick 2) sont passés comme paramètres à la fonction

on_anykey***;***

La fonction est déclenchée quand n'importe quelle touche du clavier est appuyée. Le code de balayage de la touche est passé comme paramètre à la fonction. (un code de balayage est un nombre unique de 0 à 127 assigné à chaque touche)

on_f1

La fonction spécifiée est déclenchée lorsque la touche [F1] est pressée. Les touches suivantes peuvent être assignées à des fonctions de la même façon : on_f2...on_f12 (***touches de fonctions***), on_esc (***échap***), on_tab (***tabulation***), on_ctrl, on_alt, on_shiftl (***maj gauche***), on_shiftr (***maj droit***), on_space (***espace***), on_bksp (***retour arrière***), on_cuu (***flèche vers le haut***), on_cud (***flèche vers le bas***), on_cur (***flèche vers la droite***), on_cul (***flèche vers la gauche***), on_pgup (***page préc***), on_pgdn (***page suiv***), on_home (***début***), on_end (***fin***),

on_ins (***inser***), on_del (***suppr***), on_pause, on_car (***Arrêt défil***), on_cal (***virgule***), on_enter (***entrée***), on_0...on_9, on_a...on_z.
Le nom de la fonction reflète la configuration physique de votre clavier et non ce qui est écrit sur les touches. Les noms donnés correspondent ici à un clavier allemand. Sur les claviers américains, le Y et le Z sont par exemple échangés. Les touches spéciales suivantes ont des noms alternatifs pour les claviers américains et allemands :

Touche	US	Allemand(autre nom)	Français	Anglais
on_grave	[~`]	[°^]	[²]	[¬`]
on_minusc	[_-]	[?\] (on_sz)	[°)]	[_-]
on_equals	[+=]	[`'] (on_apo)	[+=]	[+=]
on_brackl	[{[]	[Üü] (on_ue)	[¨^]	[{[]
on_brackr	[}]]	[*+] (on_plus)	[£$]	[}]]
on_bksl	[|\]	–	[µ*]	[|\]
on_semic	[:;]	[Öö] (on_oe)	[Mm]	[:;]
on_apos	["']	[Ää] (on_ae)	[%ù]	["']
on_slash	[?/]	[_-] (on_minus)	[§!]	[?/]

➢Vous trouverez en annexe différentes configurations de clavier selon votre pays.

on_server

La fonction donnée est exécutée quand le serveur reçoit un message d'un client. La variable event_type est mise au type du message. Quand le message contient une chaîne de caractères, un synonyme de la chaîne de caractères reçue est remis comme paramètre à la fonction.

Il y a 4 types d'événements serveur :

Event_join:	un client a rejoint la session. Le nom de client est remis.
Event_leave:	un client a quitté la session. Le nom de client est remis.
Event_string:	un client a envoyé une chaîne de caractères au serveur. Le synonyme de chaîne de caractères est remis.
Event_var:	un client a envoyé une variable ou un tableau au serveur.

Exemple:

```
function server_event(str)
{
	if (event_type == event_join)
	{
		str_cpy(temp_str,str);
		str_cat(temp_str," has joined");
		scroll_message(temp_str);
		return;
	}
	if (event_type == event_leave)
	{
		str_cpy(temp_str,str);
		str_cat(temp_str," has left");
		scroll_message(temp_str);
		return;
	}
	if ((event_type == event_string) && (str == message_syn))
	// message_syn a été mis auparavant à une certaine chaîne de caractères qui est employée pour l'échange de messages
	{
		// le serveur a reçu un message -> l'envoie à tous les clients
		send_string(message_syn);
		return;
	}
}

on_server = server_event;
```

on_client

La fonction donnée est exécutée quand un client reçoit un message du serveur. La variable event_type est mise au type du message. Quand le message contient une chaîne de caractères, un synonyme de la chaîne de caractères reçue est remis comme paramètre à la fonction.

Il y a 2 types d'événements de client :

Event_string: le serveur a envoyé une chaîne de caractères à tous les clients. Le synonyme de chaîne de caractères est remis.
Event_var: le serveur a envoyé une variable ou un tableau.

Exemple:

```
// client recevant un message -> on l'affiche
function client_event(str)
{
	if ((event_type == event_string) && (str == message_syn))
	// message_syn a été mis auparavant à une certaine chaîne de caractères qui est employée pour l'échange de messages
	{
		scroll_message(message_syn);
		str_cpy(message_syn,empty_str);
		return;
	}
}.
on_client = client_event;
```

Toutes les touches, la souris ou d'autres fonctions d'événement peuvent être changées n'importe quand par d'autres fonctions par exemple On_f1 = fonction;), pour que la même touche ou bouton puisse commencer des fonctions différentes plusieurs fois.

Variables, Chaînes de caractères, Pointeurs

Il y a quelques objets de base qui peuvent être définis dans un scénario pour leur utilisation dans des fonctions.

Variables et Tableaux

Les variables et les tableaux stockent des nombres à virgule fixe à 6 chiffres avec 3 chiffres après la décimale. Il y a les variables prédéfinies qui peuvent être employées par des fonctions pour changer les paramètres du jeu ou l'affichage. Vous trouverez une description de ces variables prédéfinies dans un chapitre suivant. N'importe quel nombre de variables peut être défini pour stocker des nombres pour des buts arbitraires - l'état de santé, la prouesse de combat, des habiletés magiques, la tension, le niveau d'alcool et caetera ... tout ce qui est laissé à l'imagination de l'auteur.

Une variable est définie comme suit:

var ***nam***e;
var ***name = valu***e;

Crée une variable avec le nom donné. La deuxième ligne alternative met la variable à la valeur de défaut donnée.

var ***nam***e***[***n];
var ***nam***e***[***n] **= *value_1, value_2, ... value***_n;

Crée une variable qui contient n nombres. À la deuxième ligne alternative on donne des valeurs par défaut à tous les nombres. Une telle variable de multi-nombre est appelée un tableau. Le nombre n est appelé la longueur du tableau. Un tableau peut avoir n'importe quelle longueur, mais on peut donner un maximum de 500 valeurs par défaut dans un fichier C-SCRIPT. Un tableau de longueur 3 est appelé un vecteur. Exemple:

```
var my_array[5] = 0, 10, 20, 30, 40;
```

Cela crée un tableau de 5 nombres, que l'on peut lire dans des expressions comme my_array [0] - my_array [4]. Dans l'expression le nombre entre crochets [ ] - qui est appelé index - dit lequel des 5 nombres du tableau est signifié. Notez qu'il n'y a aucun numéro my_array [5], car l'index commence par 0.

Dans une expression, un var donné sans index est considéré comme le premier nombre du tableau (my_array == my_array [0]). Pour l'utilisation de tableaux en tant que vecteurs, quelques abréviations significatives pour les trois premiers nombres peuvent être employées : .x, .y, .z, .pan, .tilt, .roll, .blue, .green, .red. Donc les références suivantes sont équivalentes à des tableaux de longueur 3:

my_vec.x == my_vec.pan == my_vec.blue == my_vec[0] == my_vec
my_vec.y == my_vec.tilt == my_vec.green == my_vec[1]
my_vec.z == my_vec.roll == my_vec.red == my_vec[2]

L'utilisation de ces abréviations produit une meilleur lisibilité et des scénarios un peu plus rapides, parce qu'alors l'index n'est pas examiné pour savoir s'il excède la longueur du tableau. La longueur d'un tableau peut être connue par son paramètre de longueur (length). L'avantage d'employer un tableau, comparé à la définition de variables simples, consiste en ce que l'on peut donner n'importe quelle expression numérique aussi comme index. Exemple:

```
temp = 0;
while (temp < my.array.length)
{
	my_array[temp] = temp; // sets the array to 0,1,2,3,... etc.
	temp += 1;
}.
```

Faites bien attention que l'index n'excède jamais sa valeur maximale, 4 dans cet exemple. Autrement vous obtiendrez un message d'erreur.

Les tableaux multidimensionnels ne sont pas un problème dans C-SCRIPT. Supposons que nous ayons besoin d'un tableau à 2 dimensions, comme une grille de valeur de hauteur de 10*20. Nous la définissons de cette façon :

```
var heightmap[200]; // 200 = 10*20
```

Et nous y accédons de cette façon :

```
temp = heightmap[j*20 + i]; // j = 0..9, i = 0..19
```

Vous pouvez employer des tableaux multidimensionnels pour ajouter des variables locales à certaines fonctions - des variables qui seront seulement employées dans cette fonction et nulle part autrement. Des variables locales peuvent être employées pour la création illimitée de variables (skill) d'entité. Supposons que les 48 variables (skill) d'entité ne sont pas assez pour un certain type d'entité - vous avez besoin de 100 variables. Il faut juste définir un tableau [52] pour obtenir un total de 100 variables) ne travaillera pas quand vous avez plus d'une entité de ce type.

Si les fonctions de C-SCRIPT soutenaient des variables locales, vous pourriez juste définir un tableau local [52] dans l'action d'entité. Bien, mais vous pouvez déjà faire cela, par la définition d'une 'association' de variables locales dans un tableau à l'extérieur de la fonction :

```
var ent100_skill[5200]; // assez pour 100 entités, 52 variables chacune
var ent100_offs = 0; // index offset à l'intérieur du tableau
define _local my.skill48; // utilise my.skill48 pour stocker index offset
action ent100 // directement attachée àl'entité
{
    _local = ent100_offs; // range l'offset courant du tableau
    ent100_offs += 52; // et réserve 52 elements du tableau pour cette action
    if (ent100_offs >= 5200) // vérifie si nous avons assez de variables pour cette entité
    { beep; beep; return; } // indique une erreur (trop d'entités)
    ...

    while (1) // entity main loop
    {
    ...
           temp = ent100_skill[_local+0]; // accède à la première variable additonnelle
           ...
           temp = ent100_skill[_local+51]; // accède à la dernière variable additonnelle
           ...
           wait(1);
    }
}
```

Ne pas oublier de remettre ent100_offs à 0 lorsque vous chargez un nouveau niveau !

Vars peut être redéfini à n'importe quelle place dans le scénario. Seule la dernière définition est employée :

```
var v1 = 7;
...
var v1 = 13;
```

Crée la variable v1 avec une valeur initiale de 13. La longueur de tableaux redéfinis ne doit pas changer.

Une sorte spéciale de variables et des tableaux est créée par :

var_info ***nam***e;

Crée une variable ou un tableau qui est sauvegardé par une instruction save_info. Les variables de renseignements sont utilisées pour les paramètres de l'utilisateur comme le volume ou la résolution d'écran que l'on suppose déterminée au début de jeu et qui ne doivent pas changer en chargeant un jeu précédemment

sauvegardé. Vous pouvez changer l'état de renseignements d'une variable dynamiquement en mettant son drapeau de renseignements (info) à 1 (on) ou à 0 (off) :

```
my_array.info = on;
save_info "state",0;
my_array.info = off;
```

Sauvegarde la variable my_array dans le fichier state0.sav. Le drapeau de renseignements lui-même n'est pas sauvegardé ou chargé, il conserve toujours sa dernière valeur.

var_nsave ***nam***e;

Comme var_info mais la variable name n'est jamais sauvegardée. Utilisez cette instruction pour les identificateurs de DLL et de sons qui ne peuvent pas être réécrite par une instruction load ou load_info.

Notez s'il vous plaît : les anciennes versions connaissaient les variables et les vecteurs sous un nom différent : SKILL. Si vous rencontrez dans un vieux fichier C-SCRIPT une définition comme cela :

```
SKILL my_skill { X 10; Y 20; Z 30; }
```

Un tableau de longueur 3 avec les valeurs par défaut de 10, 20, 30 est défini.

Strings (Chaînes de caractères)

Les chaînes de caractères (string) sont juste une séquence de caractères alphanumériques - des lettres, des nombres ou des symboles - qui peut être employé pour des messages, des menus à l'écran ou équivalent. Ils sont définis de cette façon :

string ***name = "text***";
Définit un ordre de caractère avec le nom donné et le texte contenu entre guillemets. Les retours à la ligne dans le texte sont représentés en notation C comme \n, les antislashs \ comme \\, les guillemets comme \". Si le texte s'étend sur plusieurs lignes dans le fichier de scénario, des retours à la ligne sont insérés automatiquement. Comme les variables, les chaînes de caractères peuvent être redéfinies.

string ***name[***n];
Définit une chaîne de caractères vide de n espaces.

La chaîne de caractères seule, une fois définie, n'est pas encore visible à l'écran. Pour la rendre visible, vous aurez besoin d'un objet texte, qui est traité dans un autre chapitre.

Pointeurs

Les pointeurs stockent des objets comme les variables stockent des nombres. Les pointeurs permettent, par exemple, à une même fonction de faire la même chose avec des objets différents, dépendant de l'objet qui est actuellement assigné au pointeur donné. Les pointeurs peuvent être utilisés comme synonymes ou comme nom alternatif d'un objet. Comme avec les variables, il y a quelques pointeurs prédéfinis ; le plus fréquemment employé est le pointeur my. Pour définir un pointeur à un objet il faut juste ajouter une '*' au mot clé de l'objet.

entity* ***name***
entity* name = entity;
Définit un pointeur d'entité avec le nom ***name***. Dans le premier cas l'indicateur est toujours ' vide ', dans le deuxième cas cela fait référence à une entité précédemment définie. Les pointeurs peuvent être définis pour des entités, des actions, des chaines et des bitmaps par entity*, action*, string* et bmap*.

Dans des fonctions vous pouvez employer des pointeurs comme des noms d'objet normaux. Les pointeurs d'action eux-mêmes peuvent être commencé par d'autres actions ou fonctions. Les pointeurs peuvent s'être assignés :

```
entity* ent1;
entity* ent2;
…
ent1 = my;
ent2 = ent1;
ent2.visible = off;
```

Quelques pointeurs sont prédéfinis . Vous trouverez une liste d'entre eux dans un prochain chapitre. Les deux pointeurs prédéfinis les plus importants sont my pour l'entité qui a commencé la fonction actuelle; et you pour l'entité qui a récemment causé la fonction actuelle de n'importe quelle façon, par exemple en tirant ou entrant en collision avec l'entité à laquelle la fonction a été attachée ou en créant cette entité. You peut également être mis par quelques instructions.

Deux pointeurs peuvent être comparés par l'instruction if pour vérifier si le même objet a été assigné à tous les deux. Cependant les pointeurs ne peuvent pas être comparés avec l'objet lui-même.

Notez s'il vous plait : les versions précédentes connaissait les pointeurs sous un nom différent : SYNONYM. Si vous rencontrez dans un vieux fichier C-SCRIPT une définition comme cela :

```
SYNONYM ent1_syn { TYPE ENTITY; }
```

Un pointeur d'entité est défini.

Objets Fichier

Les fichiers peuvent être nommés par des noms, pour définir des objets bitmap, fonte, animation, son ou musique et les rendre contrôlables par des fonctions. Les noms de fichier ne doivent pas être plus longs que 20 caractères, et on doit les donner sans chemin dans des parenthèses dirigées <...>. Les parenthèses dirigées sont un indicateur pour la fonction WED de publication ou de ressource pour inclure ce fichier dans le jeu publié. Ci-après les objets de fichier pouvant être définis :

bmap ***name = <filename>;***
bmap ***name = <filename>, x, y, dx, dy;***

Assigne une image bitmap - le contenu d'un fichier d'images - au nom. Le fichier peut être au format pcx, bmp ou tga, et en couleur vraie (24 bits RGB ou 32 bits ARGB) ou 256 couleurs (8 bits palettisés). La deuxième ligne alternative assigne une section rectangulaire de l'image au nom, permettant d'avoir plusieurs images bitmaps rassemblées dans un fichier simple. Les coordonnées ***x, y*** marquent le coin supérieur gauche de la section, ***dx, dy*** donne sa largeur et hauteur en pixels. L'omission des coordonnées fait que l'image bitmap entière est assignée au nom.

Si vous rencontrez un message d'erreur bitmap au démarrage, c'est que vous avez choisi un mauvais format qui n'est pas supporté par le moteur. Toutes les images bitmaps palettisées d'un niveau doivent partager la même palette de couleur, que l'on doit donner via les propriétés de carte à WED. Cependant, chaque niveau peut avoir sa propre palette. La première couleur de la palette (colour #0) doit toujours être noir et la dernière couleur (pour colour #255) doit toujours être blanc. Vous pouvez employer la couleur vraie même si le mode vidéo du moteur est commuté à la profondeur de 8 bits de couleur; dans ce cas, cependant, il y aura une perte visible de qualité.

Les images bitmaps peuvent être employées pour du recouvrement. Dans ce cas les parties avec l'index de couleur #0 d'une image bitmap palettisée ou avec les composants R, G, B au dessous de la valeur 8 pour les images bitmaps en couleur vraie, est transparente. Les images bitmaps Targa 32 bits (tga) contiennent un canal alpha qui donne une valeur de transparence pour chaque pixel simple (seulement E et supérieur). Le canal alpha est ignoré dans le mode 8 bits ou quand il n'est pas soutenu par l'édition; auquel cas le bitmap tga 32 bits est rendu comme une image bitmap 24 bits pcx ou bmp. Pour créer un canal alpha reportez vous à votre logiciel de peinture préféré. Il faut simplement ne pas oublier de sauvegarder l'image au format de Targa.

font ***name = <filename>, width, height;***

Définit une fonte de caractère de largeur fixée à partir d'un fichier d'images. Pour le format de fichier, les mêmes restrictions que bmap s'appliquent. ***Width, height*** donnent la taille d'un caractère simple en pixels. Tous les caractères doivent avoir la même taille. L'image bitmap peut aussi bien contenir 11 caractères - les chiffres 0 .. 9 et l'espace - pour des affichages numériques, que 128 ou 256 caractères pour le texte alphanumérique.

L'ordre de caractères alphanumériques dans l'image bitmap doit correspondre au jeu de caractère employé par le scénario C-SCRIPT, soit DOS/ASCII soit Windows/ANSI. Dans l'image bitmap, on peut ordonner les caractères en quatre ou huit rangées. Le nombre de caractères et leur arrangement décident automatiquement de la taille de caractère et de la taille de l'image bitmap. Mais la taille de l'image bitmap doit être exactement 11 fois, 128 fois ou 256 fois la taille de caractère donnée.

Font bitmap

sound ***name = <filename>;***

Assigne un fichier de son dans le format .wav au nom name. Un fichier wav peut être rebaptisé à .wex; dans ce cas le fichier est d'abord recherché à l'extérieur de la ressource. Le format peut être 11 ou 22 kHz, 8 ou 16 bits.

music ***name = <filename>;***

Assigne un fichier de chanson dans le format .mid au nom. Les instruments doivent correspondre à la norme midi standard.

Entités

Comme mentionné auparavant, les entités sont les objets dynamiques, les acteurs, les monstres, les véhicules et toutes ces choses qui peuvent se déplacer dans notre monde de jeu. En plus des paramètres qui peuvent être mis dans le panneau de propriété d'entité de WED, il y a beaucoup plus de propriétés qui peuvent être mises, évaluées ou changées par une action de C-SCRIPT. Ces propriétés influencent le regard des entités, leur réaction à certains événements et leur comportement de déplacement ou de collision.

Les entités peuvent être créées de trois façons :

- En les plaçant dans le niveau par WED,
- En les créant dans le niveau à l'exécution par l'instruction ent_create,
- En les définissant par une définition d'entité dans le scénario (modèles et sprites seulement).

Les entités placées dans un niveau sont nommées des entités locales; les entités définies dans un script sont des entités globales. Un type spécial, d'entités locales de très courte durée est les particules, créées par Instruction effect. Les particules ont seulement un sous-ensemble de paramètres d'entité. Paramètres d'entité qui peuvent être changés par n'importe quelle fonction; ils doivent être préfixés par le nom d'entité défini par le scénario ou un indicateur comme my ou you. Parce que le niveau s'exécute sur le serveur, les entités locales qui sont placées dans le niveau existent seulement sur le serveur et sur le client qui les a créés (si elles existent). En quittant le niveau toutes les entités locales sont supprimées automatiquement et leurs actions sont terminées.

Des entités globales existent cependant à l'extérieur d'un niveau, sur le serveur aussi bien que sur tous les clients. Elles sont visibles Même s'il n'y a aucun niveau chargé du tout. Elles peuvent être employées pour montrer les éléments 3D de l'interface utilisateur, comme une boussole tournante, une roue de gouvernail, ou une arme portée par le joueur. Les entités globales sont définies de la façon suivante :

entity ***name*** { – }

Dans la définition, on peut donner une valeur initiale à n'importe quel paramètre d'entité. Egalement une liste de drapeaux peut être définie qui est mis au début du jeu. Exemple:

```
entity shotgun_onscreen
{
	type = <shotgun.mdl>;
	layer = 2; // montre les entités au dessus de la couche 1
	flags = visible;// visible à l'écran au départ
	view = camera; // même paramètre caméra que la vue par défaut
	albedo = 50; // ombrage gouraud de position du soleil
	x = 100; // place 100 quants en avant de la vue
	y = -50; // 50 à droite
	z = 0; // et centre verticalement
}
```

Paramètres globaux

Les paramètres suivants sont employés seulement dans des définitions d'entité :

type = ***<filenam***e>;

Ici on donne le nom par défaut du fichier de l'entité mdl dans des parenthèses angulaires. Seulement le modèle et les entités sprite peuvent être définis de cette façon. Le fichier de l'entité peut être changé pendant l'exécution par une instruction morph.

layer = ***number***;

Détermine l'ordre de l'entité, si elle est visible à l'écran et se chevauche avec d'autres objets comme des panneaux ou des textes. Les éléments avec la valeur de couche plus haute seront placés sur des éléments avec la valeur de couche inférieure. Le paramètre de couche (défaut 0) ne peut pas être changé pendant l'exécution.

view = ***viewnam***e;

La vue (view) détermine les paramètres de caméra et le rectangle de coupure pour le rendu de l'entité à l'écran. Si on ne donne aucune vue, la vue (view) par défaut (la caméra) est employée. La vue ne doit pas nécessairement être visible pour ce but. On donne la position d'entité en quants par rapport au centre de la vue, avec l'axe des abscisses pointant en avant vers l'écran

Paramètres de position

Les paramètres suivants sont employés pour placer l'entité :

x, y, z

La position du centre des entités dans le système de coordonnées de la carte (en quants). Le paramètre x peut aussi être employé comme un vecteur de position. Une entité peut être téléportée ou déplacée sans détection de collision en changeant simplement ces valeurs. Pour des entités définies, ces paramètres donnent la position relativement à la position de la caméra.

pan, tilt, roll

Les angles Euler des entités qui décrivent les rotations de ses axes Z, Y et X (en degrés, 0 .. 360). Une entité peut être tournée en changeant ces valeurs.

Visual Parameters (paramètres visuels)

Les paramètres suivants influencent l'aspect des entités dans le niveau :

E lod1, lod2, lod3

Ces paramètres contrôlent les niveaux géométriques discrets de détail – LOD = Level Of Détail - d'une entité. Les LOD sont employés pour augmenter le taux d'affichage. Le rendu d'un modèle détaillé avec plusieurs milliers de polygones semble tout à fait bon quand le modèle est près de la caméra. Cependant quand le même modèle est loin du point d'oeil, les détails sont moins remarquables. Un modèle plus simple avec peu de polygones semblera aussi bon dans cette situation, mais rendra plus rapidement. La différence dans la vitesse de rendu est tout à fait remarquable dans des niveaux extérieurs énormes où des tas d'entités sont visibles, comme des arbres dans un bois ou une armée ou une ville faite d'entités de carte de maison.

Pour obtenir le meilleur de chacun de ces mondes, chaque entité peut être attachée à 4 fichiers d'entité, qui changent automatiquement en fonction de la distance du centre des entités à la caméra. Les trois nouveaux fichiers peuvent être assignés par les mots-clés lod1, lod2, et lod3 dans la définition d'une entité modèle ou d'une entité sprite, comme :

```
entity lodgun {
	type = <gun.mdl>; // employé lorsqu'on est prêt du modèle
	lod1 = <gunsimpl.mdl>; // utilisé si plus de 12.5% du clip_range, et pour les ombres près
	lod2 = <gunsimp2.mdl>; // utilisé si plus de 25%, et ombre supérieur à 12.5% clip_range
	lod3 = null; // invisible au delà de 50%, et pas d'ombre au delà de 25% clip_range
}
```

Cet exemple définit le comportement LOD pour toutes les entités <gun.mdl> dans le jeu. L'entité commute à LOD niveau 1 à une distance de 1/8 de clip_range, LOD niveau 2 à une distance de ¼ de clip_range et LOD niveau 3 à une distance de ½ de clip_range. Pour prévenir les échanges répétés entre deux niveaux LOD sur des distances critiques, le changement de niveaux de LOD se fait avec une valeur d'hystérèse de 15 %. Si l'entité est plus loin que

clip_range, elle ne sera pas montrée du tout. LOD peut aussi être employé pour la création de 4 niveaux MIP Mapping pour des sprites.

Les niveaux de LOD sont valables pour toutes les entités qui emploient le même modèle. Pour l'animation appropriée, ils doivent être de la même sorte; un modèle animé ne doit pas changer dans une carte ou un sprite. Le nombre d'encadrements et les noms d'encadrement doit être les mêmes sur tous les niveaux de LOD. Si on donne nul au lieu d'un nom de fichier, l'entité n'est pas visible pour les niveaux de LOD supérieur. De cette façon, les sous-entités d'une entité composée de plusieurs parties peuvent être supprimées au-dessus d'une certaine distance. Si un modèle a plusieurs niveaux de LOD, son ombre (shadow) est calculée pour chacun des niveaux qui affiche ce modèle. De cette façon des ombres dynamiques peuvent maintenant être rendues remarquablement plus rapidement. Si le modèle est plus loin que 50 % du clip_range, aucune ombre n'est rendue du tout.

Il n'est pas nécessaire de définir des entités LOD par une définition d'entité. Si un nom de fichier d'entité finit avec "0", on suppose que des fichiers d'entités semblables finissant avec "1", "2", "3" sont les trois niveaux LOD suivant (Exemple: pour "house_0.wmb" le moteur attend "house_1.wmb", "house_2.wmb", "house_3.wmb" comme entités LOD). Si une de ces entités LOD n'est pas trouvée, l'entité ne sera pas rendue du tout sur cette distance LOD.

scale_x, scale_y, scale_z

les facteurs d'échelle, qui déterminent la taille des entités en quants par pixel (seulement valable pour les sprites et les entités modèles). Ils peuvent être changés pendant le jeu pour grossir ou rétrécir l'entité en temps réel. Leur valeur par défaut est 1.0, ce qui signifie qu'un pixel de la texture des entités a la taille de 1 quant. Notez : le changement d'une taille des entités changera aussi ces limites, mais ne changera pas sa coque de collision (voir le chapitre suivant). S'il est réduit très petit, il peut être nécessaire de lever son centre pour empêcher l'entité de s'enfoncer dans le plancher.

frame, next_frame

Les entités modèle et sprites peuvent consister en n'importe quel nombre d'encadrements pour l'animation d'affichage. Dans le cas des entités sprites, les encadrements d'animation doivent avoir la même taille, côte à côte dans le fichier graphique. On doit donner le nombre d'encadrements du sprite à la fin du nom de fichier après un signe + (la longueur totale du nom de fichier ne doit pas excéder le format 8.3) :

Sprite animé EXPLO+7.PCX

Il y a deux façons d'animer une entité sprite ou modèle. Si l'entité consiste en plusieurs encadrements, mais n'a aucune action attachée, il bouclera automatiquement tous ses encadrements avec un taux de 8 encadrements par seconde pour les modèles et 12 encadrements par seconde pour les sprites. Si elle est par contre attachée à une action, elle ne s'animera pas toute seule. Au lieu de cela, c'est la valeur frame (1 .. nombre d'encadrements) qui donne le numéro de l'encadrement actuel. Ainsi en ajoutant de manière permanente le paramètre time au paramètre frame le modèle peut être animé, par Exemple:

```
my.frame += 1.5*time; // gives 16/1.5 = 12 frames per second
if (my.frame > 10) {
my.frame -= 10; } // don't exceed the frames
```

Alternativement, pour un modèle marchant, la distance couverte pourrait être ajoutée à frame, au lieu d'un facteur de temps.

Si le paramètre d'encadrement a une partie fractionnaire, la forme du modèle sera interpolée entre le nombre d'encadrement donné par la partie entier et un encadrement cible donné par le paramètre next_frame. Si next_frame est mis à 0, frame + 1 sera pris comme encadrement cible. De cette façon, un modèle peut sans à-coup se transformer en avant ou en arrière entre des encadrements arbitraires.

Exemple pour une fonction de boucle qui emploie next_frame pour animer sans à-coup un modèle entre des encadrements 20 et 30 :

```
my.frame = 20; // mis comme frame de départ
while (1) {
	my.frame += time; // 16 frames par seconde
	if (my.frame > 30) { my.next_frame = 20; } // interpolate avec frame de départ
	else { my.next_frame = 0; } // interpolate avec FRAME+1
	if (my.frame >= 31) { my.frame -= 10; } // si frame de fin dépassée saut arrière
	wait(1);
}
```

Normalement une entité modèle est animée par des instructions ent_frame () et ent_cycle ().

u, v

les décalages de pixel de textures d'entité de carte, en direction horizontale et verticale. En changeant ces décalages en temps réel, les effets comme de l'eau coulante peuvent être réalisés. Exemple:

```
action water_current // assigné à un bloc eau
{
	while (1) {
		my.u += 5*time;
		my.v += 5*time;
		wait (1);
	}
}
```

skin

Si une entité modèle a des peaux multiples, elles peuvent être changées par son paramètre de peau (1 ... à nombre de peaux). Si l'entité n'a aucune action attachée, elle bouclera ses peaux à 12 cycles par seconde.

ambient

paramètre (-100 .. +100, par défaut 0), une texture d'entités peut apparaître plus sombre ou plus brillante. En addition, des entités modèles sont ombrées gouraud, en fonction leur position à l'azimut du soleil et de son élévation. Les entités de carte sont en ombres plates, aussi selon la position de soleil, si elles ont la texture albedo plate.

albedo

La reflectivité d'entités carte ou modèle ou le terrain à la lumière du soleil (0 .. 100, par défaut 50). Plus grandes sont les surfaces ombrées et les entités d'albedo, plus sombre sont les ombres et plus brillantes les parties éclairées du modèle. Des modèles transparents placés dans le niveau ont un défaut albedo de 0.

lightrange

C red, green, blue (rouge, vert bleu)

Ces valeurs sont employées pour colorer des entités dynamiquement et pour l'émission de lumière dynamique colorée par des entités de niveau. Dans le dernier cas le paramètre lightrange donnera la gamme de la lumière (0 .. 2000 quants, par défaut 0); pour des entités sur écran lightrange ne doit pas être employé. Les paramètres rouges, verts, bleus (0 .. 255, défaut 0) donnent la brillance des composantes rouge, vert et bleu de la couleur de la lumière. Pour éteindre la lumière émise par une entité de niveau, lightrange doit être mis à 0. La couleur de la lumière n'est pas sur les éditions en dessous de la commerciale.

Le nombre des entités qui peuvent émettre de la lumière dynamique est limité à 32 simultanément. Si il y en a plus, la lumière « la plus ancienne » sera éteinte. Des feux dynamiques peuvent être employés pour des explosions, des fusées, des éclairs sphériques ou pour des monstres portant des torches et illumineront toutes les surfaces ombrées et les entités dans leur gamme. Dans le mode 8 bits la lumière est toujours blanche et sa brillance est calculée sur la plus grande des valeurs de ses composantes couleur Rouge, Vert et Bleu.

Les feux dynamiques influencent le taux d'affichage, donc ils doivent seulement être employés pour des effets provisoires. Plus il y a de feux, plus grande et leur rayon d'action, plus lent et le rendu. Cet effet est beaucoup moins

significatif dans les modes 16 bits ou 32 bits, mais dans le mode 8 bits des feux dynamiques peuvent réellement diviser par 2 le taux d'affichage. La qualité de rendu de la lumière dynamique dans les modes 16 bits ou 32 bits peut être augmentée en mettant la variable d3d_lightres à 1. Cela double la résolution du spot de lumière à la surface, mais augmente aussi le temps de rendu.

E alpha

Contrôle la transparence d'entités translucides dans les modes 16 ou 32 bits. Le paramètre (par défaut 50) donne un pourcentage. À 0 l'entité est totalement transparente, à 100 elle est totalement non transparente. De cette façon les entités peuvent sans à-coup apparaître en fondu avant ou arrière. Exemple pour la disparition d'une entité en fondu:

```
my.transparent = on;
my.alpha = 0;
while (my.alpha < 100) {
	my.alpha += 5*time;
	wait(1);
}
my.transparent = off;
```

visible

Seulement pour les entités définies par scénario : si ce drapeau est mis à 1 (on), l'entité sera visible à l'écran, même sans un niveau. Ne jamais mettre ce drapeau à un sur une entité locale

Invisible

si le drapeau invisible est mis à 1 (on), l'entité sera invisible dans le niveau, mais sera toujours un obstacle pour la détection de collision.

hidden (caché)

si le drapeau caché est mis à 1 (on), cette entité est seulement visible sur le poste client qui l'a créé. Pour tous les autres membres d'un système multi joueur elle est invisible. Ce drapeau peut être employé pour des modèles d'arme dans un système de multi joueur.

transparent

Si ce drapeau d'une entité sprite, modèle ou carte est mis à 1 (on), l'entité sera translucide, donc vous pouvez voir à travers. L'eau transparente peut être créée en plaçant une entité de carte transparente d'eau dans un bassin (et le niveau d'eau peut facilement monter ou descendre de cette façon).

overlay (revêtement)

Si un drapeau de revêtement des entités modèles est mis à 1 (on), les parties noires de la peau des entités ne seront pas dessinées.

E flare (éclat)

Si le drapeau éclat (flare) d'une entité sprite ou modèle est mis à 1 (on), l'entité gèrera la transparence alpha. Les parties plus sombres sont plus transparentes que les parties plus brillantes. De cette façon les explosions, les jeux de lumière ou les éclats de lentilles peuvent être produits. Si transparent est mis en même temps, la transparence est changée complètement, c'est-à-dire les parties plus brillantes sont plus transparentes que les parties plus sombres, comme pour la fumée noire. Seulement valable pour les entités sprite et seulement dans les modes 16 bits ou 32 bits. Le drapeau d'éclat (flare) doit être mis à 1 au commencement de la première action de l'entité. Toutes les entités qui partagent le même fichier doivent avoir la même transparence alpha. Si la texture d'entité contient un canal alpha, le drapeau d'éclat (flare) est mis à 1 par défaut.

E bright

En combinaison avec l'éclat ou la transparence, l'entité se fond dans l'arrière plan au lieu de se mélanger toutes les deux. De cette façon l'entité apparaît illuminée, comme du feu ou des étincelles. Dans la combinaison avec unlit l'entité reçoit une lumière d'environnement moyenne comme une lumière de soleil normale dans le niveau.

E metal

En mettant ce drapeau, la peau d'un modèle obtient une réflectance spéculaire pour la lumière de soleil dans le mode D3D, comme si elle était faite de métal brillant ou en verre. L'intensité de l'effet peut être mise par le paramètre albedo du modèle (0 .. 100, défaut 50) à une valeur entre 40 et 60. Plus grande est la valeur albedo, plus petit est le spot lumineux.

nofog

Si ce drapeau est mis, le brouillard devient beaucoup plus transparent pour cette entité.

light

Si ce drapeau est mis, l'entité est illuminée par ces propres couleur de lumière (rouge, vert, bleu) dans le mode D3D même si **lightrange** est 0. De cette façon les entités peuvent être dynamiquement colorées. Les restrictions des lumières dynamiques (maximum 32) ne s'appliquent pas ici. Exemple:

```
function flicker_red(value) {
my.lightrange = 0; // ne pas illuminer l'environnement
my.light = on; // s'illuminer soi-même
while (my.light == on) {
	while (my.red < value)
	{
		my.red += 10 * time;
		my.red = min(my.red,255); // ne pas dépasser 255
		wait(1);
		}
		while (my.red > 0)
		{
			my.red -= 10 * time;
			my.red = max(my.red,0); ne pas descendre en dessous de 0
			wait(1);
		}
	}
}
```

nofilter

Si le drapeau **nofilter** d'une entité sprite ou modèle est mise à 1 (on), l'entité est dessinée sans filtrage bilineaire et anticrénelage dans le mode D3D. Donc la texture n'obtiendra pas cette apparence floue. C'est utile pour éviter les joints noirs du sprite produit par quelques cartes 3D (Voodoo) dû à un anticrénelage à la couleur de transparence noire.

unlit

En mettant un drapeau unlit à une entité, l'entité n'est pas éclairée par son environnement. Seule sa valeur **ambient** et son drapeau **bright** influence la brillance. En mettant **ambient** à -100 l'entité devient totalement noire. De grandes entités placées dans le niveau, avec un diamètre de 250 quants ou plus, ont leur drapeau **unlit** et **bright** à 1 par défaut.

C shadow

En mettant le drapeau ombre (**shadow**) d'un modèle, son ombre est projetée avec la transparence de 50 % sur le sol. La direction de projection est donnée par l'azimut de soleil et son élévation (**sun_angle**). L'ombre est visible même si le modèle est invisible. Les projections d'ombre peuvent augmenter le temps de rendu du modèle d'environ 50 %. Le drapeau ombre (**shadow**) peut être mis par C-SCRIPT aussi bien que WED.

oriented, facing (orienté, faire face)

Si le drapeau d'orientation (**oriented**) d'une entité sprite est mis à 1 (on), l'entité seront orientée dans l'espace du monde selon ses angles **pan**, **tilt** et **roll** , plutôt que de faire face au joueur. Ce drapeau peut être employé pour créer des objets plats comme des barrières ou des fenêtres transparentes ou placer une ombre pour le sprite sur le sol. Si le drapeau 'faire face' (**facing**) est mis à 1 (on), l'entité fera toujours face à la caméra. Ce drapeau peut être employé pour des objets sphériques, comme des éclairs sphériques ou des explosions.

Si aucun drapeau n'est mis, le sprite ressemblera à un panneau d'affichage, étant debout tout droit et horizontalement se tournant toujours vers le joueur.

near (près)

Si le drapeau près (near) d'une entité modèle ou sprite est mis à 1 (on), l'entité sera beaucoup moins coupée à l'extérieur par les murs ou les autres entités . Ce drapeau peut être utilisé pour les armes portées devant la caméra, pour les éclatements etc..

Les modèles avec drapeau proche (near) sont rendus avec une valeur de z-amortisseur inférieure, pour les rendre devant d'autres objets. Cela peut mener à la coupure précédente de tels objets dans le mode D3D quand les polygones pénètrent dans le plan de caméra (comme avec des modèles d'arme). Pour empêcher cela, concevez et placez un modèle d'arme de telle façon qu'il ne pénètre pas dans le plan de caméra. Pour peaufiner le réglage, le vecteur prédéfini d3d_near peut être employé. D3d_near.x donne un facteur de décalage pour le z-buffer (0.1 .. 0.25, default0.25) et d3d_near.y donne un facteur de distance (1 .. 10, default2.5). En changeant ces valeurs le comportement de coupure près des entités modèles peut être influencé.

Détection de collision

Deux méthodes différentes de détection de collision sont employées dépendant de quelle type d'entité entre en collision avec telle autre. Pour cela, chaque entité a une boîte de limitation et une coque en forme de boîte. La coque est employée pour les collisions d'une entité se déplaçant contre des entités de carte ou de niveau. Cette sorte de collision détecte chaque polygone de l'entité cible. Pour tous les autres types de collision, les boîtes de limitation des deux entités sont évaluées l'une contre l'autre. La boîte de limitation aussi bien que la coque ne tournera pas avec l'entité, elles sont toujours alignées selon l'axe xyz.

Tandis que la boîte de limitation peut avoir une taille arbitraire, il y a seulement trois coques possibles : grasse, étroite et point de taille. Les entités avec une boîte de limitation en point de taille peuvent passer tous les blocs invisibles (mais pas les blocs qui bien que visibles, ont des surfaces invisibles par le drapeau None). La coque étroite est un cube de taille de bord de 32 quants autour du centre de l'entité. La grosse coque est un cadre de taille horizontale de 48 quants verticaux et de 64 quants horizontaux. Ces tailles correspondent à la plupart des exigences, mais pour quelques cas spéciaux elles peuvent être changées. À la différence de la boîte de limitation, on ne leur donne pas individuellement par entité, mais seulement par niveau, en entrant les options suivantes dans la ligne de compilation Shading Options de WED :

-narrow ***hsize vsize voffset***
-fat ***hsize vsize voffset***

Le premier nombre (hsize) est la taille horizontale de la coque, le second (vsize) est la taille verticale et le troisième (voffset) est le décalage vertical de la coque par rapport à l'origine des entités. Par exemple, -narrow 32 32 0 - fat 64 48 8 donnent les coques par défaut. Toutes les entités de carte doivent être construites avec les mêmes tailles de coque que le niveau. 4 vecteurs prédéfinis contiennent les tailles de boîte de coque du niveau actuel pour l'évaluation dans les fonctions de C-SCRIPT : hull_narrowmin [3], hull_narrowmax [3], hull_fatmin [3] et hull_fatmax [3]. Ils contiennent les valeurs minimales et maximales des x, y, z des coques étroites et grasses.

Pour des entités sprite, son origine est le centre géométrique du bitmap. Pour une carte ou des entités modèles, l'origine est celle donnée par l'éditeur. Si vous voulez que le modèle puisse monter des escaliers, placez simplement son origine dans une position plus élevée de son corps. Le pas le plus haut auquel le modèle peut s'élever est indiquée par la différence la limite inférieure de sa coque et ses pieds. Si vous ne voulez pas que le modèle s'élève du tout - si c'est une voiture, par exemple - placez son origine dans une position basse. Mais soyez prudent, ne le placer pas trop bas, autrement sa coque peut pénétrer dans le sol et le modèle ne sera pas capable de se déplacer du tout. L'origine est aussi le point de référence pour les rotations des modèles.

Les paramètres suivants influencent individuellement le comportement de collision des entités.

min_x, min_y, min_z, max_x, max_y, max_z

Les coins de la boîte de limitation de l'entité, relatifs à l'origine des entités, employés pour la détection de collision contre d'autres sprite ou des entités modèles (pour la détection de collision contre le niveau ou des entités de carte ce n'est pas la boîte mais la coque qui est employée). Les limites sont initialisées avec des valeurs par défaut, mais peuvent être changées. Faites attention que quand elles sont augmentées et qu'une autre entité est proche, cette dernière pourrait être coincée. C'est pour cela qu'elles sont remises à leurs valeurs par défaut chaque fois que l'entité modèle ou l'échelle est changée. Ainsi elles peuvent être changées au plus tôt, un cycle de frame après la création du modèle. La plus souvent employée par des fonctions pour déterminer la taille verticale des entités, est (max_z - min_z).

Notez que la boîte de limitation par défaut d'une entité est proche, mais non identique à la taille réelle. Elle est calculée avec un comportement de collision d'acteur en mémoire. Un axe aligné de la boîte de limitation ne tourne pas, donc il est symétrique en X et Y. Min_z et max_z sont pris de la taille réelle, tandis que le diamètre horizontal est "rétréci" légèrement pour que les acteurs puissent passer facilement par les portes.

fat, narrow (gras, étroit)

Drapeaux, employés pour la détection de collision contre le niveau ou contre les entités de carte. Déterminent quelle coque, si il y en a une, est employée par l'entité. Si une entité est plus grande que 64 quants, son drapeau gras (fat) est automatiquement mis à 1 (on) au début de jeu. Autrement c'est le drapeau étroit (narrow) qui est mis. Si l'entité est plus petite que 8 quants, comme une balle, aucun des drapeaux n'est mis - une coque de point est alors employée. Vous pouvez mettre ou enlever les drapeaux gras (fat) et étroit (narrow) manuellement, pour influencer la coque de collision employée par l'entité. Dans quelques cas vous voudrez que l'entité choisisse la coque étroite au lieu de gras, pour passer par de petites portes. Gras et étroit ne doivent jamais être mis tous les deux en même temps. Ils sont mis automatiquement si l'échelle des entités est changée ou si elle est transformée (morphe).

Exemple pour assigner manuellement une coque étroite à un joueur pour qu'il passe par des portes :

```
my.fat = off;
my.narrow = on;
```

passable

Si le drapeau passable d'une entité est mis à un, sa détection de collision est annulée. Elle ressemblera à un fantôme, se déplaçant par des murs et d'autres entités peuvent se déplacer directement comme cela. Les événements de collision ne sont pas déclenchés

push

Par la valeur de poussée (push) des entités vous pouvez déterminer plus spécifiquement qu'avec passable si l'entité sera un obstacle pour un autre ou pas. L'entité de joueur, par exemple, ne doit pas être un obstacle pour une entité de cabine d'ascenseur, autrement la cabine ne se déplacerait jamais avec le joueur. Une entité avec une valeur de poussée (push) plus haute (défaut = 0) considérera une entité avec une valeur de poussée inférieure comme n'étant pas un obstacle et ira droit sur elle.

Notez s'il vous plaît que les entités ne seront pas poussées de la voie automatiquement. La poussée de la voie, si vous le désirez, peut être exécutée par la fonction d'événement (voir ci-dessous). Les valeurs de poussée (push) des portes et des ascenseurs sont définies d'avance à 10 par les fonctions de doors.wdl.

trigger_range

Donne le rayon d'action pour lequel event_trigger sera déclenché par une entité passant par là, à la condition que cette entité ait la valeur trigger_range à non zéro. Défaut = 0.

Events (Evènements)

Une entité peut être rendue sensible à certains événements de jeu, comme des collisions, être tué ou être cliqué avec la souris. Si de telles choses arrivent à une entité, chaque fois sa fonction événement (event) est démarrée.

event

La fonction d'événement (event) peut être mise par l'action principale des entités, par exemple:

```
my.event = my_function;
```

La fonction commence aussitôt qu'un certain événement arrive avec la le drapeau correspondant enable (permis) à 1 (on). Au commencement de la fonction d'événement, la variable prédéfinie event_type peut être vérifiée pour déterminer quelle sorte d'événement est arrivé. Ainsi pour chaque type d'événement (event) il y a un drapeau enable (permis) pour faire que cette entité soit sensible à cet événement et une valeur dans la variable event_type. Fort de cela, la fonction d'événement peut alors faire réagir l'entité sur l'événement, en donnant un chemin, en répondant à un tir, en explosant ou tout autre.

Les fonctions d'événement sont en réalité exécutées immédiatement pendant l'instruction d'une autre entité qui a causé l'événement, comme les instructions ent_move, scan ou trace. La fonction d'événement elle-même doit normalement seulement transférer l'information à la fonction principale de l'entité - elle ne doit pas exécuter des instructions qui peuvent déclencher des événements eux-même, déplacer des entités ou changer autre chose dans le niveau. Ainsi les instructions comme ent_move, create, ent_remove, trace etc. doivent être évitées ici ! Autrement tous sortes de mauvaises choses peuvent arriver, comme deux entités déclenchant indéfiniment l'événement de l'autre (le jeu pourrait se geler dans ce cas). Si pour quelque raison la fonction d'événement doit exécuter des 'instructions critiques', elles doivent être précédées par un wait(1) pour les retarder à l'encadrement suivant. Alors ce sera sûr.

Les événements suivants peuvent déclencher la fonction d'événement (event), avec un nombre de mis à la variable event_type qui indique le type d'événement et peut donner un des mots-clés prédéfinis suivants pour une meilleure lisibilité.

event_block - enable_block

Collision avec une surface du niveau pendant une instruction de mouvement (move) . L'entité ne doit pas être passable et sa valeur de poussée (push) doit être inférieure ou égale à 0. Au début de la fonction d'événement (event) , le vecteur normal est mis à une direction perpendiculaire à la surface et le vecteur rebond (bounce) est mis à une direction dans laquelle l'entité ricocherait. Exemple:

```
var angle[3];

function bounce_event()
{
	if (event_type == event_block) {
		play_entsound my,whamm,50;
		to_angle angle,bounce; // rebondissent de la surface
		my.pan = angle.pan; // met ma face dans la nouvelle direction
		my.tilt = angle.tilt;
	}
	if (event_type == event_entity) {
	...
	}
	// etc. ...
}

action bounceball {
	...
	my.enable_block = on; // rend l'entité sensible à la collision de blocs
	my.enable_entity = on; // rend l'entité sensible à la collision d'entité (voir ci-dessous)
	my.event = bounce_event;
		...
}
```

event_stuck - enable_stuck

Attrapé dans un coin pendant un déplacement (move) , incapable de se déplacer plus loin. L'entité ne doit pas être passable.

event_entity - enable_entity
Collision avec une autre entité pendant le déplacement (move). L'entité ne doit pas être passable et sa la valeur de poussée (push) doit être inférieure ou égael à la valeur de poussée (push) de l'autre entité. Le synonyme you est mis à l'autre entité, normal et rebond (bounce) sont mis comme auparavant.

event_impact - enable_impact
Frappé par une autre entité qui a exécuté un déplacement (move) et avait un paramètre de poussée (push) inférieur ou égal. Aucune des 2 entités ne doit être passable. You est mis à l'autre entité. Normal et rebond (bounce) sont aussi mis pour donner la direction perpendiculaire à la surface se déplaçant et la direction dans laquelle l'entité rebondirait.

event_push - enable_push
Écrasé par une autre entité avec un paramètre de poussée (push) plus haut. Aucune entité ne doit être passable. You est mis à l'entité qui pousse. Normal et rebond (bounce) seront mis comme d'habitude. Event_impact n'est pas déclenché dans ce cas.

La fonction d'événement (event) peut maintenant contrôler le comportement de conduites auxiliaires des entités. Si l'entité poussée exécute maintenant une instruction de mouvement (move), elle ne déclenchera pas une nouvelle collision avec le propulseur, parce que l'entité you est exclue de la détection de collision de l'instruction de mouvement (move).

Si deux entités entrent en collision, la fonction d'événement (event) de chacune est déclenchée, les entités se déplaçant avec un event_entity, les entités de coup avec un event_impact ou un event_push. Si les deux entités se déplaçaient, chacune peut obtenir deux événements différents dans un ordre indéfini.

event_click - enable_click
Cliquée dessus avec le bouton gauche de la souris.

event_rightclick - enable_rightclick
Cliquée dessus avec le bouton droit de la souris.

event_touch - enable_touch
Touchée avec la souris.

event_release - enable_release
La souris a été enlevée du dessus de l'entité.

Les événements de souris seront déclenchés seulement si l'entité est dans une distance donnée par la variable mouse_range prédéfini (par défaut 1000 quants) de la position de caméra.

event_scan - enable_scan
Balayée par une instruction scan_entity. Si scan_entity a été exécuté par une entité, you est mis à 1. Result est mis à la distance du centre du cône de balayage.

event_detect - enable_detect
A exécuté une instruction scan_entity et trouvé une entité avec enable_scan de mis à l'intérieur du cône de balayage. You est mis à cette entité, et result est mis à la distance à cette entité. Pour chaque entité trouvée l'événement est déclenché séparément.

event_trigger - enable_trigger
Une autre entité a exécuté un mouvement (move) dans un certain rayon d'action. You est mis à l'entité déclenchante. Le rayon d'action est le recouvrement des 2 paramètres trigger_range des entités. Si le trigger_range des 2 entités est 0, aucune fonction n'est déclenchée.

event_shoot - enable_shoot

Frappée par une instruction trace() avec activate_shoot. You est mis à l'entité de traçage, s'il y en a une.

event_sonar - enable_sonar
Frappée par une instruction trace() avec activate_sonar. You est mis à l'entité de traçage, s'il y en a une.

event_disconnect - enable_disconnect
L'entité client a débranché et a quitté le jeu de multi joueur. La fonction évènement (event) peut être employée pour enlever l'entité.

Après le début d'une fonction d'événement (event), toutes les variables et les synonymes dépendant de cet événement, comme NORMAL etc., gardent leur valeur uniquement jusqu'à la prochaine instruction wait().Pendant la pause wait ils peuvent (et ce sera certainement le cas) être changés par d'autres fonctions. Si vous voulez les garder plus longtemps, copiez les dans des variables d'entités. Pour être clair, faite que l'action principale de l'entité fasse la plupart du travail et faites la fonction d'événement (event) la plus courte et la plus simple possible sans aucune instruction wait().

Paramètres internes

Les paramètres suivants n'ont aucune influence sur le regard ou sur le comportement de collision de l'entité elle-même, mais peuvent être employés par des fonctions :

client
Dans un jeu multi joueur chaque entité a un paramètre de client unique. Si l'entité a été créée par un client, le paramètre est mis à ce numéro de client; autrement il est à 0. Il peut être employé pour déterminer si deux entités ont été créées par le même client ou par le serveur. Si une entité a été créée par une autre entité, il hérite du paramètre de client .

skill1 ... skill48
Des variables numériques universelles pour une utilisation dans les fonctions. Les 8 premières variables d'entité peuvent être mises par WED; leur signification dépend aussi de l'action des entités. Les variables restantes peuvent être employées pour stocker des propriétés numériques internes, comme la vitesse ou la position cible. Chaque trois variables d'entité consécutives peuvent être employées ensemble comme un vecteur, qui est donné en donnant la première des trois variables.

flag1 ... flag8
Drapeaux binaires universels pour utilisation dans des actions. Peuvent aussi être mis par WED.

Paramètres des particules

Les paramètres suivants sont uniquement utilisés avec les particules, et ne doivent jamais être mises pour des entités.:

lifespan
L'espérance de vie de la particule en ticks coche (par défaut 80). Le compte à rebours est automatiquement déclenché; quand elle atteint 0 ou qu'elle est mise manuellement à 0, la particule est enlevée.

vel_x, vel_y, vel_z
Vecteur de vitesse de la particule, ou vecteur de ligne pour un rayon ou effet de bande. Au commencement de la fonction est mis au vecteur donné par l'instruction d'effet.

gravity
L'accélération de gravité de la particule en quants par tick carré (par défaut à 0). Est employé pour l'accélération Vel_z quand le drapeau de mouvement (move) est mis.

size
Taille de la particule en quants (par défaut 4)

bmap
Bitmap de la particule (si elle existe)

move
Si ce drapeau est mis, la particule se déplace avec son vecteur de vitesse et accélère avec l'accélération de gravité.

P beam
En mettant le drapeau de rayon d'une particule, on "enduit" la particule le long d'une ligne donnée par sa vitesse Vecteur et commençant à sa position (D3D mode seulement). Normalement employé pour les effets de particule simples. En combinaison avec le drapeau éclat (flare) et le drapeau brillant (bright), les rayons de lumière peuvent être créés de cette façon pour des effets de laser, de tracé d'une balle ou de lumière traînante.

P streak
Comme rayon (beam), mais bouts (streches) la particule le long de son vecteur de vitesse, au lieu de l'enduire. L'effet de bande (streak) est plus rapide que le rayon, mais semble moins spectaculaire. Arrangement(mise) de bande pour effets de multiparticule, comme Les explosions, donnent une tache de mouvement à l'effet comme l'apparition.

function
Fonction de particule. Lorsqu'il est mis à NULL, la particule continue de vivre jusqu'à ce que sa durée de vie (lifespan) arrive à 0.

skill_x, skill_y, skill_z
Vecteur de particule de but général pour utilisation dans la fonction de particule.

Interface Utilisateur: panneaux, Textes, vues (Panels, texts et Views)

Au début du jeu la vue de caméra remplit l'écran entier; sans scénario C-SCRIPT, aucun menu ou une quelconque interface utilisateur n'est visible. L'interface utilisateur doit être faite via C-SCRIPT par voie des panneaux, de textes, de nouvelles vues 3D et d'entités qui sont visibles sur écran.

Panels (Panneaux)

Les panneaux sont des zones rectangulaires avec un modèle de fond ou de revêtement et des instruments facultatifs ou des commandes sur eux. Ils peuvent être employés pour des cockpits, des tableaux de bord, des inventaire et articles d'inventaire, des boutons, des écrans d'accueil ou des images. Ils sont définis de la façon habituelle :

panel ***name*** { ... }

Définit un écran d'affichage avec le nom name. Exemple:

```
bmap compass_map = <compass.pcx>;
panel aircraft_pan
{
	pos_x = 4; pos_y = 4;
	digits = 0,0,4,digit_font,1000,player._rpm;
	digits = 60,0,4,digit_font,1,player._speed_x;
	digits = 120,0,4,digit_font,1,my_height;
	window = 200,0,40,20,compass_map,compass_pos.x,compass_pos.y;
	flags = refresh,visible;
}
```

Une définition de panneau peut contenir - dans l'ordre donné - les paramètres suivants :

bmap = ***bma***p;

Le nom de l'image bitmap pour le fond du panneau. La taille de cette image bitmap détermine la taille du panneau. Normalement il sera dessiné une seule fois. Si le panneau est déplacé sur l'écran, c'est-à-dire la position du panneau est changée, l'image de fond sera redessinée. De cette façon un panneau peut être employé pour un sprite 2D.

Les panneaux peuvent avoir n'importe quelle taille. Quand le panneau bitmap est plus grand que les restrictions données par la carte 3D, il est automatiquement découpé et rendu en plusieurs parties par le moteur.

layer = ***number ;***

Détermine l'ordre du panneau, s'il se chevauche avec d'autres objets. Les éléments avec la valeur de couche (layer) plus haute seront placés sur des éléments avec la valeur de couche (layer) inférieure. Le paramètre de couche ne peut pas être changé pendant le jeu.

pos_x = ***numbe***r; pos_y = ***numbe***r;

Distance du bord supérieur gauche du panneau au bord supérieur gauche de l'écran. Ces valeurs peuvent être changées pendant le jeu pour déplacer le panneau sur l'écran.

alpha = ***numbe***r;

Détermine la transparence d'un panneau translucide dans le mode16 ou de 32 bits. À 0 le panneau est totalement transparent, à 100 il est totalement non transparent. Les éléments de panneau comme des boutons héritent de la valeur alpha du panneau. De cette façons on peut faire apparaître ou disparaître progressivement les panneaux, les boutons etc. Le drapeau transparent du panneau (voir ci-dessous) doit être mis. Exemple pour disparition d'un panneau :

```
my_panel.transparent = on;
```

```
my_panel.alpha = 0;.
while (my_panel.alpha < 100) {
	my_panel.alpha += 20*time;
	wait(1);
}
my_panel.transparent = off;
```

flags = ***flag1, flag2...;***

Ici une liste de tous les drapeaux de panneau peut être donnée, lesquels serons mis au démarrage du jeu. Tous les autres drapeaux sont à 0 (**off**) par défaut. Les drapeaux suivants peuvent être mis:

visible

C'est seulement en mettant ce drapeau que le panneau apparaîtra sur l'écran. S'il doit apparaître sur une vue, le drapeau rafraîchir (**refresh**) doit être mis en plus.

overlay

Si ce drapeau est mis, la couleur 0 (noir) de **bmap** ne sera pas dessinée, pour que l'arrière plan du panneau apparaisse comme un recouvrement.

transparent

Si ce drapeau est mis, l'arrière plan du panneau et les boutons seront dessinés semi-transparents sur l'écran.

refresh

si ce drapeau est mis, le panneau est redessiné à chaque cycle d'encadrement. C'est exigé pour montrer un panneau au-dessus d'une vue ou avoir un texte défilant à travers le panneau sans recopier l'arrière plan. Sans la variable **refresh**, le panneau est seulement redessiné si sa position est changée et ses éléments sont seulement redessinés si la variable correspondante est changée. Particulièrement sur des panneaux énormes, redessiner constamment coûte en temps de rendu et réduit le taux d'affichage.

d3d

Si ce drapeau est mis, le panneau est rendu par le matériel en mode 16 ou 32 bits. Autrement il est rendu par logiciel. Le rendu matériel est plus rapide et à un meilleur aspect, mais consomme beaucoup de mémoire de texture.

On peut donner à un panneau plusieurs sous-éléments, pour la création de cockpits ou des menus. Dans les définitions d'élément suivantes les positions données x, y se réfèrent à la distance du coin gauche supérieur de l'élément respectif au bord gauche supérieur du panneau :

button = ***x, y, bmapon, bmapoff, bmapover, actionon, actionoff, actionove***r;

Définit un bouton sur le panneau. La taille du bouton donné correspond à la taille de l'image bitmap avec le nom donné par **bmapon**. Cette image bitmap sera visible si le bouton de souris gauche est appuyé sur le bouton. **Bmapoff** les boutons seront visible si la souris est sur le panneau, et non pas sur le bouton. **Bmapover** bouton visible tant que la souris est sur le bouton.

ActionOn sera exécutée si la souris est cliquée sur le bouton; **Actionoff** sera exécutée si la souris est maintenue pendant un certain temps et ensuite relâchée sur le bouton; et **actionover** sera exécutée si la souris touche le bouton. On doit donner des noms de fonction réels ici, pas des synonymes. À part **bmapon**, qui détermine la taille du bouton, chacun nom d'image bitmap et de fonction peut être remplacé par **null**.

Plusieurs boutons sur le même panneau peuvent partager la même fonction, s'ils sont définis avec un paramètre. Le numéro du bouton qui a déclenché la fonction est livré comme paramètre à cette fonction. Le numéro 1 correspond au premier bouton dans la définition de panneau. Exemple:

```
function count_up_skill(button_number)
{
	play_sound click,50;
	if (button_number == 1) { player.skill40 += 1; }
	if (button_number == 2) { player.skill41 += 1; }
```

```
        if (button_number == 3) { player.skill42 += 1; }
}

panel skill_pan {

_
button = 0,0,on_map,off_map,off_map,count_up_skill,null,null;
button = 0,10,on_map,off_map,off_map,count_up_skill,null,null;
button = 0,20,on_map,off_map,off_map,count_up_skill,null,null;
_
}
```

vslider = ***x, y, len, bmap, min, max, va***r;
hslider = ***x, y, len, bmap, min, max, va***r;

Curseur vertical ou horizontal, que l'on peut traîner avec la souris pour entrer des valeurs. Le nombre Len donne la hauteur ou la largeur de l'étendue du glissement en pixels. L'image bitmap bmap est employée pour le bouton de curseur, que l'on traîne avec le bouton de souris gauche appuyé. Les nombres min et max déterminent l'étendue des valeurs du curseur qui glisse. var est mise à une valeur dans cette intervalle dépendant de la position courante du curseur. Si déplacé à la position supérieure ou gauche, var est mis à la valeur min, si déplacé à la position inférieure ou droite var est mis à la valeur de max. Exemple:

```
hslider = 10,10,40,slider_map,1,8,my_var; // shifts my_var from 1 to 8 over 40 pixels
```

vbar = ***x, y, len, bmap, factor, va***r;
hbar = ***x, y, len, bmap, factor, va***r;

Affiche une barre graphique verticale ou horizontale pour la représentation d'une variable non-tableau dans le panneau. La hauteur ou la longueur de l'image bitmap varie en fonction de la valeur de la variable. ***Len*** *est la taille de la barre* en pixels. ***factor*** est un nombre à virgule fixe, qui lorsqu'il est multiplié par la variable ***var*** donne la variation de l'image bitmap en pixels. ***Bmap*** doit être au minimum de ***(len*** + ***facto***r*(valeur maximum de var)) pixels en taille verticale ou horizontale.

window = ***x, y, dx, dy, bmap, varx, var***y;

Affiche une partie découpée d'une image bitmap. Les nombres ***dx*** et ***dy*** donne la taille de la découpe en pixels. L'image source ***bmap*** ne doit pas être plus petite que ***dx*** et ***d***y. ***VarX*** and ***varY*** donne la position du coin supérieur gauche de la découpe relative à l'image. La fenêtre peut seulement être placée à l'intérieur des bords de l'image bitmap.

digits = ***x, y, len, fon***t, ***facto***r, ***va***r;

Affichage de nombres. ***Font*** est un jeu de caractère précédemment défini constitué de 11 caractères (des chiffres 0 .. 9 et l'espace) ou de 128 ou 256 caractères ASCII. La partie entière de la variable non tableau ***var*** est montrée avec le nombre de chiffres (***len)*** dans le panneau. La variable est d'abord multipliée par ***factor***. A la place de la variable on peut utiliser le nom de n'importe quel paramètre numérique d'objet.

Les 0 non significatifs sont supprimés.. Si la fonte ne contient pas le caractère moins (-), les valeurs négatives ne sont pas montrées. Au lieu d'afficher des nombres, vous pourriez employer une fonte spéciale bitmap pour montrer des symboles avec un affichage sur un caractère.

mouse_map = ***bma***p;

Curseur de souris alternatif à l'intérieur du panneau. Si on ne donne pas de nom, le curseur de souris reste celui par défaut.

on_click = ***function;***

Au lieu de la fonction on_click globale, cette fonction est exécutée par le clic gauche avec le curseur de souris n'importe où dans le panneau bitmap.

Texte

Texte est employé pour montrer des chaînes de caractères pour des menus, pour des messages ou pour le dialogue avec des acteurs.

text ***name*** { ... }
Définit un texte formatté avec le nom ***nam***e. Une définition de texte peut contenir – dans l'ordre donné – les paramètres suivants:

layer = ***numbe***r;
Détermine l'ordre du texte, s'il chevauche d'autres éléments sur l'écran. Les éléments avec une valeur plus grande de layer sont placés dessus les éléments ayant une valeur layer plus petite. Le paramètre layer ne peut pas être changé pendant le jeu

pos_x = ***numbe***r;
pos_y = ***numbe***r;
Distance entre le coin supérieur gauche du texte et le coin supérieur gauche de l'écran. La position peut être à l'extérieur de l'écran mais seule la partie visible du texte sera affichée. La position du texte peut être changée durant le jeu. Ce qui permet de faire un défilement horizontal ou vertical avec les longs textes.

size_y = ***numbe***r;
Hauteur du texte affiché en pixels (defaut et maximum = la hauteur de l'écran). Le texte est affiché uniquement dans la partie verticale de l'écran compris entre pos_y et pos_y+size_Y.

offset_y = ***numbe***r;
Le numéro de la première ligne de pixel du texte, qui sera affiché à la position pos_y. Ce paramètre vous permet de faire défiler le texte verticalement pixel par le pixel.

strings = ***numbe***r;
Nombre maximal de chaînes de caractères que text peut contenir. Ce paramètre ne peut pas être changé pendant le jeu. Les chaînes de caractères seront affichées les unes sous les autres. Des chaînes de caractères multiples sont particulièrement utiles pour des menus.

font = ***font;***
le jeu de caractère pour le texte; doit contenir 128 ou 256 caractères. Ce paramètre ne peut pas être changé pendant le jeu.

char_x, char_y
Taille d'un caractère simple de la fonte en pixels. Ces valeurs ne peuvent pas être changées, mais peuvent être employées dans des fonctions pour déterminer les limites de l'affichage du texte.

string = ***string1, string2...***;
Le texte actuel, contenant une ou plusieurs chaînes de caractères (pour lequel nous avons défini par avance le nombre maximal dans string !) . String peut être changé par des fonctions pendant le jeu. Si le nom d'une chaîne de caractères est assigné comme null, aucun texte n'est affiché. Si le texte contient plus d'une chaîne de caractères, les chaînes de caractères peuvent être accèdées individuellement par un suffixe d'index [ ], par exemple :

```
my_text.string[7] = new_string;.
```

P dataview = ***nam***e;
Assigne un objet dataview au texte. Le texte contient tous les champs de chaînes de caractères de la base de données dataview, dont le numéro du champ correspond au paramètre viewpos. Seules les chaînes de caractères de l'enregistrement choisi sont montrées et le paramètre index compte seulement les enregistrement choisis.

viewpos = ***numbe***r;
Le numéro du champ "chaîne de caractère" de la base de données que l'on veut afficher dans le texte.

E alpha = ***numbe***r;

Détermine la transparence d'un texte translucide dans le mode 16 ou 32 bits. À 0 le texte est totalement transparent, à 100 il est totalement non transparent. De cette façon les textes peuvent apparaître ou disparaître sans à-coup. Le drapeau transparent du texte (voir ci-dessous) doit être mis à 1.

flags = ***flag1, flag2...;***

Ici on peut donner une liste de tous les drapeaux de texte, qui doivent être mis au début de jeu. Tous les autres drapeaux sont à 0 (off) par défaut. Les drapeaux suivants peuvent être mis :

center_x, center_y

Mettre à 1 l'un de ces drapeaux fera que le texte sera centré horizontalement autour de pos_x et/ou verticalement autour de pos_y. Si le drapeau respectif n'est pas mis, le texte sera montré justifié gauche ou haut

condensed

Lorsque ce drapeau est mis, le texte est compressé horizontalement de 1 pixel par caractère. Plus particulièrement les fontes italiques se lisent mieux de cette façon.

narrow

Similaire à condensed, mais le texte est comprimé davantage.

transparent

Si ce drapeau est mis le texte sera affiché en semi transparence à l'écran.

visible

C'est uniquement avec ce drapeau que le texte apparaîtra à l'écran.

Views (vues)

Pour la création de rétroviseurs, de caméras de missile ou équivalent ou pour des jeux multi-joueur de PC simple, de multiples vues 3D peuvent être montrées dans la carte de niveau. Une vue, nommée camera, est déjà prédéfinie au début du jeu. Chaque nouvelle vue peut être créée par la définition de vue (view) :

view ***name*** { ... }

Définit un nom de vue ***name***. Dans la définition de vue (view) on peut donner les paramètres suivants :

layer

Détermine l'ordre de la vue, si elle chevauche avec d'autres vues. Les vues avec une valeur plus grande de layer seront placées dessus les vues ayant une valeur layer plus petite. Les panneaux et les textes sont toujours montrés par-dessus les vues indépendamment de leur valeur de couches (layer).

pos_x, pos_y

Distance entre le coin supérieur gauche de la vue (view) et le coin supérieur gauche de l'écran. La position doit être à l'intérieur de l'écran et la taille de la vue doit être également choisie pour qu'aucune de ses parties ne soit à l'extérieur de l'écran.

size_x, size_y

Taille horizontale et verticale de la vue en pixels. En mode logiciel, la taille doit être un multiple de 4 dans la direction horizontale et de 2 dans la direction verticale.

x, y, z

Position de la vue (view) dans la carte. Peut seulement être mis par une fonction. La position x peut également être employée comme un vecteur.

pan, tilt, roll
: Angles de la vue (view) dans la carte. Peut seulement être mis par une fonction. L'angle pan peut également être employée comme un vecteur..

arc
: Angle, qui donne le champ de vision; Par défaut la valeur est de 60 degrés qui correspondent à l'oeil humain. En diminuant cet angle vous obtiendrez un effet de télescope. *Si l'arc de la vue à une valeur négative*, la vue est inversée horizontalement en mode 16 ou 32 bits. C'est de cette façon que l'on peut faire des rétroviseurs.

aspect
: Détermine la proportion de taille de pixel vertical-horizontal. La valeur par défaut de 1 donne une proportion normale de 1:1.

offset_x, offset_y
: Définit une compensation du point de vision dans la direction horizontale et verticale. À 0 le point de vision est dans le centre de la fenêtre de vue comme auparavant; à-0.5 il est au bord gauche ou supérieur, à 0.5 il est au bord droit ou inférieur de la fenêtre de vue. Il ne doit pas être à l'extérieur des bords de vue. La compensation de vision peut être employée pour la conduite ou des simulateurs de vol où le conducteur ou le pilote est assis à droite ou gauche de l'écran. Exemple:

```
camera.offset_x = -0.33; // met le point de vision pour un pilote dans une deux places
```

ambient
: la valeur de lumière (0 .. 100, defaut 0) supplémentaire à tout que l'on voit dans cette vue. Peut être utilisé, par exemple, pour créer une vue infrarouge ou de radar ou changer la brillance complète de la carte.

fog
: La force (0 .. 100, defaut 50) de l'effet brouillard coloré pour cette vue. Seulement efficace dans des zones ou il y a du brouillard, où les surfaces ont obtenu une valeur de brouillard ou si le brouillard global (voir la variable fog_color) est mise. Si cette valeur est mise à 0, il n'y aura pas de brouillard. A 100, vous pouvez à peine voir la main des entités de joueur devant ses yeux. Le brouillard diminuera la vitesse de rendu en mode logiciel.

fog_start, fog_endErreur ! Signet non défini.
: Alternative pour contrôler la force du brouillard àtravers le paramètre fog, ces paramètres donnent la valeur de début et de fin pour un brouillard en mode D3D. Il n'y aura pas de brouillard plus étroit que le fog_start; le monde sera tout brumeux au delà du fog_end. De cette façon une borne très pointue de brouillard peut être donnée, par exemple pour cacher complètement tout en dehors du clip_range. Exemple:

```
camera.fog_start = 0.8 * clip_range;
camera.fog_end = 0.9 * clip_range;
```

Notez qu'en mode 8 bits certaines cartes comme la Voodoo ou la Radeon ne supportent pas des valeurs de brouillard. Faites donc bien attention que votre brouillard s'adapte à tous les cas. En mettant fog_end à 0 (défaut) c'est le paramètre fog qui est utilisé pour la force du brouillard.

P alpha = number;
: Donne la translucidité d'une vue transparente en pourcent. (50% par défaut)

genius = ***pointer***;
: Synonyme pour une entité qui est attachée à la vue. Employé pour des jeux première et troisième personne. Cette entité sera invisible si la vue est dans ses limites. La vue peut toujours se déplacer indépendamment de l'entité, mais elle ne verra pas d'objet qui ne peut pas être visible en même temps pour l'entité. En mettant genius à null, aucune entité n'est attachée à la vue.

P portal = view;

La seconde 'vue portal' qui est attaché à la vue. Un portal est une surface spéciale d'un niveau ou d'une entité carte où cette vue portique est dressée. De cette façon des fenêtres dans des secteurs différents du niveau, aussi bien que des miroirs en temps réel peuvent être créés.

Si le drapeau de miroir (seulement disponible dans WED 5, émulé par **flag8** dans WED 4) est mis pour une surface, la vue portique est mélangée sur cette surface. Seulement une entrée peut être visible à la fois. S'il y a plusieurs portals dans la vue, seulement le premier sera montré correctement.

Le pourcentage de mélange est donné par le paramètre **albedo** de la surface (0 .. à 100 %). Si la surface fait partie d'une entité de carte, alors sa valeur d'albedo module la valeur de surface, donnant une harmonie douce aux miroirs.

Aussitôt qu'une surface portique est visible dans la vue, les paramètres de la vue portal_x, portal_y et portal_z sont mis à la position du premier vertex de cette surface. Les paramètres de la vue Pnormal_x, Pnormal_y et pnormal_z sont mis au normal de la surface portique. Ces paramètres peuvent être employés pour imiter un miroir en temps réel en mettant les paramètres de vue portiques en conséquence. Exemple :

```
// Le code pour imiter un miroir horizontal en produisant une caméra inversé verticalement à
travers le plancher

view mirror { }
function init_mirror_z()
{
    camera.portal = mirror;
    mirror.noshadow = on; // supprime les ombres du miroir
    mirror.noparticle = on; // supprime les particules du miroir
    mirror.portalclip = on;
    while (1) {
    proc_late(); // place ceci à la fin de la liste des fonctions – la caméra devant être
    déplacée en auparavant
    mirror.genius = camera.genius;
    mirror.aspect = -camera.aspect; // inverse l'image dessus dessous
    mirror.arc = camera.arc;
    mirror.fog = camera.fog;
    mirror.x = camera.x;
    mirror.y = camera.y;
    mirror.z = 2*camera.portal_z-camera.z; // déplace la caméra de deux fois la distance du
    portal
    mirror.pan = camera.pan;
    mirror.tilt = -camera.tilt; // inverse l'angle vertical de la caméra
    mirror.roll = camera.roll;
    wait(1);
    }
}
```

Notez que bien que cela ressemble à un miroir réel, c'est juste une deuxième vue de caméra imaginaire. La caméra imaginaire ne doit avoir aucun obstacle devant elle et doit examiner 'un trou' dans le mur où le miroir est attaché. Le miroir lui-même doit être une entité de carte pour être invisible à la caméra imaginaire. Les murs de côté du trou de miroir ne doivent être mis à aucun mode de rendu pour être invisible aussi quand la caméra se balance autour de sa position imaginaire.

flags = ***flag1, flag2...;***

Ici on peut donner une liste de tous les drapeaux, qui doivent être mis au début de jeu. Tous les autres drapeaux de vue sont à 0 par défaut. Les drapeaux suivants peuvent être mis :

visible

Si ce drapeau est mis, la fenêtre de vue apparaîtra à l'écran. La vue de caméra a ce drapeau de mis par défaut.

P transparent

Si ce drapeau est mis, la vue est affichée translucide sur le fond (mode D3D uniquement). La translucidité peut être donnée par le paramètre alpha des vues en pourcent.

P portalclip

Si ce drapeau est mis, les surfaces de niveau sont coupées loin de la position des portal_x, portal_y et portal_z de la vue. Utilisée pourles mirroirs.

audible

Si ce drapeau est mis, les sources sonores et les sons d'entité dans le niveau près de la vue sont joués sur la carte son. La vue de caméra mets ce drapeau par défaut. Pour éviter une cacophonie de son, seule une vue doit mettre ce drapeau.

noshadow

Si ce drapeau est mis, aucune ombre modèle dynamique n'est montrée dans la vue. Utile pour des miroirs horizontaux où vous ne voulez pas que l'ombre soit reflétée aussi.

noparticle

Si ce drapeau est mis, aucune particule ne sera affichée dans la vue.

Variables du moteur (Engine Variables)

Il y a beaucoup de variables internes prédéfinies et des synonymes, qui influencent le rendu de beaucoup de façons ou qui contiennent des valeurs qui peuvent être employées dans des fonctions :

nullvector
: Les trois éléments de ce vecteur prédéfini sont à 0.

pi
: Cette variable a la valeur de 3.14159265.

version
: Cette variable contient le numéro de version du moteur(par exemple 5.120).

time
: Le temps du dernier cycle d'affichage, en ticks. Souvent employé comme un facteur de correction pour compenser les différents taux d'affichage pour des fonctions dépendant du temps. Une vitesse multipliée par le temps (time) donne une distance. Sur un taux d'affichage de 16 fps, cette variable a une valeur de 1. Pour se prémunir contre des 'secousses' extrêmes, time est limité par fps_min.

total_ticks
: Le temps qui a passé depuis le début du jeu. La valeur de cette variable est augmentée à chaque cycle d'affichage par la variable time.

total_frames
: Le nombre de frame affichée depuis le début du jeu.

sys_seconds (Secondes, 0..59)
sys_minutes (Minutes, 0..59)
sys_hours (Heures, 0..23)
sys_day (Jour du mois, 1..31)
sys_month (Mois, 1..12)
sys_year (Année, 4 caractères)
sys_dow (Jour de la semaine, 1..7, 1 = Lundi)
sys_doy (Jour de l'année, 1..365)

Ces variables s'obtiennent à partir de l'horloge interne du PC.

sound_vol
: Le volume pour tout les bruitages wav, valeur 0 .. 100.

midi_vol
: Le volume pour les chansons midi, valeur 0 .. 100. Par défaut, le fait d'appuyer sur la touche [F12] allume/éteint alternativement le son et le volume de la musique.

midi_playing
: Tant qu'une chanson midi est jouée, cette variable est à non-zéro.

movie_frame
: Donne le numéro d'encadrement du fichier d'animation de film actuellement joué ; Est à 0 si aucun film n'est joué.

cd_track

Le numéro de la piste audio du CD actuellement joué ou 0 si rien n'est joué. Est uniquement mis à jour par l'instruction play_cd.

freeze_mode
Si cette variable est mise à 1, toutes les actions d'entités s'arrêteront, toutes les entités dans le niveau se «gèleront ». Si la variable est mise à 2, ce sont les fonctions en cours d'éxécution qui s'arrêteront. Cette variable est normalement utilisée pour faire une pause dans le jeu en éxécution. La réinitialisation de la variable à 0 (le défaut) fera que toutes fonctions arrêtées reprendront au point ou elles se trouvaient lorsque freeze_mode a été mis.

fps_max
Limite le taux de rafraîchissement à la valeur donnée (par défaut 70 fps), pour éviter des différences de vitesse et d'autres effets qui peuvent arriver sur des extrêmement hauts taux de rafraîchissement en raison de pauvres scénarios C-Script.

fps_min
Cette variable (default 4) limite la variable time à un taux de rafraîchissement minimal. En dessous de ce taux, time reste constant.

pos_resolution
Cette variable choisit la résolution des entités positions. À 0 la résolution est de 0.125 quants. À 1 (le défaut) la résolution est de 0.001 quants, pour tenir compte des niveaux extrêmements énormes de centaines de kilomètres carrés avec des modèles très réduits. Les objets fixés très près de la caméra, comme une arme, se déplacent de façon plus lisse en mode haute résolution. L'inconvénient est une augmentation de 25 % du temps de réponse pour des jeux multi-joueur. Pos_resolution doit être mis avant le chargement du premier niveau et ne doit pas être changé pendant le jeu.

app_name
Cette chaîne de caractères contient le nom de l'application, par exemple "office" quand le nom de fichier C-SCRIPT principal est office.wdl.

warn_level
Si cette variable (par défaut 1) est mise à 0, les avertissements de texture, les tailles d'entités et mauvaises instructions C-SCRIPT ne sont jamais publiées. S'il est mis à 2, les avertissements concernant de possibles problèmes de taille de texture sont toujours publiés - même si la carte 3D actuelle peut les montrer. C'est utile si vous voulez vérifier si vos textures sont assez petites pour être montrées même sur des vieilles cartes 3D, comme la Voodoo3.

num_actions
Donne le numéro des actions qui s'exécutent actuellement (incluant des actions de particule). Il est aussi montré dans le panneau de qualité [D].

Les variables et les synonymes suivants peuvent être employés pour définir des configurations globales pour le rendu, l'apparition des surfaces et des effets visuels dans le niveau :

video_mode
Le mode vidéo actuel. Ne peut pas être mis directement par une fonction, mais peut être défini avec une valeur de départ pour choisir le mode vidéo dans lequel le moteur doit commencer. Pendant le déroulement du jeu, cette variable peut seulement être changée par l'instruction switch_video. Par défaut, l'appui de la touche [F5] bascule la résolution vidéo. Les résolutions suivantes sont disponibles :

320x200	-	1	
320x240	-	2	(mode par défaut pour le survol (fly-thru))
320x400	-	3	(n'est pas supporté par certaines cartes graphiques)
400x300	-	4	(n'est pas supporté par certaines cartes graphiques)
512x384	-	5	(ne fonctionne pas sur les ordinateurs portables)

640x480	-	6	(mode par défaut pour le jeu)
800x600	-	7	
1024x768	-	8	
1280x960	-	9	
1600x1200	-	10	

video_depth

La profondeur de couleur pour le mode vidéo actuel (8, 1 ou 32 bits). Dans le mode 8 bits, le moteur emploie l'outil de rendu logiciel interne et a besoin de DirectX 5.0 ou supérieur. Dans les modes de 16 bits et 32 bits, le moteur emploiera D3D et a besoin de DirectX 7.0 ou supérieur pour rendre l'image. Cette variable ne peut pas être mise directement par une fonction, mais peut être redéfinie avec une valeur de départ pour choisir la profondeur de couleur et le mode dans lequel le moteur doit commencer. Exemple:

```
ifdef d3d;
var video_depth = 16; // le mode D3D est démarré avec -D D3D
ifelse;
var video_depth = 8; // autrement mode logiciel 8 bits
endif;
```

Pendant le jeu cette variable peut seulement être mise par l'instruction switch_video.

video_screen

Le mode vidéo d'affichage actuel (1 = en plein écran, 2 = dans une fenêtre). Quelques vieilles cartes 3D (Voodoo) ne sont pas capables de rendre dans le mode de fenêtre. Pendant le déroulement du jeu, cette variable peut seulement être mise par l'instruction switch_video. Par défaut, en appuyant sur les touches [Alt-Enter] on bascule le mode d'écran.

screen_size.x, screen_size.y

La résolution d'écran actuelle. Ne peut pas être mis directement, mais sera changée automatiquement par l'instruction switch_video.

d3d_triplebuffer

Sur des cartes 3D avec plus de 8 MO de mémoire vidéo, 3 tampons sont employés dans le mode plein écran, sinon on utilise 2 tampons. Trois tampons peuvent augmenter le taux d'affichage, mais consomme plus de mémoire vidéo pour un troisième écran tampon. On peut empêcher les 3 tampons en redéfinissant la variable d3d_triplebuffer à 0 et au contraire on peut la forcer en la redéfinissant à 1. D3d_triplebuffer ne doit pas changer pendant le jeu. Exemple:

```
var d3d_triplebuffer = 0; // empêche les 3 tampons
```

d3d_vsync

Si cette variable est mise à 0, le moteur n'attend pas la synchro verticale avant le basculement des mémoires tampons écran. Cela empêche le taux d'affichage d'être bloqué à la moitié ou au tiers de la fréquence vidéo et peut augmenter le taux d'affichage même au-dessus de la fréquence vidéo. L'inconvénient consiste en ce que sans l'option d3d_triplebuffer ou quand le taux d'affichage est plus haut que la fréquence vidéo, des déchirements peuvent être visibles à l'écran. La valeur par défaut de d3d_vsync est 0 si d3d_triplebuffer est activé, sinon elle est de 1.

d3d_mode

Cette variable en lecture seule reflète la qualité de la carte 3D; 0 = incapable, 1 = faible, 2 = lentement, 3 = bon. Elle peut être employée pour réduire le LOD sur les mauvaises cartes 3D et sur les lentes, augmenter la lumière ambiante pour compenser l'image peu contrastée des cartes 3D faibles ou donner le conseil à l'utilisateur d'acheter une meilleure carte.

d3d_panels

En mettant ce drapeau à 1 (on), tous les panneaux sont dessinés par le matériel D3D en mode 16 bits, même si leur drapeau D3D n'est pas mis. Dans le mode de 32 bits les panneaux sont toujours dessiné par le matériel. Dessinner les panneaux par le matériel offre l'avantage d'augmenter le taux d'affichage et les panneaux sont dessinés dans de vraies couleurs, mais l'inconvénient est qu'ils consomment de la mémoire vidéo.

d3d_texmemory

La quantité de mémoire vidéo disponible sur la carte 3D, en kilo octets. Est également indiquée par le moteur au début de jeu. Notez s'il vous plaît que la quantité réelle disponible pour des textures 3D est quelque MO de moins, en raison de la mémoire vidéo nécessaire pour les mémoires tampon d'écran. Souvent, la mémoire vidéo disponible est autour de 29 MO sur des bonnes cartes 3D (TNT2) et 5 ou 9 MO sur de vieilles cartes (3dfx, Voodoo).

d3d_texreserved

Quantité de mémoire vidéo réservée pour les textures non-gérées. Ces textures ne sont pas échangées avec la mémoire virtuelle par le manager de texture de DirectX et ne peuvent pas donc produire "de secousse de vue" en raison d'un accès de disque dur pendant l'échange. Toutes les textures qui ne sont pas placées dans l'espace de texture réservé sont échangées. La valeur par défaut de cet espace de texture réservé est la moitié de la quantité de mémoire vidéo disponible (d3d_texmemory) sur des bonnes cartes 3D et 0 sur des cartes plus mauvaises.

D'abord les cartes d'ombre du niveau sont allouées dans l'espace de texture réservé (elles sont les plus critiques pour l'échange), puis les textures de surface puis les entités. Le ciel et les cartes de scène sont alloués dans l'espace de texture réservé avec la priorité plus haute que les textures superficielles, si leurs synonymes sont mis avant le chargement de niveau.

Plus il y a de mémoire vidéo de réservée pour les textures non-gérées, moins il y a d'espace de disponible pour le manager de texture, aboutissant à plus d'échange pour le reste des textures gérées. S'il n'y a aucun espace de laissé dans la mémoire vidéo, aucune des textures restantes ne peut être allouée. C'est indiqué par un message d'erreur et le niveau ne démarrera pas. Même si plus de mémoire vidéo est réservée par d3d_texreserved qu'il y en a de disponible sur la carte 3D, au moins environ 2 MO sont toujours tenus libre pour des textures gérées. De cette façon, de petites textures peuvent toujours être allouées, même avec une mauvaise valeur de d3d_texreserved.

Pour réduire au minimum les secousses d'échange, mettez d3d_texreserved à la taille nécessaire pour le plus grand des niveaux de jeu dans l'action principale avant le chargement de niveau. La taille peut être décidée par d3d_textotal (voir ci-dessous). Considérez s'il vous plaît que quelques cartes de type voodoo/3dfx ne démarreront pas avec aucun espace de texture réservé. Faites donc attention à toujours fournir une option de démarrage pour l'utilisateur final pour mettre hors de service l'espace de texture réservé. Exemple:

```
function main()
{
ifndef notex; // met hors service d3d_texreserved en démarrant avec -d notex sur des cartes faibles
	d3d_texreserved = min(12000,d3d_texmemory/2); // le niveau le plus grand à besoin de 12 Mo pour l'espace texture
endif;
	sky_map = my_sky; // mets la carte du ciel avant le chargement de niveau pour leur donner la priorité
	cloud_map = my_clouds;
	scene_map = my_scene;
	load_level <office.wmb>;
...
}
```

d3d_textotal

Le total de mémoire vidéo locale (en KO), consommé dans le mode D3D pour les textures superficielles, les entités, les cartes d'ombre et les panneaux bitmaps D3D du niveau actuel. Cette variable est visible dans le panneau de mise au point de défaut (en appuyant [D]). Si cette quantité excède d3d_texreserved, les textures peuvent être échangées avec la mémoire virtuelle, qui peut aboutir à des secousses à certains endroits dans le niveau. Notez que selon la carte 3D, cette variable peut augmenter pendant le jeu; c'est le comportement normal et n'indique pas de faute de logiciel ou une perte de mémoire.

d3d_texlimit

Une autre variable pour ces utilisateurs malheureux de cartes Voodoo/3dfx. Au début de jeu il donne la taille de texture maximale que la carte 3D peut montrer. Sur des cartes 3D normales c'est 1024 ou 2048. Des mauvaises cartes 3D comme Voodoo3 ou inférieur ont une taille de texture maximale de 256. On fera alors rétrécir automatiquement toutes les textures plus grande que la taille donnée à la moitié de la taille. De cette façon, des cartes Voodoo/3dfx peuvent montrer des textures superficielles même plus grandes jusqu'à 512x512, cependant avec une qualité réduite.

Un deuxième but de cette variable est de permettre aux niveaux énormes de fonctionner sans à-coup sur le système avec une mémoire de texture très limitée. D3d_texlimit peut être mis manuellement à une valeur inférieure. Si vous le mettez à une valeur comme 64, toutes les textures plus grandes que 64 seront réduites, mais les plus petites tiennent leur qualité originale. De cette façon la mémoire totale de texture nécessaire pour le niveau peut être réduite de plus de 50 %.

d3d_lightres

En mettant cette variable à 1 la qualité des lumières dynamiques dans le mode D3D est augmentée, cependant aux dépens de la diminution du taux d'affichage.

d3d_mipmapping

Détermine la méthode de mipmapping dans le mode D3D :

0 – mipmapping est désactivé
1 – mipmapping polygone (défaut dans le mode 8 bit)
2 – mipmapping balayage de ligne
3 – mipmapping trilinéaire (défaut)

Mipmapping est utilisé pour le 'brouillage' des surfaces lointaines et réduit de ce fait les effets de crènelage. Le mipmapping balayage de ligne rend mieux que le mipmapping polygone mais il n'est pas accepté par certaines vieilles cartes 3D comme Voodoo. Le mipmapping trilinéaire est bien entendu le mieux mais il ne fonctionne que sur les toutes nouvelles cartes 3D. La variable de d3d_mipmapping doit être placée avant le chargement de niveau. Sa valeur par défaut est 3 si la carte 3D supporte le scanline ou le mipmapping trilineaire, autrement elle est à1.

mip_flat, mip_shadedErreur ! Signet non défini.

Utilisez ces variables (défaut 1.5, resp 1.0) pour déterminer la distance relative pour le basculement des textures mipmap en plates ou en ombrées dans le mode mipmapping polygone. Elles peuvent également être mises à différentes valeurs dépendat de video_depth.

d3d_near

Les modèles avec le drapeau near (près) sont rendus avec une faible valeur du tampon z, pour les rendre devant d'autres objets. Cela peut mener à la coupure précoce de tels objets dans le mode D3D quand les polygones pénètrent dans le plan de caméra (comme avec des modèles d'arme). Pour peaufiner la coupure, ce vecteur prédéfini peut être employé. D3d_near.x donne un facteur de compensation du tampon z (0.1 .. 0.25, defaut 0.25) et d3d_near.y donne un facteur de distance (1 .. 10, defaut 2.5). En changeant ces valeurs, le comportement de coupure près des entités modèles peut être influencé.

d3d_lines

Si cette variable est mise à 1, tous les polygones seront entourés par des lignes rouges dans le mode D3D. De cette façon le partage des surfaces par l'arbre BSP et par le processus de mosaïque de lumière dynamique peut être examiné.

d3d_monochrome

Si cette variable est mise à 1 au démarrage du jeu, les lumières dynamiques et statiques seront blanches comme dans la version standard où extra. Cette variable ne peut pas changée pendant le jeu.

d3d_entsort

Si cette variable prédéfinie est mise à 1, les entités transparentes sont triées avant le rendu en mode D3D. Cela empêche les fautes de tri visibles des entités presque intransparente avec une haute valeur alpha ou les canaux alpha. L'inconvénient est quelques pour cent diminution du taux d'encadrement quand beaucoup d'entités transparentes sont sur l'écran. La valeur par défaut de cette variable est 1.

gamma

Peut être employé pour éclaircir ou obscurcir la palette en mode de 8 bits, pour adapter l'image au moniteur ou au niveau de brillance de la pièce. Par défaut, l'appui sur la touche [F11] bascule la variable gamma dans différentes valeurs. En modes 16 bits et 32 bits, où aucune palette n'existe, gamma peut être employé pour éclaircir ou obscurcir les cartes d'ombre, donnant ainsi au niveau une atmosphère plus brillante ou plus sombre. Dans ce cas la variable gamma doit être définie avec une nouvelle valeur dans le scénario C-SCRIPT. Son changement par [F11] ou par une fonction n'a aucun effet.

render_inflate

Cette variable (0 .. 2.5) peut être employée pour compenser des petits trous entre les polygones qui sont produits par des drivers de carte D3D imprécis, particulièrement Voodoo, Banshee ou des cartes ATI. Quand vous mettez une valeur supérieure à 0, les trous sont remplis, mais le rendu est un peu plus lent.

logo

Variable de 1 à 4 pour choisir le coin où le logo de filigrane du moteur sera montré sur les éditions Standard et Extra.

clip_range

Cette variable donne l'étendue de visibilité maximale et les distances LOD d'objets (par défaut 100 000 quants). Les entités à l'extérieur de cette valeur et les blocs à l'extérieur de deux fois cette valeur (sauf des blocs de ciel) ne sont pas visibles dans la vue. Utile pour augmenter le taux d'affichage en cachant les entités lointaines dans le brouillard ou le sombre à l'extérieur des niveaux

clip_factor

Cette variable (par défaut 2) donne la gamme de polygones de coupure de l'arbre BSP, en unités clip_range. Exemple :

```
clip_factor = 1.0; // clips world polygons at same range as entities
```

clip_size

Cette variable (par défaut 0.5) donne le nombre minimal des pixels qui doivent être couverts par un modèle triangle pour rendu. Avec la valeur par défaut, les modèles triangles ne sont pas rendus si Ils sont plus petits que la moitié d'un pixel. Cela augmente le taux d'encadrement, mais peut mener "à des trous" dans des modèles construits de hauts polygone. En employant des modèles de hauts polygones, clip_size doit être mis à 0.

floor_range

Contrôle l'étendue dans laquelle la lumière des entités dépend de la brillance et les ombres de la surface du sol au dessous (par défaut 1000 quants). Si la hauteur de l'entité au-dessus du plancher excède cette valeur, l'entité ne recevra pas de valeurs lumière, albedo et brouillard de la texture de plancher. De cette façon vous pouvez empêcher qu'un modèle d'avion dans un simulateur de vol change soudainement sa brillance en fonction des ombres sur le terrain.

mip_flat, mip_shaded

Employez ces variables (par défaut 1.5 resp. 1.0) pour déterminer les distances relatives pour commuter la texture mipmap sur surfaces plates ou sur surfaces hachurées. Elles peuvent aussi être mises à des valeurs différentes dépendant de video_depth. Une valeur de 0 commute mip-mapping à 0.

tex_share

Si cette variable (par défaut 0) est mise à 1 avant le chargement d'un niveau, les entités de carte partagent leurs textures avec le niveau et avec chacun si elles ont le même nom et la même taille. Cela économise la mémoire de texture.

max_entities
La quantité maximale d'entités dans un niveau; doit être mis avant le chargement de ces niveaux. La valeur par défaut est de 10 nexus.

max_particles
Cette variable (défaut 2000) détermine le nombre maximal de particules si redéfini à une autre valeur en début de jeu.

num_particles
Indique le nombre de particules actuellement existantes.

num_vismappolys
num_visentpolys
num_visents
Ces variables indiquent le nombre de carte visible et de polygones du niveau, de modèle visible et des polygones de sprite, et des entités visibles rendues par affichage. Elles peuvent être employées pour optimiser le taux d'affichage en mettant la topographie de niveau et le clip_range en conséquence. Le panneau de qualité, qui apparaît en pressant [D], montre ces variables.

scene_map
Synonyme pour l'image bitmap scène enveloppée autour de l'horizon. Elle apparaîtra sur toutes les surfaces de ciel et peut être employée pour des montagnes ou pour un paysage de ville à l'horizon. C'est un revêtement, c'est-à-dire que les parties noires (la couleur 0) sont transparentes. Cela doit être une puissance de deux dans la taille horizontale et les bords horizontaux doivent se correspondre sans raccord. scene_map peut être plus grand que la limite de taille de texture de la carte 3D. L'échelle et la position verticale peuvent être ajustées par les deux variables suivantes:

scene_field
Donne le secteur horizontal couvert par scene_map en degrés (par défaut 90), déterminant ainsi l'échelle du bitmap. Si cette valeur est inférieure à 360, la scène bitmap sera étirée horizontalement.

scene_angle.tilt
L'angle d'élévation du bord inférieur de scene_map en degrés (defaut -5). Un angle de 0 place scene_map exactement sur la ligne d'horizon.

scene_nofilter
Si ce drapeau est mis à 1, scene_map est dessiné sans filtrage bilineaire et sans anticrènelage dans le mode D3D. De cette façon cela ne semblera pas flou et n'aura aucune couture noir produite par quelques cartes 3D (Voodoo) en raison de l'anticrènelage à la couleur de transparence noire.

cloud_map
Le synonyme pour une image bitmap nuage de recouvrement, qui apparaît sur toutes les surfaces de ciel derrière et au-dessus de la scène bitmap et remplacera le revêtement de nuage, que l'on peut donner à WED. Il est normalement employé pour des nuages se déplaçant et est projeté sur un dôme autour de la carte. Cela doit être une puissance de deux dans la taille, ses parties de couleur #0 sont transparentes et c'est une texture arrangée côte à côte, donc tous les bords doivent correspondre de façon homogène.

sky_map
Le synonyme pour le ciel bitmap, qui apparaît sur toutes les surfaces de ciel derrière l'image bitmap nuage et remplacera la texture de ciel à l'origine donnée à WED. Il peut être employé pour des étoiles ou pour une deuxième

couche de nuages se déplaçant. Ce doit être une puissance de deux dans la taille et c'est une texture arrangée côte à côte, donc tous les bords doivent correspondre de façon homogène.

Le sky_map et cloud_map peuvent individuellement être éteints en les mettant à **null**. S'ils sont à **null** avant le chargement d'un niveau, ils sont automatiquement mis à la texture de ciel dans le niveau. Si **sky_map** est **null**, **scene_map** est dessiné dans le mode de non-recouvrement. S'il n'y a ni ciel, ni nuage, ni scène, aucun ciel n'est dessiné du tout. De cette façon ses propres modèles de sphère de ciel peuvent être activés. En employant les drapeaux **flare** ou **transparent** pour des sphères de ciel, un nombre arbitraire de couches de ciel peut être créé.

Les variables suivantes déterminent l'apparence de l'arrière plan et le bitmap de nuage et de ciel:

sky_color
Les composantes rouges, verts et bleus de ce vecteur de couleur (par défaut 255,255,255 = blanc) sont multipliées par la texture de ciel dans le mode D3D. De cette façon le ciel peut être foncé pour les mutations jour - >nuit de façon homogène.

scene_color
Les composantes rouges, verts et bleus de ce vecteur de couleur (par défaut 255,255,255 = blanc) sont multipliées par la texture de scène dans le mode D3D. De cette façon la scène peut être foncée pour les mutations jour - >nuit de façon homogène.

bg_color
Par les composantes rouges, vertes et bleues de ce vecteur de couleur, une couleur de fond pour le mode D3D peut être mise. Cette couleur est visible quand aucun ciel n'est dessiné ou quand aucun niveau n'est chargé. Si les trois composantes sont mises à 0 (le défaut), aucune couleur de fond n'est dessinée.

sky_clip
Cet angle d'élévation donne la frontière inférieure du sky_map et cloud_map en degrés. La valeur par défaut 0 correspond à l'horizon. Le ciel au-dessous de cet angle ne sera pas dessiné dans le mode D3D. Cela augmente le taux d'affichage. Exemples :

```
sky_clip = -90; // le ciel entier est dessiné même en dessous du sol
sky_clip = 45; // seulement le segment supérieur du ciel est dessiné
```

sky_scale
L'échelle de **cloud_map** et **sky_map** (par défaut 1).

sky_curve
'La pente raide' du dôme de ciel (par défaut 1).

cloud_speed.x, cloud_speed.y
Donne la vitesse des nuages se déplaçant dans la direction X et la direction Y (par défaut 3)

sky_speed.x, sky_speed.y
Donne la vitesse de mouvement du ciel bitmap dans la direction X et la direction Y (par défaut 1). Pour un ciel d'étoile ne se déplaçant pas, mettez ces valeurs à 0.

sun_angle.pan, sun_angle.tilt
Azimuth (0...360, defaut 0) et élevation (0..90, defaut 30) du soleil. Peut être changé pendant le jeu et détermine la direction de la lumière pour l'ombre des textures plates, pour l'ombrage gouraud de modèles et pour des ombres dynamiques.

sun_pos
Ce vecteur prédéfini contient la position du soleil. Il ne peut pas être mis, mais est calculé automatiquement par les angles de soleil.

sun_light
La force (0 .. 100, defaut 50) de l'effet du soleil sur des surfaces en texture plate et sur des modèles. Si cette variable est mise à 0, l'ombrage des surfaces plates ne dépendra désormais plus de la direction de soleil.

fog_color
Nombre de la couleur brouillard ou obscurité (1..5, defaut 0) Le brouillard ou l'obscurité peuvent être activés localement, en donnant un numéro de couleur de brouillard par la valeur de la texture albedo ou généralement en mettant cette variable à 1.

Une valeur de 1 .. 4 active une des 4 couleurs de brouillard définies dans les propriétés de carte et 5 active l'obscurité. Les scénarios de calibre (template) emploient les numéros de brouillard suivants :

1 – brouillard distant (blanc)
2 – brouillard sous l'eau (bleu)
3 - lave (rouge)
4 – brouillard défini par l'utilisateur (libre)
5 - obscurité / brouillard total (noir)

Le brouillard ou l'obscurité laissent les surfaces (sauf le ciel) et les entités s'estomper à la couleur de la cible dans la distance. La force du brouillard ou l'effet d'obscurité peut être contrôlé par le paramètre de brouillard de la vue. Dans le mode 8 bits et sur des cartes 3D faibles ou vieilles (Voodoo), le brouillard non noir ne se combine pas bien avec des ombres. Des vieilles cartes 3D montrent aussi différemment du brouillard clair sur des surfaces ombrées, non ombrées et totalement noires, qui sont visibles dans le niveau. Ainsi en employant le brouillard clair, faites attention à mettreune légère lumière dans le niveau (ambient dans propriétés de carte) qu'il n'y ait aucune surface totalement noire dans le niveau. Des cartes 3D normales n'ont aucun problème pour montrer le brouillard et des ombres en même temps.

turb_speed
Donne la vitesse de vague des textures de turbulence (par défaut 1).

turb_range
Donne l'étendue de mouvement de vague des textures de turbulence (par défaut 1).

Grâce aux variables suivantes la communication de multijoueur peut être optimisée pour le type de jeu ou pour le réseau. Les variables sont seulement pour des utilisateurs avancés qui ont l'expérience de DirectPlay - normalement ils doivent être tenus à leurs valeurs de défaut. Elles ne peuvent pas être mis directement pendant le jeux, mais peuvent être définies à une certaine valeur.

dplay_protocol
Si cette variable (par défaut 0) est redéfinie à 1, le moteur utilisera le protocole UDP de DirectPlay pour des messages fiables au lieu du protocole TCP. Le protocole DirectPlay a l'avantage de soutenir des messages fiables aussi pour des connexions modem et IPX.

dplay_servermode
Si cette variable est à 1 (défaut), le serveur fonctionnera en mode DPSESSION_MIGRATEHOST. S'il est redéfini à 2, le serveur fonctionnera en mode DPSESSION_CLIENTSERVER. S'il est redéfini à 0, aucun de ces 2 modes n'est actif.

dplay_optimize
Si cette variable est à 1 (défaut), le serveur fonctionnera en mode DPSESSION_OPTIMIZELATENCY. S'il est redéfini à 2, le serveur fonctionnera en mode DPSESSION_OPTIMIZELATENCY et NOPRESERVEORDER. S'il est redéfini à 0, aucun de ces 2 modes n'est actif. Cela réduit la bande passante dans les jeux multijoeurs mais augmente le temps de latence.

dplay_smooth

Cette variable (défaut = 0.3) place le facteur de prévision pour le système de 'compte mort' pour des extrapolations de mouvement d'entité en mode multijoueurs. Le 0.3 est l'influence de l'anticipation prévue du mouvement d'entité des valeurs reçues du serveur. Si le positionnement est 0 0, aucune prévision est exécutée du tout.
C'est pour des buts de tests seulement.

dplay_unreliable

Si cette variable (défaut = 0) est mise à 1, tous les messages sont envoyés dans un mode sans garantie. Uniquement à des fins de tests.

recbuf_size, recbuf_cycles

Ces variables donnent le nombre d'octes dans le tampon de réception en mode mulijoueurs et le ratio entre le nombre de cycle d'affichage entre le serveur et le client. Sont utilisées pour des tests et apparaissent dans la 4ème colonne du panneau d'affichage de mise au point (debug panel) sous la taille de la mémoire D3D

Les variables suivantes et des synonymes sont responsables du traitement des périphériques d'entrée :.

mouse_mode

Variable d'activation de souris. Si mise à 0 (le défaut), le pointeur de souris n'est pas visible à l'écran; si mise à 1 ou plus, le pointeur apparaît et peut être employé pour toucher ou cliquer sur des articles. Si mis à 2, les mouvements de souris ne changent pas la variable mouse_force, permettant au pointeur de souris d'être déplacé indépendamment. Si l'indicateur de souris est actif et dans une fenêtre de vue active, le taux d'affichage baissera considérablement, parce que de la puissance de calcul complémentaire est nécessaire pour détecter les entités touchées par la souris.

mouse_pos.x, mouse_pos.y

Position horizontale et verticale du pointeur de souris en pixels, par rapport au coin supérieur gauche de l'écran. Si le pointeur de souris est mis par mouse_mode, il peut être déplacé sur l'écran en changeant ces variables.

mouse_map

Synonyme Bitmap pour le curseur de souris. Le point chaud sera dans le coin gauche supérieur par défaut. Si mis à null, la souris peut toujours être active, mais est invisible.

mouse_spot.x, mouse_spot.y

Les coordonnées du point chaud du pointeur de souris en pixels par rapport au coin supérieur gauche de l'image bitmap du curseur (par défaut 0,0).

mickey.x, mickey.y

Mouvement de la souris en pixels dans le dernier encadrement.

pointer.x, pointer.y

Les coordonnées absolues de la souris en pixels; normalement employé pour mettre les variables mouse_pos.x et mouse_pos.y pour déplacer l'indicateur de souris sur l'écran. Exemple:

```
bmap arrow,<arrow.pcx>;

function mouse_toggle() { // commute la souris oui/non
	mouse_map = arrow;
	mouse_mode += 2;
	if (mouse_mode > 2) {
		mouse_mode=0;
	}
while (mouse_mode > 0) { // la déplace à l'écran
	mouse_pos.x = pointer.x;
	mouse_pos.y = pointer.y;
	wait(1);
	}
}
```

mouse_range
Rayon dans lequel clic, touche ou évènements de mise à jour peuvent être déclenchés (defaut 1000 quants).

mouse_moving
Cette variable indique si la souris se déplace (1) ou a été immobile pendant ¼ de seconde (0).

mouse_calm
Mouse_calm La distance maximale en pixels pour la considérer comme immobile pour mouse_moving (par défaut 3).

mouse_time
Temps en ticks qui est employé pour mesurer la distance mouse_calm pour déterminer mouse_moving (par défaut 4)

mouse_left, mouse_middle, mouse_right
L'état des 3 boutons de souris; 0=non appuyé, 1=appuyé.

mouse_force.x, mouse_force.y
La vitesse avec laquelle la souris est actuellement changée dans la direction verticale et horizontale, dans une étendue -1 ... +1.

mouse_ent
Le synonyme de l'entité (si disponible) touché par la souris.

joy_raw.x, joy_raw.y, joy_raw.z, joy_rot.x, joy_rot.y, joy_rot.z
joy2_raw.x, joy2_raw.y, joy2_raw.z, joy2_rot.x, joy2_rot.y, joy2_rot.z
La position brutes des axes de levier de commande, étendue -255 ... +255. Jusqu'à 6 axes peuvent être utilisés. Par une simple arithmétique l'étendue peut être changée, Exemple:

```
throttle = 50 - (50/255)*joy_raw.z; // donne à throttle une étendue 0…100
```

num_joysticks
Donne le nombre de joysticks de connecté (0,1,2).

joy_force.x, joy_force.y
Traduit, la position calibrée du levier de commande dans la direction verticale et horizontale, étendue autour de -2.0 ... +2.0. Automatiquement calibré à 0 dans la position centrale.

joy_1, joy_2, joy_3, joy_4, joy_5, joy_6, joy_7, joy_8, joy_9, joy_10
joy2_1, joy2_2, joy2_3, joy2_4, joy2_5, joy2_6, joy2_7, joy2_8, joy2_9, joy2_10
L'état jusqu'à 10 boutons du premier ou deuxième levier de commande; 0 = non appuyé, 1 = appuyé.

key_f1...key_f12 (***touches de fonctions***), key_esc (***échap***), key_tab (***tabulation***), key_ctrl, key_alt, key_shiftl (***maj gauche***), key_shiftr (***maj droit***), key_space (***espace***), key_bksp (***retour arrière***), key_cuu (***flèche vers le haut***), key_cud (***flèche vers le bas***), key_cur (***flèche vers la droite***), key_cul (***flèche vers la gauche***), key_pgup (***page préc***), key_pgdn (***page suiv***), key_home (***début***), key_end (***fin***), key_ins (***inser***), key_del (***suppr***), key_pause, key_car (***Arrêt défil***), key_cal (***virgule***), key_enter (***entrée***), key_0...key_9, key_a...key_z.

L'état des touches du clavier; 0 = non pressée, 1 = pressée. Le nom de la touche reflète la configuration physique de votre clavier et non ce qui est écrit sur les touches. Les noms donnés correspondent ici à un clavier allemand. Sur les claviers américains, le Y et le Z sont par exemple échangés. Les touches spéciales suivantes ont des noms alternatifs pour les claviers américains et allemands :

Touche	US	Allemand(autre nom)	Français	Anglais
key_grave	[~`]	[°^]	[²]	[¬`]
key_minusc	[_-]	[?\] (key_sz)	[°)]	[_-]

key_equals	[+=]	[`'] (key_apo)	[+=]	[+=]
key_brackl	[{[]	[Üü] (key_ue)	[¨^]	[{[]
key_brackr	[}]]	[*+] (key_plus)	[£$]	[}]]
key_bksl	[|\]	–	[µ*]	[|\]
key_semic	[:;]	[Öö] (key_oe)	[Mm]	[:;]
key_apos	["']	[Ää] (key_ae)	[%ù]	["']
key_slash	[?/]	[_-] (key_minus)	[§!]	[?/]

➢Vous trouverez en annexe différentes configurations de clavier selon votre pays.

key_force.x, key_force.y
: La force lissée appliquée en appuyant les touches de curseur, étendue autour-1 ... +1. Lorsque la touche [Maj] est pressée, la force est multipliée par shift_sense.

shift_sense
: facteur de multiplication de key_force lorsque la touche [maj] est enfoncée (par défaut 2).

key_any
: Cette variable prend la valeur 1, quand n'importe quelle touche, bouton de souris ou de de levier de commande est appuyé, autrement il est 0.

key_lastpressed
: Cette variable contient le scan code de la dernière touche pressée ou du dernier bouton pressé.

enable_key
enable_mouse
enable_joystick
: Si ces drapeaux ne sont mis à 0, aucune action de bouton de clavier, de souris ou de levier de commande ne sera exécutée. C'est utile, par exemple, pour fermer le clavier pendant la lecture AVI pour empêcher des menus surgissant à l'arrière-plan en frappant des touches aléatoires. Exemple:

```
play_moviefile("intro.avi");
enable_key = off; enable_mouse = off;
enable_joystick = off;
while (movie_frame != 0) {
	// permet d'interrompre l'intro en frappant n'importe quelle touche (recommendé!!)
		if (key_any != 0) { stop_movie(); }
		wait(1);
}
enable_key = on;
enable_mouse = on;
enable_joystick = on;
```

Les variables suivantes et les synonymes sont mis ou employés par des instructions :

result
: Cette variable est mise comme un code de retour par quelques instructions C-SCRIPT pour indiquer le résultat ou le succès ou l'échec.

my
: Pointeur pour l'entité attachée à l'action actuelle. Cela restera valable pendant l'action entière et toutes les fonctions commencées avec lui. Si la fonction actuelle n'a pas été attachée à une entité, ce pointeur est non défini.

you
: Le pointeur pour l'entité qui a causé ou a déclenché la fonction actuelle - cela peut être en tirant ou entrant en collision avec l'entité à laquelle est attachée la fonction ou être en créant cette entité ou en émettant cette particule. Autrement, ce pointeur est non défini. Il sera seulement valable jusqu'à la première instruction wait().

target, normal, bounce
: Vecteurs, souvent employés pour indiquer une position cible, une surface normale et une direction de rebond par plusieurs instructions. Ils seront seulement valables jusqu'à la première instruction wait().

event_type
: Variable mises au commencement d'une fonction événement d'entités pour indiquer la sorte d'événement.

in_solid, in_passable, on_passable
: Jeu de variables mises à 0 ou 1 par quelques instructions, décrivant la position actuelle des entités.

move_friction
: (valeur de 0 à 1, par défaut 0.25) détermine la friction pour les glissements le long des surfaces. A 0 il n'y a pas de friction du tout, à 1 l'entité colle à la surface et il n'y a pas de glissement du tout.

level_name
: Cette chaîne de caractères contient le nom du niveau actuel, par exemple "office.wdl".

app_name
: Cette chaîne de caractères contient le nom l'application, par exemple "office".

player_name
: Cette chaîne de caractères contient le nom du client dans un jeu de multijoueur. On peut donner cela par l'option de ligne de commande -pl; autrement un nom unique est produit automatiquement.

server_name
server_ip
: Ces chaînes de caractères prédéfinies tiennent le nom et la première adresse IP du serveur dans un jeu de multijoueur. Sur un client les chaînes de caractères sont vides. Ils peuvent être employés pour montrer l'information que les clients doivent entrer dans le champ() IP pour se connecter. Pour la connexion sur un réseau local le nom de serveur peut être employé; pour la connexion sur Internet l'adresse IP ou le nom de domaine doivent être employés. Exemple:

```
ifdef SERVER;
while (connection == 0) { wait(1); }
str_cpy(temp_str,server_name);
str_cat(temp_str," has the IP: ");
str_cat(temp_str,server_IP);
scroll_message(temp_str);
endif;
```

session_name
: Contient le nom de la session courante en mode multijoueurs.

connection
: Sa variable est mise à 0 au début de jeu. Aussitôt qu'une connexion de multi joueur est établie, elle est automatiquement mise à 1, 2, or 3, dépendant du moteur fonctionnant dans le serveur, le client ou en mode de client/serveur.

Predefines (Préféfini)

Les définitions suivantes peuvent être utilisées dans un scénario C-SCRIPT. Elles sont parcourues avant dêtre compilées par le précompilateur pour générer des modèles de base pour nos applications et les futures éxécution de nos scénarios:

include ***<filename>;***

Lit un scénario C-SCRIPT complémentaire à partir du fichier donné et continue ensuite à parcourir le fichier original C-SCRIPT. De cette façon des scénarios C-SCRIPT prédéfinis peuvent être inséré.. On peut donner un maximum de 40 fichiers C-SCRIPT par include lorsqu'on produit le jeu par publish. Le nombre de include n'est pas limité lorsque le jeu est généré par resource.

bind ***<filenam***e>;

le fichier donné sera inclus dans la ressource de jeu par l'emballeur de ressource. Seulement pour buts spéciaux ; tous les fichiers entre les parenthèses angulaires seront inclus. Vous pouvez donner n'importe quel nombre de fichiers liés.

savedir ***"dirname";***

Nomme le répertoire par défaut pour la sauvegarde des jeux et des copies d'écran. Si ce dossier n'existe pas, il sera d'abord créé puis le jeu sera sauvegardé. Notez que l'on doit donner des antislashs ("\") dans la notation C ++, c'est-à-dire comme des antislashs doubles (par exemple " C: \\ masauvegarde "). Le nom de dossier donné peut ici être ignoré par l'option de ligne de commande -dir.

path ***"dirname";***

tous les fichiers complémentaires - bitmaps, sons ou entités - seront d'abord cherchés dans le dossier réel et ensuite dans le chemin donné ici (de nouveau, on utilisera les antislashs comme" \\ "). Vous pouvez spécifier jusqu'à 16 noms de chemin, qui seront recherchés dans l'ordre donné.

resource ***"resourcename";***

Comme pour la déclaration path mais donne le nom d'un fichier de ressource .wrs duquel des fichiers supplémentaires pourront être lus. Les fichiers de ressources sont balayés dans l'ordre dans lequel ils sont déclarés. De cette façon les fichiers appartenant à un jeu peuvent être coupés en plusieurs ressources. Une autre possibilité consiste à distribuer un pack de scénario ou de modèles ou … à d'autres utilisateur de GameStudio qui pourront les utiliser mais qui ne pourront pas les extraire et les éditer. Ces ressources peuvent être lues par toutes les éditions de GameStudio mais ne peuvent être crées que par la version professionnelle. Exemple :

```
resource "models.wrs";
resource "bmaps.wrs";
resource "warlock.wrs";
```

Un fichier de resource séparé peut être créé par un script WDL constitué de déclarations bind ou d'autres qui donnent le nom entre parenthèses angulaires <> comme ceci :

```
//resource distribuable pour le modèle warlock
bind <warlock.mdl>;
bind <staff.mdl>;

action warlock {
	my.fat = off;
	my.narrow = on; // set narrow hull
	my._walkframes = 1; // enable frame name animation
	my._force = 0.5; // He should be not too fast
	my._movemode = _mode_walking;
	my.__jump = on;
	my.__duck = on;
```

```
    my.__strafe = on;
    my.__trigger = on;
    player_move();
    ent_morph(me,"warlock.mdl");
    ent_create("staff.mdl",nullvector,staff_prog);
}
```

Pour créer de telles ressources ouvrez un nouveau projet dans WED, assignez lui le script comportant les instructions **bind** dans Map Properties et exécutez la fonction ressource. Le fichier .WRS est créé dans le sousrépertoire .CD. Dans le script original, les fichiers contenus dans les diiférentes ressources doivent être donnés entre "" à la place de <> pour éviter qu'ils soient inclus dans la ressource principale. Si des fichiers avec le même nom sont contenus dans différentes ressources, la priorité de ces fichiers dépend de l'ordre de leur déclaration **resource**, commençant avec la ressource principale. Notez que les noms des fichiers contenus dans des ressources doivent être au format DOS compatible (max 8+3 caractères).

print ***text;***

Affiche le texte dans la fenêtre de démarrage du moteur durant la compilation du script (moteur de développement uniquement). De cette façon on peut suivre à la trace l'évolution de la complilation. Exemple :

```
print version_1;
```

Affichera ...version_1.... Entre les points dans la fenêtre de démarrage

define ***nam***e;
define ***name, replacemen***t;

Defines peut aussi être donné par l'option de ligne de commande **-d**. La deuxième ligne vous permet de rebaptiser des noms et des paramètres dans le fichier C-SCRIPT; la première est pour l'inclusion ou l'exclusion de lignes C-SCRIPT selon des conditions (voir **ifdef** ci-dessous). Chaque fois que le nom (**name**) apparaît dans le scénario C-SCRIPT après l'instruction **define**, il sera remplacé par **replacement**, qui peut être un autre nom ou un nombre ou rien. Le changement de nom rend les fonctions plus 'lisibles', par exemple en donnant quelques noms significatifs aux 40 variables / paramètres à but général des entités (**skill**xx). Exemple:

```
define mynumber,-123;
define damage,skill3; //damage (dommage) est plus parlant que skill3 !!!
action kill {
    result = my.damage;
    my.damage = mynumber;
}
```

Les caractères comme "{}; () < >" ne peuvent pas être redéfinis et **defines** ne peut pas être emboîté.

undef ***nam***e;

Un nom défini par **define** peut être 'non défini' par **undef** pour toutes les lignes du scénario C-SCRIPT qui suivent.

ifdef ***nam***e; ifndef ***nam***e; ifelse; endif;

Les noms définis peuvent être employés pour sauter certaines lignes de scénario C-SCRIPT dépendant d'un précédent **define** ou - particulièrement - de l'option de ligne de commande **-d**. Toutes les lignes de scénario entre **ifdef** et **endif** sont sautées si le nom n'a pas été défini auparavant. Toutes les lignes entre **ifndef** et **endif** sont sautées si le nom a été défini auparavant. L'instruction **ifelse** change complètement la ligne à sauter ou à ne pas sauter. De cette façon, vous pouvez 'inventer' des nouvelles options de ligne de commande pour le moteur, qui a un effet arbitraire sur le jeu. Exemple:

```
define hires_d3d; // ou démarre avec l' option -d hires_d3d
...
ifdef hires_d3d;
var video_mode = 6; // 640x480 haute resolution
var video_depth = 16; // mode couleur 16 bits
ifelse;
var video_mode = 2;} // 320x240 basse resolution
var video_depth = 8; // mode couleur 8 bits
```

```
endif;
```

Les noms suivants sont définis au début de l'application:

develop
: Prédéfini dans la version de développement du moteur et non défini dans la version publiée qui est généré par les fonctions WED Publish ou Resource. De cette façon le panneau de mise au point peut-être affiché sous conditions comme ceci :

```
Ifdef develop;
	Set_debug();
Endif;
```

server
: Defini si le moteur est démarré en tant que serveur (commande ligne option-sv).

client
: Defini si le moteur est démarré en tant que client (commande ligne option-cl).

caps_color
caps_flare
caps_shadow
: Definisse si le moteur supporte les lumières colorées (color), les signaux lumineux (flare) ou les ombres (shadow) Elles peuvent être employées pour des fonctions universelles ou être éxécutées sur des éditions différentes d'une façon différente. Exemple:

```
IFDEF CAPS_FLARE; // Edition Extra ou plus
	create(<shadspr.pcx>,my.x,move_shadowflare);
IFELSE;
	create(<shadflat.pcx>,my.x,move_shadowflat);
ENDIF;
```

direct3d
: Defini si le moteur s'éxécute en mode D3D. Peut être utilisé pour charger différentes images bitmaps ou niveaux en fonction du mode. Exemple:

```
IFDEF DIRECT3D;
	bmap panel_map = <map24bit.pcx>; // bitmap pour le mode d3d
IFELSE;
	bmap panel_map = <map8bit.pcx>; // bitmap pour le mode logiciel
ENDIF;.
```

Database and Dataview (base de données et vue de la base)

Par des bases de données de texte externes, les chaînes de caractère peuvent être chargées dépendant du fichier texte externe (l'édition professionnelle seulement). Dans des versions futures, des variables, des images bitmaps, des sons et des entités pourront être chargés aussi. Une telle base de données est définie par les noms suivants :

P database ***name*** { ... }
: Définit une base de données. La base de données est copiée en enregistrements (rangées) et en champs (colonnes) comme une table de tableau. Actuellement tous les champs de la base de données doivent être des chaînes de caractère ASCII ; dans des versions futures ils pourront aussi contenir des nombres ou des noms de fichier. On peut donner les noms suivants dans les accolades de la définition de base de données :

datafile ***<filenam***e>;
: Fichier de données ; le nom du fichier de base de données, qui contient les données de texte formaté en ASCII dans des champs et des enregistrements.

separator ***"***c";
: Caractère ASCII, qui sépare les champs de chaque enregistrement (par exemple ";" - défaut : ","). Les enregistrements eux-mêmes sont séparés par des retours à la ligne.

string ***name;***
: Définit un nom pour un champ qui contient une chaîne de caractères (actuellement le seul type des champs soutenu). Tous les champs d'un enregistrement doivent être définis de cette façon, dans le bon ordre.

Pour choisir ou non certains enregistrements d'une base de données, dataview est employé :

P dataview ***name*** { database ***nam***e; }
: Crée une structure de sélection pour une une base de données avec le nom donné. Pour chaque enregistrement de la base de données, un drapeau interne dit si l'enregistrement est choisi ou non. Par défaut tous les enregistrements sont choisis. Un nombre arbitraire de dataviews peut être employé pour la même base de données.

Dataviews peut être assigné aux objets de texte pour montrer certaines chaînes de caractère d'une base de données. Vous pouvez employer les instructions select, unselect, and_select et or_select pour déterminer quelles rangées de la base de données seront montrées et celles qui ne le seront pas.

Par exemple, un fichier data.txt contient les trois lignes de texte suivantes:

```
Smith,John,New York
Mueller,Karl,Hamburg
Manchu,Fu,Alma Ata
```

Vous pouvez utiliser les objets database et dataview pour afficher les chaînes de caractères sélectionnées, et l'instruction select pour sélectionner certains enregistrements :

```
database people_db {
	datafile <data.txt>;
	separator ",";
	string "Name";
	string "Firstname";
	string "City";
}

dataview people_view { database people_db; }

text city_txt {.
```

```
	font standard_font;
	dataview tst_view;
	viewpos 2;
	flags visible;
}

string select_str "John";

function select_people() {
select people_view,"firstname",select_str;
}.
```

Multiplayer Applications (applications multi joueurs)

Avec l'édition commerciale et professionnelle GameStudio vous pouvez connecter 4 (commerciale) ou un nombre arbitraire (professionnelle) de PC par un réseau local ou Internet ou deux PC par un modem ou une connexion série. Dans une configuration multi utilisateurs un des PC est serveur (server), les autres sont clients.

Le serveur se charge du monde de jeu, tandis que les clients sont assignés aux joueurs. Le serveur transmet automatiquement les positions d'entité et d'autres propriétés aux clients. De cette façon chaque client a une image cohérente de l'état actuel du niveau. C'est comme si vous aviez plusieurs vues sur un PC simple. Chaque client lui-même peut aussi transmettre certains événements au serveur, pour créer et contrôler une ou plusieurs entités dans le monde de jeu ou pour d'autres buts, comme la causerie ou l'échange de messages.

Un système multiposte doit contenir un serveur et au moins deux clients. Il est possible de diriger un serveur et un client simultanément. Donc la configuration multiposte minimale est deux PC pour deux joueurs qui jouent ensemble l'un contre l'autre. Le serveur a la plupart du travail à faire et doit donc être la machine la plus rapide. Le même scénario C-SCRIPT doit être chargé sur tous les PC. Dans les scénarios C-SCRIPT le nom server est automatiquement prédéfini sur le serveur et client sur les clients. De cette façon, des scénarios différents peuvent être exécutés sur le serveur et sur les clients via les déclarations ifdef server; ou ifdef client ; Cependant un grand soin doit être pris ici : le nombre d'éléments comme des fonctions, des variables, des sons etc. doivent bien sûr être les mêmes sur le serveur que sur les clients.

Un nombre arbitraire d'applications multi utilisateur - nommé sessions – peuvent fonctionner sur le même réseau local ou la même adresse Internet. Un client se connecte toujours à la session qui éxécute le même scénario C-SCRIPT. De cette façon un serveur de jeu de multi-niveau peut être réalisé.

Le serveur doit être lancé en premier, en employant l'option de ligne de commande -sv. Une fois le serveur lancé, les clients peuvent se connecter (option de ligne de commande -cl). Les clients peuvent se connecter et partir à tout moment. En donnant les deux -sv et -cl le moteur est démarré comme serveur et comme client simultanément. Aussitôt que la connexion est établie, la variable prédéfinie connection est mise à 1, 2, or 3, dépendant du moteur s'éxécutant comme serveur, client ou les deux.

Par défaut, le protocole de TCP/IP, nécessaire pour les connexions Internet, est employé. D'autres types de connexions par le protocole de réseau IPX, un modem ou un câble série sont aussi possibles. Dans des connexions TCP/IP chacun a besoin pour démarrer le jeu du nom du domaine du serveur ou le nom du réseau ou l'adresse IP pour se connecter. Le nom de réseau et l'adresse IP du serveur sont contenus dans les chaînes de caractère prédéfinies server_name et server_ip.

Chaque client a un nom individuel, que l'on peut donner à l'option de ligne de commande -pl. Autrement un nom unique est créé par le moteur. A la connexion de chaque client, l'événement prédéfini on_server est déclenché sur le serveur, avec event_type de mis à event_join et un jeu de synonyme chaîne de carctères remet le nom du client comme argument à la fonction d'événement. Ainsi le nom de chaque client joint peut être indiqué sur le serveur, comme cela :

```
function server_event(str)
{
	if (event_type == event_join)
	{
		str_cpy(temp_str,str); // contient le nom du client
		str_cat(temp_str," vient de nous joindre");
		scroll_message(temp_str);
		return;
	}
}.
```

Le statut actuel du monde est transféré à chaque client qui nous joint et sera mis à jour après chaque frame du serveur. La mise à jour avec plus de 20 encadrements par seconde gaspillera seulement un peu de bande passante, aussi le taux d'encadrement d'un serveur doit être limité par fps_max. Une vue d'ensemble du protocole de multi joueur peut

être trouvée dans le Manuel du Programmeur GameStudio. Le protocole emploie un algorithme d'éstimation qui prévoit la vitesse et l'accélération des entités. Cela 'lisse' les mouvements d'entité sur des systèmes de client/serveur lents avec un haut temps de latence. Sur des connexions de TCP/IP, le protocole TCP fiable est employé pour des messages essentiels comme la jonction ou la transmission de variables et des chaînes de caractères, tandis que le protocole UDP plus rapide mais moins fiable est employé pour des messages moins importants comme des mises à jour d'entité. La taille de paquet est limitée à 1400 octets, parce que les routeurs ou autres équipements sur l'Internet retardent souvent les plus grands paquets UDP. Toutes ces mesures font que les applications multijoueur de GameStudio sont très rapides et fiables.

Les variables et les chaînes de caractères ne sont pas transférées automatiquement. Ainsi elles peuvent contenir des valeurs différentes sur le serveur et sur les clients, si elles ne sont pas synchronisées exprès par les instructions send_var ou send_string. Les instructions send_string et execute peuvent aussi être employées pour envoyer des commandes arbitraires C-SCRIPT, comme pour le changement de niveau, du serveur vers les clients.

Si une entité a été créée par un client par l'instruction create, elle apparaîtra sur le serveur et ses fonctions principale (main) et d'événement seront éxécutées seulement là. Mais il se souviendra de son créateur et s'il envoie des variables d'entité par l'instruction send, il recevra les variables de ce client. De cette façon chaque client peut créer et contrôler sa propre entité de joueur sur le serveur. Si le client sedéconnecte, il ne peut pas envoyer de nouvelles intentions, donc son entité de joueur restera immobile. event_disconnect peut alors être employé pour enlever l'entité.

Le niveau office est préparé pour le mode multijoueur. Pour l'essayer, reliez deux ou plusieurs PC via un réseau local avec le protocole IPX. Commencez une session sur le plus rapide des PC en mode client/serveur (-sv -cl). Aussitôt que la session est lancée, démarrez les clients (-cl). Si un client est connecté, son entité apparaîtra dans le niveau d'office toujours à la même position (donnée dans la fonction principale), un peu derrière la position du joueur normal. Donc il doit s'éloigner un peu aussitôt que possible pour empêcher les nouvelles entités de client de se coller les unes sur les autres. Dans un vrai jeu multijoueur cela doit être traité par un scénario. Le combat de multijoueur et le port d'arme sont aussi possibles, mais non encore mis en oeuvre dans les scénarios de calibre (template). [F4] active un mode de bavardage pour l'envoi de messages aux autres joueurs.

Starter Window Definitions (définition de la fenêtre de démarrage)

Avec l'édition professionnelle, vous pouvez changer le comportement du moteur avant même le début de niveau et après la fin du jeu. Par défaut, au démarrage une petite fenêtre sera montrée, qui donne quelques messages système et offre un bouton pour le premier arrêt. Cependant, vous pouvez créer votre propre fenêtre, avec des boutons différents, des images, des messages de bienvenue, le logo de votre société ou une barre de progression. Une telle fenêtre est définie de cette façon dans le scénario C-SCRIPT :

P window winstart { ... }
ou
P window winrun { ... }
ou
P window winend { ... }

La première ligne définit la fenêtre de départ, la deuxième la taille de la fenêtre de jeu, la troisième la fenêtre qui apparaît après la fin du jeu. Si la fenêtre winrun n'est pas définie, la résolution vidéo initiale sera prise comme la taille. Si la fenêtre winend n'est pas définie, il n'y aura aucune fenêtre de fin de montrée.

Dans les accolades, on peut donner les noms suivants :

TITLE ***"tex***t";
Texte qui apparaît dans la barre de titre de la fenêtre.

SIZE x, y;
La taille de la fenêtre en pixels. Si on ne donne pas la taille, la résolution vidéo initiale sera employée.

MODE ***nam***e;
Mode de fenêtre. Un de ceux là standard, image, fullscreen ou use_oldstyle.

BG_COLOR rgb***(***r, g, b);
Couleur de fond de la fenêtre; ***r, g, b*** = 0..255.

SET FONT ***"fon***t", rgb***(***r, g, b);
Pour donner une fonte et une couleur pour le nom text suivant. La fonte est une fonte Windows. Si elle n'est pas disponible ou si on donne "", la fonte standard est prise.

Text ***"tex***t", x, y;
Production de texte simple à la position donnée. On peut donner des retours à la ligne dans le texte "\" .

TEXT_STDOUT ***"fon***t", rgb***(***r, g, b), x, y, ***widt***h, ***height***;
Champ de texte déroulant pour la production standard de moteur.

FRAME ***styl***e, x, y, ***widt***h, ***height***;
Un encadrement à la fenêtre. Pour ***style*** les noms ftyp1, ftyp2, ftyp3 ou ftyp4 peuvent être utilisés.

BG_PATTERN ***<filenam***e>, opaque;
Remplit en entier le fond de la fenêtre avec le motif donné par le nom de fichier (fichier .pcx ou .bmp, 8 bits seulement).

PATTERN ***<filenam***e>, opaque, x, y, ***widt***h, ***height***;
Remplit la zone donnée avec le motif de fond donné par le nom de fichier (fichier .pcx ou .bmp, 8 bits seulement).

PICTURE ***<filenam***e>, opaque, x, y;
Place une image à la position donnée.

PROGRESS rgb(r, g, b), ***element***s, x, y, ***widt***h, ***height***;

Montre une barre de progression. Elements correspond au nombre de points que vous voulez montrer pendant la production standard de moteur.

Les définitions de fenêtre deviendront seulement valables lorsqu'un fichier starter aura été créé par WED.

Reference: Prefabricated Scripts (Les scénarios prédéfinis)

La plupart des fichiers de scénario suivants sont automatiquement inclus dans chaque nouveau scénario. Donc toutes les fonctions ci-dessous sont disponibles pour WED aussi bien que pour vos propres scénarios, tant que vous n'effacez pas les lignes include. Les fichiers sont placés dans le répertoire template. Si vous voulez les changer pour vos propres buts, copiez-les juste dans votre dossier de travail auparavant. Les fichiers trouvés dans le dossier actuel seront employés avec la priorité sur ceux avec le même nom dans le dossier de référence template.

Au lieu de d'attacher directement une action à une entité, vous pouvez combiner plusieurs actions d'entité le de base (si elles sont d'une sorte différente et n'interfèrent pas ainsi les unes avec les autres) en les commençant d'une action principale. Attachez alors cette action principale pour composer un acteur de plusieurs comportements. Vous pouvez aussi mettre les variables d'entité et les drapeaux par cette action, au lieu du panneau de propriété de WED. Exemple:

```
action robot1 {
	my._walkframes = 1;
	my._entforce = 0.7;
	my._firemode = damage_explode + fire_ball + hit_explo + bullet_smoketrail;
	patrol(); // met le comportement de mouvement
	drop_shadow(); // met l'ombre
	actor_fight(); // metle comportement de combat
}
```

Au début de chaque fichier de scénario, sont d'abord définis les fichiers de son ou des graphiques nécessaires (tous placés dans le dossier template) et ensuite les valeurs de départ de quelques variables qui peuvent avoir une influence sur l'intérieur des fonctions. Vous pouvez changer les deux dans votre scénario C-SCRIPT principal, sans changer le fichier de scénario prédéfini. Pour changer des variables ou les redéfinir dans votre fichier C-SCRIPT principal après la liste include ou leur donner une valeur différente au commencement de votre jeu (fonction main()).

Movement.wdl

Ce fichier contient les actions de base pour le mouvement de joueur.

player_move()

Cette action fera de l'entité à laquelle il est assigné un joueur. Il peut marcher, conduire, mitrailler, se tourner, regarder en haut et en bas, sauter, se baisser, ramper, avancer, nager et plonger. La gravité est appliquée et l'entité peut monter un escalier. Sur une pente assez raide, elle sera tirée en bas par la gravité. Si la pente est trop raide, elle sera poussée en arrière pour qu'elle ne puisse pas y monter. Si elle entre dans de l'eau, elle commencera à nager.

Pour tester cela, en appuyant sur [0] le joueur contrôlera l'entité comme une caméra; le fait de presser [0] de nouveau, déplace la caméra hors de l'entité (version de développement seulement; le contrôle de caméra direct prédéfini est mis hors de service dans le jeu publié). La pression sur la touche [F7] bascule entre mode première et troisième personne. Le synonyme player est mis à l'entité, pouvant être utilisé dans d'autres fonctions. Soyez prudent pour ne pas assigner cette action à plus d'une entité!

L'origine des entités doit être placée assez haut pour être capable de monter à l'escalier. L'animation modèle peut contenir des noms d'encadrement commençant par :

"stand"	- animation d'attente debout
"walk"	- animation de marche
"run"	- animation de course
"crawl"	- animation rampante
"swim"	- animation nageante
"dive"	- animation plongeante
"duck"	- animation se baissant

"jump" - animation sautant

le nombre d'encadrements par cycle d'animation peut être arbitraire. L'entité passera de la marche à la course à une vitesse de plus de 12 quants par tick. En pressant [Début] et [Fin] on amorce le fait de se baisser ou de sauter tant que l'on est sur la terre ferme et une rotation verticale pour la plongée en bas tant qu'on est dans l'eau. Si l'entité est sous le brouillard d'eau, le brouillard bleu (numéro 2 pour fog) est activé.

Vous pouvez changer quelques particularités en mettant les variables suivantes et les drapeaux dans le panneau d'entité. Ces mêmes variables et drapeaux peuvent aussi être mis par des fonctions, permettant au joueur de changer son comportement de mouvement pendant le jeu. Les déclarations define sont employées pour donner des noms significatifs aux variables :

_walkframes (skill1)
Doit être mis à 1 pour permettre l'animation avec des noms d'encadrement.

_entforce (skill5)
Le facteur de force, détermine la vitesse des entités; défaut = 1.

_banking (skill6)
Le facteur de bord, détermine l'angle de rouleau (roll) dans des courbes. Peut être mis à une valeur négative (la moto, l'avion) ou une valeur positive (la voiture) ; défaut = 0.

_movemode (skill7)
Mode de mouvement ; _mode_walking (1), _mode_driving (2), _mode_swimming (3) sont disponibles.

__fall (flag1)
Si mis à 1, le joueur peut prendre des dégâts en tombant. S'il tombe, son temps de chute est calculé. Il ne sera pas endommagé si son temps de chute est alors inférieur à 5 ticks. Autrement, une quantité de (10 + int ((my_falltime - 5) * 1.75)) sera soustraite de sa valeur _health (skill9).

__wheels (flag2)
Prevent turns without moving, like a vehicle on wheels.

__slopes (flag3)
Adapte l'angle d'inclinaison (tilt) et de roulis (roll) à la pente du sol, comme un véhicule sur des roues.

__jump (flag4)
Autorise le saut lorsqu'on appuie sur la touche [Début].

__bob (flag5)
Hochement de tête pendant la marche.

__strafe (flag6)
Valide le mitraillage en utilisant les touches [,] et [.].

__trigger (flag7)
Met la variable trigger_range du joueur pour exploiter automatiquement les portes et les plateformes lorsqu'il s'en approche.

camera_move
Cette fonction fait que le joueur prend le contrôle direct de la vue de caméra, indépendante de l'acteur joueur. [PgUp] / [PgDn] changeront l'angle d'inclinaison, [Alt+Curseur gauche] et [Alt+Curseur Droit] changeront l'angle de rouleau.

client_move

Cette fonction doit être lancée sur le client dans un jeu de multi joueur, avant ou après la création de l'entité de joueur. Il rassemble les forces de mouvement, entrées par le clavier du client ou la souris et les envoie à l'entité de joueur sur le serveur qui a été créé sur le client.

drop_shadow
Cette fonction peut être employée pour attacher un sprite d'ombre plat à une entité. Le sprite d'ombre par défaut est simplement une ellipse semi-transparente sombre (shadow.pcx) projetée sur le sol horizontal, mais cela peut être changé par l'utilisateur. L'ombre apparaîtra aussi longue que le _movemode (skill7) de l'entité si sa valeur est non-zéro et l'entité n'aura pas d'ombre (drapeau shadow).

_player_intentions
Cette fonction parcourt le clavier, la souris et le levier de commande et calcule le vecteur force (pour le mouvement du joueur) et aforce (pour la rotation de joueur). Pour la nouvelle définition des touches ou le changement du comportement de mouvement, une fonction alternative _player_intentions peut être entrée dans le fichier C-SCRIPT principal.

move_view
Cette fonction place la vue de caméra à la position de l'entité joueur, dans le mode 1ère ou 3ème personne, dépendant du contenu de la variable person_3rd.

attach_entity
Peut être employé pour la création des entités qui sont attachés à l'entité my.

Actors.wdl

Ce fichier de scénario contient les fonctions de base pour les mouvements de l'acteur NPC (NPC = Non player Character c'est à dire personnage qui n'est pas le joueur et que nous n'avons donc pas sous contrôle direct).

patrol
L'action de patrouille laisse une entité modèle se promener selon un chemin entre des positions de départ, employant les mêmes variables et drapeaux que l'action player_move. Au début de jeu, l'entité cherchera la position de départ la plus proche pour le prochain départ dans un rayon de 2000 quants. Il marchera à cette position. Quand il sera arrivé, il cherchera la position suivante dans une direction approximative à celle indiquée par la première position de départ. Ce sera répété jusqu'à ce qu'il ne trouve plus désormais de position de départ. Alors l'entité s'arrêtera.

Pour faire un chemin circulaire pour l'entité, placez quelques positions de départ dans une distance inférieure à 2000 pas l'une de l'autre. Ajustez les angles pour que chaque position pointe sur la suivante et que le dernier pointe de nouveau sur la position du premier.

patrol_path
L'action de patrouille laisse un modèle d'entité se promener sur un chemin fermé, employant les mêmes variables et drapeaux que l'action player_move. Au début du jeu, l'entité cherchera le plus proche chemin dans un rayon de 1000 quants. Il marchera vers le premier but. Quand il sera arrivé, il marchera vers le but suivant et cetera. Regardez cette action pour avoir une idée de comment écrire une fonction de mouvement pour une entité.

actor_turnto(***angl***e)
Cette fonction fait tourner une entité modèle horizontalement vers l'angle cible donné.

actor_move
Cette fonction laisse une entité modèle se déplacer en avant à une vitesse dépendant du vecteur force. La montée de marche, aussi bien que l'animation qui passe de la marche à pied à la course seront traitées.

Weapons.wdl

Ce fichier de scénarios contient les fonctions de base pour porter des armes, tirer avec et gérer les munitions.

gun

L'action gun fait qu'une entité devient arme. Elle peut être prise en la touchant, en cliquant avec la souris ou en pressant la barre [Espace] lorsqu'on est tout près. Si elle est prise, elle sera assignée à la vue de caméra via la fonction carry (portant). Les différents types d'armes à feu peuvent être choisis en appuyant les touches numériques. Si la touche [Crtl] ou le bouton de souris gauche sont appuyés, l'arme tirera. Afin que l'embouchure de feu apparaisse à la bonne place, le centre de l'arme doit être à l'intérieur de son baril.

L'action gère aussi le transfert d'armes à feu dans un autre niveau. Dans chaque niveau, tous les modèles d'arme à feu qui sont déjà prises par le joueur doivent être placés quelque part. L'action d'arme à feu vérifie si le joueur a pris une arme à feu avec le même numéro d'arme auparavant et si c'est le cas la transfère automatiquement comme étant en possession du joueur au début de niveau.

Des armes à feu individuelles peuvent être construisent en mettant les drapeaux et les variables suivants et ensuite en appelant l'action gun :

__rotate (flag1)

Si ce drapeau est mis, le modèle d'arme à feu tourne avant d'être pris.

__silent (flag2)

Si ce drapeau est mis, le message de prise d'arme à feu sera supprimé.

__bob (flag5)

Si ce drapeau est mis, l'arme à feu se balance un peu si la caméra se déplace. La période de balancement est la même que celle employée pour le hochement de tête dans movement.wdl. Si le drapeau n'est pas mis, l'arme à feu ne bouge pas.

__repeat (flag7)

Si ce drapeau est mis, c'est une mitrailleuse; autrement une arme à feu d'action simple.

skill1 ... skill3

La position avant, droit et bas de l'arme à feu, quand on la porte, en quants par rapport à la caméra.

_ammotype (skill4)

type de Munitions (1 .. 7) dont ont besoin les armes à feu. Si 0, l'arme à feu ne consomme pas de munitions. La partie après la décimale, multiplié par 100, donne la quantité de munitions à ajouter quand l'arme à feu est prise. Par exemple 2.30 = type de munitions 2 et 30 sont déjà dans l'arme à feu.

_bulletspeed (skill5)

Vitesse de la balle, defaut = 200 quants / tick. Si l'arme à feu n'émet pas de particules ou aucune balle du tout, sa portée en quants est deux fois la vitesse de la balle. La partie après la virgule multipliée par 100, donne la force de recul. Si 0, il n'y aura pas de recul. Si supérieure à 0, le recul est fait en glissant en arrière de la quantité calculée en quants. Si inférieure à 0, alors le recul est fait en balançant vers le haut de la quantité calculée en degrés.

_weaponnumber (skill6)

Le numéro de l'arme determiné par la touche ([1]–[7]) que l'on presse pour sélectionner l'arme à feu, si elle a été prise auparavant.

_firetime (skill7)

Le temps (en ticks) nécessaire à l'arme pour se recharger. Inclus le temps pour le recul de l'arme et l'animation si existent.

_firemode (skill8)

Mode de feu et effet de dégâts de l'arme à feu. Il peut être composé en ajoutant les nombres suivants qui sont définis comme des noms :

damage_shoot (1) - Les dommages sont appliqués par event_shoot, sans balles.
(dégats de tir)
damage_impact (2) - Les dégâts sont appliqués par l'event_impact de la balle.
(dégats d 'impact)
damage_explode (3) - Dégâts à plusieurs cibles par event_scan au point d'impact.
(dégats d'explosion)
fire_particle (4) - l'arme à feu produit une traînée de vapeur vers la cible.
(particules de tir)
fire_ball (12) - coups de feu oranges avec lumière radiante.
(traînée de balle)
fire_rocket (16) - Tire une rocket <rocket.mdl> qui laisse une trainee de fumée.
(tir de rocket)
fire_laser (20) - un rayon laser <lbeam.mdl> qui est émis vers la cible.
(rayon laser)
bullet_smoketrail (32) - les coups de feu laissent des traînées d'étincelles.
(traînée d'étincelles)
hit_flash (128) - il y aura un flash au point d'impact.
(flash au point d'impact)
hit_explo (256) - il y aura une explosion au point d'impact.
(explosion au point d'impact)
hit_smoke (512) - un nuage de fumée montera au point d'impact.
(fumée au point d'impact)
hit_scatter (1024) - points d'impacts multiples comme de la chevrotine.
(éparpillement au point d'impact)
gunfx_brass (2048) - l'arme à feu éjecte des douilles <gbrass.pcx>.
(èjection des douilles)
hit_sparks (8192) - production d'une fontaine d'étincelles au point d'impact.
(étincelles au point d'impact)
hit_hole (16384) - un trou de balle <bulhole.pcx> dans le mur .
(trou de balle)

La partie après la décimale, multipliée par 100, donne la quantité de dégâts que la balle produit (défaut = 10). Si la balle explose, le rayon d'explosion est cinq fois la valeur de dégâts.

_gun_source_x (skill11)
_gun_source_y (skill12)
_gun_source_z (skill13)
definissent la position d'origine du coup de feu. Si _gun_source_x == 0, alors on utilise la position par défaut (vecteur gun_muzzle).

Exemple:

```
action launcher { // construit des specifications pour l'arme
	my._bob = on; // balencement
	my._repeat = on; // mitrailleuse
	my._ammotype = 3.20; // munition type 3, 20 pièces
	my._bulletspeed = 100.05; // vitesse de la balle 100, recul 5 quants
	my._weaponnumber = 4; // touche 4 pour la sélectionner
	my._firetime = 5; // environ 3 coups à la seconde
	// explosion de la balle avec étincelles, dommage 40, rayon de l'explosion 200 quants
	my._firemode = damage_explode+fire_ball+hit_explo+bullet_smoketrail+0.40;
	gun();
}
```

ammopac

Cette action fait que l'entité est un paquet de munitions. Il peut être pris en le touchant, en cliquant avec la souris ou en pressant la touche [Espace] lorsqu'on est tout près. Les propriétés du paquet de munitions sont mises par les drapeaux et les variables suivants :

__rotate (flag1)
Si ce drapeau est mis, l'entité tourne avant d'être prise.

__silent (flag2)
Si ce drapeau est mis, le message indiquant que l'article est pris sera supprimé.

_ammotype (skill4)
Type de munitions (1..7).

skill5
Quantité de munitions à ajouter en prenant le paquet.

medipac
Cette action fait que l'entité est une trousse de premier secours. Elle peut être prise en la touchant, en cliquant avec la souris ou en pressant la touche [Espace] lorsqu'on est tout près. Les propriétés de la trousse de premier secours sont mises par les drapeaux et les variables suivants :

__rotate (flag1)
Si ce drapeau est mis, l'entité tourne avant d'être prise.

__silent (flag2)
Si ce drapeau est mis, le message indiquant que l'article est pris sera supprimé. _rotate (flag1)

skill5
Le nombre de points de vie à jouter lorsque la trousse de premier secours est prise.

bullet_shot
Cette action peut être attachée à une entité de balle lors de sa création. Elle dirige la balle dans la direction donnée par le vecteur shot_speed. Les effets dedégâts que la balle fait au moment de l'impact sont donnés par la variable damage. Ces effets - explosion ou pas, traînée de fumée ou pas et cetera – sont donnés par la variable fire_mode, qui accepte les mêmes valeurs que la variable skill8 d'une arme à feu (voir l'arme à feu – gun -).

pan_cross_show
pan_cross_hide
Montre ou cache un reticule au centre de l'écran.

War.wdl

Ce fichier de scénario contient les fonctions de base pour des batailles entre le joueur et les acteurs.

actor_fight
Cette action peut être assignée à une entité qui se bat avec le joueur employant une arme à feu. Il observe le joueur, l'attaque, gère son armure et sa santé, esaye de s'échapper et meure.

Les cycles de l'animation du modèle doivent contenir des encadrements pour l'animation de l'attaque ("attack") et l'animation de la mort ("death"). La plupart des variables et drapeaux ont la même signification que player_move, avec les exceptions suivantes :

_hitmode (skill6)

Définit le comportement en réception de coups et en mourant. Actuellement on peut seulement donner les valeurs 0 et 256, le dernier signifiant que l'entité éclatera après avoir joué son animation mourante et sera fracassée en plusieurs <gibbit.mdl> qui voleront en l'air et retomberont sur le sol.

_firemode (skill8)
Définit le mode de feu de l'arme à feu des entités et les dégâts qu'un coup produira. On peut donner les mêmes firemodes que pour les armes à feu (gun).

_health (skill9)
Quantité de santé Initiale de l'entité. Diminue sur des coups si l'entité n'a aucune armure. Si cette valeur atteint zéro, l'entité meurt.

_armor (skill10)
Quantité d'armure initiale de l'entité. Diminue sur des coups. Si cette valeur atteint zéro, l'entité commence à perdre des points de santé.

_muzzle_vert (skill32)
Numéro de vertex de la bouche de feu de l'arme pour des coups de feu des modèles ennemis. Employé pour laisser le tir partir de la bouche de feu plutôt que du centre de l'arme.

Aux fins de test, la variable freeze_actors peut être mise à 1 pour empêcher que les acteurs attaquent. Si elle est mise à 2 les acteurs sont passables.

player_fight
Cette action peut être employée pour une entité qui sert comme un joueur dans un environnement peu amical. Il gère la perte de points d'armure et de santé, et la mort. Les variables et des drapeaux ont la même signification que player_move.

Doors.wdl

Ce fichier de scénario contient toutes les fonctions pour toutes les parties de déplacement d'un niveau, comme la porte, l'ascenseur ou les entités de pont-levis.

send_handle
Cette fonction est assignée à la touche [Espace] par défaut et enverra un signal de l'identificateur du client au serveur. L'entité de joueur qui a reçu ce signal emploiera l'instruction de balayage pour ouvrir ou fermer n'importe quelle porte ou fera fonctionner n'importe quel ascenseur, dans un rayon de 200 quants de l'entité.

door
Cette action peut être assignée aux entités de carte de porte. La porte s'ouvrira en tournant horizontalement autour de sa charnière, qui est placée en son centre. La porte s'ouvre quand une instruction de balayage a été exécutée tout près par send_handle ou quand elle a été cliquée avec la souris ou quand une entité qui a _trigger de mis s'approche ou quand un changement éloigné a été opéré. Les propriétés de la porte sont mises à travers les drapeaux et les variables suivants :

_endpos (skill3)
L'angle d'ouverture de la porte est donné par skill3 (en degrés). Si on n'en donne aucun, le défaut est 90 degrés.

_keytype (skill4)
Numéro de clef (1..8). Si une clef est nécessaire pour faire fonctionner la porte, skill4 doit être mis au numéro de cette clef.

_force (skill5)
Donne la vitesse de la porte (en degrés par tick). Si skill5 est négatif, la porte s'ouvrira en sens inverse des aiguilles d'une montre.

_trigger_range (skill7)
Donne le rayon en quants dans lequel une entité doit se trouver pour ouvrir la porte automatiquement à son approche et la fermer derrière lui. Le drapeau _trigger de l'entité doit être mis à 1. Si on ne donne aucune valeur, la porte ne sera pas déclenchée automatiquement.

_switch (skill8)
Détermine quelles entités commutateurs ouvriront cette porte, si il y en a. La variable est composée d'une somme de puissance de 2 (1, 2, 4, 8, 16, 32, 64 ...) . Les commutateurs dont la valeur _switch contiennent une de ces puissances de 2 ouvriront la porte sur leur fonctionnement.

gate (la porte 'gate' à ne pas confondre avec la porte 'door'. 'gate' s'apparente plus à une porte de château type herse qui monte)
L'action de porte peut être assignée aux portes se déplaçant verticalement (entités de carte). La porte (gate) se déplacera vers le haut dans les mêmes conditions que décrit pour l'entité de porte (door); son rayon de mouvement est pré mis à 90 % de sa longueur. Skill5 donne la vitesse de porte (en quants par tick), le temps qu'il attend à sa position supérieure avant de se fermer de nouveau est donné par skill6. Si skill6 est à zéro, la porte se fermera seulement si il y a une nouvelle instruction de balayage de reçue. Des nouvelles propriétés de porte sont mises par les drapeaux et les variables suivants:

_keytype (skill4)
Numéro de clés (1..8). Si une clé est nécessaire pour ouvrir une porte, skill4 doit être mis au numéro de la clé.

_trigger_range (skill7)
Donne le rayon en quants à l'intérieur duquel une entité peut déclencher automatiquement l'ouverture d'une porte. Si aucune valeur n'est donnée, la porte ne s'ouvrira pas automatiquement.

lid (couvercle)
Cette action peut être assignée à des entités de carte type couvercle ou trappe, aussi bien que pour des ponts-levis. Le couvercle s'ouvrira en tournant verticalement autour de sa charnière dans les mêmes conditions que décrit pour l'entité de porte (door). Le centre des entités de couvercle doit être placé à sa position de charnière. Les propriétés de couvercle sont les mêmes que pour une entité de porte.

Elevator (ascenseur)
Cette action peut être assignée à une entité de carte type plateforme, qui peut se déplacer horizontalement, verticalement ou dans n'importe quelle direction. L'ascenseur peut transporter des passagers entre la position à laquelle il se trouve et une position cible arbitraire. Les coordonnées XYZ de la cible sont données par skill1, skill2, et skill3. Si une de ces variables est à 0, l'ascenseur ne se déplacera pas le long de cet axe.Si l'ascenseur entre en collision avec une entité ou le joueur, il se déplacera tout droit à travers lui – l'écrasant ou le poussant de côté, cela pouvant être traité par une fonction séparée event_push.

SI une clé est nécessaire pour démarrer l'ascenseur, skill4 doit être mis au numéro de la clé (1..8). La vitesse de l'ascenseur est donnée par skill5 (en quants par tick), Les temps d'attente à chaque extrémité sont données par skill6. Si skill6 est à zéro, l'ascenseur ne bougera que lorsqu'il recevra un scan (par l'appui de la touché [Espace] lorsqu'on est à côté) ou si on clique dessus avec la souris; autrement il se déplacera en permanence comme un ascenseur paternoster. De nouvelles propriétés d'ascenseur peuvent être mises avec les drapeaux et variables suivants :

_keytype (skill4)
Numéro de clé (1..8). Si une clé est nécessaire pour faire fonctionner l'ascenseur, skill4 doit être mis à ce numéro de clé.

_force (skill5)
Vitesse de l'ascenseur en quants par tick

_trigger_range (skill7)
Donne le rayon en quants à l'intérieur duquel une entité peut appeler automatiquement un ascenseur de sa positon de fin. Si cette valeur est à 1 ou plus, l'ascenseur démarre automatiquement dés que l'on marche dessus.

_switch (skill8)
Determines quel interrupteur démarrera la plateforme, s'il y en a. La variable est composée d'une somme de puissance de 2 (1, 2, 4, 8, 16, 32, 64...). Les interrupteurs dont la valeur _switch contient une de ces puissances de 2 mettra l'ascenseur en marche.

__remote (flag6)
Si ce drapeau est mis, l'ascenseur peut être appelé de ses positions de fin en recevant un balayage (scan).

teleporter
Cette action peut être assignée à une entité de carte, qui téléportera d'autres entités en touchant une position cible dont les coordonnées sont données par skill1, skill2, et skill3.

ent_rotate
Cette action est utile pour des ventilateurs ou des roues à eau. Elle fera tourner de manière permanente l'entité à laquelle elle est assignée autour de son axe Z par sa valeur skill1 (en degrés par tick), autour de son axe Y par sa valeur skill2 et autour de son axe X par sa valeur skill3.

key
Cette action peut être assignée aux entités clefs, qui peuvent être prises et employées pour ouvrir des portes. Bien sûr, elles ne sont pas obligées de ressembler à des clefs. Une entité clef peut être prise en la touchant, en cliquant avec la souris ou en appuyant sur la touche [Espace] lorsqu'on est près. Si elle est prise, une des variables key1.. . key8 sera mise à 1, dépendant du numéro de clef donné par skill4 (variable clef). Les propriétés de la clef sont mises par les drapeaux et les variables suivants:

__rotate (flag1)
Si ce drapeau est mis, l'article clef tourne avant d'être pris.

__silent (flag2)
Si ce drapeau est mis, le message indiquant que l'article est pris sera supprimé.

_keytype (skill4)
Numéro de la clé (1..8).

doorswitch
Cette action peut être attachée aux commutateurs pour des portes et des ascenseurs. Sa valeur de _switch (skill8) doit être mise à une puissance de deux (1,2,4,8,16, etc). En cliquant sur un commutateur, toutes les portes qui ont la même valeur de puissance de 2 dans la variable _switch seront ouvertes.

Messages.wdl

les fonctions dans ce fichier de scénario sont chargées de montrer des panneaux et des messages d'affichage sur l'écran.

msg_show (string,sec)
msg_blink (string,sec)
la fonction msg_show montrera la chaine donnée par string pendant le temps donné par sec en seconde dans le coin supérieur gauche de l'écran. La fonction blink_message fera de même, mais le texte clignotera. Exemple:

```
msg_show("ceci est mon message!",5);
```

scroll_message()

Cette fonction peut être employée pour faire défiler des messages de ligne multiples sur l'écran. Pour ajouter une nouvelle ligne au fond, employez les instructions suivantes :

```
set_string scroll.string, this_string;
scroll_message();
```

Le nombre maximal de lignes visibles est donne par la définition scroll_lines au commencement de message.wdl.

enter_message
: En commençant cette fonction un joueur peut entrer à l'écran un message d'une seule ligne dans un jeu multi-joueur et l'envoyer à tous les autres joueurs. Par défaut, cette fonction est lancée avec la touche [F4].

show_panels
: Active un panneau au coin d'écran gauche inférieur, où l'armure du joueur, la santé et les munitions restant dans l'arme actuelle sont montrés.

Particle.wdl

Ce fichier de scénario est responsable du traitement d'explosions de particule, des traînées de fumée et d'autres effets de particule.

particle_scatter
: Cette fonction peut être assignée aux particules pour créer une explosion. L'image bitmap <particle.pcx> est employée pour ces particules.

particle_range
: Comme particle_scatter, mais les particules montreront une couleur, au lieu d'une image bitmap et changeront leur couleur pendant leur durée de vie. La couleur de départ est donnée par les valeurs rouge, verte, bleue du vecteur couleur scatter_color, le pas pour changer de couleur est donné par le vecteur couleur scatter_range.

particle_trace
: Comme particle_scatter, mais les particules laisse un peu de traînées de fumée, faites de nouvelles particules.

particle_line
: Produit une traînée de vapeur s'effaçant entre les positions données par les vecteurs p et p2.

snowfall (chute de neige)
: Commence une chute de neige autour de la caméra.

Menu.wdl

Ce fichier de scénario se charge de traiter l'interface utilisateur.

menu_main (menu principal)
: Cette fonction montrera un menu de jeu standard. Le menu peut être actionné par la souris aussi bien que par le clavier. Il y a des options pour commencer un nouveau jeu, sauvegarder et charger un jeu, mettre le son et le volume de musique et la résolution vidéo, invoquer un panneau d'aide ou quitter. Cette fonction de menu peut être employée comme un calibre pour des menus de jeu individuels. Par défaut, le menu de jeu sera activé en appuyant sur la touche [Echap].

yesno_show
: Cette fonction affiche un panneau à l'écran, où un bouton oui / non peut être choisi via la souris ou le clavier. Par défaut, c'est une simple image bitmap noir et blanc. Le contact d'un bouton avec la souris le mettra en surbrillance. En cliquant sur le bouton oui ou en pressant la touche [Y], [Z] ou [Entrée] la fonction s'exécutera; le déclic sur le bouton non ou l'appui sur la touche [N] ou [Echap] fera disparaître le panneau. La chaîne de caractères et la fonction doivent être mis auparavant via yesno_txt.string et yesno_do.

menu_show

Cette fonction fera apparaître un menu à l'écran, où jusqu'à sept sujets peuvent être sélectionné via la souris ou le clavier. Tant que le menu est visible, le jeu sera gelé. Toucher un article avec la souris ou les touches de direction [haut/bas] le mettra en évidence. Le déclic de cet article ou le fait de presser la touche [Entrée] exécutera la fonction assignée à cet article. Par défaut, le menu disparaîtra en appuyant la touche [Echap].

Les chaînes de caractères et les fonctions correspondantes pour le menu doivent être mises auparavant via menu_txt1.string .. menu_txt7.string et menu_do1 .. menu_do7. La variable menu_max doit être mise au nombre d'articles dans le menu.

console

La fonction console() affichera un curseur sur l'écran où vous pouvez taper des instructions C-SCRIPT, qui seront exécutées pendant le jeu. Des instructions multiples peuvent être tapées, jusqu'à 80 caractères sur une ligne simple. Chaque instruction doit être terminée par un point-virgule. Par défaut, la console sera activée en appuyant la touche [Tab].

mouse_on, mouse_off, mouse_toggle

Ces fonctions bascule l'indicateur de souris en je te vois, je ne te vois pas. Par défaut, l'indicateur de souris apparaît lorsqu'on appuie sur le bouton droit de la souris.

Venture.wdl

Ce fichier de scénario contient les fonctions qui peuvent être employées pour vous aider à créer des jeux de type aventure (RPG). Il n'est pas automatiquement inclus dans un nouveau fichier de scénario, et doit donc être inclus manuellement. Ce scénario est employé par le jeu Adeptus. Venture peut être personnalisé pour mieux adapter vos besoins en employant les choses suivantes :

Predefines

Venture.wdl contient plusieurs valeurs par défaut. Beaucoup de ces images bitmaps, sons, variables, etc. peuvent être changés en les prédéfinnissant dans votre scénario de jeu avant que vous n'incluiez (include) venture.wdl. Pour prédéfinir un jeu de valeurs vous devez d'abord définir (define) le nom du groupe, puis vous devez prédéfinir toutes les valeurs dans ce groupe (si vous ne voulez pas changer toutes les valeurs, coupez et collez la section entre "define adv_def" ... et "endif" dans votre scénario et éditez les valeurs que vous voulez changer).

Par exemple, pour redéfinir les sons de combat tapez les lignes suivantes dans votre scénario de jeu avant "include <venture.wdl>;":

```
define adv_def_combat_snd;// sons de combat prédéfinis
sound weapon_swing_snd, <my_swing.wav>;
sound weapon_hit_snd, <my_hit.wav>;
sound player_hurt_snd, <my_pain.wav>;
```

Overrides (remplace)

Ce sont les fonctions qui sont conçues pour être redéfinies dans votre scénario de jeu après que venture.wdl soit inclus. Ces fonctions remplacent les fonctions par défaut de venture (qui sont ordinairement vide).

Managers

Les fonctions de Venture sont réparties en groupes appelés gestionnaires (manager). Chaque gestionnaire gère un aspect de venture (l'inventaire, le combat, les stats joueur, etc). Toutes les fonctions qui appartiennent à un gestionnaire commencent par le préfixe de ce gestionnaire. Les préfixes sont comme suit : Dialogue (vdia_), Entrée – input - (vinp_), Inventaire (vinv_), Ecran – Screen - (vscr_),Sauvegarde d'images - Picture Save - (vpic_), Qualités du joueur - Player Stats - (vpst_), Les articles – Items - (vitm_), Combat (vcom_), Les charmes – Spells - (vspl_), Touch Text (vttx_), et divers – Miscellaneous - (vmsc_).

Screen Manager (gestionnaire d'écran)

Employé pour gérer l'écran du joueur. C'est là où les panneaux d'interface du joueur sont définis (statbar_pan, screen_pan, blood_pan, blacktop13_pan ... blacktop16_pan).

Screen Predefines (écrans prédéfinis)

Les chaînes de caractères suivantes peuvent être définies:

```
define adv_def_adventure_text;
string wrong_key_str, "cette clé ne va pas!";
string wrong_item_str, "essayez un autre objet!";
string wrong_type_str, "rien n'est arrivé!";
string level_info_str, "vous avez fait un niveau! ...".;
string start1_str, "...".;
string bonus_info_str, "\nvous pouvez partager les points de bonus! fini";
string char_pan_str, " variable du personnage ...";
string help_str"commandes du clavier ...".;
string menu_str,"...";
string admenu_str, "....";
string exit_str, "...";
string vstr_exit_txt,"Merci ...";
string credit_str, "...";
string vent_blank_str, "...";
```

Les panneaux suivants sont prédéfinis:

```
define adv_def_screen_maps;
bmap button_map, <ventbutt.pcx>; // bitmap bouton
bmap char_button_map, <ventback.bmp>, 535,0,60,100; // icone pour bouton du personnage
bmap inv_button_map, <ventback.bmp>, 530,120,70,85; // icone pour bouton inventaire
bmap menu_button_map, <ventback.bmp>, 570,210,70,70; // icone pour bouton menu
bmap screen_map, <ventback.bmp>; // la carte d'écran en entier
bmap hpbar_map, <redbar.pcx>; // carte de la barre de coups
bmap strbar_map, <grnbar.pcx>; // carte de la barre de force
bmap manabar_map, <bluebar.pcx>; // carte de la barre mana
bmap info1_map, <ventinfo.bmp>; // info1 panneau bitmap (utilisé pour les sauvegardes et les chargements)
bmap info2_map, <ventdia.bmp>; // petit panneau info-texte

panel_font, <ventfont.pcx>,8,10; // panneau bitmap de fonte

define screen_stat_hp_bar_x 540; // décalage X pour la barre de coups
define screen_stat_hp_bar_y 10; // décalage Y pour la barre de coups
efine screen_stat_str_bar_x 555; // décalage X pour la barre de force
define screen_stat_str_bar_y 10; // décalage Y pour la barre de force
define screen_stat_mana_bar_x 570; // décalage X pour la barre de mana
define screen_stat_mana_bar_y 10; // décalage Y pour la barre de mana
define screen_stat_bar_height 80; // la hauteur des barres de stats
define screen_stat_bar_factor 1.0; // facteur de décalage vertical
bmap blood_splash_map <redf.pcx>; // éclaboussure de sang rouge (montrée lorsque le joueur est blessé)
bmap black_map <blackf.pcx>; // image noire(pour les fondus)
```

fonctions d'écran (Screen functions)

vscr_fade_out

Fondu de l'écran entier vers le noir.

vscr_fade_in

Fondu de l'écran entier depuis le noir.

vscr_black_out

Ecran tout noir.

vscr_toggle_menu
vscr_toggle_help
vscr_toggle_credits

Bascule les écrans menu, aide, et crédit en affiché / non affiché (on et off).

vscr_show_menu
Affiche le menu principal (dans le menu du jeu).

vscr_show_startmenu
Affiche le menu de démarrage (menu affiché au démarrage du jeu).

vscr_show_admenu
Montre le menu d'après la mort du joueur (menu qui doit faire que le joueur ne peut pas continuer le jeu).

vscr_show_help
Affiche l'écran d'aide.

vscr_show_credits
Affiche l'écran des crédits

vscr_show_exit
Affiche l'écran de sortie.

vscr_show_info
Affiche l'écran d'info. my doit avoir une chaîne de caractères valide (string2) (cette chaîne sera affichée dans la fenêtre).

vscr_close_all
Cache tous les panneaux, initialise les entrées du clavier du joueur (voir vinp_init_keys), et valide les icônes du joueur.

vscr_close_all_at_start
Cache tous les panneaux, invalide les entrées clavier du joueur (voir vinp_reset_keys), et invalide les icônes du joueur.

vscr_close_all_at_end
Cache tous les panneaux, invalide les entrées clavier du joueur (voir vinp_reset_keys), et valide les icônes du joueur.

Picture Save Manager
Gère la fonction de sauvegarde / chargement. Chaque jeu peut-être sauvegardé avec une copie d'écran et une option de ligne de texte. Jusqu'à 4 jeux peuvent être sauvegardés.

Picture Save Predefines

Les choses suivantes peuvent être prédéfinies:

```
define adv_def_save_load;
define picsize_x 64; // 320x240 divisé par 5
define picsize_y 48; //          "

bmap slot1_map <white.pcx>,0,0,picsize_x,picsize_y; // image de sauvegarde par défaut
bmap slot2_map <white.pcx>,0,0,picsize_x,picsize_y;
bmap slot3_map <white.pcx>,0,0,picsize_x,picsize_y;
bmap slot4_map <white.pcx>,0,0,picsize_x,picsize_y;

define picsave_bkgd; // le panneau a un fond
bmap picsavebk_map <black.pcx>,0,0,170,140;
define picloadbk_map picsavebk_map;
define panel_font,standard_font;

define picslot1_x 10; //positionne l'image
define picslot1_y 10;
```

```
define picslot2_x 84;
define picslot2_y 10;
define picslot3_x 10;
define picslot3_y 68;
define picslot4_x 84;
define picslot4_y 68;
define picexit_x 160;
define picexit_y 130;
// DEFINE PICSAVE_TEXT; // pour avoir un titre à chaque emplacement
```

Picture Save functions

vpic_save

Bascule le panneau de sauvegarde du jeu en oui / non. Lorsque le panneau de sauvegarde du jeu est sur oui, le joueur peut choisir un emplacement en cliquant dessus. Une copie d'écran est automatiquement générée, le joueur peut saisir un petit texte ((défini dans 'picsave_text'), et l'état du jeu est sauvegardé.

vpic_load

Bascule le panneau de chargement du jeu en oui / non. Lorsque le panneau de chargement du jeu est sur oui, le joueur peut sélectionner un jeu sauvegardé en cliquant sur son emplacement.

NPC Manager

Utilisé pour gérer les personnages autre que le joueur (NPC = Non Player Characters).

NPC Predefines

Les choses suivantes peuvent être prédéfinies:

```
define adv_def_npc;
var vsk_nav_speed[3] = 10, 0, 0; // vecteur de vitesse utilisé dans la navigation aléatoire
```

NPC functions

vnpc_random_navigation

Employé pour contrôler la cible invisible de navigation. Faîtes que votre NPC suivent cette cible si vous voulez qu'il donne l'impression de 'errer'.

Player Stat Manager

Utilisé pour gérer les qualités du joueur (coups, mana, force, agilité, bravoure, intuition, expérience, points de bonus, et niveau).

Player Stat Predefines

Les choses suivantes peuvent être prédéfinies:

```
define adv_def_playerstats;

define player_level_two_exp 1000; // experience nécessaire pour les 5 premiers niveaux
define player_level_three_exp 2500;
define player_level_four_exp 5000;
define player_level_five_exp 8000;
// ces valeurs sont ensuite doublées pour chaque niveau suivant
// (ex. 6 = 16000, 7 = 32000, 8 = 64000)

var player_hp = 20; // nombre de coups de base pour le joueur
var player_mana = 0; // niveau mana de base pour le joueur
var player_str = 15; // force de base pour le joueur
var player_gew = 15; // agilité de base pour le joueur
var player_mut = 15; // bravoure de base pour le joueur
var player_int = 15; // intuition de base pour le joueur

define player_rand_hp 5; // nombre aléatoire de points ajoutés aux coups
efine player_rand_mana 0; // nombre aléatoire de points ajoutés à mana
define player_rand_str 25; // nombre aléatoire de points ajoutés à la force
define player_rand_gew 25; // nombre aléatoire de points ajoutés à l'agilité
```

```
define player_rand_mut 25; // nombre aléatoire de points ajoutés à la bravoure
define player_rand_int 25; // nombre aléatoire de points ajoutés à l'intuition

define player_bonus_per_lvl 10; // nombre de points de bonus gagné à chaque fin de niveau

define player_periodic_hp 0.008; // nombre de points de coups qui se régénèrent chaque tick
define player_periodic_str0.050; // nombre de points de force qui se régénèrent chaque tick
define player_periodic_mana 0.008; // nombre de points de mana qui se régénèrent chaque tick
```

Player Stat functions

vpst_init_stats
Initialise la base des qualités du joueur

vpst_show_char
Montre le panneau des qualités du joueur.

vpst_show_bonus
Montre le panneau des qualités du joueur et lui permet d'affecter ses points de bonus à augmenter ses variables qualités.

vpst_show_create_character
Initialise le personnage (vpst_init_stats) et montre le panneau de bonus (vpst_show_bonus).

vpst_toggle_char
Bascule le panneau des qualités du joueur sur oui / non.

vpst_enable_periodic_update
Régénère les qualités (force, coups, et mana) et met à jour les barres dans le panneau de vie.

vpst_check_exp
Ajoute l'expérience donnée (stockée dans la variable give_exp avant l'appel de cette fonction) à l'expérience du joueur (player_exp) et vérifie si le joueur peut passer au niveau suivant. Si le joueur a atteint le niveau suivant, on incrémente le niveau du joueur (player_lvl) et lui affecte l'expérience à obtenir pour le niveau suivant (next_lvl_exp), donne les points de bonus (player_bonus_per_lvl) pour ses qualités, et montre le panneau bonus.

Item Manager
Utilisé pour le gestionnaire d'articles.

Item Predefines (articles prédéfinis)
Les choses suivantes peuvent être prédéfinies:

```
define adv_def_items;
sound item_hit_snd,<sack.wav>; // un son est joué quand un objet tombe
```

Item Overridden functions (fonction qui remplace les objets)

vitm_create_item
Crée un objet dependant selon la variable courante mouse_object. Chaque objet d'inventaire doit être inclus dans cette fonction. Exemple:

```
function vitm_create_item()
{
if (mouse_object == m_fish)
{
        create <fish_ov.pcx>, my_pos, entity_fish_event;
}
if (mouse_object == m_key)
{
        create <key_ov.pcx>, my_pos, entity_key_event;
```

```
}
...
}
```

Item functions

vitm_init_item

Initialise un objet qui doit être utilisé dans venture. Appelé après que vous ayez donné une valeur aux paramètres d'objet. Exemple:.

```
// Desc: action de l'entité poisson (attachée au poisson)
action entity_fish
{
	my.ent_id = m_fish; // met l'identificateur d'entité à poisson
	my.facing = on; // toujours face au joueur
	my.ent_scale = 1.0; // met à l'échelle (sans jeu de mot ah ah –scale en anglais voulant également dire écaille (de poisson) :) )
	MY.AMBIENT = 60; // valeur de lumière
	MY.STRING1 = fish_str; // texte touche
	vitm_init_item(); // initialise les objets (venture.wdl)
}
```

vitm_throw_item

Jette un article dans la direction du pointeur de souris. My doit être mis à l'article avant l'appel.

vitm_drop_item

Laisse tomber un article aux pieds du joueur. My doit être mis à l'article avant l'appel.

vitm_release_item

Libère l'article que le joueur tient actuellement (mouse_object).

Inventory Manager (gestionnaire d'inventaire)

Utilisé pour gérer l'inventaire du joueur.

Inventory Predefines

Les choses suivantes peuvent être définies:

```
define adv_def_invent_p;

bmap invent_map, <invent.bmp>; // panneau d'inventaire de 12 cases
bmap invent_item_map, <items.bmp>;// objets de l'inventaire

bmap i1_map, <invent.bmp>, 2, 2, 62, 70; // 1 des 12 cases du panneau d'inventaire
bmap i2_map, <invent.bmp>, 66, 2, 62, 70;
bmap i3_map, <invent.bmp>, 2, 72, 62, 62;
bmap i4_map, <invent.bmp>, 66, 72, 62, 62;
bmap i5_map, <invent.bmp>, 2, 135, 62, 62;
bmap i6_map, <invent.bmp>, 66, 135, 62, 62;
bmap i7_map, <invent.bmp>, 2, 197, 62, 62;
bmap i8_map, <invent.bmp>, 66, 197, 62, 62;
bmap i9_map, <invent.bmp>, 2, 260, 62, 58;
bmap i10_map, <invent.bmp>, 66, 260, 62, 58;
bmap i11_map, <invent.bmp>, 2, 320, 62, 62;
bmap i12_map, <invent.bmp>, 66, 320, 62, 62;

define invent_p_invent_temp_item_dx 70; // largeur de chaque objet dans invent_item_map

define invent_p_x 387; // coin supérieur gauche du panneau d'inventaire
define invent_p_y 4;

define invent_p_win_dx 70; // coupe la fenêtre à la bonne taille
define invent_p_win_dy 70;

define invent_p_row_1 2; // décalage ligne pour panneau d'inventaire
define invent_p_row_2 72;
define invent_p_row_3 135;
```

```
define invent_p_row_4 197;
define invent_p_row_5 260;
define invent_p_row_6 320;

define invent_p_col_1 2; // décalage colonne pour panneau d'inventaire.
define invent_p_col_2 66;
```

Inventory Overridden functions (les fonctions remplaçante de l'inventaire)

vinv_reset_critical_items

Remets à 0 les articles critiques. Exemple:

```
function vinv_reset_critical_items()
{
	// démarre sans épée ni diamant
	__has_sword = false;
	__has_diamond = false;
}
```

vinv_critical_items

Vérifie pour voir si un objet critique a été pris. Exemple:

```
function vinv_critical_items()
{
	if (invent_box_content == m_sword)
	{
		__has_sword = true;
		on_space = vcom_create_sword;
	}
	if (invent_box_content == m_gem)
	{
		__has_diamond = true;
	}
}
```

Inventory functions (fonctions d'inventaire)

Ces fonctions peuvent être employées pour gérer l'inventaire du joueur.

vinv_reset_inventory

Remet l'inventaire du joueur à 0 (zéro à toutes les valeurs).

vinv_show_inventory

Cette fonction montre le panneau d'inventaire du joueur.

vinv_hide_inventory

Cette fonction cache le panneau d'inventaire du joueur.

vinv_toggle_inventory

Bascule entre **show_inventory** (si le panneau d'inventaire est actuellement caché) et **hide_inventory** (si le panneau d'inventaire est actuellement visible).

vinv_put_item_inventory

Pose l'article actuellement stocké dans **'invent_temp_item'** dans la case vide suivante dans l'inventaire du joueur. Si aucun emplacement n'est disponible l'article est relaché (**vitm_release_item**).

vinv_delete_item_inventory

Enlève le premier article dans l'inventaire du joueur qui correspond à la valeur d'article actuellement stockée dans **'invent_temp_item'**. **venture_return** est mis à 1 si l'article a été trouvé, 0 autrement.

Combat Manager

Utilisé pour gérer le corps à corps.

Combat Predefines.

Les choses suivantes peuvent être prédéfinies:

```
define adv_def_combat_snd;

sound weapon_swing_snd, <swing.wav>;
sound weapon_hit_snd, <hit.wav>;
sound player_hurt_snd, <ahh.wav>; // son joué quand le joueur prend des dégâts
```

Combat functions (fonctions de combat)

vcom_player_defense
Cette fonction est appelée pour gérer les attaques sur le joueur. Employez les variables **vent_npc_str**, **vent_npc_mut** et **vent_npc_gew** (mis par l'entité attaquante avant l'appel de cette fonction) pour calculer une attaque sur le joueur. Le joueur emploie ses variables **player_mut**, **player_gew** et **player_int** pour se défendre. Les dégâts du joueur sont calculés (**player_current_hp** est réduit) et (si les dégâts sont plus grands que zéro) **vcom_player_hit** est appelé.

vcom_player_attack
Cette fonction est appelée pour gérer les attaques faites par le joueur. Employez les variables **player_mut**, **player_gew** et **player_int** du joueur pour calculer l'attaque. Employez les variables **vent_npc_int**, **vent_npc_mut**, et **vent_npc_gew** (mises par l'entité de défense avant l'appel de cette fonction) pour calculer la défense à l'attaque du joueur. Les dégâts sont calculés et appliqués à la valeur **_health** de l'entité appelante. Si la valeur de l'entité < 0, **_state** est mis à **_state_dead** (mort).

vcom_player_hit
Cette fonction est appelée pour gérer les coups reçus par le joueur. Si le coup a réduit la variable **coup** du joueur à 0 ou inférieur, il appelle **vcom_player_dead**. Autrement, il joue le son de 'mal' et 'flashe' le panneau de sang.

vcom_player_dead
Cette fonction est appelé pour gérer la mort du joueur. Remet à 0 les touches, montre le panneau de sang, gèle le jeu et montre le menu 'après la mort '.

vcom_create_sword
Crée un objet épée à **my_pos**. La touche [**Espace**] enlève l'épée.

vcom_direct_attack
Applique les dégâts directs à une cible en réduisant **my_health** de la valeur actuellement stockée dans **hp_minus**. S'il tue la cible, mettre son drapeau à **_npc_dead**

Dialog Manager

Utilisé pour gérer les dialogues des NPC.

Dialog Overridden functions (fonctions remplaçantes de dialogue)

vdia_make_talk
Utilisé pour se brancher sur différents dialogues du NPC. Exemple:

```
function vdia_make_talk()
{
	vscr_close_all();
	vinp_reset_keys();

	__icon_active = off; // ne pas laisser le joueur utiliser les icônes boutons
```

```
	if (npc_entity.ent_id == m_testnpc).
	{
		if (npc_entity._dialog_state == 11)
		{
			dialog_txt.string = testnpc11_str;
			dialog1_button_panel.visible = on;
			actor_dialog_b1 = 12;
			actor_dialog_b2 = 13;
			exit1 = off;
			exit2 = off;
		}
		...
	}
	vdia_show_dialog();
}
```

Dialog functions (fonctions de dialogue)

vdia_show_dialog
Montre le panneau de dialogue du NPC.

Input Manager
Utilisé pour gérer les entrées du joueur.

Input Predefines
La chose suivante peut être redéfinie:

```
define adv_def_cursor_map;
bmap mouse_l_map, <arrow.pcx>; // image du curseur de la souris
```

Input Overridden functions (fonctions remplaçantes d'entrée)

vinp_set_mouse_map_to_item
Employé pour mettre le pointeur de souris sur l'objet courant mouse_object. Exemple:

```
functon vinp_set_mouse_map_to_item()
{
	// met le 'point chaud' de la souris.
	mouse_spot.x = 4;
	mouse_spot.y = 4;
	if (mouse_object == m_obj_1)
	{
		mouse_map = object1_map;
	}
	if (mouse_object == m_obj_2)
	{
		mouse_map = object2_map;

	...
}
```

Input functions (fonctions d'entrée)

vinp_clear_mouse_to_inventory
S'il y a un objet sur le pointeur de la souris du joueur, l'enlève et le place dans le premier emplacement vide de l'inventaire du joueur.

vinp_mouse_init
Initialise les paramètres de la souris.

vinp_mouse_toggle
Bascule le pointeur de la souris oui / non.

vinp_show_mouse
Montre le pointeur de la souris.

vinp_hide_mouse
Cache le pointeur de la souris.

vinp_reset_keys
Remet à 0 les entrées clavier.

vinp_init_keys
Mets par défaut la carte des touches du clavier.

Note: vous pouvez remplacer cette fonction si vous voulez utiliser des touches différentes dans votre jeu.

touch text manager
Utilisé pour gérer le 'touche texte' (affiche le texte de l'objet touché).

vttx_show_touch_text
Montre le texte de l'objet touché.

vttx_hide_touch_text
Cache le texte de l'objet touché.

vttx_show_wrong_item_text
Utilisé pour indiquer que l'objet que vous employez n'est pas le bon. Cache le texte actuel de l'objet touché, montre le texte qui vous indique mauvais objet pendant 2 secondes et remontre l'ancien texte de l'objet touché.

Misc Manager
Utilisé pour gérer les fonctions diverses.

vmsc_exit
Fondu vers le noir et sortie du jeu. Affiche le texte de sortie (vstr_exit_txt)

Appendix

Annexe A: les pièges des scripts

C-SCRIPT est un langage facile - d'habitude. Mais la création d'un jeu en temps réel produit parfois des problèmes que vous ne rencontrerez pas normalement dans des tâches de programmation simples, comme de faire quelques effets de page Web avec java script. Par exemple, dans un jeu commercial des milliers de fonctions et des actions tournent en même temps et peuvent s'immiscer les unes avec les autres. Si vous ne pensez pas à cela vous pouvez avoir des ennuis. Dans ce qui suit nous avons rassemblé quelques problèmes communs arrivant non seulement aux débutants, mais aussi aux experts!

La redoutée instruction wait()

Le code suivant qui doit produire sept sons ne fonctionnera pas. Et pourtant elle fonctionne si nous supprimons la ligne wait(1) ! Pourquoi ?

```
function seven_chimes()
{
	temp = 0;
	while (temp < 7) // répète jusqu'à ce que temp soit égal à 7
	{
		play_sound chime_snd, 50; // joue le carillon
		temp += 1; // incrémente la variable temp
		wait(1);
	}
}
```

La variable temp est employée par beaucoup de fonctions pour des résultats intermédiaires. Plein d'autres fonctions s'exécutent pendant l'instruction wait(). A la première rencontre wait(1), temp a certainement la valeur 1 - mais après cela, elle peut avoir n'importe quelle valeur, selon ce que les autres actions ont fait avec elle.

Note: Considérez wait() comme une instruction qui change définitivement toutes les variables externes et les synonymes. Seul le synonyme my est garanti de conserver sa valeur à l'intérieur d'une action.

Ces maudits pointeurs invalides

Les deux actions suivantes sont assignées à deux entités dans le niveau. Hier elles ont travaillé parfaitement. Mais aujourd'hui, après le changement de quelque choses dans le niveau à une place totalement différente, le message d'erreur " empty pointer " surgit soudainement pour la deuxième action! Comment cela se fait-il?

```
entity* monster

action become_monster {
	monster = my; // met un synonyme global qui peut être utilisé par d'autres fonctions
}

action get_monster_height {
	my.z = monster.z; // utilise le synonyme pour donner la position verticale de my au monstre
}
```

La deuxième action exige que le pointeur monster soit déjà mis. Cependant cela dépend si la première action a été commencée avant la deuxième - une chance (ou malchance) sur deux. On ne peut pas définir l'ordre dans lequel les actions assignées aux entités seront commencées dans le chargement de niveau. Remplacez votre deuxième action par :

```
action set_monster_height {
	while (monster == null) { wait(1); }// attendre jusqu'à ce que ce soit monster
```

```
	my.z = monster.z;
}
```

Note: En employant un pointeur - même simple comme my ou you - pensez toujours à la possibilité que cela pourrait être non défini à ce moment là!

Mauvais chronométrage

L'utilisation d'une variable pour le passage de l'information à une fonction ne travaille pas parfois. Je veux émettre une particule bleu et une particule verte, mais j'obtiens toujours deux particules vertes ! Qu'est-ce qui ne va pas dans le code suivant ?

```
var particle_mode = 0;

...
particle_mode = 2; // indique une particule bleue
effect(particle_start,1,my.x,nullvector);
particle_mode = 1; // indique une particule verte
effect(particle_start,1,my.x,nullvector);
...
function particle_start()
{
	if (my_age == 0)
	{
		if (particle_mode = 0) { my_color.red = 255; my_color.green = 0; my_color.blue = 0; }
		if (particle_mode = 1) { my_color.red = 0; my_color.green = 255; my_color.blue = 0; }
		if (particle_mode = 2) { my_color.red = 0; my_color.green = 0; my_color.blue = 255; }
	...
}
```

À la différence de l'appel direct d'une fonction, les fonctions appelées indirectement par des instructions comme effect, ent_create, load_level, load etc. ne sont pas garanties de commencer immédiatement après cette instruction. Elles peuvent commencer jusqu'à deux cycles d'encadrement plus tard et parfois même pas sur le même PC dans des jeux multi joueur! La fonction particule, par exemple, tourne sur les postes clients quand effect est exécuté sur le serveur. Dans notre exemple, les deux actions de particule commencent par particle_mode de mis à 1 (ou à une autre valeur particle_mode qui arriverait ensuite). La solution serait d'employer des fonctions de particule différentes.

Mauvaises mathématiques

J'ai trouvé la formule suivante pour calculer la distance bi-dimensionnelle de l'origine dans mon livre de mathématiques - cependant dans C-SCRIPT, cela donne parfois un résultat faux!

```
dist2d = sqrt(my.x*my.x + my.y*my.y);
```

La valeur absolue de toutes les variables et des résultats intermédiaires doit être comprises entre 0.001 et 1000 000. Donc notre formule devient invalide si my.x ou my.y excèdent 1000 (parce qu'alors le résultat intermédiaire my.x*my.x excède 1000 000). Solution : si vous pensez que des résultats intermédiaires de formules peuvent excéder 1000 000, multipliez-les d'abord par un choix judicieux de facteur et ensuite corrigez le résultat entier.

```
dist2d = 10000 * sqrt((0.01*my.x)*(0.01*my.x) + (0.01*my.y)*(0.01*my.y));
```

Cette formule mathématiquement identique, donne my.x et my.y dans un intervalle valable entre 10 et 100 000 et donne zéro s'ils sont en dessous 10. Dans les cas normaux les variables de jeu n'excéderont pas les frontières critiques, la seule exception peut arriver en multipliant deux distances.

Le piège des évènements

La fonction suivante est destinée à téléporter le joueur à l'emplacement (100,200,0) s'il touche une entité téléporteuse avec l'action teleport1 qui lui est attachée. Cependant elle ne fonctionne pas :

```
function tp1_event()
{
	you.x = 100;
	you.y = 200;
	you.z = 0;
}

action teleport1 {
	my.enable_impact = on;
	my.event = tp1_event;
}
```

La fonction d'événement est déclenchée par event_impact de l'intérieur d'une instruction ent_move. Donc ent_move n'a pas été fini au moment où la fonction d'événement a été exécutée. La fonction d'événement change la position de l'entité, mais ent_move fait de même et place l'entité à sa position cible calculée sur la fin. De cette façon le déplacement par la fonction d'événement n'a aucun effet. Solution : insérez une instruction wait(1) au commencement de la fonction d'événement, pour que ent_move puisse être fini auparavant :

```
function tp1_event()
{
	wait(1);
	you.x = 100;
	you.y = 200;
	you.z = 0;
}
```

Note: n'employez Jamais des instructions qui changent quelque chose dans le niveau ou peuvent produire des événements eux-mêmes, dans le premier cycle d'une fonction d'événement! Cela signifie aussi que les instructions comme ent_create, remove, ent_move, trace, shoot, scan... insèrent toujours un wait(1) en les employant dans des événements.

Attacher des entités les unes aux autres

Nous avons un modèle entité avec une lumière rouge vif en son centre. Pour améliorer l'effet de lumière, nous avons attaché un sprite d'éclat à ce modèle :

```
action red_model
{
	ent_create <light.pcx>,my.x,flare_light;// attache le sprite lumière
	patrol(); // enclenche l'évènement de déplacement
}

action flare_light {
	my.flare = on; // met le drapeau flare
	my.bright = on; // rend la lumière brillante
	my.facing = on; // toujours face à la camera
	while (1) {.
		my.x = you.x;// place le sprite à la position xyz de l'entité qui l'a créé
		my.y = you.y;
		my.z = you.z;
		wait(1);
	}
}
```

Nous nous sommes attendus à ce que le sprite lumière soit fixé au centre du modèle et se déplace avec lui. Cependant il semble avoir toujours une frame de retard. Et même plus mauvais, le modèle se déplace maintenant à seulement la moitié de la vitesse d'avant! Qu'est-ce qui arrive ?

Le deuxième problème est facile à régler : nous avons oublié de rendre la lumière passable. Donc le modèle entre de manière permanente en collision avec sa propre lumière, ce qui explique la moitié de la vitesse. Les entités attachées à d'autres doivent avoir le drapeau passable :

```
action flare_light {
```

```
    my.passable = on;
    my.flare = on; // set the flare flag
    ... // etc.
```

Mais qu'est-ce qui peut expliquer ce retard de temps étrange ? Si deux entités s'influencent, il est important de garder à l'esprit l'ordre de leurs deux actions fonctionnant simultanément. Elles ne s'éxécuteront pas vraiment en même temps. Dans l'exemple, pendant une frame le planificateur de fonction exécute d'abord l'action de **flare_light** et ensuite l'action de **patrol** du **red_model**. Les fonctions qui sont commencées d'abord s'éxécuteront d'abord. Donc **flare_light** est toujours placé à la position qu'avait **red_model** dans la frame précédente. La solution est simple : En changeant l'ordre des actions ...

```
action red_model {
    patrol(); // enclenche l'évènement de déplacement AVANT
    ent_create <light.pcx>,my.x,flare_light;// attache le sprite lumière et démarre l'action APRES cela
}
```

... La lumière se tient parfaitement au centre de l'entité. Pour la même raison, la fonction **drop_shadow()** doit toujours être appelée après la fonction de mouvement d'une entité.

Annexe B: Démarrage du moteur

La version de développement du moteur A4 ou A5 - acknex.exe - est placée dans le sous dossier BIN. Sauf de l'intérieur de WED, le moteur peut également être lancé à partir du répertoire dans lequel se trouve votre scénario ou vos cartes en donnant la commande :

acknex.exe ***mapnam***e.wmp ***[option***s]
ou
acknex.exe ***scriptnam***e[.wdl] ***[option***s]

Ou vous pouvez créer un raccourci Windows et y assigner cette ligne de commande. Sauf du jeu lui-même, il y a encore trois fichiers exigés pour que le moteur puisse fonctionner. Ceux-ci sont aussi normalement placés dans le dossier BIN et peuvent ou être copiés du répertoire BIN vers le dossier de jeu ou recréés directement :

acknex.mdf	- le fichier des messages d'erreur du moteur.
acknex.wdf	- le fichier de définition du lanceur, qui peut aussi être personnalisé par la version professionnelle WED,
palette.raw	- le fichier de palette, qui peut aussi être produit en rebaptisant simplement le fichier levelname.raw qui est créé par le processus de compilation

Acknex.exe fait partie du système de développement et ne doit jamais être donné. Pour la distribution d'un jeu, les fonctions publish et resource de WED créeront une version de distribution EXE du moteur.

Les options de Ligne de commande

Les mêmes options de ligne de commande sont valables pour le moteur pendant l'exécution EXE aussi bien que pour le moteur de développement acknex.exe. Ainsi le jeu publié peut être commencé avec les options de ligne de commande suivantes, donné derrière le nom du module EXE :

-nx ***number***
Taille du nexus en megabytes. Le nexus est la structure de données interne pour le rendu d'images. La taille du nexus dépend de la taille et de la complexité du plus grand niveau du jeu. Plus grand sera le nexus et plus on pourra montrer de scènes complexes et plus on pourra utiliser d'entités - mais plus de mémoire virtuelle est employée - si la mémoire réelle n'est pas disponible, elle est prise du disque dur. La valeur par défaut pour nexus est 40 méga-octets.

Un nexus trop petit pour une scène peut causer une baisse soudaine du taux d'affichage ou même donner un message d'erreur du moteur. Si vous jamais rencontrez un message " nexus too small " (nexus trop petit) ou "too much entities" (trop d'entités), le moteur doit être lancé en donnant une taille de nexus plus haute que celle que vous avez employé auparavant (par exemple -nx 80 pour la taille de nexus de 80 MO et 800 entités).

-dir ***name (nom de répertoire)***
Donne le répertoire défini pour la sauvegarde des jeux et des copies d'écran ; remplacera n'importe quel nom de savedir défini dans le scénario C-SCRIPT.

-d ***name***
Définit le nom name pour la nouvelle évaluation dans le scénario C-SCRIPT par ifdef. Cela a le même effet que define ***nam***e; dans le scénario C-SCRIPT. De cette façon, des options de jeu arbitraires, comme la résolution d'écran de départ, D3D ou pas, la catégorie de difficulté etc., peuvent être mises via la ligne de commande.

-diag
Produit des messages diagnostiques dans le fichier acklog.txt dans le dossier actuel.

-wnd

Commencera en mode fenêtre au lieu du mode plein écran. Tant que vous n'avez pas redéfini la fonction on_enter, vous pouvez toujours commuter entre le mode fenêtre et le mode plein écran par [Alt-Entrée]. Notez s'il vous plaît qu'en utilisant le mode de profondeur vidéo 16 bits ou 32 bits (mode D3D), le mode fenêtre travaille seulement si la profondeur de couleur du bureau est la même (haute ou couleur vraie). Dans le mode vidéo 8 bits la fenêtre peut être ouverte dans toutes les résolutions du bureau.

-n3d

Forcera le rendu en mode vidéo 8 bits et de ne pas employer D3D, même si on donne le mode vidéo 16 bits ou 32 bits dans le scénario C-SCRIPT.

-s3d

Force l'outil de restitution de logiciel D3D externe dans les modes 16 bits ou 32 bits. L'outil de restitution de logiciel D3D est mortellement lent, mais peut être employé pour faire des copies d'écran ou pour déterminer si une certaine faute visible est causée par le moteur ou par les drivers de carte D3D.

-w3d

Forcera le mode D3D faible, même quand la carte 3D feint de soutenir des textures multiples. Cela peut être nécessaire pour quelques vieilles cartes 3D. Voir la section suivante.

-crd ***number***

La carte 3D avec le numéro donné sera choisi, au lieu de 1 par défaut. C'est utile pour des systèmes avec plusieurs cartes 3D. Seulement des dispositifs 3D actifs seront choisis. L'ordre des cartes n'est pas déterminé, donc pour choisir une deuxième carte il faudra parfois mettre "1" et parfois "2" (ou 3 si 3 cartes). Le message de démarrage donnera le nom de la carte.

-nj

Met hors de service le levier de commande.

-nm

Met hors de service la souris.

P C –sv

Lance une nouvelle session multi joueur en mode serveur.

P C -cl

Se joint à une session de multi joueur dans le mode client.

-pl ***name***

Par cette option on peut donner un nom individuel à chaque client (jusqu'à 16 caractères). Autrement le moteur générera le nom d'un client

-com ***number***

Utilise une liaison série avec le numéro du port donné (1...4) pour une session à 2 joueurs.

-modem

Comme -com, mais utilise une connexion modem. Au début de jeu vous serez invités à dire lequel de vos modems vous employez et quel numéro de téléphone doit être composé. Si vous n'entrez pas à de numéro de téléphone, la connexion commencera comme serveur et attendra qu'un autre PC l'appelle pour se connecter.

-ipx

Comme -com, mais utilise un réseau local (LAN) avec le protocole IPX pour une session multi joueurs.

-tcp

Comme -ipx, mais utilise un réseau local (LAN) avec le protocole TCP/IP ou une connexion Internet active. Au début de jeu vous serez invités à entrer le nom de domaine du serveur ou l'adresse IP à connecter. Si vous n'entrez rien, le PC ouvrira la session comme un serveur et attendra d'autres PC pour se connecter. Si votre serveur n'a pas de nom de domaine, vous pouvez trouver votre adresse IP actuelle avec le programme WINIPCFG dans votre dossier de Windows. Sur un réseau local l'adresse IP des PC connectés peut être trouvée avec le programme NETSTAT. Ces deux programmes sont disponibles sur tous les PC Windows.

-sn *name*

A travers cette option de ligne de commande, le nom de la session peut être donné au démarrage du serveur ou d'un client en mode multi-joueurs. Le nom de la session par défaut est le nom du fichier script principal sans l'extension. Le nom donné ou l'adresse sont valables durant toute la durée du jeu dans la chaîne session_name.

-port *number*

A travers cette option de ligne de commande, une adresse de port pour la communication tcp/ip client/serveur est donnée. Les ports valides sont dans la plage 2300 à 2400. Si aucun port n'est donné, un port par défaut est assigné. Une adresse de port ne peut être donnée que pour un serveur. Le client se connectera sur le bon port automatiquement.

-ip *name*

A travers cette option de ligne de commande, le nom du serveur, du domaine ou l'adresse IP lorsqu'on démarre un client en mode multi-joueur. Dans ce cas la boîte de dialogue demandant d'entrer l'adresse du serveur n'apparaîtra pas. Exemple :

Acknex office –cl –ip 169.254.73.28

Se connectera au serveur avec l'adresse donnée sur le réseau ou sur internet. Le nom donné ou l'adresse sont valables durant toute la durée du jeu dans la chaîne server_name.

Si elles ne sont pas changées par C-SCRIPT, les touches suivantes sont actives dans le jeu ou à l'intérieur d'un niveau 'walkthrough' ('marche à travers') :

[F2]	Sauvegarde le jeu dans le fichier "save_0.sav".
[F3]	Charge le dernier jeu sauvegardé.
[F5], [Shift-F5]	Bascule la résolution vidéo.
[Alt-Entrée]	Bascule entre mode fenêtre et mode plein écran.
[F6]	Pose les copies d'écran dans le fichier "shot_n.pcx".
[F10]	Quitte le jeu.
[F11]	Bascule la valeur gamma (mode 8-bit uniquement).
[F12]	Bascule musique et sons en 3 paliers.

Les touches suivantes sont actives dans le moteur de développement, mais pas dans le moteur de distribution:

[0]	Bascule le mode de déplacement direct du joueur et la caméra en 3 paliers.
[D]	Active un panneau par défaut de mise au point et de statistiques, qui montre les informations suivantes : valeur courante du taux d'affichage en fps, la position XYZ de la caméra, l'angle de caméra (pan, tilt et roll) la mémoire texture, le taux de remplissage du tampon client / serveur, le ratio fps entre le serveur et le client, le nombre de polygones de carte visibles, les polygones d'entités, les entités visibles, le nombre de fonctios actives, le nombre de particules, le temps de rendu 3D, le rendu des panneaux, la mise à jour du tampon écran, les scripts, la communication client / serveur, la position x,y de la souris. Lorsqu'on appuie une seconde fois sur [D] les polygones sont mis en valeur par des lignes rouges.
[Tab]	Active un curseur clignotant où des instructions C-SCRIPT peuvent être entrées. Quelques instructions comme exit ou load_level() ne peuvent pas être entrées directement.

Considerations D3D

En mode 16 et 32 bits un accélérateur 3D est activé, si aucun n'est trouvé. Pour employer toutes les particularités de D3D, DirectX 7.0 est exigé (s'il n'est pas déjà installé, la dernière version de DirectX peut être téléchargée du site de Microsoft). Les accélérateurs 3D diffèrent dans leurs particularités, la vitesse et dans la taille de texture maximale qu'ils peuvent montrer. Selon les caractéristiques du matériel, le moteur emploie des algorithmes différents pour produire des textures, des ombres et des effets. Le moteur essaye toujours de sortir la meilleure exécution et la meilleure qualité d'image du matériel. Un bon indicateur pour la performance de votre matériel et la taille de la texture maximale est le message dans votre fenêtre initiale:

no d3d device - 16/32 bit modes disabled
: D3D n'est pas correctement installé sur votre PC ou la version DirectX est trop vieille. Le moteur peut seulement fonctionner en mode de 8 bits.

8 bit software renderer activated (reduced quality)
: Le moteur est démarré en mode de 8 bits ou il n'y a pas de matériel 3D de détecté. C'est normalement le cas avec certains portables. Le rendu est donc exécuté par logiciel.

rgb emulation device detected - 1024x1024 textures supported
: D3D est installé, mais aucun matériel 3D n'a été trouvé. C'est normalement le cas avec la plupart des ordinateurs portables. Les modes 16 bits sont émulés par logiciel, les modes 32 bits ne sont pas disponibles. Le rendu 16 bits est lent et doit seulement être utilisé pour des copies d'écran.

d3d hal incapable device detected - 1024x1024 textures supported
: Un très vieil accélérateur D3D a été détecté, qui manque des particularités 3D essentielles. Des modes 16 bits sont possibles, mais non recommandés - les images manqueront d'ombres et de quelques autres effets. Les modes 32 bits ne sont pas disponibles.

d3d hal weak device detected - 256x256 textures supported
d3d hal slow device detected - 256x256 textures supported
: Un dispositif 3D bon marché, périmé a été détecté, comme une carte Voodoo/3dfx. Le message 'slow' est cependant meilleur que le message 'weak'. Des textures, des sprites ou des modèles de triangles de peau plus grand que 256x256 pixels ne seront pas montrés (mais ils existent également des dispositifs lents ou faibles qui supporte 1024x1024). Les modes 16 bits et 32 bits sont remarquablement lents et ont une apparence plus que mauvaise.

d3d hal good device detected - 1024x1024 textures supported
: Un dispositif 3D moderne, capable d'opérations de mélangeage de multitexture, a été détecté. Votre application peut tourner avec plus de 65 fps sur un écran en 1024x768.

Quelques vieux accélérateurs 3D feignent la capacité de multitexture, mais ne le soutiennent pas vraiment. Cela peut aussi arriver si vous changez votre carte 3D et mélangez les drivers de votre vieille carte avec les drivers de la nouvelle carte par une désinstallation incorrecte. De tels cas sont normalement détectés par le moteur et il fonctionnera dans le mode faible. Mais si une telle carte réussit à tromper le moteur, le mode D3D ne montrera pas les lumières et les ombres correctement ou ne fonctionnera pas du tout. Dans de tels cas l'option de ligne de commande **-w3d** doit être employée pour forcer le mode faible. Notez s'il vous plaît que quelques vieilles versions Voodoo/3dfx peuvent seulement fonctionner en mode plein écran (maximum 640x480) et ne supporte pas l'espace de texture réservé.

Error Messages

Très souvent vous rencontrerez des messages d'erreur ou d'avertissement en compilation ou en exécution d'un niveau. La plupart des messages d'erreur arrivent pendant le balayage du fichier C-SCRIPT et indiquent des erreurs de syntaxe. Votre avez mal tapé quelque chose ou faites référence à un fichier ou un objet qui n'existe pas. Le fichier C-SCRIPT et la ligne en question sont donnés (s'il y a), donc vous pouvez facilement corriger l'erreur. Toutes

les autres erreurs qui ne s'expliquent pas d'elles-mêmes et qui ne sont pas évidentes sont listées ici. Les avertissements et des messages d'erreur sont seulement publiés par le système de développement, pas par le module pendant l'exécution du jeu final.

Erreurs pendant le démarrage de moteur

Missing export ddraw dll
: Si vous avez un message de ce type, DirectX n'est pas installé correctement sur votre ordinateur. 3D GameStudio nécessite DirectX 7.0 o supérieur..

error e351: corrupted engine (possible virus!)
: Une tentative de tailler ou modifier le module pendant l'exécution a été détectée, probablement en raison d'une infection virale.

error e356: problem with wdl script
: Le scénario ne pouvait pas être exécuté. Soit il contient une erreur de syntaxe qui est indiquée. Soit il n'a pas été trouvé dans le répertoire actuel. Soit il a été modifié après la publication.

error e1003: missing basic palette (palette.raw)
: Pour quelque raison le fichier de palette initial manque. Vous pouvez le remplacer en rebaptisant la palette levelname.raw qui a été créé par la dernière compilation.

error e1004: insufficient memory or disk space
: Votre disque dur est plein. Effacez des trucs ou achetez en un plus grand !!!

error e1005: invalid level
: Le module d'exécution a été créé pour un jeu différent - le fichier de ressource ne va pas. Créez-le à nouveau.

error e1241: d3d device open failure
error e1242: video device open failure
: DirectX n'est pas correctement installé sur votre PC. Installez DirectX 7.0 ou au-dessus.

Erreurs pendant le chargement de niveaux, d'entités ou d'animation

error e1100: can't open level (ne peut pas ouvrir le niveau)
error e1101: bad wmb format (mauvais format WMB)
error e1103: bad colormaps (mauvaise carte de couleurs)
error e1109: level contains no palette (le niveau ne contient pas de palette)
: Le fichier WMB donné n'est pas trouvé ou est corrompu. Reconstruisez-le.

error e1301: can't open file (ne peut pas ouvrir le fichier)
error e1302: bad mdl format (mauvais format mdl)
error e1303: bad mdl bounds (mauvaise limites mdl)
: le fichier modèle donné n'est pas trouvé, corrompu ou a un faux format.

error e1199: unknown entity type (type d'entité inconnu)
error e1321: invalid entity (entité invalide)
: Vous avez essayé de charger une entité qui n'est ni sprite, ni carte, ni modèle valable.

error e1104: invalid texture size (taille de texture invalide)
error e1105: bad texture animation (mauvaise animation de texture)
: Le fichier WMB contient des textures qui ont une taille invalide ou leur cycle d'animation ne commence pas par +0 ou est interrompu.

warning w1107: texture size no power of 2 (avertissement : la taille de la texture n'est pas une puissance de 2)

Cet avertissement est publié seulement si la variable warn_level est mise à 2 ou au-dessus. La texture est montrée, mais dans une qualité moindre.

warning w1108: texture too big for 3d card (texture trop grande pour la carte 3D)
warning w1114: sprite too big for 3d card (sprite trop grand pour la carte 3D)

Cet avertissement est seulement publié en employant une carte 3D Voodoo ou si la variable warn_level est mise à 2 ou au-dessus. Des cartes voodoo ne sont pas capables de montrer des textures de plus de 256 pixels en taille. Par une astuce logicielle le moteur peut faire que la Voodoo affiche des textures superficielles jusqu'à 1024x1024 (pas les sprites), mais dans une qualité moindre.

error e1111: invalid pcx/bmp format (format de pcx/bmp invalide)
error e1112: invalid size (taille invalide)
error e1115: bad tga format (32 bit rle only) (mauvais format TGA (32 bits rle seulement)

Le fichier bitmap contient un format invalide ou une fausse taille de texture. La profondeur de couleur doit être 8 bits ou 24 bits sur des textures normales et 32 bits sur des textures avec le canal alpha.

error e1113: bitmap too big for 3d card (image trop grande pour carte 3D)

L'image bitmap est si grande qu'elle ne tient pas dans la mémoire vidéo (par exemple: bitmaps de 4096x4096 ou plus grand)

warning w1305: triangle too big for 3d card (triangle trop grand pour la carte 3D)

Cet avertissement est seulement publié en employant une carte 3D Voodoo ou si la variable warn_level est mise à 2 ou plus. Des cartes Voodoos ne sont pas capables de montrer les triangles modèles de plus de 256 pixels en taille.

warning w1306: entity too big for this level (entité trop grande pour ce niveau)

Cet avertissement est seulement publié si la variable warn_level est mise à 2 ou plus. L'entité est trop grande et coupée dans trop de parties par l'arbre BSP à la position actuelle. Donc la sélection de BSP ne peut pas être employée pour cette entité, aboutissant à un taux d'encadrement plus lent. Le nombre de découpe BSP pour une entité peut être réduit au minimum en plaçant son origine à sa position de centre.

error e1310: bad scale value (mauvaise valeur d'échelle)

Vous a donné une échelle d'entité négative ou autrement étrange.

error e1320: action not found (action non trouvée)

Vous avez rebaptisé ou effacé l'action de comportement d'entité du fichier C-SCRIPT.

warning w1351: can't open file (ne peut pas ouvrir le fichier)
warning w1352: can't play sound (ne peut pas jouer de son)
warning w1361: not found or no valid avi (fichier AVI invalide ou non trouvé)

Le son ou le fichier d'animation n'a pas été trouvé ou a un format invalide.

Runtime Errors (erreurs pendant l'éxécution)

Error e1201..1211: nexus too small (nexus trop petit)

Le message est clair: Vous devez augmenter la valeur nexus (-nx comme option dans la ligne de commande) pour ce niveau.

error e1220: general error (erreur générale)

Un accident interne au moteur est arrivé. Notez les circonstances et mettez-vous en rapport avec Conitec pour de l'aide.

error e1230: insufficient memory for d3d textures (mémoire insuffisante pour textures d3d)

Votre carte 3D n'a pas assez de mémoire pour pouvoir exécuter ce niveau.

error e1231: insufficient disk space left (espace disque insuffisant)
Votre disque dur est plein, faites de la place.

error e1240: palette missing for transparency (palette manquante pour la transparence)
error e1243: palette missing for fog (palette manquante pour le brouillard)
Vous avez essayé de faire de la transparence ou du brouillard en mode 8 bits sans avoir chargé de niveau avec une palette valide pour le brouillard.

error e1244: 3d card requires weak mode (-w3d) (la carte 3D exige un mode dégradé –w3d)
Votre carte 3D ne supporte pas des particularités données par sa propre liste de capacité de dispositif. Vous devez forcer le mode faible par l'option de ligne de commande -w3d.

error e1245: 3d card internal memory failure (erreur mémoire sur la carte 3D)
Votre carte 3D à un défaut interne. Mettez vos drivers à jour à partir du site constructeur. En attendant démarrez dans le mode dégradé, ça aide parfois.

error e1299: demo time expired. order the full version at www.conitec.net
En anglais dans le texte mais le message est clair. Vous ne pensiez tout de même pas que la version de démonstration allait durer éternellement !!! Il est temps de passer à la caisse, vous ne le regretterez pas, ce produit est vraiment fantastique. (NDT)

error e1340: more entities than max_entities (plus d'entités que max_entities)
Le message est clair : vous devez augmenter le nexus ou la valeur de max_entities.

warning w1342: more than 4096 entities potentially visible (plus de 4096 entités potentiellement de visibles)
Trop d'entités sont dans l'angle de vue de la caméra en raison d'un niveau mal conçu.

Client/Server communication errors (erreurs de communication client/serveur)

warning w1401: client/server communication trouble (ennui de communication client/serveur)
warning w1402: client/server buffer overflow (débordement du tampon client/serveur)
La connexion client/serveur a été interrompue pour quelques secondes ou trop d'entités ont été créées en même temps.

warning w1403: too much clients (trop de clients)
Il y a plus de clients qui essayent de connecter que d'entités disponibles dans le niveau. Le client est rejeté. Augmentatez max_entities.

warning w1404: incompatible engine versions (versions de moteur incompatibles)
Le client emploie un moteur avec une version de protocole différente et est rejeté.

warning w1405: session already exists - can only join it as client (une session existe déjà – impossible de joindre le client)
Un serveur a essayé d'ouvrir une session qui existe déjà sur le réseau.

warning w1406: session doesn't exist - can only open it as server (la session n'existe pas – peut seulement l'ouvrir comme serveur)
Un client a essayé de joindre une session inexistante.

warning w1407: client joined wrong level (un client a joint un mauvais niveau)
Ce message d'avertissement est publié par le serveur si un client connecté avec lui éxécute un niveau de jeu différent.

Bad C-SCRIPT code (mauvais code C-SCRIPT)

warning w1501: empty pointeur used (un pointeur vide est utilisé)
Vous utilisez un pointeur qui n'est pas initialisé et qui n'existe donc pas. Reportez vous au manuel à propos des pointeurs et de leur utilisation.

warning w1502: endless loop (boucle sans fin)
Vous avez probablement oublié un wait()dans une boucle de l'action donnée. Lire le tutorial C-SCRIPT au sujet des boucles.

warning w1503: invalid array index (index de tableau invalide)
Vous avez donné un index qui excède la longueur du tableau.

warning w1504: negative log (logarithme négatif)
warning w1505: negative square root (racine carré négative)
warning w1507: division by 0 (division par 0)
Vous employez une fonction arithmétique avec un mauvais argument.

warning w1506: too much actions (trop d'actions)
Plus d'actions s'exécutent que le planificateur peut manipuler. Vous avez probablement créé des actions dans une boucle infinie.

warning w1508, w1510: can't load from file (ne peut pas charger à partir du fichier)
Vous essayez de charger une sauvegarde de jeu ou le fichier de renseignements qui a été sauvegardé par un jeu différent ou le fichier C-SCRIPT.

warning w1509: avoid save/load in same frame (évitez les sauvegardes / chargement dans une même frame)
Vous essayez d'exécuter plusieurs chargement (load) ou des opérations de chargement de niveau (load_level) en même temps. Seule une opération de chargement est possible par cycle d'encadrement - les autres ne seront pas exécutées.

warning w1511 - dangerous instruction in event (instruction dangereuse dans l'évènement)
Cet avertissement sera publié quand la fonction d'événement donnée contient un mauvais code C-SCRIPT - une instruction interdite qui change quelque chose dans le niveau ou peut déclencher un événement lui-même, comme move, ent_remove, ent_create, trace, scan etc. dans le premier cycle de l'événement. Insérez un wait(1) avant l'exécution de telles instructions dans une fonction d'événement. Cet avertissement est seulement publié si la variable warn_level est mise à 2 ou au-dessus.

error e1513: wdl crash in... (wdl plante dans...)
Un plantage est arrivé en raison du mauvais code de C-SCRIPT dans une certaine fonction, comme une division par 0 ou une racine carrée d'un nombre négatif. Notez que telles erreurs sont seulement attrapées et indiquées dans le moteur de développement. Dans la version distribuable ils provoqueront juste le plantage programme.

error e1514: invalide handle in... (mauvais identificateur...)
Un paramètre invalide est passé à un identificateur ou à une instruction ptr_for_handle.

WED / Map Builder warnings and errors (WED, avertissements et erreur du compilateur de cartes)

load error: error before line xxx (erreur au chargement, erreur avant la ligne xxx)
Le fichier ouvert par WED ne contient aucun contenu WMP valable. Il peut y avoir plusieurs raisons, comme un plantage du PC pendant l'opération de sauvegarde, une faute de disque dur ou une édition / recopie manuelle du fichier WMP. Si le fichier WMP est détruit, il peut être récupéré : WED produit automatiquement deux fichiers de secours, un fichier .bak à chaque opération de sauvegarde et le fichier .$$m à chacun opération de compilation. Le dernier est seulement pour les cas d'urgences, comme l'information de groupe est perdue.

warning w010: vertex out of plane (vertex en dehors du plan)
Le compilateur n'aime pas la forme d'un bloc particulier. D'ordinaire inoffensif.

warning w011, w201, w202: null length edge
Un bloc a un bord qui est trop court - deux vertices sont trop proches l'un de l'autre. Si on donne le numéro du bloc, cherchez le bloc (Edit -> Find Block) et effacez-le.

warning w013: can't create block (ne peut pas créer le bloc)
Le bloc est mal formé ou a des plans doubles ou quelques autres problèmes. Ce ne sera pas visible dans le niveau et peut produire des erreurs. Trouvez le bloc et effacez-le.

warning w022: map must be rebuilt (la carte doit être reconstruite)
Vous avez essayé de mettre à jour les textures d'une carte, mais le niveau lui-même ou une taille de texture a été changé. La carte doit être reconstruite.

warning w064: block with inconsistent surfaces (bloc avec des surfaces contradictoires)
Vous avez essayé d'appliquer des textures passables (liquide) et solides au même bloc. Vous devez vous décider si le bloc entier doit être liquide ou non.

warning w066: surfaces too big for shaded mode (surfaces trop grandes pour le mode ombré)
Vous avez des surfaces dans votre niveau qui sont trop grandes pour le mode de rendu choisi. D'extrêmement grandes surfaces doivent être rendues dans le mode plat, autrement elles consommeront trop de mémoire) et réduiront le taux d'affichage. Vous pouvez ignorer l'avertissement, mais alors le niveau sera très lent dans la compilation et le rendu.

warning w071: duplicate plane (plan dupliqué)
Le bloc donné a deux côtés identiques ou très semblables. Trouvez le et effacez-le.

warning w074: duplicate blocks (bloc dupliqué)
Deux blocs identiques ont la même position, probablement en raison d'une opération de copie échouée. Une autre possibilité pour la création de blocs doubles et que vous ayez déplacé un bloc à l'intérieur d'un autre et exécuté ensuite une opération de soustraction. Effacez un des blocs.

warning w090: region leaking (une région a une fuite)
La boîte de ciel n'est pas serrée ou un objet est à l'extérieur ou il n'y a aucune boîte de ciel du tout. Ce n'est normalement pas un problème. Cependant les surfaces extérieures de la boîte de ciel doivent avoir un mode de rendu de mis à none dans un tel cas pour empêcher un taux d'affichage réduit en raison du balayage des surfaces invisibles.

warning w149: invalid surface size (taille de surface invalide)
La texture est appliquée à une surface qui est trop grande ou trop petite ou appartient à un bloc mal formé. La surface sera montrée avec une texture par défaut.

warning w177: can't create portal (ne peut pas créer de portal)
warning w110: misplaced portal (portal égaré)
Une erreur interne est arrivée qui pourrait mener à un problème de coupure dans le niveau à une certaine place. Surtout dû aux erreurs de blocs qui ont été indiqués dans les messages d'avertissement précédents. Surtout inoffensif.

critical error: map size (taille de carte)
critical error: ...overrun (…embouteillage)
critical error: too much ... (trop …)
La carte est trop grande ou contient trop des polygones compliqués pour être compilée. Effacez beaucoup de blocs. Remplacez des objets de bloc ou prefabs par des entités de carte.

critical error: can’t create portals (ne peut pas créer de portal)

La carte contient de mauvais blocs ou d'autres problèmes. Si les erreurs de bloc ont été indiquées auparavant, effacez ces blocs. D'autres problèmes pourraient être aussi des blocs allongés ou des blocs aigus.

Annexe C: If Something Doesn't Work (si quelque chose ne travaille pas)

Le développement de jeu n'est pas en réalité la tâche d'un débutant. Essayez d'employer 3D GameStudio sans lire le manuel ou sans quelques connaissances de base du PC, Windows, le graphisme et les palettes peut rapidement aboutir à la frustration. Ici nous avons rassemblés quelques problèmes communs auxquels les débutants peuvent se heurter.

Q. Bien que j'ai placé la lumière dans mon niveau, elle ne se montre pas – le niveau apparaît pleine brillance.

R. Voici quelques raisons communes pour que les lumières ne soient pas montrées:

- Vous n'avez pas compilé votre niveau après le placement des lumières.
- Aucune de vos lumières ne touche une surface dans son rayon d'action.
- Vous avez mis vos lumières trop noires.
- Votre carte 3D n'est pas correctement installée ou est trop vieille (indiquée comme "incapable").
- Vous avez mis toutes vos textures à flat.

Q. O.K., mes lumières sont à présent visibles mais elles sont monochromes.

R. Queques raisons possibles:

- Vous avez seulement l'édition standard ou extra
- Vous avez coché Monochrome dans l'écran de dialogue de Build.
- Vous exécutez votre jeu en mode 8-bit.

Q. Mon jeu n'a pas une belle apparence et s'exécute très lentement.

R. Vous l'exécutez probablement dans le mode de 8 bit avec l'outil de restitution logiciel. Pour fonctionner dans le mode D3D, assurez-vous que vous avez une carte 3D correctement installée. Dans votre fichier de script définissez la variable video_depth à 16.

Q. Le moteur montre une baisse remarquable du taux affichage si la caméra fait face à un certain mur. Derrière ce mur se trouve une pièce très complexe, mais elle n'est pas visible d'ici et ne devrait pas influencer le taux d'affichage.

R. Vous avez compilé la carte avec preview de coché. Dans le mode preview, le taux d'affichage dépend lourdement de la complexité des cartes (visibles ou pas).

Q. Dans le mode D3D 16 bits ou 32 bits, quelques surfaces sont noires comme la poix ou ne sont pas montrées du tout.

R. Vos textures sont plus grandes que 256x256. Quelque matériel 3D, Banshee ou Voodoo3 par exemple, ne soutiennent pas de tailles de textures au-dessus de 256x256.

Q. J'ai essayé de construire mes propres fichiers wad en ajoutant des textures de 8 bits que j'ai créées, mais je continue à obtenir de fausses couleurs dans le niveau, comme dans WED.

R. Vous avez employé la mauvaise palette dans votre programme de peinture pour créer vos propres textures avec. Ou vous les avez convertis au format 8 bits palettisés sans spécifier de palette. Adaptez juste vos textures à la palette de niveau. Dans Paint Shop Pro, c'est simplement " Charge Palette ". Cela doit être la même palette que vous avez assignée dans les Propriétés de Carte.

Q. J'ai créé une nouvelle palette et y ai adapté mes textures. Mais maintenant les textures ont un peu de points noirs et blancs en mode plein écran. Dans le mode fenêtre elles sont parfaites.

R. Votre palette ne respecte pas la restriction "couleur 0 = noir, couleur 255 = blanche". Editez votre palette, changez la couleur 0 et la couleur 255 et adaptez toutes vos textures à nouveau.

Q. Pourquoi le taux d'affichage baisse-t-il quand j'emploie une petite échelle de texture (< 0.1 quants/texel) ?

R. Cela arrive sur des surfaces ombrées. Plus l'échelle est petite, plus de cache pour les surfaces est créé, et plus vous avez besoin de cache, plus souvent il a besoin d'être rempli à nouveau surtout si votre nexus est trop petit.

Q. J'ai créé un module d'exécution pour mon jeu, employant la fonction publish ou resource, mais il ne démarre pas.

R. Vérifiez si vous avez une erreur de fichier manquant pendant le processus publish ou resource. Si les fichiers manquent, aucun module d'exécution ne sera créé. Une autre possibilité consiste en ce que vous ayez changé un de vos fichiers dans le répertoire CD par la suite. Dans ce cas vous devez lancer le processus publish ou resource de nouveau.

Annexe D: Juridique

Vous avez maintenant fini votre jeu, fini les tests bêta, trouvé un éditeur ou un distributeur et vous attendez que votre jeu apparaisse dans les magasins. Mais vous n'avez pas oublié quelque chose ? Au sujet du prix de la licence à payer pour distribuer le module d'exécution du moteur A4 ou A5 avec votre jeu à l'intérieur ?

Que nenni. Pour avoir acheté légalement 3D GameStudio, vous avez le droit de créer autant de jeux que vous voulez, les distribuer, gagner de l'argent et devenir riche, sans devoir vous donner la peine du paiement de droits d'auteur ou des montants de la licence. Vous avez aussi le droit d'employer toutes les textures libres, les cartes, les modèles, les armes et les sons sur le CD-ROM.

En échange de ces droits, nous exigeons trois choses quand vous allez distribuer votre jeu en public:

- Envoyez-nous une copie de votre jeu distribué.
- Montrez le logo du moteur pendant une seconde au début de jeu.
- N'employez pas 3D GameStudio pour un jeu qui propage intentionnellement le génocide contre un certain groupe, nation, race ou religion existant dans le monde réel d'aujourd'hui.

Dans les éditions standard ou extra, il y a un logo minuscule 'en filigrane' du moteur qui sera montré automatiquement dans un coin d'écran de votre choix. Il est semi transparent et vous pouvez facilement l'intégrer dans un panneau pour le cacher.

C'est à vous de créer l'écran d'accueil avec le logo du moteur. Vous pouvez ajouter vos propres logos de jeu et texte. Le logo doit être montré pendant au moins 1 seconde (16 ticks) au commencement du jeu. Il y a un choix de deux logos - sombre (logodark.bmp) et brillant (logolite.bmp). Ils montrent la version du moteur - A4 ou A5 - dans un cube. Vous les trouverez dans le dossier template.

Le logo doit être placé près du centre de l'écran d'accueil. Aucune partie du cube ne peut être couverte par d'autre graphisme ou texte. Il peut être adapté à votre palette de niveau et redimensionné pour correspondre à votre résolution d'écran de départ, mais le cube doit couvrir au moins un tiers de la taille d'écran horizontale. Vous pouvez montrer plus d'écrans d'accueil avec vos propres logos, mais l'écran d'accueil du logo du moteur doit être le premier de votre jeu.

En plus de ces trois obligations, il y a des choses légales supplémentaires que vous devez garder à l'esprit.

Avec le logiciel vous recevrez un mot de passe qui autorise des mises à jour libres à partir de notre site web. Il ne vous est pas permis de donner ce mot de passe à d'autres personnes.

Peut-être ne travaillez-vous pas seuls au jeu, mais ensemble avec une équipe. Vous avez le droit de faire les copies de 3D GameStudio pour des buts de secours, mais il est illégal de travailler avec. Si plusieurs membres de votre équipe travaillent avec 3D GameStudio, vous ne devez pas acheter un original pour chaque membre - vous pouvez acheter une licence d'équipe. Pour des groupes de développeurs ou des classes d'école/collège, les licences d'équipe sont une façon de économiser sur les dépenses. Tous les membres de l'équipe doivent employer le logiciel pour le même projet et le logiciel ne doit pas être employé pour des projets séparés ou à l'extérieur de l'équipe. La

licence d'équipe inclut une copie de 3D GameStudio, qui peut être installé sur le PC de chaque membre d'équipe. L'acheteur d'une licence d'équipe est responsable de mettre à jour le logiciel et ne doit pas donner le numéro de série et le mot de passe de mise à jour à un autre membre de l'équipe. Pour plus de détails voir le tarif en ligne sur notre bon de commande d'Internet.

Si vous employez des textures, des sons, des modèles ou d'autre matériel des niveaux de démonstration du système de développement, notez s'il vous plaît que vous devez les employer pour des applications GameStudio seulement. Il est illégal de les employer pour un autre but. Il est aussi illégal de distribuer une autre partie du système de développement ou modifier le module d'exécution distribuable de n'importe quelle façon.

Le droit pour la distribution gratuite n'inclut pas le droit d'employer ou d'émettre des applications 3D GameStudio pour des buts commerciaux dans des lieux publics, comme des spectacles télévisés, des machines d'arcade, des sites de jeu de multi joueur en ligne commerciaux et cetera. Si vous projetez un tel événement, entrez s'il vous plaît en contact avec le support pour découvrir si une licence dans votre cas est exigée ou non. Pour l'utilisation d'applications GameStudio en public comme des expositions, il est normalement suffisant de montrer le logo de moteur dans des intervalles réguliers entre les jeux.

Pour des sociétés de développement de jeux professionnels, il y a quelques avantages à acheter une licence même pour la distribution normale. L'écran d'accueil d'affichage du logo pendant une seconde n'est pas exigé pour des jeux autorisés. Des modules d'exécution autorisés sont seulement disponibles pour l'édition professionnelle. Pour en acheter un, vous aurez besoin de la Clef Magique que WED montre pour votre fichier de ressource. Nous vous enverrons par la poste ou vous enverrons par courrier électronique le module normalement en un jour ouvrable. Le montant de la licence dépend du prix pour l'utilisateur final et la largeur de distribution et s'étend de 1000 $ pour un jeu en shareware, jusqu'à 10,000 $ pour un jeu de bonne taille distribué dans le monde entier. Pour des détails et pour des licences de jeux multiples, entrez s'il vous plaît en contact avec notre support.

Questions fréquemment posées

Q. J'ai besoin d'une particularité ou d'un effet pour mon jeu, qui semble ne pas être disponible dans le moteur. Puis-je payer pour le faire mettre en oeuvre ?

R. Nous prendrons votre argent avec le plaisir. Contactez simplement notre support. Cependant, peut-être la particularité dont vous avez besoin est déjà prévue pour la mise à jour gratuite suivante ou une version intermédiaire est déjà disponible pour l'évaluation. Vous trouverez une liste de particularités prochaines sur le forum utilisateur.

Q. Pourquoi le logiciel consiste-t-il en CD-ROM plus un disque clef ? Pourquoi pas un CD-ROM seul ?

R. Empêcher la piraterie du logiciel. Le disque clef contient votre fichier clef personnalisé avec votre numéro de série à l'intérieur. De cette façon nous pourrions tracer n'importe quelle copie illégale - ou n'importe quel jeu fait avec cela – jusqu'au client qui l'a acheté, le persuader avec des arguments forts de ne pas commettre de piraterie désormais dans sa vie et se battre ainsi avec le mal dans le monde.

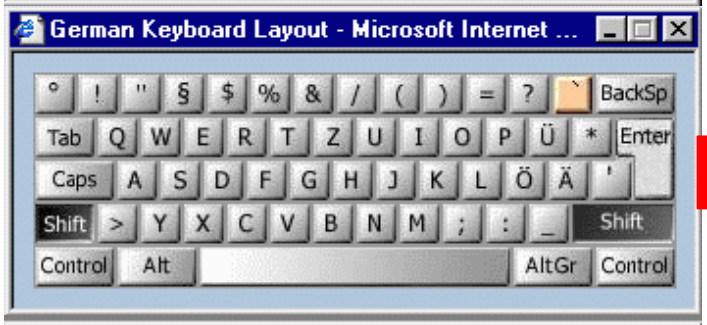
P.S. Pensez 5 minutes à la tête que vous ferez lorsque c'est votre jeu, sur lequel vous venez de passer des jours et des nuits qui sera piraté… no comment ! (NDT)

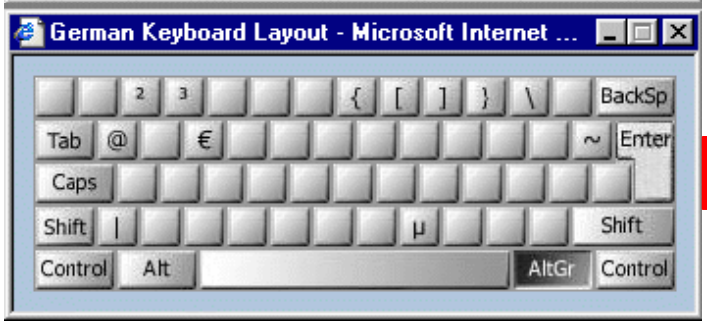
Q. J'ai envoyé une question technique au support, mais je n'ai obtenu qu'une réponse désinvolte et vague.

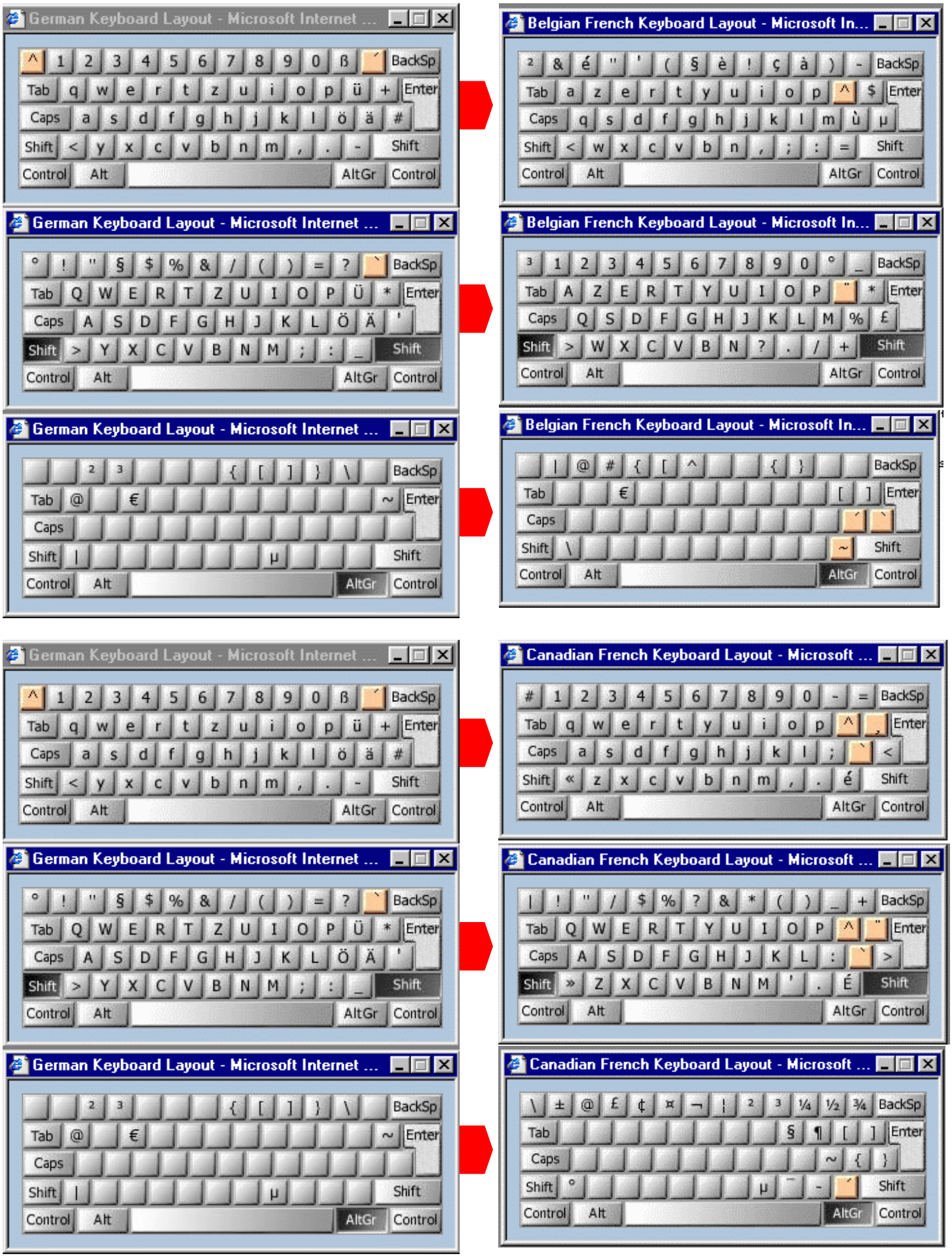
R. Vous avez probablement demandé : "envoyez-moi un programme C-SCRIPT qui fait cela ou ceci." Le support à alors normalement du vous répondre : "S'il vous plaît, lisez le manuel et programmez le vous-même. ".

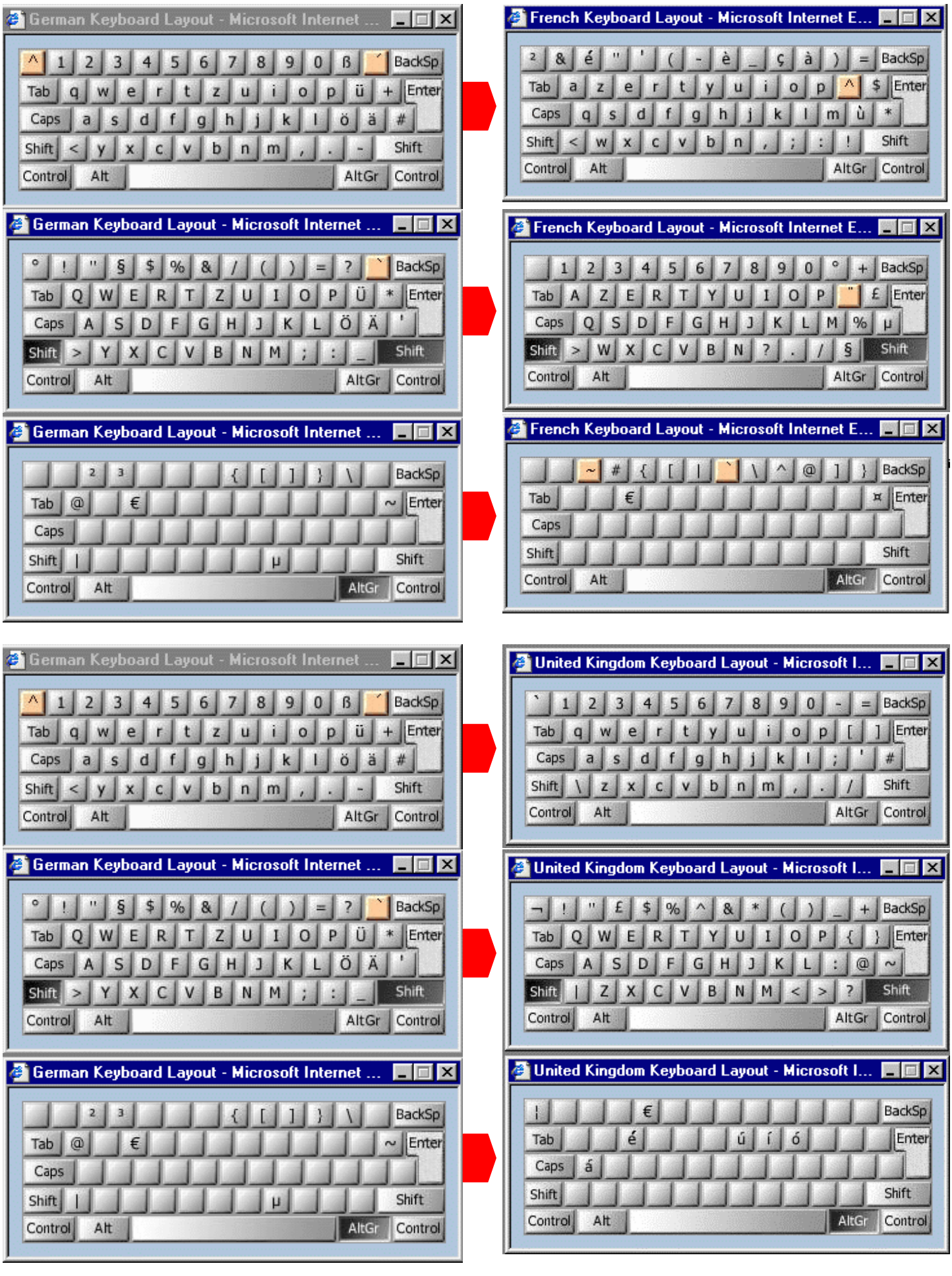
Ce sera le mot de la fin.

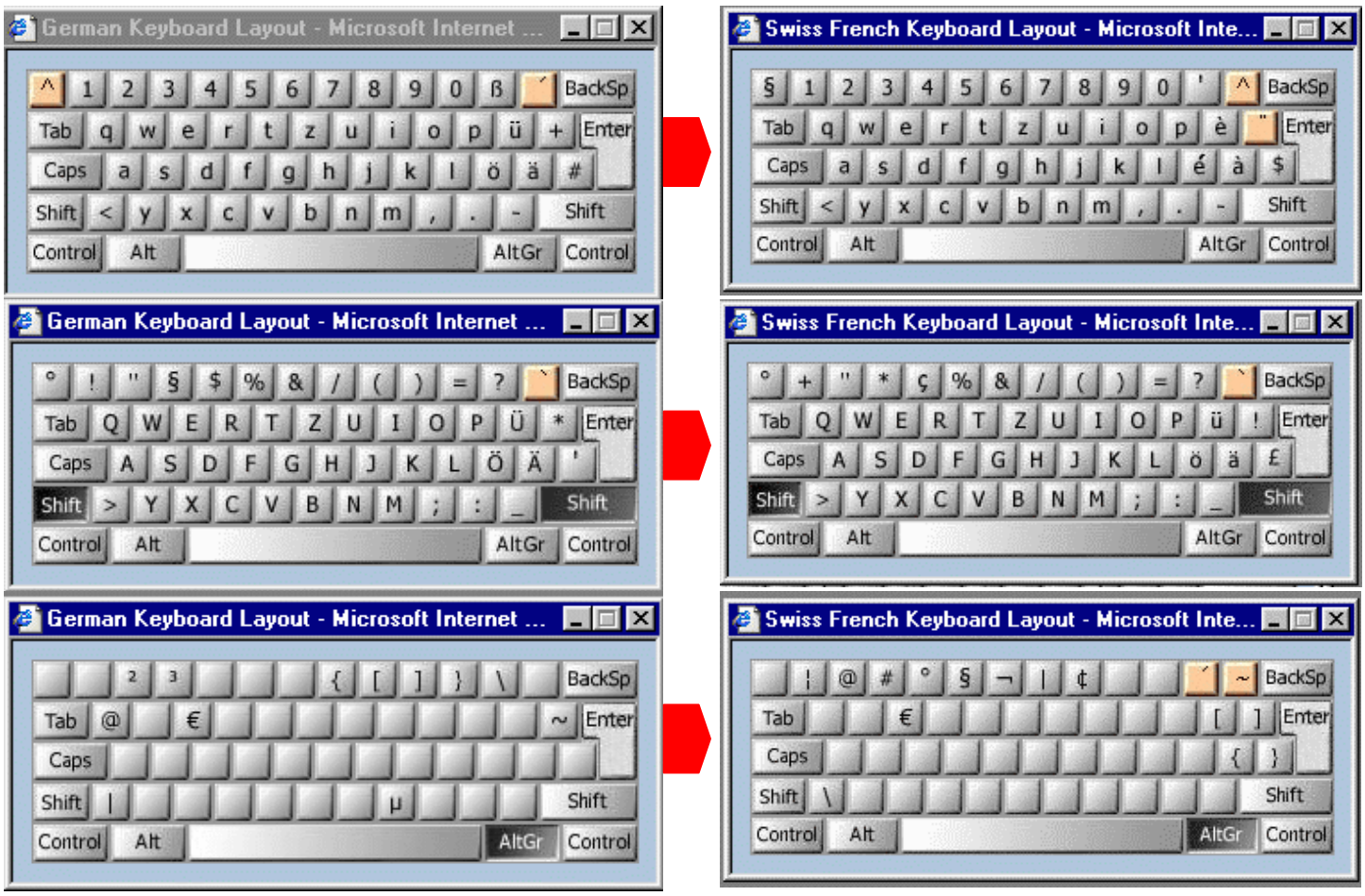
Annexe E – les claviers






German Keyboard Layout - Microsoft Internet ...
Belgian French Keyboard Layout - Microsoft In...
German Keyboard Layout - Microsoft Internet ...
Belgian French Keyboard Layout - Microsoft In...
German Keyboard Layout - Microsoft Internet ...
Belgian French Keyboard Layout - Microsoft In...
German Keyboard Layout - Microsoft Internet ...
Canadian French Keyboard Layout - Microsoft ...
German Keyboard Layout - Microsoft Internet ...
Canadian French Keyboard Layout - Microsoft ...
German Keyboard Layout - Microsoft Internet ...
Canadian French Keyboard Layout - Microsoft ...


German Keyboard Layout - Microsoft Internet ...
French Keyboard Layout - Microsoft Internet E...
German Keyboard Layout - Microsoft Internet ...
French Keyboard Layout - Microsoft Internet E...
German Keyboard Layout - Microsoft Internet ...
French Keyboard Layout - Microsoft Internet E...
German Keyboard Layout - Microsoft Internet ...
United Kingdom Keyboard Layout - Microsoft I...
German Keyboard Layout - Microsoft Internet ...
United Kingdom Keyboard Layout - Microsoft I...
German Keyboard Layout - Microsoft Internet ...
United Kingdom Keyboard Layout - Microsoft I...


German Keyboard Layout - Microsoft Internet ...
Swiss French Keyboard Layout - Microsoft Inte...
German Keyboard Layout - Microsoft Internet ...
Swiss French Keyboard Layout - Microsoft Inte...
German Keyboard Layout - Microsoft Internet ...
Swiss French Keyboard Layout - Microsoft Inte...

INDEX

D

E

F

G

H

I

'

I

J

K

L

M

T

U

V

W

X

Y