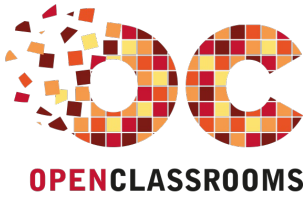
Une map avec Hammer !

Par Loulimi
et Mathieu Nebra (Mateo21)



www.openclassrooms.com

Dernière mise à jour le 24/08/2011

Sommaire

Une map avec Hammer !

Le tutoriel que vous êtes en train de lire est en **bêta-test**. Son auteur souhaite que vous lui fassiez part de vos commentaires pour l'aider à l'améliorer avant sa publication officielle. Notez que le contenu n'a pas été validé par l'équipe éditoriale du Site du Zéro.

Par  Mathieu Nebra (Mateo21) et  Loulimi

Mise à jour : 24/08/2011
Difficulté : Facile

Vous voulez créer votre propre map pour **Counter-Strike** ? Une nouvelle aventure pour **Half-Life** ?

Vous êtes sur la bonne page : c'est un cours de mapping pour débutants !
Contrairement à ce qu'on pourrait croire, ce n'est pas si sorcier que ça. Si vous prenez le temps de bien lire ce cours, je peux vous assurer qu'à la fin la création de maps n'aura plus de secrets pour vous ! Et comme on part de **Zéro**, je suppose que vous ne savez rien de rien sur la création de maps, et je vous apprends TOUT dessus.

Pourquoi ne pas essayer !?

Ce tutoriel est exclusivement destiné à **Half-Life 1** et ses mods, ce qui signifie qu'un grand nombre des méthodes décrites (notamment la compilation) et des logiciels donnés dans ce tutoriel ne sont pas valables pour Half-Life 2 et Counter-Strike : Source.
Si vous voulez apprendre à mapper pour le moteur Source, vous pouvez consulter le tutoriel de Thunderseb sur le Mapping Source, ou consulter le site web Modding-Area.

Partie 1 : Fonctions de base de Hammer

*(Initiez-vous à Hammer et apprenez à manier les blocs.
C'est par là que doivent commencer les débutants)*

Installation et configuration

Je vous l'avais dit, mapper sous Half-Life (ou un de ses mods vous avez compris)nécessite un certain nombre de logiciels et de programmes!
Hé bien justement, il est temps de télécharger ses fameux programmes & logiciels!
Une fois que vous les aurez téléchargé, il faudra les configurer.

Télécharger et installer

Bien. La première chose à faire c'est d'installer Valve Hammer Editor, le logiciel permettant la création de maps sous Half-Life (par Half-Life je désigne Half-Life et ses mods)!

Télécharger Valve Hammer Editor

C'est un exécutable. Double cliquez dessus pour installer Hammer. Une fois Hammer installé, téléchargez sa mise à jour, un exécutable nommé Hammer.exe, que vous placerez dans le répertoire d'Hammer!

Télécharger Hammer.exe v3.5

Vous remplacerez ainsi celui qui s'y trouvait déjà!

Une fois ceci fait, téléchargez les zhlt. Ce sont des compilateurs qui rendront votre map jouable sous Half-Life ou un de ses mods. Leur utilité ne vous est pas indispensable pour l'instant, mais nous en aurons besoin bientôt pour configurer Hammer.

Télécharger les zhlt

Une fois ce dossier dezipé (clic gauche), placez le dans le répertoire d'Hammer.

Pour dezipper un fichier, il vous faut 7zip téléchargeable sur le web!

Nous allons maintenant télécharger les FGD, indispensable eux aussi pour configurer Hammer.
Entres les mods d'Half-Life, il y a des différences. Si on veut mapper pour l'un ou pour l'autre, il faudra alors un FGD qui permet de mapper pour le mod que l'on shouaite.
Ainsi grâce au FGD de Counter-Strike vous pourrait créer des maps pour Counter-Strike!

Télécharger le FGD de Counter-Strike, utile aussi pour le mapping CS:CZ

Les autres fgd sont dans le dossier "fgd" de Hammer.

Pour ce qui est des fgd des autres mods, rendez vous sur cette page de mapping-area, téléchargez les FGD de votre choix et mettez les dans le dossier "fgd" de Hammer.
Si vous comptez faire une map pour un autre mod, il vous faudra chercher le fichier FGD correspondant qui est quelques fois "livré" avec le mod, ou sinon au créateur du mod de vous l'envoyer.
D'autres FGD se trouvent cependant dans ce pack.

La dernière chose à faire concernent un .wad. La seule chose à savoir pour l'instant sur un .wad c'est que c'est ce fichier qui contient les textures.

Créez un dossier dans le répertoire d'Hammer nommé "wad" et mettez dedans le fichier zhlt.wad qui se trouve dans le dossier "zhlt" précédemment téléchargé et que vous avez mis dans le répertoire d'Hammer.

Configurer Hammer

Cette partie concerne la configuration de Hammer pour le système WON!
Si vous utilisez **Steam**, lisez à la place cette partie du tutoriel de Mister-J ou regardez les vidéos de Comatrix. Il est cependant conseillé de lire d'abord cette partie puis celle de Mister-J!

Nous pouvons maintenant démarrer Hammer. La première fois, il vous invite à le configurer. Répondez "Yes" pour ouvrir la fenêtre de configuration. Vous pourrez revenir à cette fenêtre en passant par le menu Tools/Options.

Onglet "Game Configurations"

Dans le premier onglet "Game configuration", cliquez sur le bouton "Edit" tout en haut. Cliquez sur "Add" dans la fenêtre qui vient de s'ouvrir, et donnez un nom à votre nouvelle configuration (le nom du mod).
Imaginons que l'on veut configurer pour half-life.
Entrez "Half-life", puis faites "Close".

De retour dans la première fenêtre, cliquez sur le Bouton "Add" de "Game data Files", puis indiquez où se trouve le fichier FGD du mod.

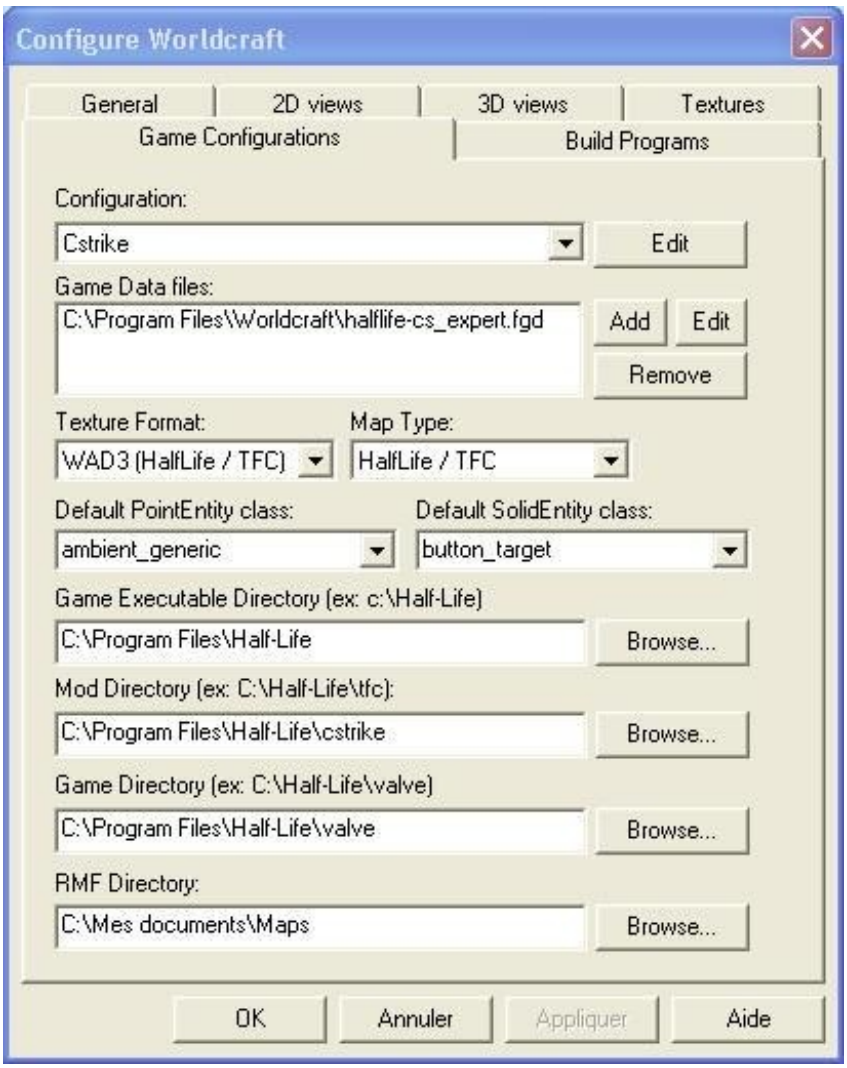
Normalement vous avez mis les FGD dans le dossier "fgd" qui se trouve dans le répertoire d'Hammer!

Hammer devrait alors lire les informations du FGD et afficher quelques informations supplémentaires un peu plus bas.

Il nous reste encore 4 lignes à remplir :

- Game executable directory : choisissez le répertoire où HL (Half-life) est installé.
- Mod directory : c'est le répertoire où est installé le mod. Pour HL, c'est "valve" ; pour Counter strike : "cstrike"...
- Game directory : il correspond au dossier de valve (ex. : C:\Half-life\valve).
- RMF Directory : c'est le répertoire où vous stockez vos maps en cours de fabrication.
  En effet, vous mapperez et enregistrerez votre map sous le format *.rmf, sous ce format elle ne fonctionne pas sous Half-Life et vous ne pourrez pas jouer dessus.
  Mais vous apprendrez plus tard ce qu'on appelle la compilation qui transforme votre map jouable!
  Moi, j'ai un dossier C:\Mes documents\maps. Vous faites comme vous voulez, du temps que c'est facile d'accès 

Répétez ces opérations depuis le début pour chaque mod qui vous intéresse. Voici ce que ça donne pour Counter Strike :



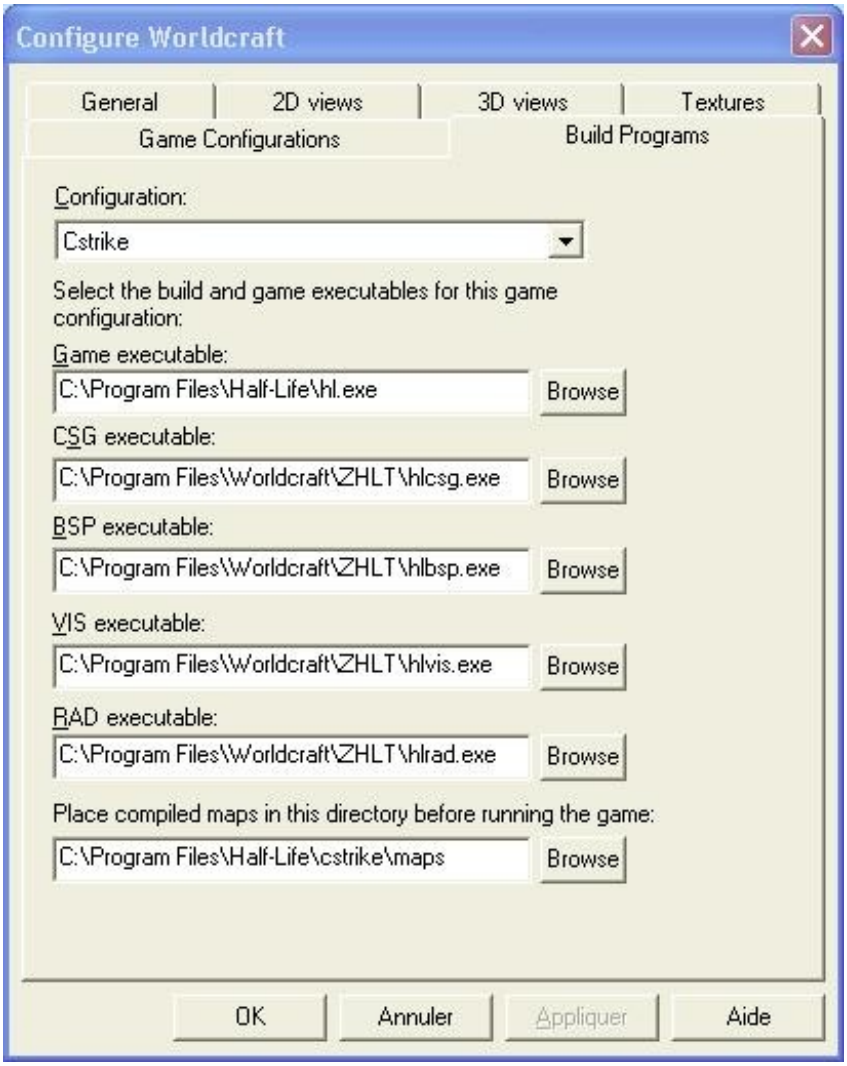
Onglet "Build Programs"

Intéressons-nous maintenant à cet onglet. Vous devez d'abord choisir le mod en haut, puis remplir toutes les cases. Une fois que c'est fait, recommencez pour chaque mod (bon, au début si vous configurez seulement Half-Life et Counter-Strike ça nous suffira).
On doit indiquer, de haut en bas :

- Game Executable : c'est l'exécutable de Half-life. Son nom est "hl.exe", et vous le trouverez dans le répertoire où vous avez installé Half-life (C:\Program Files\Half-Life\hl.exe par exemple).
- CSG Executable : c'est un des 4 compilateurs qui sont nécessaires pour créer un fichier BSP (une map compilée!). Vous le trouverez dans le répertoire où vous avez installé les ZHLT. Son nom est "hlcsg.exe".

  Attention ! Hammer a installé d'autres compilateurs, plus anciens que les ZHLT (qcsg.exe par exemple). Vous ne DEVEZ pas utiliser ceux-là, même si ça fonctionne. Les ZHLT sont meilleurs, ils vous aideront à retrouver des erreurs plus facilement.
- BSP Executable : un autre compilateur. Son nom est "hlbsp.exe".
- VIS Executable : le troisième compilateur, "hlvis.exe".
- RAD Executable : le dernier compilateur : "hlrad.exe" (vous l'auriez deviné je suppose ;))
- Place compiled maps in this directory before running the game : c'est le répertoire des maps du mod que vous devez indiquer. Pour HL, c'est "Half-life\valve\maps", pour Counter-Strike "Half-life\cstrike\maps", etc etc...

Voilà ce que ça doit donner pour Counter-Strike :



Onglet "Textures"

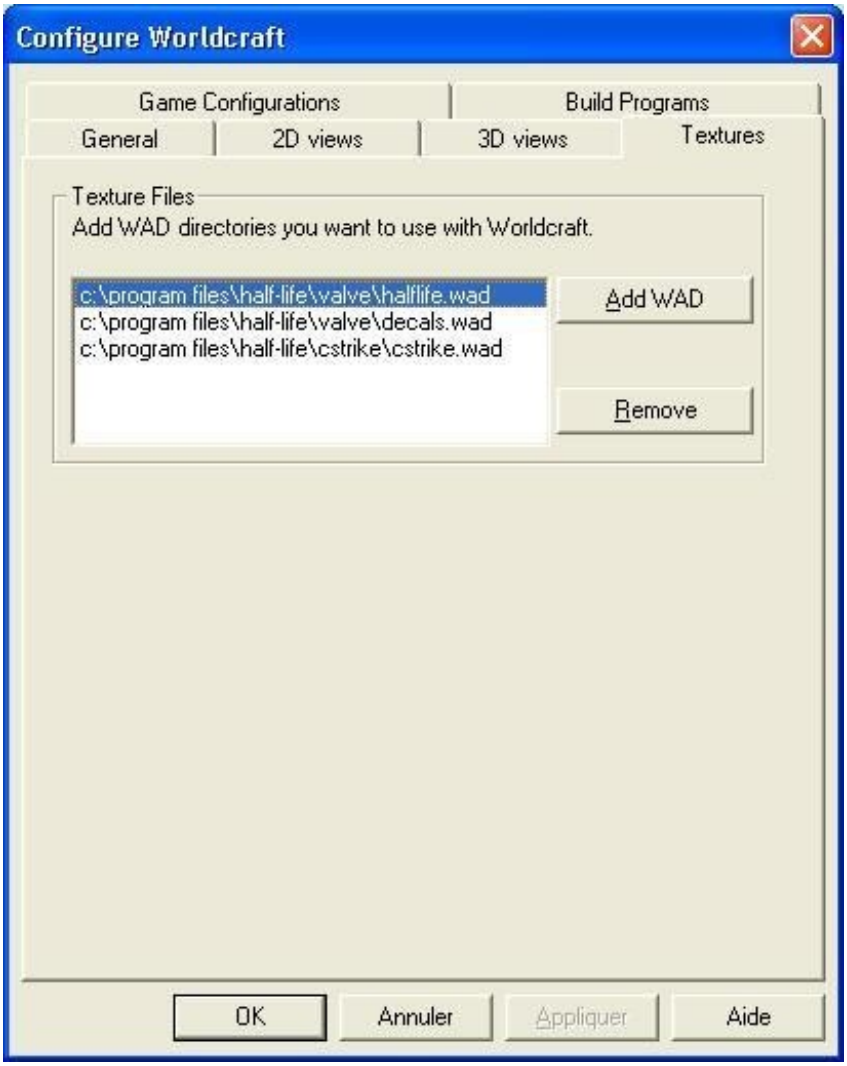
Enfin, le dernier onglet que l'on configurera (les autres ce n'est pas vraiment la peine d'y toucher) : les textures. Je vous ferai un peu plus loin un chapitre entier sur les textures, mais pour le moment vous avez juste besoin de savoir qu'une texture est un dessin répété en mosaïque, servant à colorier les murs, le sol... On a par exemple des textures d'eau, de bois, de porte...

Les textures sont enregistrées dans des fichiers à l'extension *.wad. Vous devez donc indiquer quelques WAD à Hammer pour qu'il vous propose un choix de textures lorsque vous ferez votre map.

Vous trouverez beaucoup de fichiers WAD, mais pour l'instant je vous demanderai de mettre seulement :

- Celui de Half-Life : Half-life\valve\halflife.wad
- Celui des decals (on verra ce que c'est ne vous inquiétez pas) : Half-life\valve\decals.wad.
- Et pourquoi pas celui de Counter-Strike, si vous faites une map pour ce mod : Half-life\cstrike\cstrike.wad

Vous devriez donc avoir chargé au moins ces fichiers dans l'onglet "Textures" :



Voilà. Hammer est correctement installé et configuré : vous êtes enfin prêt à créer des maps. Nous allons nous y intéresser dans le prochain chapitre...

Les bases de toute map

Dans ce chapitre très important, vous allez en apprendre un peu plus sur Half-Life et son moteur graphique. Hé oui, on ne joue plus !

Vous allez devoir connaître des détails techniques sur Half-Life pour pouvoir maîtriser pleinement le "mapping".

C'est parti !

Le moteur de Half-Life

Tout d'abord, même si ça peut paraître évident, il faut avoir joué à Half-Life pour utiliser Hammer. Il faut que vous ayez fini le jeu au moins une fois pour avoir une idée de ses capacités... Et si ça fait un moment que vous n'y avez plus joué, je ne saurais trop vous recommander de recommencer l'aventure.

Mais bon, je ne vais quand même pas trop vous en demander ;).

Premier constat : Half-Life est un vieux jeu. Vieux, certes, mais il vieillit bien.
Pourquoi ? Parce qu'il est facile de créer des maps et même des mods. Du coup, il existe de nombreux mods, dont la plupart sont vraiment excellents (je pense à Counter-Strike et à Team Fortress Classic en particulier, mais il y en a d'autres).

Toutefois, son moteur graphique vieillit. Discrètement, mais il vieillit.
Vous ne savez pas ce qu'est le moteur graphique d'un jeu ? Imaginez que c'est son coeur : c'est ce qui permet de le faire fonctionner. Sans moteur, vos maps ne servent à rien. Le moteur définit les graphismes du jeu et "plonge" le joueur dans un univers plus ou moins réaliste, plus ou moins complexe, plus ou moins beau...
De nombreux mods améliorent considérablement les graphismes d'Half-Life (Half-Life Gold, Paranoia, Half-Life : FX, ARRANGEMENT, et bien d'autres...)

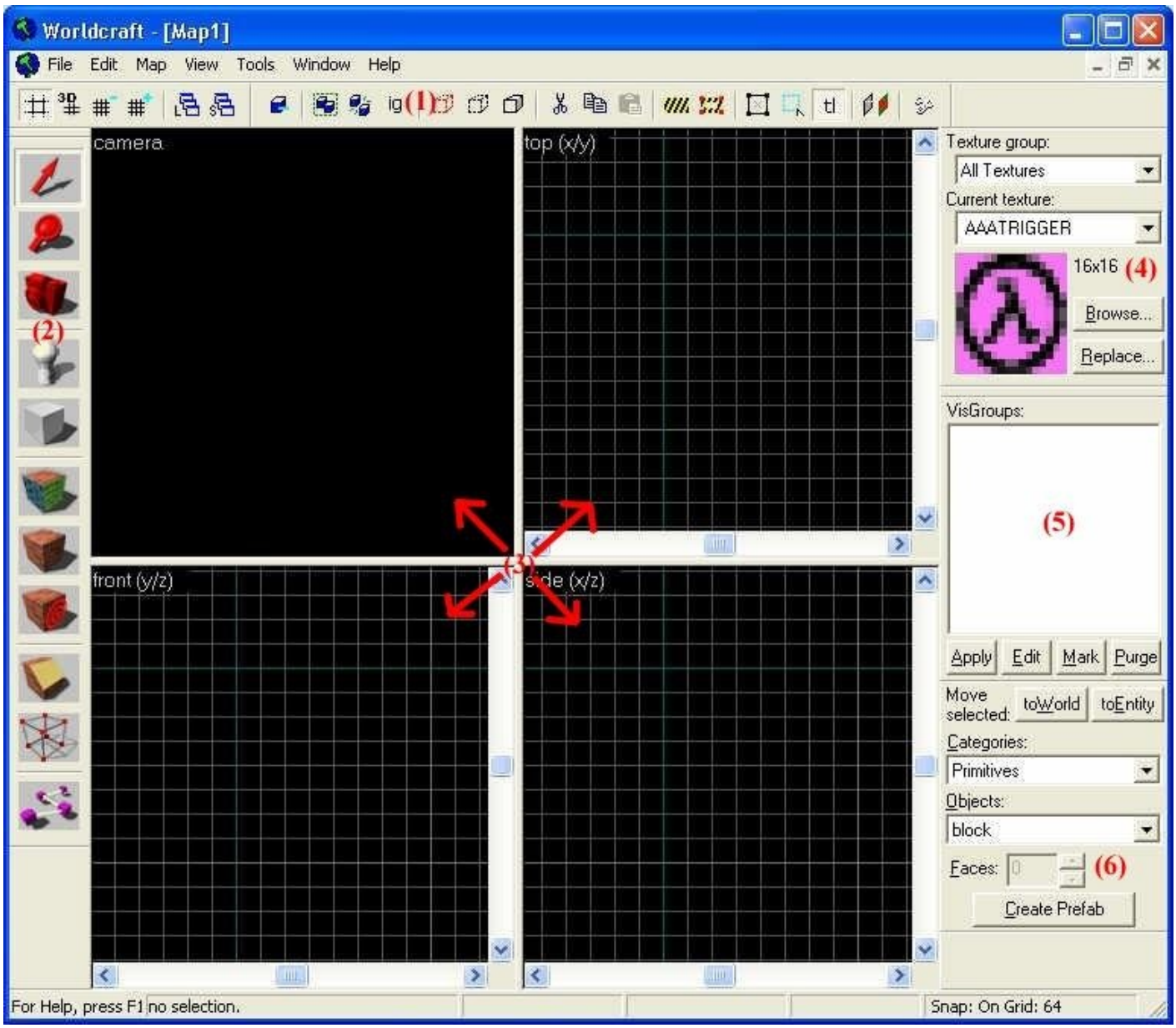
Et peu de gens le savent : le moteur de Half-Life n'a pas été créé de toutes pièces. A l'origine, il s'agit du moteur de... Quake II !
Valve (les développeurs du jeu) a repris en quelque sorte à ID Software le moteur de Quake II. Ils l'ont largement modifié (ça n'a presque plus rien à voir), mais tout de même... ça fait vieux.

Le moteur de Half-Life est donc loin d'être le moteur ultime. Il ne faudra pas trop le brusquer, rester raisonnable au début lorsque vous mapperez.

Quoiqu'il en soit, il est fortement conseillé de commencer dès la première map un vrai "projet". Faire un plan au brouillon, réfléchir au gameplay etc... Si vous faites une map pour CS par exemple, évitez à tout prix les speedmaps, qui ne vous apprendront pas grand chose sur le mapping. Je vous conseille dès le début de faire une vraie "de_", "cs_", "as_", "es_"... Mais surtout évitez les "awp_", "ka_", ou encore "csde_"...

L'environnement de travail

Passons aux choses sérieuses. **L'environnement de travail** est l'espace dans lequel vous allez créer vos maps. Il s'agit ni plus ni moins de Hammer, tout simplement.
Voici à quoi doit ressembler votre environnement de travail :



L'objectif de cette section est de vous décrire chacun de ces boutons qui apparaissent à votre écran. Ainsi, vous saurez avec quoi vous travaillez, et vous serez prêts à attaquer le mapping pur et dur

Avant toute chose, il est recommandé de mettre la résolution de son écran à 1024 x 768 pixels au moins. Vous disposerez ainsi d'un espace plus grand et plus confortable pour créer vos maps.
Pour ce faire, cliquez avec le bouton droit de la souris sur le bureau, puis cliquez (avec le bouton gauche cette fois sur "Propriétés". Dans l'onglet "Paramètres", faites glisser le curseur jusqu'à "1024 par 768 pixels" (si ce n'était pas déjà le cas).

Pour obtenir la même fenêtre que moi, cliquez sur le menu File / New. Vous passerez toujours par là pour créer une nouvelle map.

Un truc super important : lorsque vous enregistrerez votre map sous Hammer en faisant File / Save ou Ctrl+S, Hammer crée un fichier .rmf (c'est votre map) . Vous DEVEZ toujours ouvrir et travailler sous Hammer avec les .rmf.
Comme vous le verrez plus tard, Hammer génère aussi un .map qui ressemble beaucoup au rmf, mais il ne faut pas travailler dessus (beaucoup de mappeurs ont perdu tout leur travail en ouvrant le .map au lieu du .rmf !)

Nous allons, dans un premier temps, analyser chacune des parties de la fenêtre de Hammer. Reprenons l'image ci-dessus : j'ai placé à plusieurs endroits de la fenêtre des numéros (de 1 à 6), correspondants à des parties de Hammer. A quoi correspondent-elles ?

- **(1)** Tout en haut (juste en-dessous des menus), il y a la barre d'outils. Elle est composée de plusieurs boutons permettant d'accéder rapidement aux fonctions les plus courantes du programme :
  Le premier bouton doit rester enfoncé, il permet d'afficher une grille pour les vues 2D. Le bouton suivant affiche une grille dans la vue 3D, il n'est pas très utile. Les deux boutons qui suivent sont importants, ils modifient la précision de la grille (nous y reviendrons). Le 5° et le 6° bouton n'ont pas d'intérêt, vous ne les utiliserez pas.

Tous les autres boutons sont importants et permettent d'effectuer des fonctions complexes dans Hammer. Nous les verrons plus tard, lorsque vous en saurez suffisamment. Le dernier, "Run map !", vous permet de compiler votre map. Ne foncez pas dessus tête baissée, un chapitre entier y sera consacré, car ce n'est pas une mince affaire.

N'oubliez jamais que l'on peut retrouver ces fonctions dans les menus, et que d'autres fonctions moins courantes (mais tout aussi importantes) se retrouvent dans les menus.

- **(2)** C'est une autre barre d'outils qui doit normalement se trouver sur votre gauche. Qu'a-t-elle de différent ? Ces boutons sont plus gros car beaucoup plus souvent utilisés. Pour la plupart, ils permettent d'effectuer des fonctions de bases, que nous étudierons dans le prochain chapitre.
- **(3)** Au centre, c'est la partie la plus importante : votre map. Vous avez sûrement constaté qu'elle est divisée en 4 parties, ce qui peut vous avoir fait peur si vous n'avez jamais créé de mondes en 3D. Pourquoi 4 parties, et pas une seule ?

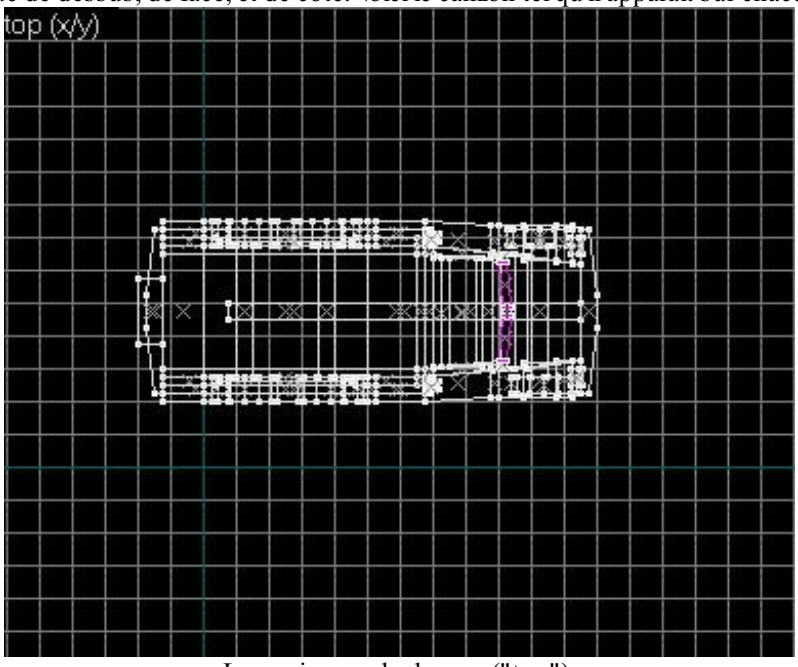
Si le monde avait été en 2D, ç'aurait été plus simple... Votre écran ne peut afficher que 2 dimensions. Oui oui, si votre jeu est en 3D, l'écran vous donne juste l'illusion d'une troisième dimension ; mais lui, il reste bel et bien en 2 dimensions (à moins d'avoir des lunettes 3D ou un écran importé du futur ;).
Donc, un jeu en 2D avec un écran 2D, c'est très simple. Mais un jeu 3D avec un écran 2D ? Ca se corse... Il a fallu diviser le monde en 3 vues 2D : ce sont les fenêtres "Top", "Front" et "Side". Il y a donc une vue de Dessus, de Face, et de Côté.

Si vous aimez la géométrie dans l'espace, sachez que ces 3 vues sont 3 plans différents, de coordonnées (x/y) pour "Top", (y/z) pour "Front", et (x/z) pour "Side". Je ne dis pas qu'il faut absolument vous replonger dans vos cours de Maths ;), mais je vous le conseille si vous voulez bien comprendre les fonctions avancées de Hammer...

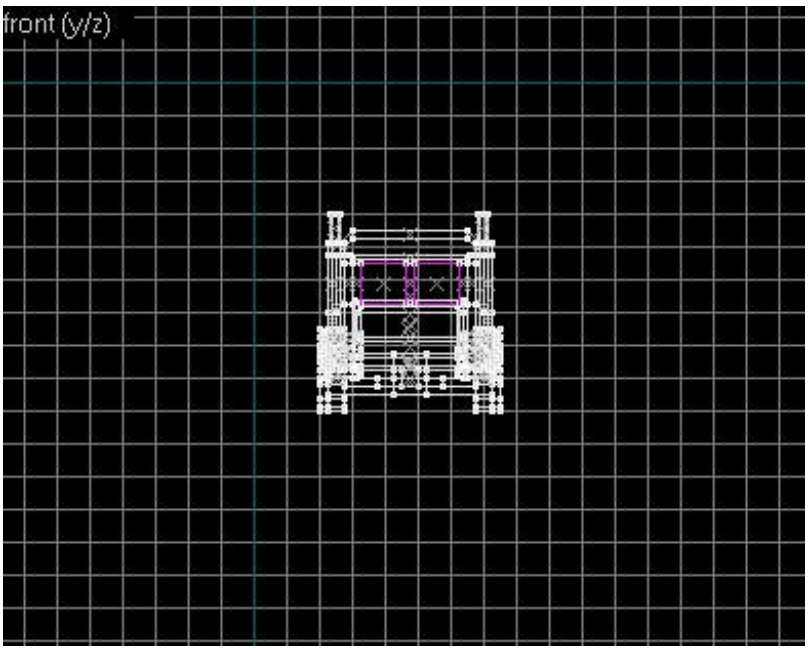
Ces 3 vues réunies forment un monde en 3D (coordonnées x/y/z), que la vue "Caméra" restitue en temps réel ! Vous ne travaillerez presque jamais sur la vue 3D (c'est juste un aperçu) : la plupart du temps ce sont les vues 2D qui vous servent à créer votre map. C'est un peu déroutant au début, mais c'est en fait super-simple.

Les vues 2D

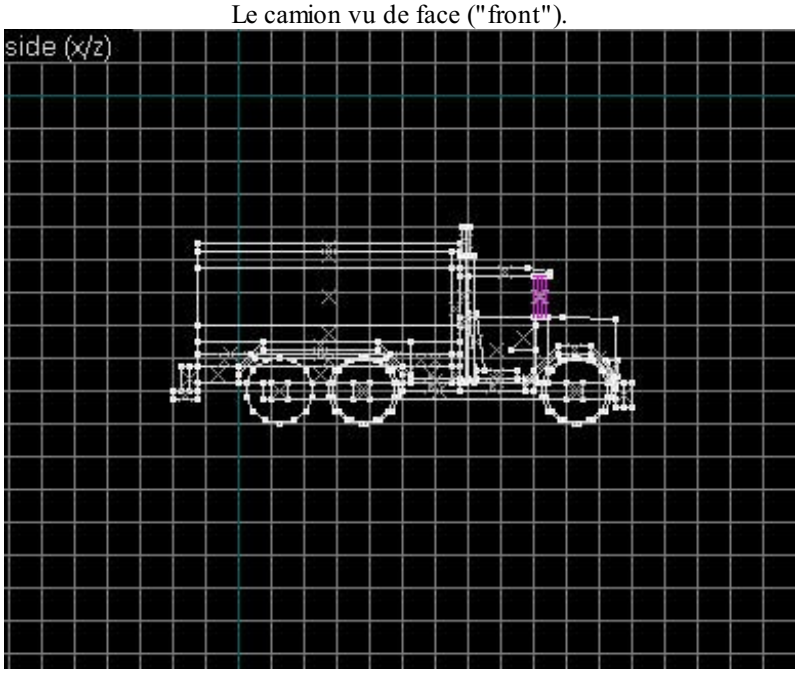
Si l'explication théorique que je viens de vous donner est trop complexe pour vous, alors voici un exemple. J'ai commencé une nouvelle map vierge (File / New), et, grâce aux vues 2D, j'ai créé un camion (ce qui n'est pas évident, mais là n'est pas la question).
J'ai donc eu besoin d'une vue de dessus, de face, et de côté. Voici le camion tel qu'il apparaît sur chacune de ces vues :



Le camion vu de dessus ("top").



Le camion vu de face ("front").



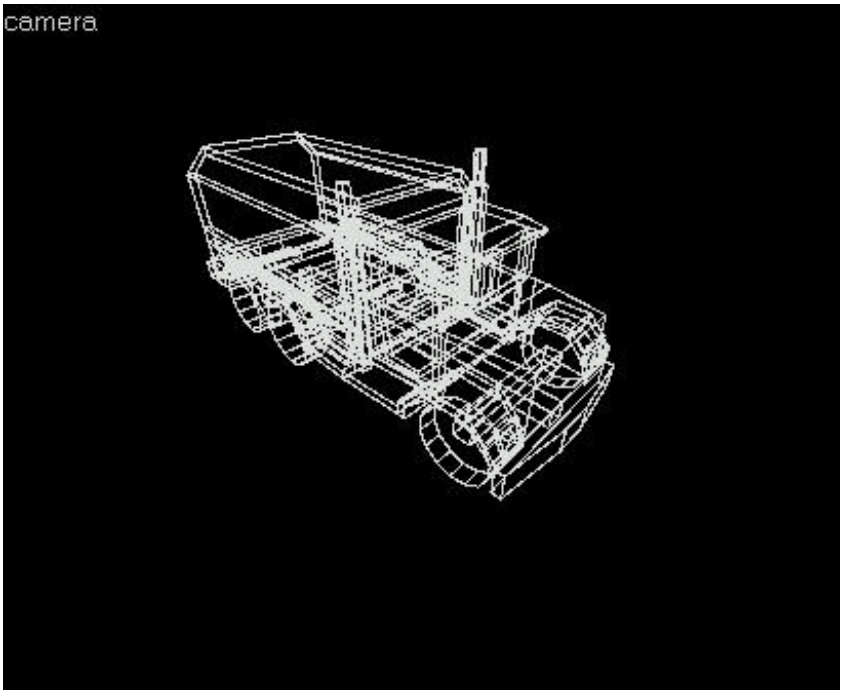
Le camion vu de côté ("side").

La vue 3D

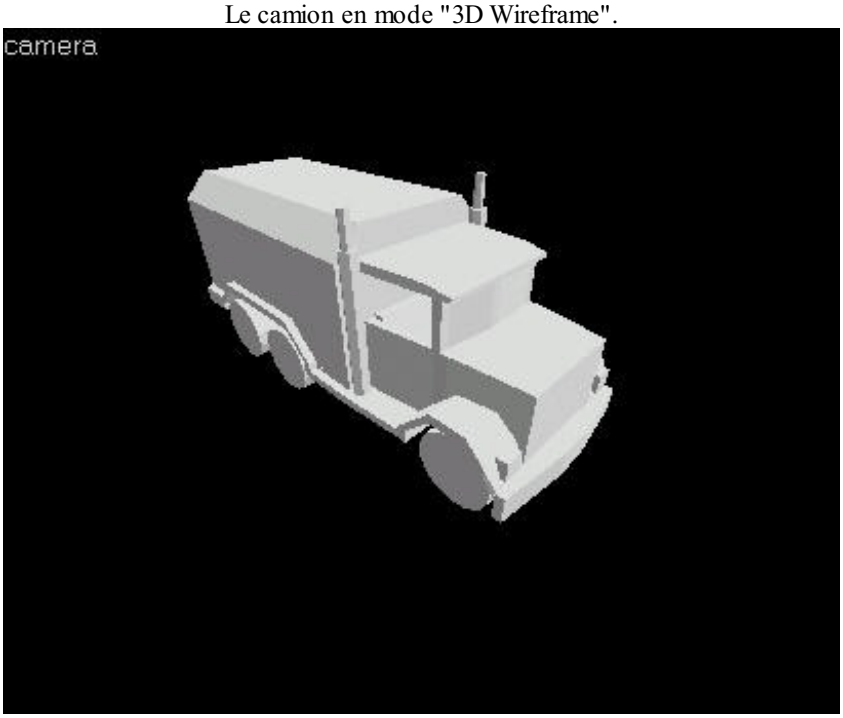
En combinant ces vues en 2D, on peut obtenir un objet en 3D : le camion.
C'est le rôle de la caméra de nous montrer cet objet en 3D. La caméra sous Hammer peut afficher la 3D de 3 manières différentes. Pour passer de l'une à l'autre, allez dans le menu "View", puis cliquez sur le mode de votre choix :

- "3D Wireframe" : ce mode n'affiche que les arêtes des cubes. On peut donc voir à travers les murs. Il est assez difficile de se repérer dans ce mode car les objets ne sont pas "remplis" : ils sont "vidés" de leur contenu.
- "3D Filled Polygons" : cette fois, tous les objets de votre map sont remplis. On ne peut plus voir à travers. Par contre, ils sont seulement coloriés avec une couleur unie.
- "3D Textured Polygons" : le mode le plus réaliste. Tout comme "Filled Polygons", les objets sont remplis. Mais ici, leurs faces affichent chacune la texture qu'elles possèdent (à la place d'une couleur unie).

La première est celle qui requiert moins de puissance à votre ordinateur, et la dernière est celle qui en requiert le plus. Toutefois, si votre PC ne date pas de l'Antiquité, vous n'aurez aucun problème à travailler avec le mode "Textured Polygons", ce qui est conseillé vu que c'est lui le plus réaliste. Sous le jeu, tout sera texturisé comme le mode "Textured Polygons", quelle que soit le mode avec lequel vous avez pu travailler.
Voyons maintenant comment le camion est affiché en fonction du mode choisi :



Le camion en mode "3D Wireframe".



Le camion en mode "3D Filled Polygons".

Le camion en mode "3D Textured Polygons".

Se déplacer dans la vue 3D

Si vous voulez vous déplacer dans la vue 3D, sachez que ça ne fonctionne pas du tout comme si vous étiez en train de jouer dedans :

- Primo : vous vous déplacez très vite (ce qui est plutôt pratique si vos maps sont grandes)
- Segundo : vous n'êtes pas un joueur mais une caméra. Vous avez donc la possibilité de traverser les murs (un peu comme le code de cheat "noclip"), et de voler dans les airs.
- Tercio : le déplacement est plus complexe. On n'utilise pas les habituelles touches de direction (Haut, bas, gauche, droite ; ou Z, S, Q, et D).

Pour vous déplacer, vous devez placer le curseur de la souris dans la vue 3D, puis maintenir une ou deux touches enfoncées pendant que vous cliquez avec un ou plusieurs boutons de la souris... Pas évident hein ?
Voici la liste des déplacements possibles :

- Espace + Clic gauche : en maintenant le bouton gauche de la souris enfoncé ainsi que la touche Espace, la caméra ne se déplace pas mais fait une rotation sur elle-même. Vous pouvez regarder autour de vous à 360°, en haut et en bas.
- Espace + Clic droit : avec le bouton droit, la caméra se déplace latéralement, de gauche à droite ou de haut en bas.
- Espace + Clic gauche + Clic droit ou Espace + Shift + Clic droit : bon, c'est un peu compliqué au début je reconnais ;). Vous avez 2 possibilités, moi je préfère la seconde (avec la touche "Shift" : elle se trouve en-dessous de la touche majuscule et représente une flèche vers le haut). Ce déplacement vous permet d'avancer et de reculer dans votre map, ce qui est très important.

Entraînez-vous ! Vous en aurez besoin...
Vous pouvez aussi en maintenant barre espace et clic gauche vous déplacer avec la molette!

- **(4)** En haut à droite, c'est la partie des textures. C'est ici que s'affiche la texture que vous utilisez (bois, gazon, carrelage...), et que vous pouvez sélectionner une texture différente. Nous reparlerons des textures dans un prochain chapitre.
- **(5)** La partie "VisGroups" est de loin la moins utilisée, ce qui ne veut pas dire que vous n'y toucherez jamais. Cela sert, en gros, à créer des groupes d'objets ayant une fonction commune... ce qui est assez complexe, et que nous ne verrons que dans un chapitre avancé de ce cours pour Hammer.
- **(6)** Enfin, cette dernière partie vous permet de créer une entité et de sélectionner le type de bloc à créer. C'est ce à quoi nous allons nous intéresser dès maintenant.

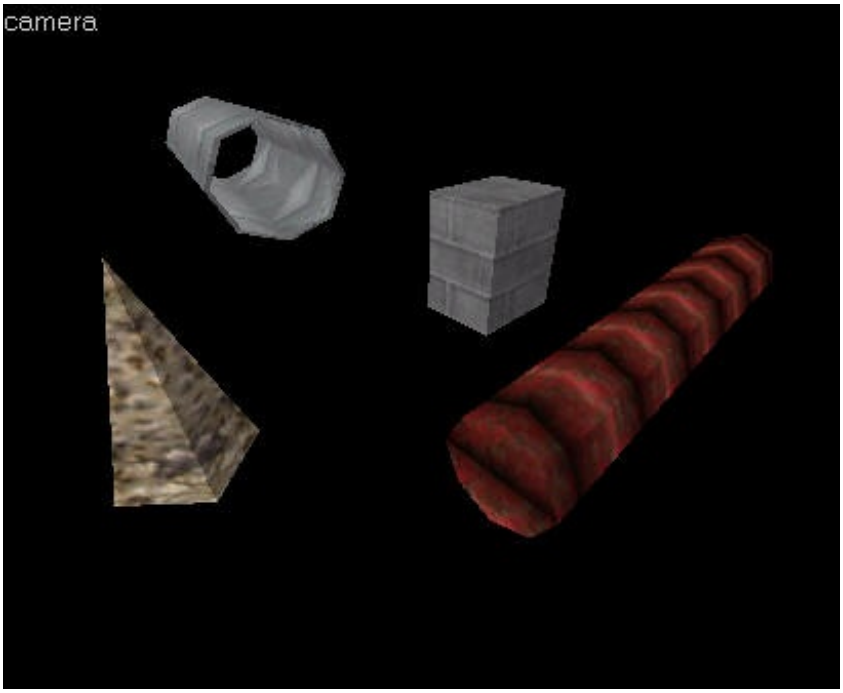
Les constituants d'une map

Ce qui suit est extrêmement important !

Nous allons essayer de répondre à la question : que puis-je mettre dans ma map, de quoi est constitué une map? En effet, pour l'instant la vue 3D affiche du vide, et les vues 2D le sont aussi ;). De quoi est constituée une map ?
Il existe 2 constituants différents :

Tout d'abord, il y a les **blocs**. Ce sont des cubes plus ou moins gros, pouvant prendre différentes formes (parallélépipède rectangle, cylindre, arcs...). Ils sont toujours inertes dans Hammer comme sous le jeu(impossible de les voir bouger ou de les déplacer). Ils constituent le squelette d'une map (c'est quelque part sa structure).

Ils possèdent obligatoirement une texture pour les décorer. Ils sont la base même de toute map : en effet, ils permettent de construire les murs, le sol, le plafond... Voici un aperçu avec la vue 3D de quelques blocs de différentes formes :



Enfin, il y a les **entités**. Vous vous demandez peut-être à quoi elles peuvent bien servir ces entités. Eh bien, les entités, c'est tout le reste. Ca fait un peu fourre-tout en effet
Qu'est-ce que c'est que "le reste" ? Il s'agit d'éléments permettant le bon fonctionnement de la map, tels que : la lumière, les spots, les caméras, les ennemis, les armes, les munitions, le son d'ambiance...

Pour la plupart, leur forme n'est pas clairement définie comme les blocs (exemple : la lumière, le son...), certains ont la capacité de bouger (les scientifiques, les ennemis, les boîtes...), d'autres peuvent déclencher des actions (les boutons...). Ce sont donc les organes de votre map, puisqu'elles permettent de lui donner vie.

Les entités sont très nombreuses et très diverses : on aura vite étudié les blocs en quelques chapitres, mais les entités nous occuperont pour tout le reste du cours. Elles seront classées par thèmes (la lumière, le son, les ascenseurs, les portes...), car il est impossible de les étudier par ordre alphabétique.

Dans la vue 3D, certaines entités apparaissent sous forme de petits dessins ou de cubes de couleur unies (ce qui est moins pratique). En fait, tout dépend du fichier FGD que vous utilisez (les entités de Counter-Strike ne sont pas les même que celles de Half-Life).

Notez qu'il y a en fait 2 sortes d'entités : la première, vous venez d'en voir un aperçu (c'est une entité-point). Quant à la seconde (entité-bloc), elle est un peu plus complexe et je ne vous en parle pas de suite pour ne pas vous embrouiller : c'est un bloc transformé en entité pouvant ainsi bouger ou faire d'autres choses.

Pour l'instant, vous ne travaillerez que sur les blocs, vous ne pourrez donc pas tout de suite créer une vrai map, car les entités sont indispensables pour créer une map potable!
Voilààà ! On y est presque : maintenant que vous savez ce qu'il y a dans une map, vous êtes parés pour faire un peu de pratique. Dans le prochain chapitre, on va travailler sur les blocs, et on va en apprendre un peu plus sur la barre d'outils de gauche.

Les blocs (A : création)

Bon, assez discuté. Maintenant on passe à la pratique

Vous l'avez vu, les blocs sont le squelette de votre map, et les entités les organes (les muscles...). Nous nous intéresserons donc au squelette pour commencer.

Créer et cloner

Avant tout, demandez à Worldcraft une nouvelle map vierge (menu File / New). Nous allons nous entraîner à créer des blocs dans une map vide.
Si vous avez configuré Worldcraft pour plusieurs mods différents de Half-Life (voir le premier chapitre), il vous demandera de choisir la configuration que vous voulez. Ainsi les paramètres du mod (le fichier FGD) seront chargés. La fenêtre devrait ressembler à ça :

Pour ce que nous allons faire, le choix du mod n'a strictement aucune importance. Les blocs sont communs à tous les mods de Half-Life (ce qui n'est pas le cas pour les entités). Là, je vais choisir ma configuration Cstrike (Counter-Strike), mais j'aurais très bien pu choisir Half-Life...

Maintenant, les 4 vues doivent afficher du vide. Toutes vos maps sont vides au départ.

Ironie du sort : pour que l'on puisse jouer sur vos maps, il faudra ab-so-lu-ment cacher ce vide au joueur. Je vous en reparlerai plus tard, mais sachez dès à présent que ce vide est votre ennemi, et qu'il est INTERDIT de le montrer au joueur. C'est la première et principale contrainte que doit remplir votre map avant d'être compilé!

Pour l'instant, notre but n'est pas de faire des maps jouables, mais des blocs de toutes formes. Ne vous préoccupez donc pas du vide et concentrez-vous sur la création de blocs.

Au début, la caméra est en mode "3D Wireframe". Vous devriez la faire passer en "3D Filled Polygons" (dans le menu View, comme vous avez vu). Si je vous demande exceptionnellement de ne pas vous mettre en "3D Textured Polygons", c'est que ce mode afficherait les textures des blocs et ça ne doit pas vous perturber pour l'instant.

Si vous déplacez un peu la caméra en arrière, vous allez vous rendre compte qu'il y a déjà quelque chose dans votre map. Vous devriez voir ceci :

C'est le repère vectoriel de votre map... Je vois d'ici vos têtes vertes Rassurez-vous, comme dans tout monde en 3D, le mappeur (vous en l'occurence) a besoin d'un repère. Mais il n'est pas question ici de Maths
Cet endroit est juste le point de départ de la caméra (lorsque vous chargez votre map pour la première fois).

Créer un bloc

Regardez la barre d'outils à gauche : c'est ici que se trouvent les outils dont nous avons besoin :


Pour l'instant, vous êtes en mode "Selection Tool". C'est le mode normal : votre curseur a la forme d'une flèche.


Je vais vous demander de sélectionner le "Bloc Tool". C'est lui qui nous permet de créer des blocs.

Si vous passez la souris sur une des vues 2D, votre curseur doit avoir changé de forme : c'est une croix accompagnée d'un rectangle.

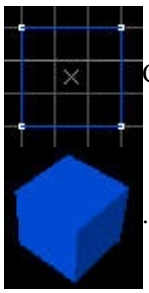
Comme je vous l'ai déjà expliqué, c'est dans les vues 2D que nous allons travailler. Oui, mais par laquelle commencer ? En ce qui me concerne, je préfère utiliser d'abord la vue "top" (celle de dessus) pour commencer mon bloc, puis j'utilise soit la vue "front", soit la vue "side" pour compléter la forme du bloc (sinon il serait tout plat !).

Voici les étapes à suivre pour créer un bloc simple, ayant la forme d'un cube :

- Pointez le curseur dans la vue "top".
- Cliquez et maintenez le bouton de la souris enfoncé.
- Déplacez la souris de façon à créer un bloc d'une taille de 3 x 3 carreaux.
- Lâchez le bouton.
- Pointez le curseur dans la vue "front".
- Agrandissez le bloc de manière à ce qu'il aie aussi une taille de 3 x 3 carreaux.
- Vérifiez que le bloc est semblable dans la vue "side".
- Tapez "Entrée" pour valider la création de votre bloc.

Le curseur de votre souris doit se trouver dans une vue 2D au moment où vous tapez "Entrée", sinon le bloc ne sera pas créé ! C'est une erreur courante quand on débute, alors souvenez-vous en.

Si à un moment vous souhaitez annuler la création de votre bloc (avant d'avoir tapé "Entrée"), il suffit d'appuyer sur la touche "Echap" en haut à gauche du clavier.
Si vous avez déjà tapé "Entrée", il faudra appuyer sur la touche "Suppr" pour le supprimer.


Chacune des vues 2D devrait afficher un carré comme celui-ci...

... tandis que la caméra affiche un cube parfait.

Si votre cube ou votre carré n'a pas la même couleur que moi, ça n'a absolument aucune importance. La caméra en mode "3D Filled Polygons" attribue des couleurs différentes à chaque fois.

Bien entendu, vous n'êtes pas obligés de faire des cubes : rien ne vous empêche de faire des rectangles plus ou moins gros dans chacune des vues. Entraînez-vous à créer des blocs de toutes sortes pour vous habituer à travailler sur les 3 vues 2D.

Sélectionner un bloc

Avec le "Selection Tool" (la souris), vous pouvez sélectionner le bloc (sans blague !).
Pour sélectionner un bloc, vous pouvez vous servir soit d'une vue 2D, soit de la vue 3D (la caméra).

- Dans la vue 3D, cliquez simplement sur un des blocs que vous voyez pour le sélectionner (rappelez-vous : il faut que le "Selection Tool" soit activé !).
- Dans une vue 2D, vous pouvez cliquer sur un des côtés du bloc (les segments bleus dans mon cas) ou bien sur la croix au centre. Ailleurs, vous ne sélectionnerez rien !

Sélectionner plusieurs blocs

Pour sélectionner plusieurs blocs, c'est à peu près pareil... Il faut juste laisser la touche "Ctrl" (Control) enfoncée pendant que l'on clique sur les blocs.

Plus rarement utilisée, il existe une méthode (toujours avec le "Selection Tool") pour sélectionner une partie de la map. Cette fois, vous devez obligatoirement vous placer dans une vue 2D. Cliquez et maintenez le bouton gauche de la souris enfoncé jusqu'à créer un cadre de la taille souhaitée. Agrandissez-le dans toutes les vues 2D comme bon vous semble, puis tapez "Entrée". Les blocs qui se trouvaient dans cette zone sont maintenant sélectionnés.

Le clonage

Bah ouais, ça s'appelle comme ça j'y peux rien Le clonage est une technique qui consiste à reproduire les mêmes blocs... Un peu comme le copier-coller.

D'ailleurs, le copier-coller marche ! Sélectionnez un ou plusieurs blocs, puis faites Ctrl+C pour copier, et Ctrl+V pour coller.

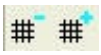
Mais il y a plus simple encore : la touche Shift. Cette touche se trouve en-dessous du cadenas, elle est représentée par une flèche.
Sélectionnez un bloc, laissez la touche Shift enfoncée, déplacez la souris jusqu'à l'endroit souhaité et lâchez la touche Shift. C'est tout !

Selon le cas, si votre PC n'est pas très puissant et que vous clonez un nombre important de blocs à la fois, ça risque de ramer à mort. Soyez patients et attendez que l'affichage s'actualise avant de lâcher la touche Maj.

Zoom & précision de la grille
Précision de la grille

Maintenant que vous savez créer des blocs simples, vous aurez peut-être besoin de plus de précision. La grille vous empêche "d'affiner" la taille de vos blocs.
L'idée est donc d'avoir une grille plus fine, pour plus de précision.

Ne confondez pas le zoom et la précision de la grille. Lorsque vous augmentez la précision, vous ne zoomez pas ! Vous avez simplement la possibilité de faire des blocs plus précis.

 Nous utiliserons ces boutons de la barre d'outils du haut.

Le premier permet d'avoir une précision plus importante, tandis que le second réduit la précision.

En bas de la fenêtre de Worldcraft, un texte vous indique le niveau de précision : 

. Le niveau le plus précis est 1. En général, je travaille au niveau 8, et j'augmente des fois la précision jusqu'à 1 lorsque je dois créer des blocs très petits.
A vous de voir selon le cas la précision qu'il vous faut. Je pense toutefois que vous n'irez jamais au-dessus de 16 unités, parce qu'après c'est trop gros et ça devient ingérable.

Ces unités dont je vous parle sont des unités propres à Worldcraft et Half-Life. Ce ne sont ni des mètres, ni des centimètres...

Zoom

Le zoom, ça vous connaissez. Ca permet de vous rapprocher de l'objet et de le voir de plus en plus gros. Notez qu'en général on utilise conjointement le zoom et la précision de la grille pour obtenir une précision maximale à l'atome près

 Le "Magnify Tool" se trouve dans la barre d'outils de gauche. C'est la loupe permettant de zoomer.

L'utilisation est très simple :

- Clic gauche : zoom avant.
- Clic droit : zoom arrière.

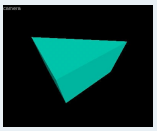
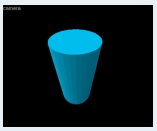
Les différentes formes de blocs

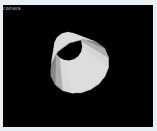
Les blocs que vous avez appris à créer ressemblent tous à un cube. Ce serait dommage de n'utiliser que cette forme, alors que le moteur de Half-Life en supporte bien d'autres...

Quelles autres formes peut-on créer ? Worldcraft vous laisse le choix entre 5 formes d'objets différentes. Vous pouvez sélectionner celle que vous voulez à droite de la fenêtre, dans la section "Objects", comme le montre l'image ci-dessous :

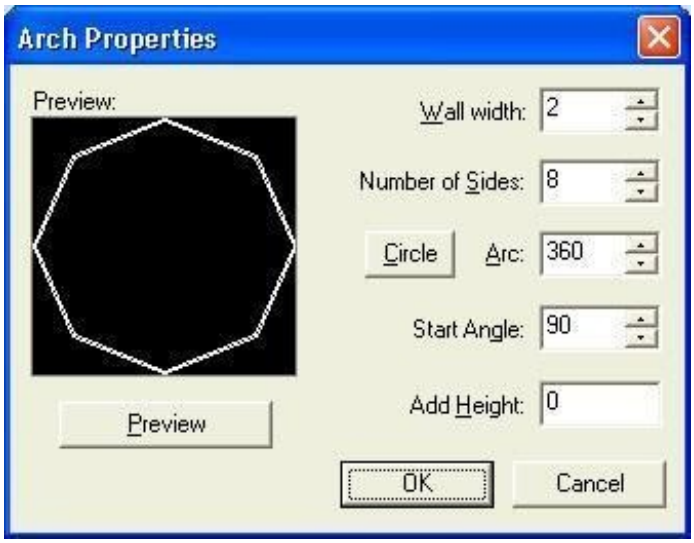
Notez que les boutons qui se trouvent à côté ne nous intéressent pas pour le moment. Cela permet d'insérer des Prefabs dans votre map. Nous aurons l'occasion d'en reparler plus tard.

Vous avez le choix entre ces 5 formes :

	**block** : c'est celle que nous venons d'utiliser. Elle permet de créer des cubes, des rectangles... C'est de très loin la forme que l'on utilise le plus (99 % du temps).On s'en sert pour faire le sol, les murs, les caisses, les chaises, les trottoirs etc...
	**wedge**: ce sont des triangles rectangles. Cette forme est très rarement utilisée.En fait, on peut l'obtenir différemment en utilisant une autre méthode que vous verrez dans le prochain chapitre. Ce sera plus simple et encore plus précis.
	**cylinder** : ce sont des cylindres. On utilise assez souvent cette forme, qui permet de varier les décors. On peut en faire des pylônes, des tuyaux...Vous pouvez augmenter le nombre de faces pour rendre le cylindre encore plus arrondi : le champ "Faces" accepte jusqu'à 32 faces. N'en abusez pas quand même, sinon votre map pourrait faire ramer votre PC.Par exemple, quand le cylindre est gros, il vaut mieux qu'il aie plus de faces pour que ça ne soit pas trop moche ; quand il est petit, il est inutile de mettre plus de 8 faces.

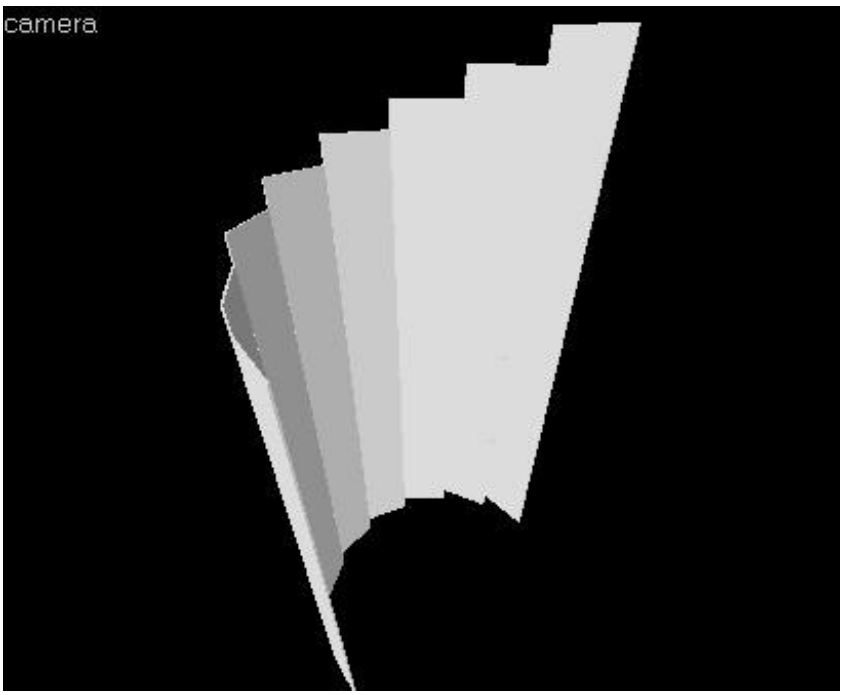
	**spike** : un cône, dont vous pouvez augmenter le nombre de faces. On l'utilise plus rarement, mais c'est une forme quand même pratique que j'aime bien :).Si vous pensez ne pas en avoir vu beaucoup dans le jeu Half-Life, détrompez-vous ! Les "spike" sont rarement positionnés comme sur cette image : ils sont plutôt petits et permettent d'affiner un objet "cylinder" par exemple.
	**arch** : la forme la plus complexe. Elle a été créée spécialement pour le moteur d'Half-Life. Cela ressemble à cylindre, mais creux.Le plus souvent, on s'en sert pour faire des tuyaux à travers lesquels le joueur peut passer. On l'utilise de temps en temps, surtout pour des maps Half-Life (c'est plus rare pour Counter-Strike). D'ailleurs, vous pouvez en voir dans la map d'entraînement de Half-Life...Lorsque vous tapez "Entrée" pour valider le bloc, Worldcraft ne le crée pas tout de suite. Il affiche une nouvelle fenêtre que nous allons voir...

La création d'un bloc de forme "arch" étant très complexe, vous devez rentrer certaines informations pour pouvoir le créer. La fenêtre ci-dessous devrait s'afficher :



A gauche, une image vous donne un aperçu de ce que vous allez créer (c'est très pratique puisqu'après avoir cliqué sur OK vous ne pourrez plus revoir cette fenêtre). Cliquez sur le bouton "Preview" pour mettre à jour l'affichage.
A droite, les paramètres que vous pouvez modifier :

- Wall width : c'est l'épaisseur du mur de votre tuyau ("arch"), en unités de Worldcraft. Généralement, les murs ne sont pas très épais. A vous de tester pour voir si ce n'est pas trop gros...
- Number of Sides : le nombre de côtés de votre tuyau. Comme pour un cylindre : augmentez cette valeur si votre bloc est gros, et diminuez-la s'il est petit.
- Arc : la forme "arch" ne permet pas seulement de faire des tuyaux entiers en forme de cercle, on peut aussi faire des arcs de tuyaux ;). Si vous voulez un cercle (c'est plus fréquent), alors cliquez sur le bouton "Circle" ou tapez 360°.
- Start Angle : ce n'est utile que si vous créez un "arch" qui a une forme d'arc (et non pas de cercle). Dans ce cas, vous pouvez faire pivoter l'arc pour le positionner comme vous voulez.
- Add Height : l'"arch" étant en fait formé de plusieurs petits "blocs", il est possible de déplacer chacun d'eux. En mettant une valeur supérieure à 0, vous pouvez faire un effet d'escalier, comme ceci :



Ouf ! Voilà, vous savez créer des blocs de toutes formes !
Mais vous ne savez pas tout sur les blocs : on peut encore modifier leur structure, les tourner, les couper, les étirer etc... Toutes ces tortures vous sont dévoilées dans le prochain chapitre

Les blocs (B : modification de la structure)

Au programme de ce chapitre : les fonctions basiques et avancées de restructuration

En clair, lorsque votre bloc a été créé, il est encore possible de le modifier : découpage, rotation, carving etc... sont autant de fonctions que propose Hammer pour faire varier la forme de vos blocs... et rendre vos niveaux bien plus originaux !

Là encore, laissez la caméra en mode "3D Filled Polygons", car les textures ne doivent pas vous déranger. On n'en parlera que dans le prochain chapitre.

Fonctions simples

Vous avez la possibilité de faire subir quelques modifications très simples à vos blocs. Je dis très simples parce que vous n'aurez pas de difficulté à imaginer à quoi ça sert. Ce sera plus difficile ensuite lorsque je parlerai des fonctions avancées de Hammer...


Après avoir créé un bloc (de forme "block" pour commencer), sélectionnez-le avec le "Selection Tool"

A partir de là, vous avez 3 fonctions basiques :

- Le redimensionnement
- La rotation
- L'étirement

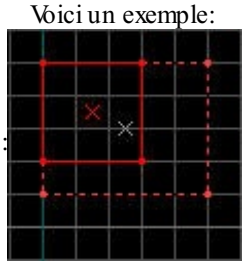
Le redimensionnement

Mouarf, c'est tellement simple que j'hésite à vous en parler ;o) Mais bon, après tout, vous êtes sur le Site du Zér0, et il est en mon devoir de T-O-U-T vous expliquer

Si vous trouvez votre bloc trop gros ou trop petit, le redimensionnement vous permet de modifier sa taille. Oui, vous allez me dire que vous pouviez le faire avant la création du bloc, mais l'intérêt ici c'est que vous pouvez le faire après sa création ! Pour cela :

1. Placez le curseur de la souris dans la vue 2D qui vous intéresse.
2. Sélectionnez le bloc.
3. Cliquez sur un des 8 carrés situés autour du bloc.
4. Maintenez le bouton de la souris enfoncé tout en déplaçant le curseur jusqu'à l'endroit voulu.
5. Lâchez le bouton.

Voici un exemple:



Et voilà ! C'est bon, votre bloc a été redimmensionné !

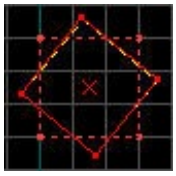
La rotation

Bon, là c'est une fonction un peu plus intéressante : vous pouvez faire tourner un objet sur lui-même. Cela vous permettra de donner une impression de désordre dans vos maps, ce qui est très important si vous ne voulez pas que tout soit parfaitement aligné au millimètre près

Pour utiliser cette fonction, vous devez cliquer une deuxième fois sur le bloc après l'avoir sélectionné. Les carrés qui se trouvaient autour se sont transformés en ronds. Utilisez alors la même méthode que tout à l'heure pour le redimmensionnement.

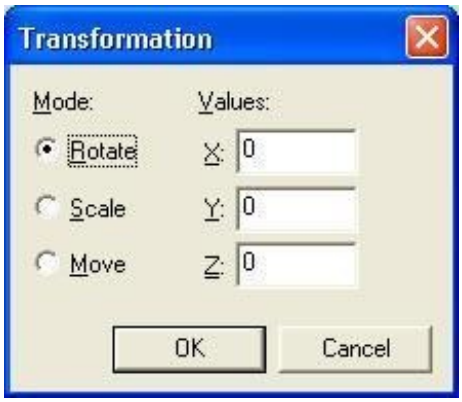

Méfiez vous ! Selon la vue 2D sur laquelle vous faites la rotation, le bloc ne tournera pas de la même manière !

Cela donne :



Par ailleurs, si vous souhaitez effectuer une rotation précise (par exemple à 90°), je vous déconseille d'utiliser cette technique. En effet, ce ne sera jamais une rotation parfaite de 90° (ce sera 89° ou 91°) et vous aurez des problèmes pour retoucher votre bloc par la suite.

D'autant plus qu'il existe une fenêtre dans laquelle vous allez pouvoir commander une rotation précise, alors on ne va pas s'en priver ! Pour y accéder, allez dans le menu Tools/Transform (raccourci clavier Ctrl + M). La fenêtre suivante apparaît :



Il faut distinguer 2 colonnes : à gauche, le "Mode". Pour l'instant, seul le mode "Rotate" nous intéresse. Nous verrons "Scale" lorsque nous parlerons des textures. Le mode "Move" vous permet de déplacer votre bloc d'un nombre précis d'unités (peu utilisé).

Sélectionnez "Rotate" comme j'ai fait, puis entrez une valeur dans le champ qui vous intéresse (tout dépend de quelle vue vous vous placez) : X, Y ou Z. Par exemple, si vous tapez 90 dans le champ Z, le bloc effectuera une rotation de 90° dans la vue "Top".

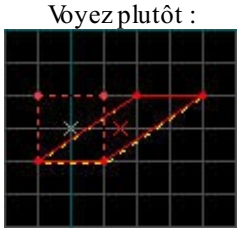
L'étirement

Pour faire un étirement, vous l'auriez deviné, il faut cliquer une troisième fois sur le bloc. Des petits carrés devraient apparaître, mais disposés cette fois à des endroits un peu différents.

C'est une technique un peu spéciale parce qu'on n'a pas l'habitude de l'étirement. Cela permet de déplacer une face du bloc sur une même ligne, tout en la gardant parallèle à l'autre face.

Pas évident à imaginer hein ?

Voyez plutôt :



L'étirement est une technique que les mappeurs ont tendance à délaisser. Ne l'oubliez pas ! Ca peut vous être très utile dans certains cas : des fois, les techniques les plus avancées ne peuvent pas vous aider pour réaliser la forme dont vous avez envie, et il suffit en général d'un étirement pour régler le casse-tête.

Grouping/Ungrouping

Beuh, c'est pas vraiment une technique de transformation des objets, mais c'est le meilleur endroit que j'aie trouvé pour en parler. Le **Grouping** consiste à assembler plusieurs blocs pour ne faire qu'un seul bloc au final. C'est une fusion, en quelque sorte.

 Nous utiliserons ces boutons de la barre d'outils du haut.

- Sélectionnez plusieurs blocs, comme je vous l'ai appris. Pour les grouper, il faut cliquer sur le premier de ces boutons. Maintenant, lorsque vous cliquerez sur un des blocs du groupe, tous les autres blocs seront automatiquement sélectionnés ! Vous pourrez ainsi les redimmensionner tous d'un coup !
- Si vous en avez marre, cliquez sur le second bouton, et vos blocs seront dégroupés (séparés). Ils redeviendront comme avant.
- Si vous voulez sélectionner un seul bloc d'un groupe d'objets, sans pour autant les dégrouper, cliquez sur le troisième bouton : "Ignore Groups". Vous pourrez alors retravailler sur certains blocs, tout en les laissant groupés.

Voilà pour les fonctions simples. Les fonctions suivantes sont un peu plus avancées.
Le terme "avancé" ne doit pas vous faire peur  Ca ne veut pas dire que c'est difficile à utiliser, mais que ça permet d'obtenir de meilleurs effets (plus complexes).
Là encore, nous allons travailler sur des blocs de forme "block". Ce n'est pas impossible à faire avec d'autres formes, mais j'ai préféré utiliser ici des blocs simples pour des raisons de simplicité.

Hollowing et Carving

Très important! L'Hollowing et le Carving sont deux méthodes à proscrire en tant que débutant car elles présentent beaucoup d'inconvénients! Une fois plus expérimenté, on peut s'en servir à juste dose mais pour l'instant il est conseillé de les oublier! A la place, il faudra utiliser la technique suivante le clipping! Si je vous en parle c'est surtout pour que vous sachiez qu'elles existent.

Prenez un bloc simple. Il faut qu'il soit assez grand.
Imaginons que vous voudriez vider le bloc de son contenu pour le rendre vide, et donc créer d'un seul coup une pièce à l'intérieur.
Vous avez deviné, c'est à ça que sert l'**hollowing**.

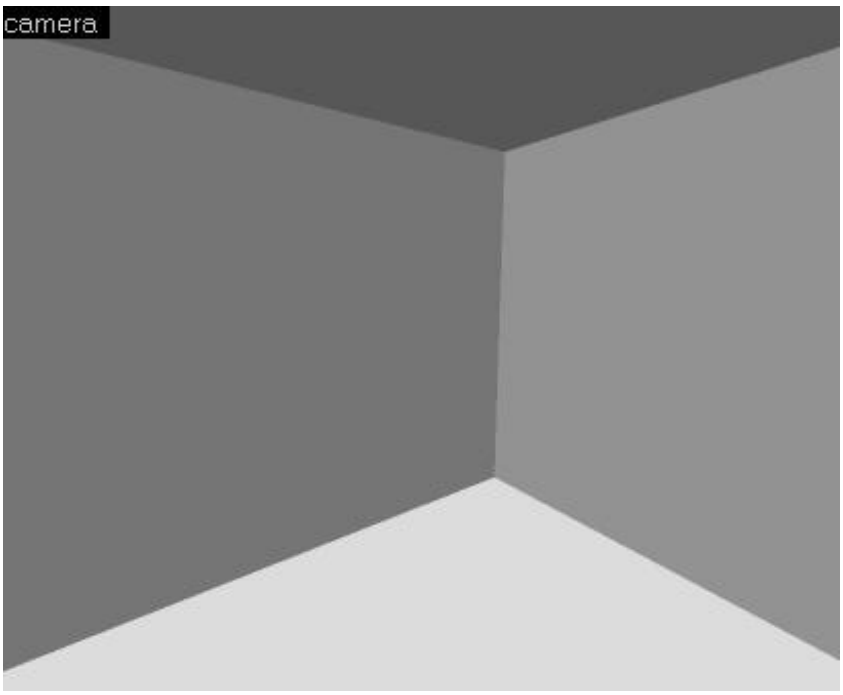
Après avoir créé le bloc, cliquez dessus avec le bouton droit de la souris. Cliquez sur "Hollow". La fenêtre suivante apparaît :

Dans la majorité des cas, on utilise la méthode Hollow pour créer une pièce. Hammer vous demande l'épaisseur des murs qui entoureront la pièce (en unités de Hammer).
Vous avez 2 possibilités :

- Vous entrez une valeur positive, les murs seront créés à l'intérieur du bloc.
- Si vous entrez une valeur négative, les murs seront créés à l'extérieur du bloc. C'est ce que je vous recommande de faire en général : vous êtes ainsi sûrs que la pièce aura un volume égal à celui du cube.

Une épaisseur de 16 unités me semble raisonnable, voire même 8. Si vous entrez 32 c'est un peu gros...
Habituez-vous à utiliser les unités de Hammer, ce n'est pas évident.

Voilà un exemple de cube "Hollowé" (vu de l'intérieur) :



Notez que Hammer a en fait transformé votre bloc en un groupe de 6 blocs. Vous pouvez donc utiliser les techniques de Grouping/Ungrouping si vous le désirez.

Passons maintenant au carving....

Ouïe ouïe ouïe ! Le **Carving** est une technique complexe, mais fort utile. Elle est même un peu trop utilisée... Elle consiste à "trouer" un bloc à un endroit précis, à l'aide d'un autre bloc...

Prenons un bloc simple comme tout à l'heure. Ce sera le bloc **carvé**. Retenez bien ce terme, il est important.
Prenons maintenant un autre bloc (à côté) de plus petit taille. Ce sera le bloc **carvant**.
Vous l'avez compris, il y a un carvé et un carvant.

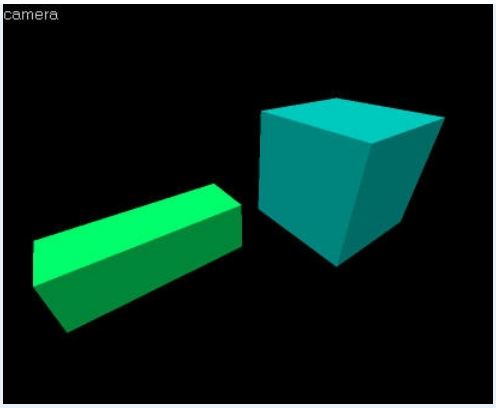
Maintenant, déplacez le bloc carvant de manière à ce que tout ou une partie du bloc rentre en contact avec le bloc carvé. En général, on ne fait rentrer qu'une partie du bloc carvant à l'intérieur du bloc carvé.

1. Sélectionnez le bloc carvant
2. Cliquez dessus avec le bouton droit de la souris
3. Puis, cliquez sur "Carve".
4. C'est fait ! Vous pouvez maintenant retirer ou même supprimer le bloc carvant.

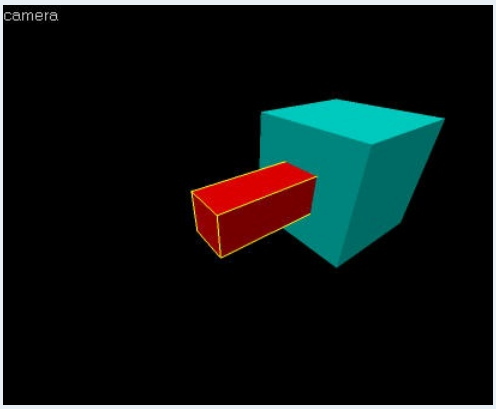
Le bloc carvé est maintenant troué à l'endroit où vous vouliez. Utile pour créer des passages dans des murs, comme des conduits d'aération !

Le Carving est source d'erreurs. Vous risquez en effet d'avoir des problèmes à la compilation si vous abusez du Carving sur un même bloc...

Bien, je préfère vous résumer tout ça dans un tableau pour être sûr que vous ayez compris, car c'est une technique vitale.

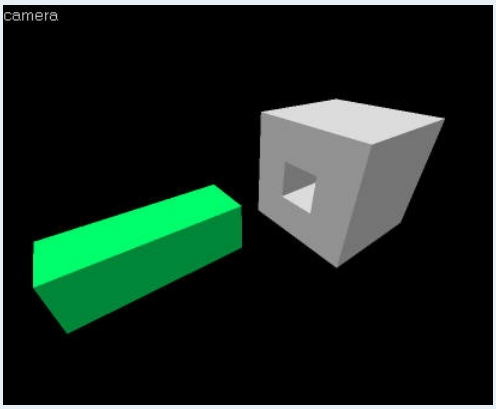


1/ On a 2 blocs :- Un carvé : c'est le bloc bleu.



2/ On place le bloc carvant dans le bloc carvé. Il sont l'un dans l'autre.

- On sélectionne le bloc carvant.
- On clique avec le bouton droit de la souris dessus...
- ... et on clique sur "Carve".



3/ On retire le bloc carvant (en général on le supprime carrément, il ne sert plus à rien).
Il apparaît alors que le bloc carvé est troué : ça pourrait être par exemple un passage dans un mur, dans lequel le joueur peut se faufiler.
Les possibilités d'utilisation sont très nombreuses, ayez un peu d'imagination !

Le bloc carvé est maintenant constitué d'un groupe de plusieurs blocs. Cela augmente considérablement le nombre de polygones à afficher à l'écran, et si le joueur a une petite config', ça risque de ramer du boudin

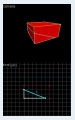
Clipping

Encore une technique pour modifier la structure de vos blocs. Rassurez-vous, celle-ci est plus facile à utiliser, et elle est tout aussi pratique !

Le **clipping** consiste à couper un bloc selon une droite. C'est tout simple, et la forme change vraiment (surtout qu'il n'est pas interdit de couper plusieurs fois).

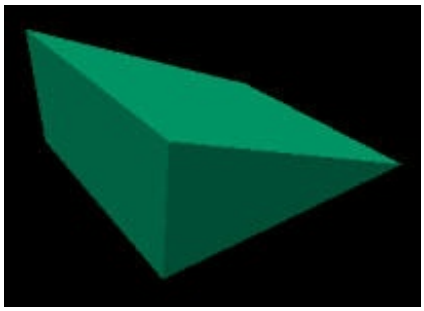
Sélectionnez un bloc, puis actionnez le "Clipping Tool".

Le "Clipping Tool" se trouve dans la barre d'outils de gauche.



Votre curseur prend la forme d'un croix. Servez-vous en dans une vue 2D pour créer un segment de 2 points (qui est en fait une droite).Ce segment servira a couper votre bloc en 2 morceaux.Le morceau qui sera gardé est en blanc. Le reste disparaîtra. Si vous voulez que ce soit l'inverse, il faut faire la combinaison clavier Shift+X (autant de fois que ce sera nécessaire pour avoir en blanc la bonne partie à garder).Une fois que c'est bon, tapez simplement "Entrée". Pfiou ! Et voilà un bloc coupé en 2 !Par ailleurs, notez qu'il n'est pas obligatoire de placer les points du segment sur un des côtés du bloc (comme sur la capture d'écran à gauche). Il est possible d'écarter encore plus les points pour avoir encore plus de précision.

Voici à quoi ressemble un bloc coupé :



Morphing

Morphing... Brrr... Rien que ce nom doit vous faire frémir...
Non ? Ca ne vous fait pas frémir ? Eh bien ça devrait.

Le **morphing** est une technique excessivement complexe (la plus complexe d'ailleurs), qui consiste à déplacer les vertices d'un polygone.
Les **vertices** ? Bonne question... Les vertices (vertex en anglais) définissent un polygone (sa position, sa taille, sa forme...) : c'est la technique de base qu'utilisent nos chères cartes 3D pour afficher des mondes en 3 dimensions. Beuh, autant vous dire qu'il n'y a rien de plus difficile...

Toutefois, le morphing permet d'obtenir des résultats surprenants, des polygones qui défient votre imagination

C'est la technique la plus avancée de Hammer. Les très bons mappeurs l'utilisent assez souvent parce qu'elle permet de faire beaucoup de choses.
Si vous êtes un vrai débutant, oubliez le morphing pour l'instant, vous aurez le temps de vous y mettre plus tard.

Retenez simplement que si vous ne maîtrisez pas PARFAITEMENT le morphing, la structure du bloc sera invalide dans 90 % des cas. La compilation de la map risque donc de planter.

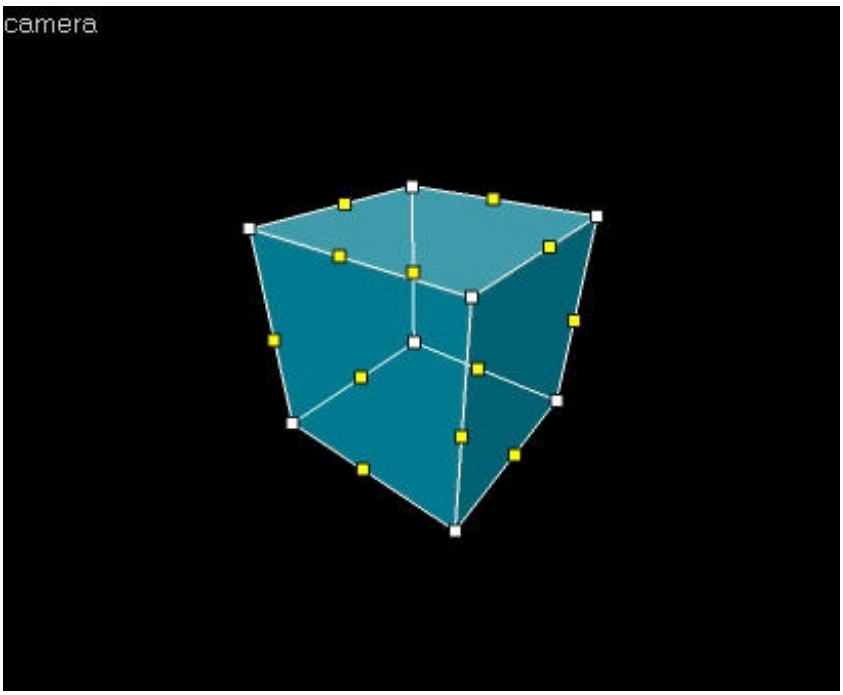
Bon, si vous êtes toujours décidés, sélectionnez le bloc à modifier.


Activez le mode Morphing en cliquant sur le bouton "Vertex Tool".

Vous avez 2 possibilités :

- Soit vous travaillez dans les vues 2D, et vous ferez un peu moins d'erreurs
- ... soit vous travaillez dans la vue 3D. C'est plus pratique pour se faire une idée de la nouvelle forme, mais on fait vite une erreur rendant la structure du bloc invalide...

Pour ma part, je choisis la vue 3D. Le cube apparaît transparent :



Tous ces points qui entourent le cube sont des vertices.
Certains sont jaunes, d'autres sont blancs. Si vous ne souhaitez afficher que les vertices blanches, cliquez une 2° fois sur le bouton "Vertex Tool". Si vous cliquez une 3° fois, seuls les vertices jaunes apparaîtront.

Quelle est la différence entre ces deux types de vertices ? Les blanches sont situés sur les sommets du polygone, et les jaunes au milieu de chaque arête.
Selon que vous déplacez une vertice blanche ou jaune, il ne se passera pas la même chose : les blanches déplacent un sommet du polygone, tandis que les jaunes déplacent un côté du polygone.
Pratique, mais dangereux !

Le mieux est de tester vous-même le morphing pour voir quand ça marche et quand ça fait une erreur. Si le bloc est invalide, il apparaît une erreur dans la fenêtre des "Problems" de Hammer (Alt + P). Nous reparlerons de cette fenêtre dans le chapitre sur la compilation.

Le mode "Morphing" vous permet aussi de diviser une face en 2, pour avoir une nouvelle partie du bloc à travailler. Pour cela, sélectionnez 2 vertices jaunes d'une même face et faites Ctrl + F. Une nouvelle face a été créée !

Enfin, vous avez la possibilité de fusionner des vertices, pour n'en garder qu'un seul. Pour cela, placez une vertice blanche sur un autre vertice blanc. La fenêtre suivante devrait apparaître :

Si vous cliquez sur Oui, les vertices fusionneront. Essayez pour voir !
Et voilà ! Hollowing, clipping, carving, morphing... Ca sonne beaucoup en -ing tout ça... C'est vrai que ça fait pro de dire : "J'ai fait un carving sur mon bloc, il ne me reste plus qu'à faire un petit clipping en bas et je serai paré pour attaquer un morphing de l'arête du haut."

Vous êtes maintenant des experts en manipulation de blocs (si si, je vous assure que de nombreux mappeurs n'en savent pas autant que vous en savez maintenant !).

Vous commencez mine de rien à apprendre un nombre impressionnant de choses sur le mapping... Faites une petite pause avant de continuer, relisez tranquillement ce chapitre et le précédent (imprimez-les si nécessaire, c'est pratique).

Le prochain chapitre sera un peu plus cool, histoire de reprendre des forces avant d'attaquer le mastodonte : les entités.

Maintenant que vous savez créer des blocs, prenez l'habitude d'en diminuer si possible le nombre!
Car, quand votre map deviendra jouable sous Half-Life, si il y a trop de blocs, elle pourra ramer!

Les textures (A : Utiliser des textures)

Bon! Vous savez maintenant créer et manier les blocs!
Mais un bloc sans texture sous Half-Life, ce n'existe pas, tous en ont forcément une, alors intéressons nous dès maintenant aux textures.
Vous pouvez activer le mode "3D Textured Polygons" du menu View. Nous allons voir quelle place prennent les textures dans le mapping, comment les appliquer à nos bloc, les choisir...

Préparez-vous à en voir de toutes les couleurs

Choisir une texture

Bon, avant de parler des textures, il faudrait définir ce mot, non ?
Une texture est une image décorant les blocs. En jeu, chaques blocs en a une obligatoirement. Sous Hammer, il faut activer le mode "3D textured polygones" pour les voir et les choisir.
Une texture est généralement faite pour être répétée en mosaïque : on peut ainsi agrandir les blocs sans problème.
Il existe des textures de toutes sortes : pour les sols, les murs, les escaliers, la végétation, l'eau... Certaines ne sont pas faites pour être répétées en mosaïque : les portes, les posters, les graffitis...

Notre objectif pour le moment est de voir à quoi ressemblent ces textures, et de les repérer pour pouvoir rapidement les réutiliser plus tard (parce qu'il y en a beaucoup !).
Première chose importante : vérifiez que vous avez bien configuré Hammer. Dans l'onglet "Textures" des options, il faudrait que vous ayez au moins les fichiers suivants :

- halflife.wad : c'est une bonne partie des textures d'Half-Life (télécharger"
)
- decals.wad : ce sont des graffitis, nous en reparlerons plus bas...
- cstrike.wad : et pourquoi pas le fichier de textures de Counter-Strike, pour ceux qui veulent faire une map cs.
  Le fichier le plus important de tous est de loin halflife.wad, et si vous voulez faire une map cs, vous avez intérêt à l'inclure quand même : le fichier cstrike.wad est assez pauvre en textures...

  Mais à quoi servent donc ces fichiers .wad ? Ils contiennent plusieurs textures, que Hammer doit charger pour que vous puissiez les utiliser.

  Si vous utilisez des textures d'un fichier wad (pourquoi pas le vôtre), et que le joueur ne possède pas ce fichier, ça va pas aller du tout ! Mais alors là pas du tout ! Pensez à distribuer les nouveaux wads avec votre map !!! Vous verrez plus tard que c'est le cas d'autres fichiers.

  Bon, maintenant que c'est configuré, penchons-nous sur la fenêtre "Textures" de Hammer. Repérez ceci sur votre écran :

- "Texture Group" : vous pouvez choisir d'afficher seulement les textures d'un wad précis.
- "Current Texture" : c'est le nom de la texture choisie qui est utilisée en ce moment. Cliquez sur ce menu déroulant pour en choisir une autre.
- La texture choisie apparaît dans ce carré, pour vous donner un aperçu...
- C'est la taille de la texture en unités de Hammer. Notez qu'il est possible de faire des blocs plus gros que 64x64, puisque la texture est faite pour être répétée en mosaïque.
- "Browse" : ce bouton très important affiche une nouvelle fenêtre pour choisir vos textures. L'avantage, c'est qu'elle est plus grande et qu'elle permet de faire une recherche de textures...
- "Replace" : ce bouton affiche une autre fenêtre très pratique. Imaginons que vous utilisiez depuis le début dans votre map une texture de mur que vous n'aimiez pas trop... Et maintenant vous venez de trouver la texture que vous auriez aimé utiliser depuis le début... Eh bien, cette fenêtre remplace les textures de tous ces blocs par la nouvelle texture.

Pour afficher plus de textures à la fois, cliquez sur le bouton "Browse...". La fenêtre suivante apparaît :

C'est bien plus pratique pour choisir sa texture non ?
Choisissez la texture qui vous plaît (et vous avez le choix !). Moi je prends "out_sndbag4".
Double-cliquez dessus pour valider, et c'est bon. Vous venez de choisir votre première texture ! Bravo !!!

Au début, on se dit : "Il y a beaucoup trop de textures, comment faire pour trouver celle que je cherche ???".

Bah, vous êtes pas arrivés ! Y'en a pour des semaines à éplucher tout ça !
Heureusement que je suis là Au bout d'un certain temps, j'ai fini par connaître les noms des textures qui m'intéressent. Je perds beaucoup moins de temps à rechercher mes textures.

Il existe des groupes de textures. Chaque texture d'un groupe a une partie de son nom commune à toutes les autres. Exemple : les textures d'ordinateurs ont toutes le mot "comp" dans leur nom.

Pour afficher uniquement ces textures, il faudra taper le nom du groupe dans la zone de texte "Filter" en bas de la fenêtre.

Ce tableau devrait vous aider à choisir une texture :

Exemple	Préfixe	Commentaires
	wall ou w	Si vous recherchez une texture de murs, vous en trouverez dans les groupes "wall" ou "w".
	flr	"flr" pour floor (sol en anglais). Les textures pour le sol sont regroupées ici;
	ceil	Les textures de plafond sont rares. Il n'y en a pas beaucoup dans ce groupe... On utilise alors parfois des textures de murs pour faire le plafond.

	door ou dr	Les textures de portes se trouvent dans ces groupes.
	lgt, lght ou light	Ces trois groupes contiennent des textures pour les lampes, parfaites pour décorer l'intérieur d'un laboratoire ou d'une maison (et pour l'éclairer aussi)
	water	Des textures d'eau ?! Mais bien sûr que ça existe ! Mais bon, on n'a pas beaucoup de choix. Après tout, de l'eau ça reste de l'eau
	glass	Des vitres ! A noter qu'elle ne sont pas automatiquement transparentes, nous verrons comment faire dans un futur chapitre.
	comp ou cmp	Pour mettre des ordinateurs dans votre map, c'est par là !
	sign	Il y a pas mal de panneaux comme celui-ci dans le groupe "sign". Pratique pour avertir le joueur d'un danger ou pour décorer les murs !
	generic	Il y a beaucoup de textures intéressantes dans ce groupe très diversifié!
	crate	Si vous voulez mettre des boîtes dans vos niveaux, vous trouverez votre bonheur ici !
	out	De nombreuses textures très utiles si votre map se situe à l'air libre (à l'extérieur quoi).
	cstrike	La plupart des textures de cstrike.wad sont du groupe "cstrike". Ces textures vous aideront à créer des niveaux pour Counter-Strike, mais il n'y en a pas beaucoup... alors on réutilise souvent des textures de halflife.wad
	fifties	Ces textures s'intègrent très bien dans les niveaux de Counter-Strike. Elles sont utilisées pour décorer l'intérieur des maisons.

Je crois que ce petit tableau vous sera très utile au début pour vous aider à choisir vos textures. Heureusement qu'il y a du choix !

Nous allons maintenant voir comment faire pour appliquer ces textures aux blocs.

Appliquer une texture

Avant la création du bloc

Le moyen le plus simple pour appliquer une texture à un bloc, c'est tout simplement de créer ce bloc ! Toutes ses faces seront recouvertes de la texture actuellement sélectionnée (comme nous avons vu plus haut).
Si la vue 3D est configurée en mode "3D Textured Polygons", vous verrez d'un seul coup le résultat.

J'ai pu ainsi créer en quelques secondes les murs et le sol d'une pièce !

Pour obtenir cette pièce, j'ai tout d'abord créé un bloc pour faire le sol (texture C1A2_FLR2).

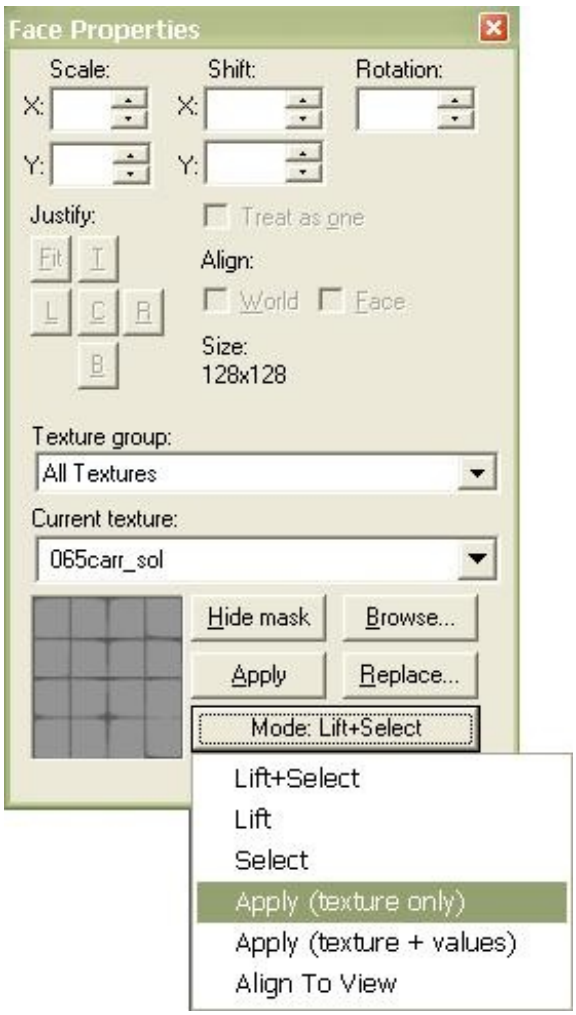
Puis, j'ai placé un mur (FIFTIES_WALL14U) sur un des côtés du bloc du sol. Je l'ai cloné pour le reproduire à l'identique en face. Il me restait encore 2 murs à placer : j'ai recloné encore une fois le bloc et je lui ai fait faire une rotation à 90° (menu Tools/Transform, raccourci clavier Ctrl + M). J'ai de nouveau cloné le bloc (celui qui a subi une rotation) pour le placer en face.

Après la création du bloc

Si vous avez créé un bloc et que vous voulez en changer sa texture après sa création, choisissez la nouvelle texture voulue comme expliqué précédemment, sélectionnez votre bloc, et cliquez sur "Apply Curent Texture"().

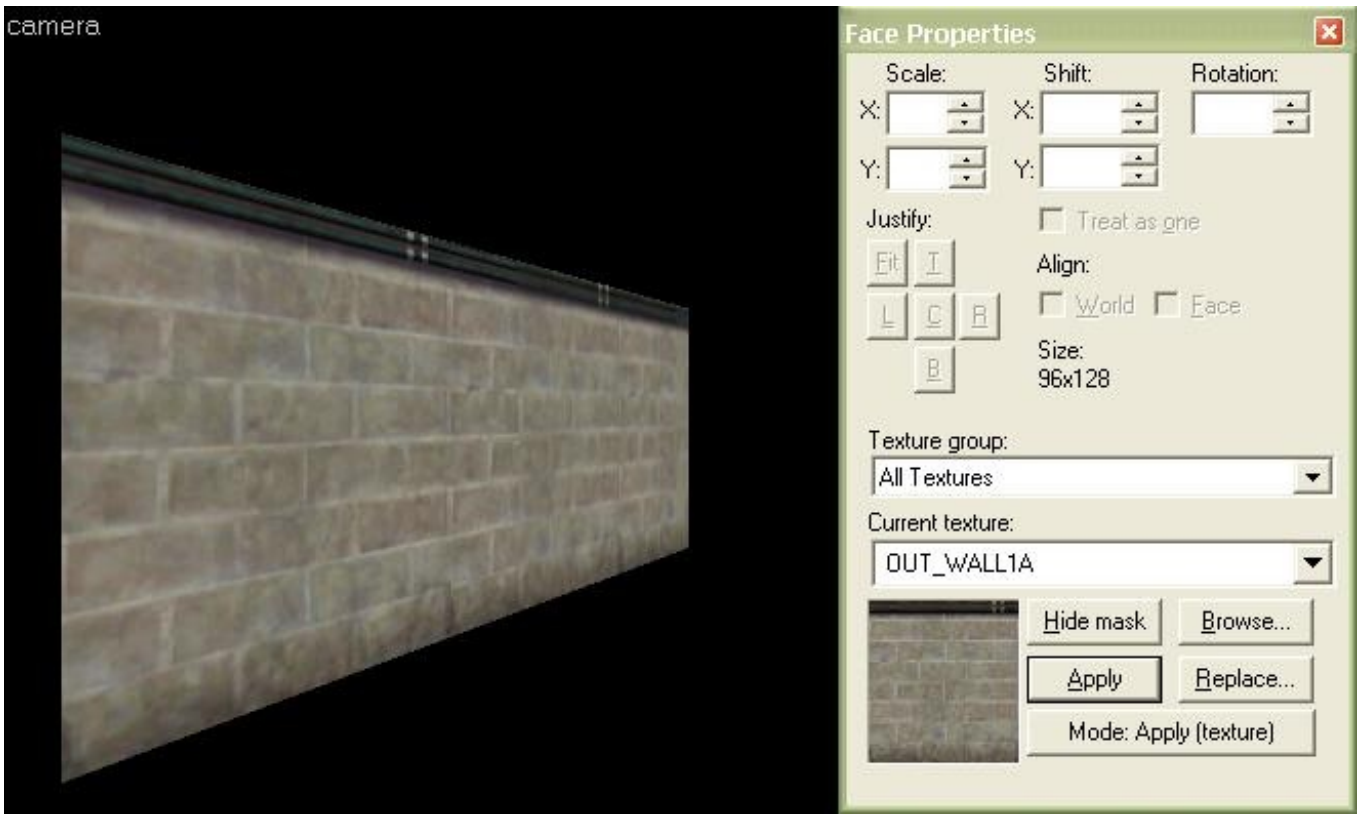
Mais ce n'est pas tout.
Imaginons que vous vouliez faire le mur d'une maison par exemple. Ce mur aura deux côtés: un vers l'extérieur et un vers l'intérieur.
Il est normal que la face du mur tourné vers l'extérieur ait une texture différente de celle vers l'intérieur.
Comment faire sous Hammer pour appliquer une texture sur une face particulière?
C'est tout simple, mais ça nécessite un outil:
 : Toggle Texture Application, dans le menu de gauche de Hammer.
Cliquez dessus. Une fenêtre, "Face Properties", apparaît.

Tout en bas de cette fenêtre, il y a un bouton "mode:" suivi de quelque chose. Cliquez dessus et une liste apparaît. Dans cette liste, choisissez "**Apply (texture only)**".



Ensuite, c'est tout simple. Si vous n'avez pas déjà sélectionné la texture que vous voulez appliquer sur une face particulière, cliquez sur "Browse" pour chercher votre texture.
Trouvez votre texture et double-cliquez dessus. Vous revenez sur "Face Properties" qui affiche maintenant votre texture à gauche de "Browse".

Là, il ne vous reste plus qu'à cliquez dans la vue "Camera" sur la face du mur en question pour appliquez la texture dessus.



Textures spéciales

Vous croyez avoir tout vu sur les textures ? Détrompez-vous ! On a encore pas mal de choses à voir dans ce chapitre.

Il existe en effet des textures spéciales, qui fonctionnent différemment de toutes les autres textures. Elles sont plus rares, mais il est important de les connaître.

La texture "Clip"

Voici la texture en question. Son nom est CLIP.

En fait, ce n'est pas vraiment une texture, vous vous en doutez. Elle représente le visage d'un homme en gris.

Si vous créez des blocs avec cette texture, ils ne seront pas visibles. On a rarement des blocs de ce type dans les maps, mais ça arrive.
Ils permettent de bloquer un passage au joueur : le bloc est invisible, mais le joueur ne peut plus avancer. On s'en sert généralement aux extrémités de la map, lorsqu'on ne veut pas que le joueur aille plus loin. Cependant les balles des armes passent à travers !

La texture "Origin"

La texture ORIGIN ressemble à CLIP, mais cette fois la couleur du bonhomme est verte fluo...

Là aussi, vous vous en doutez, la texture n'est pas visible quand on joue.

Sauf que là, je suis embarrassé pour vous expliquer son fonctionnement. Elle est utilisée conjointement avec les portes et autres objets rotatifs pour définir leur axe de rotation.
On n'en verra l'utilité que lorsque nous parlerons des entités. Il faut juste savoir que cette texture existe et qu'on en a parfois

besoin.

La texture "AAATRIGGER"

Voici la texture AAATRIGGER !

Si elle a un nom bizarre, c'est pour qu'on s'en rappelle plus facilement (AAA).
Cette texture n'est pas vraiment une texture spéciale : on pourrait l'appliquer à n'importe quel bloc... En fait, on l'utilise sur des blocs spéciaux (les entités-blocs que nous verrons dans un autre chapitre) pour se rappeler dans Hammer que ce n'est pas un bloc ordinaire. En général, le bloc n'est pas visible lorsqu'on joue, et la texture n'apparaît plus.

La texture "Sky"

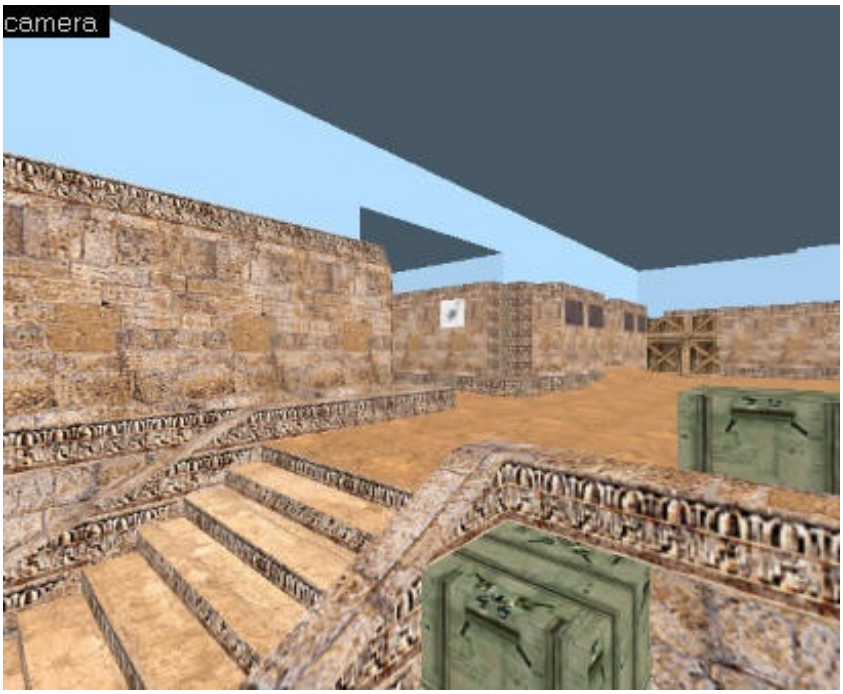
Cette texture s'appelle SKY (ciel en anglais). C'est une texture unie bleu clair...

Ca, pour une texture spéciale, c'est une texture spéciale !

Elle ne doit être utilisée que si votre map se déroule à l'extérieur, ce qui est très fréquent dans les maps Counter-Strike.
Les blocs recouvert de cette texture affichent du ciel, mais pas n'importe comment. La position des décors reste fixe, contrairement à toutes les autres textures. C'est un peu difficile à expliquer, je le reconnais. Heureusement, Half-Life s'occupe de tout, vous n'avez rien à faire : placez juste du ciel là où vous voulez qu'il y en aie.

Le ciel ne se trouve pas forcément au-dessus de la tête du joueur, il y en a tout autour de lui et même en bas, comme des murs pour cacher le vide qui se trouve derrière.

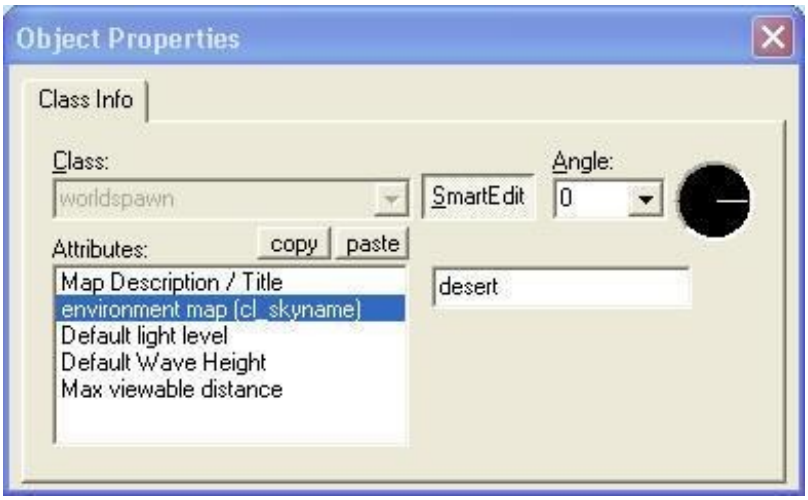
Voici par exemple la map de_dust telle qu'elle apparaît dans Hammer :



On reconnaît le point de départ des terroristes. Si vous observez quelques secondes cette image, vous vous rendez-compte que le sky est partout présent. Il entoure chaque pièce de la map, et Half-Life se charge d'afficher le ciel quand on joue.
Si vous voulez voir la diffèrence entre le sky sous Hammer et sous Half-Life, regardez cette map pas encore compilé qui est sous Hammer. Elle a été compilé et voilà le résultat (vous verrez plus tard comment compiler une map et la rendre jouable, pour l'instant vous ne pouvez que mapper sous Hammmer.).
Sous Hammer

Sous le jeu une fois compilé:

Oui, mais quel ciel ? Vous avez le choix en effet entre plusieurs ciels (vous pourriez même créer le vôtre !). Pour choisir un ciel, rendez-vous dans le menu Map/Map Properties :



Vous pouvez modifier quelques propriétés générales sur votre map, notamment son nom, celui de son auteur...
Ce qui nous intéresse, c'est "environment map (cl_skyname)". Vous devez taper à droite le nom du ciel à utiliser.

Si vous laissez le champ vide ou si vous mettez un nom incorrect, Half-Life utilisera le ciel par défaut pour votre map. Il s'agit du ciel avec les montagnes Rocheuses en fond, comme on en voit souvent dans l'aventure de Half-Life.

Les textures aléatoires "-0" "-1" "-2" "-3" ...


Voici un exemple de texture aléatoire : cette boîte à outils ne se répète pas avec la même texture à chaque fois.

Ce sont des textures assez particulières : elles fonctionnent par groupe. Elles se répètent en mosaïque comme d'habitude mais elles changent de texture à chaque fois.
Inconsciemment, le joueur va avoir une impression de dynamisme, et surtout, ça évite d'avoir l'impression quand on joue que c'est la même texture qui est appliquée au bloc. Ca donne un effet intéressant dans une map...

Les textures de ce type commencent par un tiret "-". Elles sont constituées en groupes. Dans un même groupe, les textures ont le même nom mais un numéro différent. Ici, nous avons les textures suivantes :

- -0OUT_CUBY
- -1OUT_CUBY
- -2OUT_CUBY
- -3OUT_CUBY
- -4OUT_CUBY
- -5OUT_CUBY

Ces textures appartiennent au même groupe et se ressemblent. Mais ce ne sont pas exactement les mêmes...

Vous pouvez mettre à votre bloc n'importe laquelle de ces textures, c'est Half-Life qui se chargera de la changer tout seul au cours du jeu.

Les textures animées "+0" "+1" "+2" "+3" ...


Ceci est une texture animée : lorsqu'on joue on voit une véritable chute d'eau !

Si si ! Ca existe ! Il y a bel et bien des textures animées dans Half-Life !!!
Bon, OK, y'en a pas beaucoup. Mais l'effet que ça peut donner dans vos niveaux est quand même indéniable : des chutes d'eau, des écrans d'ordinateur qui oscillent etc etc...

Le problème, c'est qu'il est impossible de paramétrer le temps qui sépare chaque image de l'animation. Celui-ci est déjà défini, et on ne peut pas le modifier. Bah, ce n'est pas très grave...

Les textures animées commencent par un plus "+". Chaque image d'une même animation a le même nom, et un numéro différent. Ici, la texture s'anime dans cet ordre :

- +0LASER1FALL
- +1LASER1FALL
- +2LASER1FALL
- +3LASER1FALL

Mettez n'importe laquelle de ces textures dans Worldcraft : Half-Life l'animera lorsque vous y jouerez.

Les textures associées à des boutons (toggled) "+A" "+0"

Cette texture, par exemple, n'est pas allumée au départ. Il faudra appuyer sur un bouton pour que l'oxygène s'allume.

Cette fois, c'est vous mappeur qui décidez de la texture à afficher.
Par exemple, au cours du jeu, si on ouvre la vanne à oxygène, alors la texture change et la lumière bleue de l'oxygène s'allume.
Nous verrons en pratique comment faire cela dans un autre chapitre.

Ces textures fonctionnent par paires : "+A" et "+0". Si c'est la texture A qui est actuellement affichée, alors elle est remplacée par la texture 0 (et réciproquement).

Les textures d'eau "!"

Une texture d'eau, parmi la dizaine disponible...

Les textures d'eau sont un peu spéciales parce qu'elles ont la possibilité de bouger, pour faire les remous de l'eau.

La plupart des textures d'eau commencent par un point d'exclamation "!". C'est une convention, c'est-à-dire que ce n'est pas obligatoire mais qu'il faut essayer de la respecter.
Bref, si vous tapez un "!" dans la ligne "Filter", vous verrez les textures d'eau compatibles avec l'eau : elles pourront bouger pour les remous.


Ce n'est pas parce que votre bloc possède une texture d'eau commençant par un "!" que le bloc sera transformé en eau.
C'est un peu plus compliqué que ça, comme nous le verrons dans un futur chapitre.

Il n'y a rien d'autre à dire sur les textures d'eau, vous verrez plus tard comment transformer un bloc en eau...

Les textures transparentes "{"

Cette texture est faite pour être rendue en partie transparente.

Les textures de ce type ont été créées pour être rendue transparentes. Tout du moins en partie : seule la partie bleue sera transparente, on pourra donc voir à travers ! Pour la rendre transparente, nous aurons recours à une entité, comme nous le verrons plus loin.

Leur nom commence par une accolade : "{".

La texture NULL

Ha la fameuse texture NULL!
Si vous ne l'avez pas dans vos tetxures, c'est que vous n'avez pas ajouté le fichier "zhlt.wad" que vous avez normalement mis dans le dossier "wads" qui se trouve dans le répertoire de Hammer. Si ce n'est pas le cas ajoutez-le à Hammer comme vous avez ajouté les autres wads. Ce fichier .wad contient d'autres textures que nous verrons plus tard, contentons nous de la texture "NULL".
Que peut on faire avec cette texture? Hé bien rien! Oui, si vous créez un bloc avec cette texture il n'apparaitra pas, il restera invisible! Toutes les faces recouvertes de cette texture seront invisibles. Cependant, elles restent solides et bloquent le joueur.
Cette texture sera utile lorsque vous apprendrez à maitriser les entités ou à compiler votre map.

Les decals

Nous venons de voir à l'instant les textures transparentes. Il faut savoir qu'il en existe de 2 sortes : certaines contiennent du bleu, d'autres du blanc.

Ce sont celles qui contiennent du blanc qui nous intéressent maintenant : ce sont des decals.

Voici par exemple un decal utilisé dans les maps DE de Counter-Strike. Vous remarquerez que le nom commence là aussi par une accolade "{"

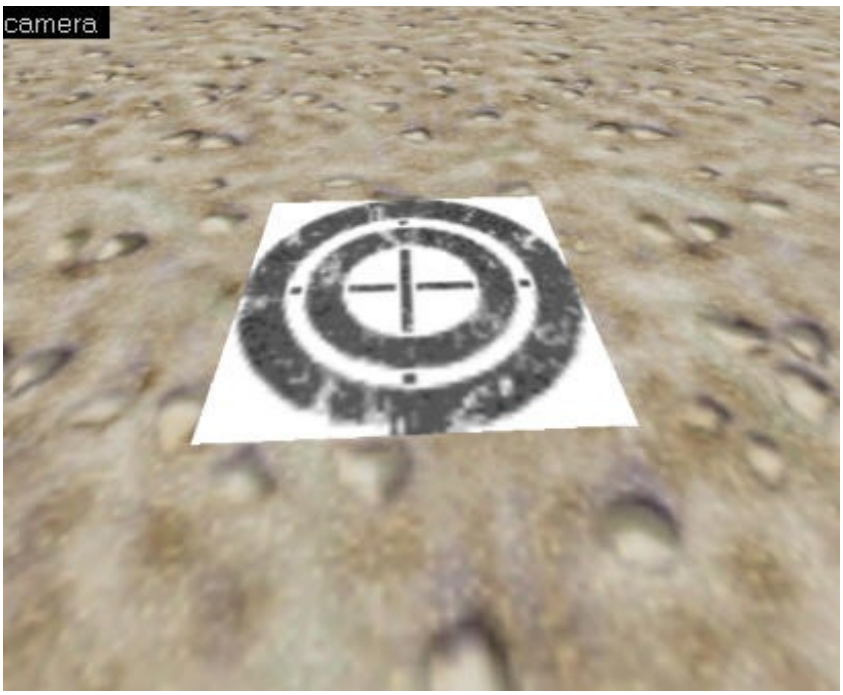
Oui, mais il reste une question essentielle : c'est quoi un decal ?

Un **decal** n'est pas vraiment une texture : on ne peut pas le répéter en mosaïque, ni créer des blocs avec ces textures.
Un decal s'applique sur un bloc déjà créé : ça fonctionne comme les tags que les joueurs de Counter s'amusent à afficher un peu partout.


Pour appliquer un tag à un bloc, on utilisera le "Decal Tool".

Une fois le "Decal Tool" activé, ultilisez la vue 3D pour vous déplacer, puis cliquez sur un bloc pour appliquer le decal.

Là, je me suis amusé à mettre le decal de cible sur un sol sablonneux :



Lorsqu'on jouera, le blanc aura disparu et la cible sera de couleur rouge.

Un decal ne fonctionne pas comme un bloc : impossible par exemple de lui faire subir une rotation. Vous pouvez par contre le sélectionner et le supprimer.
Si votre decal apparaît à l'envers sur un mur, vous n'aurez pas d'autre choix que de le créer ailleurs (sur une autre face) pour qu'il se remette dans le bon sens.

Ouf ! Le chapitre sur les textures est terminé... enfin presque ! Je vais maintenant vous montrer comme faire pour créer vos propres textures, et après je crois qu'on aura fait le tour de la question

Les textures (B : Créer ses textures)

Et si vous voulez créer vos propres textures ? Eh oui, c'est tout à fait possible !
On utilisera pour cela un autre logiciel, appelé Wally.

Selon les textures que vous faites, vous n'êtes pas obligés d'êtres des pros du dessin. Vous pouvez même importer des JPEG dans Wally !

Présentation de Wally

Dès que l'on veut avoir une map "personnelle", qui se détache des maps habituelles, la création de textures est le meilleur moyen de donner un style unique à votre map.
C'est même d'ailleurs recommandé pour tous ceux qui veulent faire un "bonne" map Counter-Strike : en effet, les meilleures maps ont toutes leurs propres textures. Sinon, les décors rappelleraient beaucoup trop l'aventure d'Half-Life, et ce n'est pas recommandé.

Worldcraft vous permet de faire des niveaux pour Half-Life, mais pas des textures (vous vous en seriez doutés). Pour cela, on a recours à un autre logiciel (très bien fait d'ailleurs) répondant au doux nom de Wally.

Faites-moi le plaisir de télécharger ce programme, il est devenu aujourd'hui quasiment indispensable pour créer ses textures :

Wally_v1.55.exe (1,09 Mo)

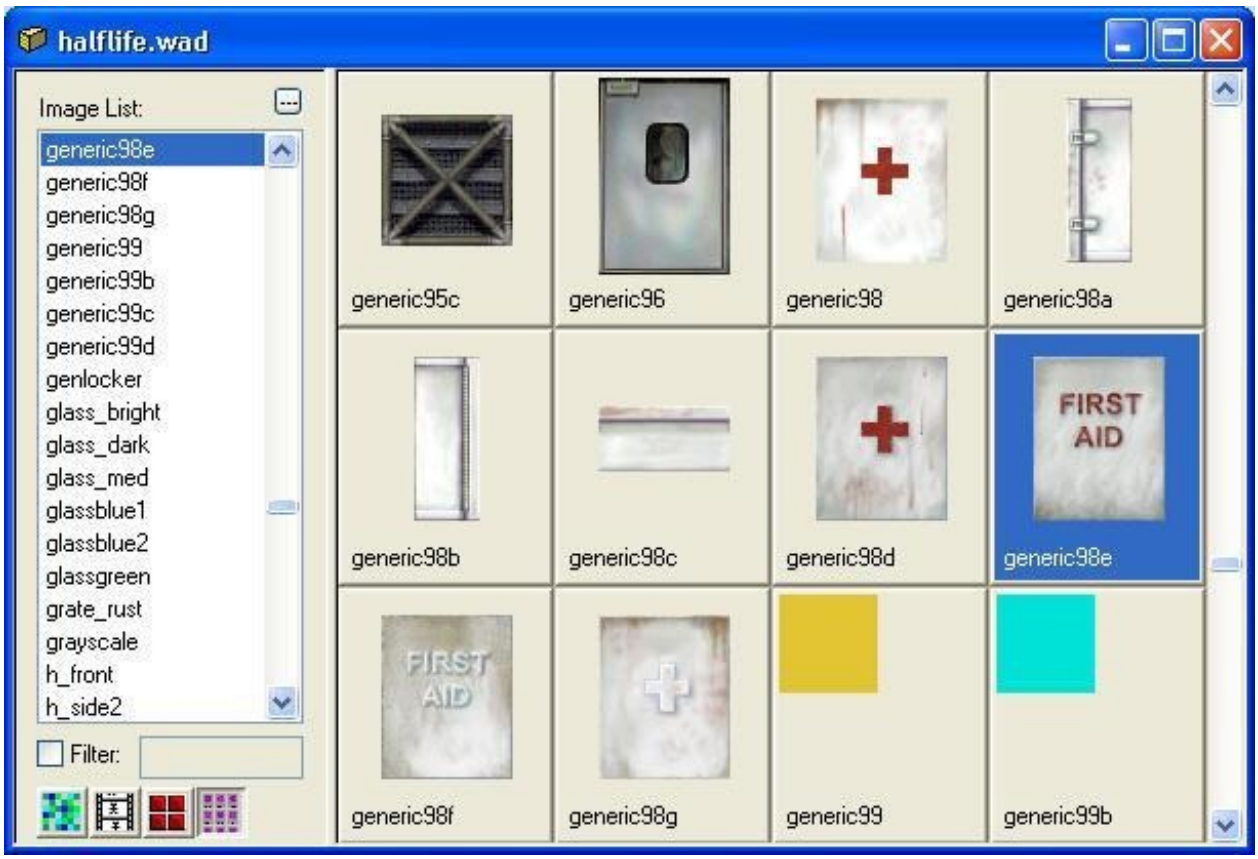
Ce fichier est un autoextractible de Winzip : dézippez tous les fichiers dans un répertoire créé à cet effet (ex. : C:\Worldcraft\Wally), puis créez un raccourci sur le bureau ou dans le menu Démarrer.

Dès le premier chargement de Wally, vous allez vous rendre compte qu'il n'est pas fait que pour Half-Life. Rappelez-vous ce que je vous ai dit : Half-Life est basé sur le moteur de Quake II !

Ce logiciel permet donc de faire des textures pour Half-Life, Quake I, Quake II... Mais pas Quake III ! Faut pas rêver : Quake III utilise des textures beaucoup plus avancées (codées sur 16 ou 32 bits, c'est-à-dire 65536 ou 4294967296 couleurs !!!), alors que Quake II et Half-Life ne supportent que 256 couleurs dans les textures.

Bon, que permet de faire Wally concrètement ? Il permet de visionner, créer et modifier des fichiers wad. Ces fichiers à l'extension *.wad peuvent contenir plusieurs textures (3116 pour halflife.wad !).

Pour commencer, ouvrez le fichier halflife.wad (dossier Half-Life\valve), histoire de voir comment ça marche. Cette fenêtre s'ouvre :



A gauche, on a la liste des noms des textures contenues dans ce wad. A droite, les textures apparaissent. Vous pouvez double-cliquer sur une texture pour la modifier (dans ce cas Wally se transforme en un outil de dessin comme un autre).
En bas, il y a 4 boutons intéressants :

1. Le premier active la gestion des textures aléatoires (si votre texture commence par un tiret "-")
2. Le second permet l'animation de la texture (si c'est une texture animée "+0" "+1" "+2"...)
3. Le troisième affiche la texture en mosaïque (très pratique)
4. Enfin, la dernière affiche la liste des textures comme nous sur la capture d'écran ci-dessus.

Pour créer un nouveau wad, faites File/New, et choisissez Half-Life Package.

Il existe 2 moyens pour ajouter de nouvelles textures dans votre wad :

- Vous pouvez importer des BMP ou des JPEG dans le wad. Ainsi, si vous trouvez sur Internet un fond d'un site web qui vous plaît, une image qui ferait une belle texture etc... vous allez dans Wally, vous ouvrez votre Wad et vous importer l'image que vous voulez.
- Vous pouvez aussi, si vous vous en sentez le courage, créer votre texture "à la main" sous Wally. Le programme dispose en effet d'un mode "édition de texture" qui ressemble un peu à Paint, mais en mieux je vous rassure

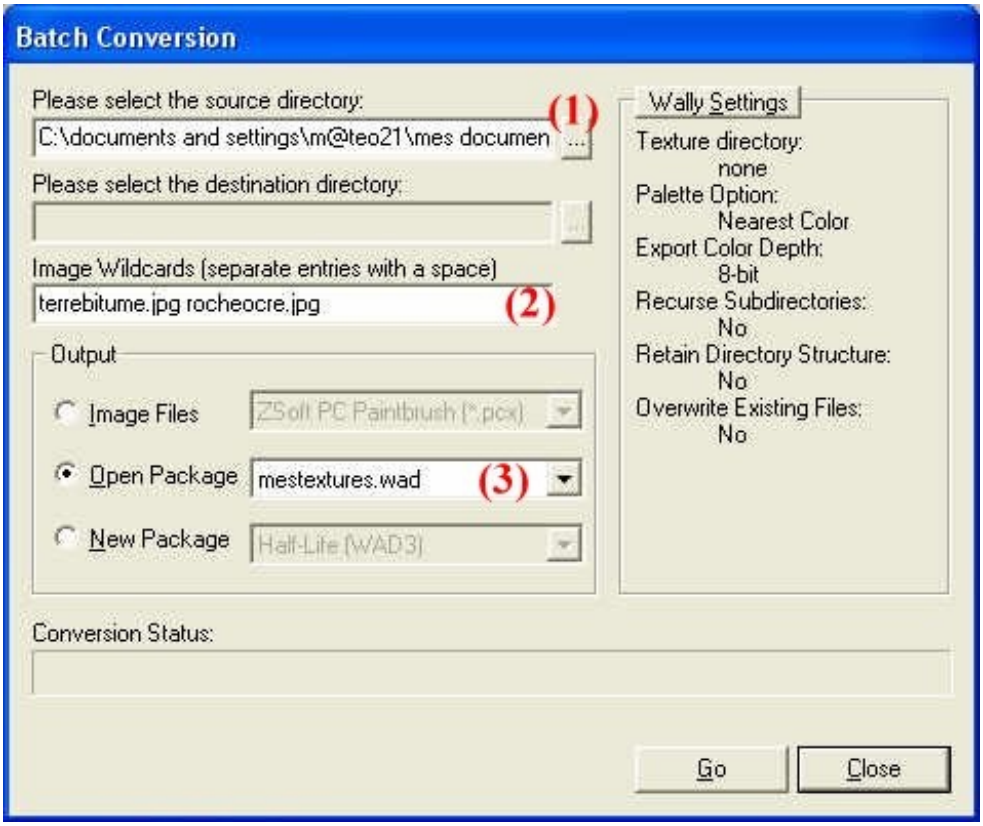
Importer des textures dans un wad

Nous allons voir ici le premier cas : l'importation d'une image (texture) dans un wad.

Si vous préférez créer vos textures avec un autre logiciel plus puissant dont vous avez l'habitude (comme Corel...), ou bien que vous avez trouvé une image qui ferait une jolie texture, il est tout à fait possible de la reprendre dans Wally.

Si vous importez des images dans Wally, et que celles-ci contiennent plus de 256 couleurs, elles devront être converties en 256 couleurs (8 bits). Votre image risque de perdre de sa qualité !

Il faut que vous ayez sur votre disque l'image au format bmp, jpg ou png.
Dans Wally, allez dans le menu Package/Add Images. Vous devriez voir ceci :



1. Répertoire source : indiquez dans quel dossier se trouvent les images que vous voulez importer.
2. Image Wildcards : là c'est un peu particulier, il ne faut pas se tromper. Si vous mettez *.jpg, toutes les images au format jpg de ce dossier seront importées.
   Si vous voulez importer une seule image, tapez le nom de cette image (ex : "monimage.jpg").
   Enfin, si vous voulez importer plusieurs images précises, tapez leurs noms en les séparant d'un espace (ex : "image1.jpg image2.jpg image3.jpg")...
3. Output (sortie) : vous devez indiquer ici ce que Wally doit faire de vos images.
   - Image Files : vos images seront converties dans un autre format. Exemple, vos JPG seront transformés en BMP. Je suppose que ce n'est pas ça que vous cherchez à faire ici
   - Open Package : vos images seront importées dans le wad que vous indiquez. Par défaut, c'est le wad que vous avez ouvert (celui que vous venez de créer) qui recevra les nouvelle textures. C'est donc cette option-là qu'il faut cocher.
   - New Package : vos images iront dans un nouveau Wad.

Cliquez sur "Go" et patientez. Patientez longtemps. Prenez de quoi lire ou de quoi manger en attendant, parce que c'est assez long

Donc surtout, laissez Wally faire et ne vous impatientez pas (même si vous avez l'impression que ça n'avance pas). Il vous indiquera quand il aura terminé l'importation.

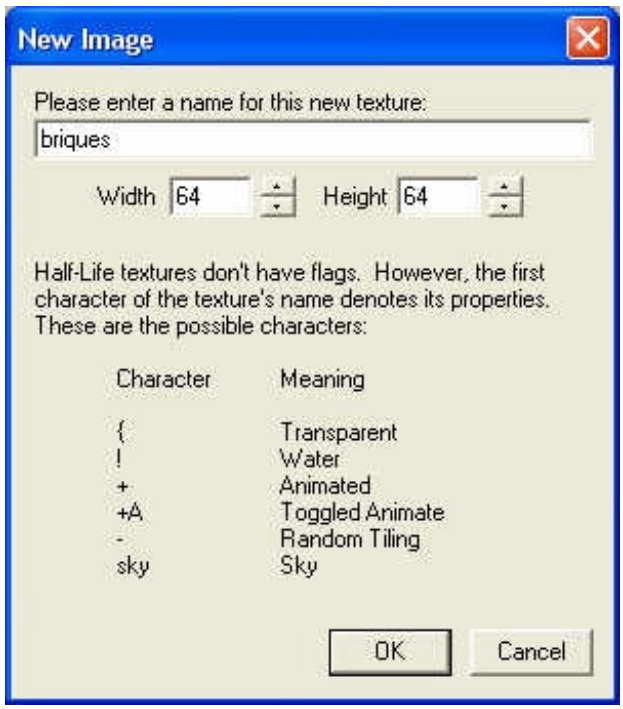
Par ailleurs, vous pouvez aussi faire un copier-coller de vos images. Copiez vos images à partir de l'explorateur Windows (Ctrl + C) et, sous Wally, allez dans le menu Edit / Past / Past as New Texture.
La conversion des images est là encore plutôt lente, alors soyez patients.

Une règle à respecter, toutes les images que vous importé doivent avoir comme dimension un multiples de 16pixels!

Créer sa texture sous Wally

Supposons maintenant que vous vouliez dessiner votre texture sous Wally. Tout a été prévu !
Bon, ce n'est pas évident quand on débute et quand on est nul en dessin comme moi, mais il ne faut pas avoir peur pour autant. On peut faire d'assez belles choses avec un minimum d'efforts

Pour créer une nouvelle texture pour votre wad, allez dans le menu Package / Create New Image. Cette fenêtre s'ouvre :

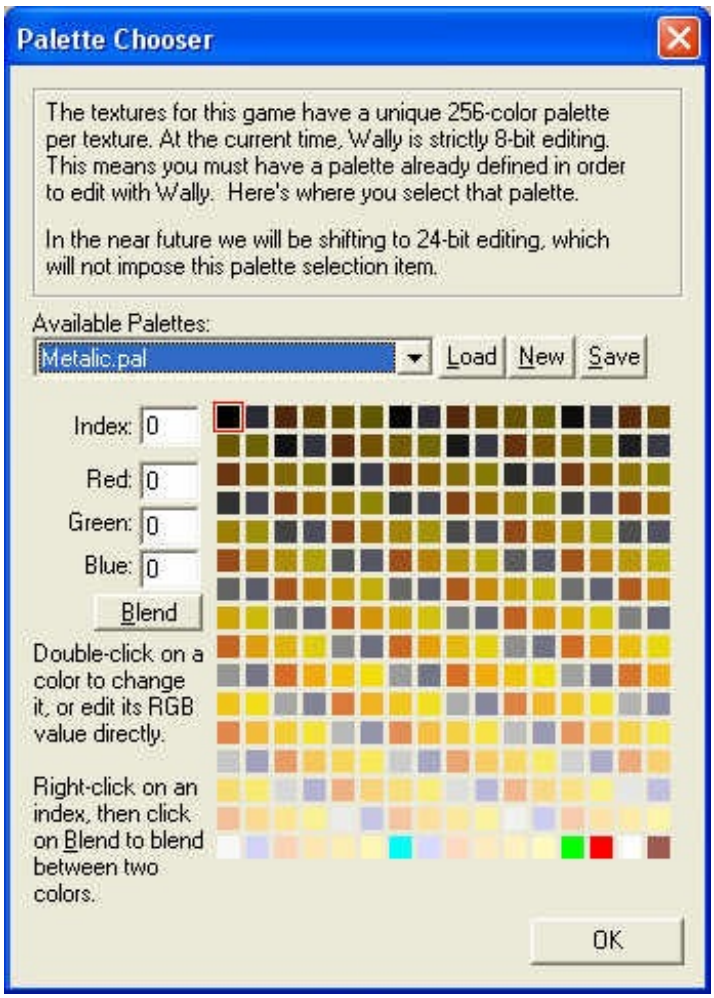


Vous devez taper en haut le nom de votre nouvelle texture.

Attention ! Ici le nom de la texture est très important. Si c'est une texture normale, sans particularités, vous pouvez l'écrire comme vous voulez. Mais si c'est une texture spéciale (comme nous l'avons vu dans le chapitre précédent), il faudra faire précéder le nom d'un symbole (ex : "{" pour une texture transparente).

Choisissez ensuite la taille de votre texture. 32*32, 64*64, 128*128, 256*256 grand maximum... La taille de 64*64 me semble être bien pour commencer.

Cliquez sur OK, et une nouvelle fenêtre s'ouvre :



Cette fenêtre est très importante, alors réfléchissez bien à ce que vous allez faire...
Vous devez sélectionner la **palette de couleurs** appropriée à votre image. Etant donné que vous êtes limité à 256 couleurs, il faut bien choisir ces couleurs.

Par exemple, la palette "Blue" ne contient que des couleurs bleues (bleu clair, bleu foncé etc...). Pareil pour la palette "Green", "Red", "Brown"... D'autres palettes vous offrent le choix entre plusieurs couleurs, mais avec moins de précision : "Smooth", "Max", "Standard"....
Enfin, des palettes sont optimisées pour certains types de textures. Exemple : "Metalic", que vous voyez ci-dessus, contient des couleurs adaptées aux objets métalliques. "Earth" contient des couleurs adaptées pour des textures de terre, d'arbres etc...

Si vous cliquez sur OK, Wally passe en mode édition de texture.
A gauche, vous avez une panoplie d'outils très complets. Je vous laisse les découvrir, vous verrez qu'ils sont très pratiques. Par exemple, le pistolet permet de simuler les trous occasionnés par des balles.
A droite, vous avez les couleurs à votre disposition...
Et au centre, vous devriez avoir un gros carré noir dans lequel vous devez dessiner. Si vous voyez d'autres carrés noirs à côtés, ce sont des vues plus petites de votre texture. Pour les mettre à jour, cliquez sur le bouton tout en bas à gauche de l'écran (ReMip Image).

Notez que vous trouverez de nombreuses fonctionnalités intéressantes dans le menu "Image".
Je vous conseille aussi, si vous faites une image qui doit se répéter en mosaïque, d'aller dans le menu "View / Tiled (Ctrl + T)". C'est très pratique

Pour revenir dans la vue générale du Wad, allez dans le menu "File / Close" (et pensez à enregistrer avant). Et voilà ! Votre image est ajoutée au Wad !
Pour retourner l'éditer plus tard, cliquez simplement 2 fois sur l'image miniature, et Wally reviendra en mode Edition.

Les sons des textures

Vous avez peut-être remarqué que les bruits de pas sont différents en fonction du sol sur lequel vous marchez... Si c'est du sable, vous entendrez des bruits de pas sur le sable. De même pour la neige, le métal etc...

Oui mais voilà, lorsque vous créez une texture de neige par exemple, et que vous testez votre map, que se passe-t-il ? Eh non, Half-Life ne reconnait pas tout seul qu'il s'agit de neige et donc du coup vous entendez un bruit de pas horrible (sur du parquet), qui n'a rien à voir avec votre texture de neige !

Comment les autres mappeurs font-ils alors ? Ils se servent d'un fichier qui se trouve dans le fichier Half-Life\NomDuMod\soundmaterials.txt. C'est lui qui indique quel son Half-Life doit jouer lorsque le joueur marche sur une texture.

Voici les différents types de sons possibles, avec la lettre qui va avec :

'M' metal
'V' ventillation
'D' dirt
'S' slosh liquid
'T' tile
'G' grate (Concrete is the default)
'W' wood
'P' computer
'Y' glass
'N' snow

Ca c'était les différents types de sons. Voici maintenant comment vous devez indiquer à Half-Life le son à jouer pour une texture :

D cubed
N cubrusha
M cudumpsterc
D cugroundb
T cuiflra
T cuiflrb
T cuiflrc
T cuiflrd
T cuiflre
T cuiflrf
M custovetop
M cutrashb
W cuwoodm

Ce fichier fonctionne comme suit : vous indiquez par une lettre le type de texture (neige, vitre, metal...), puis vous tapez le nom de votre texture.
A gauche, le type de son. A droite, le nom de la texture.
C'est compris ?

Il est toutefois recommandé de ne pas modifier ce fichier. En effet, ce fichier est redistribué à chaque nouvelle version de Counter-Strike par exemple, et si vous le modifiez vous devrez le distribuer avec votre map ! Le joueur n'a peut-être pas la version originale du fichier et vous risqueriez alors d'écraser ses données !

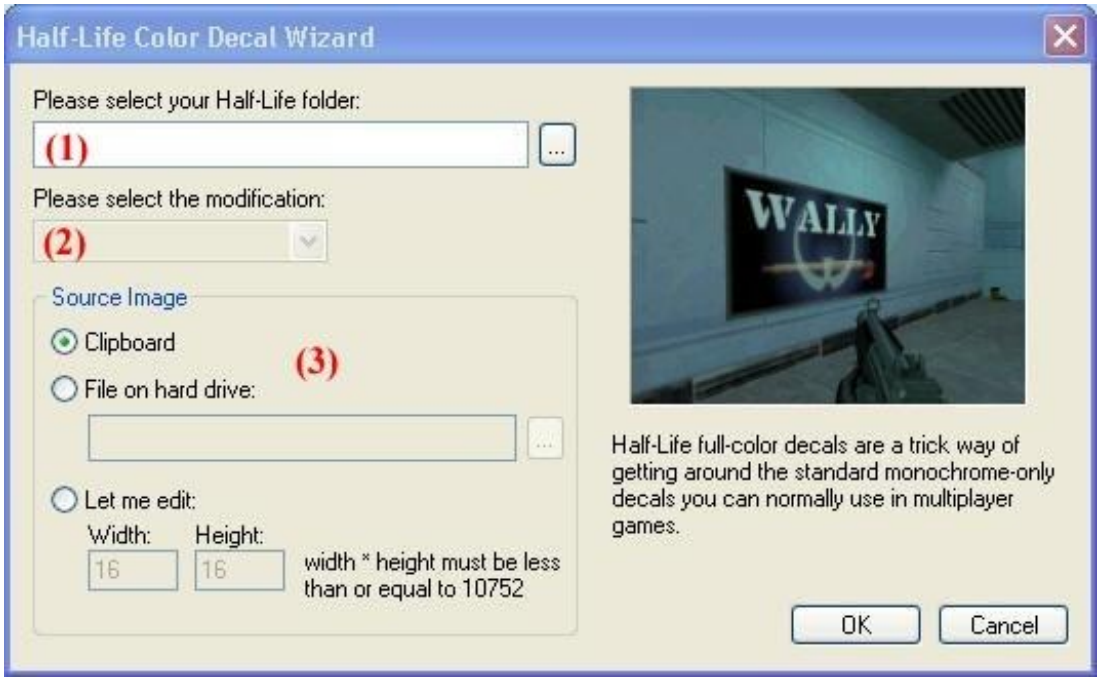
Comment faire alors ? C'est très simple : il vous suffit de nommer votre texture avec un nom déjà existant et qui correspond au son que vous voulez entendre.
Par exemple, si je voulais que ma texture ait un son de bois, je n'ai qu'à la nommer "cuwoodm" (voir plus haut) car cette texture produit déjà un son de bois comme c'est écrit dans le fichier materials.txt !

Ainsi, vous ne modifiez pas le fichier materials.txt, mais vous renommez votre texture avec un nom déjà existant. Et là elle produira le bon son. Elle est pas belle la vie ?

Créer son tag

Euh, ce n'est pas du tout en rapport avec le mapping, mais je suis persuadé que certains seront heureux d'apprendre que Wally vous permet de créer vos tags !
Que ce soit pour Counter-Strike, Half-Life etc... il est compatible avec tous les mods

C'est très facile à faire : allez dans le menu Wizard/HL Color Decal. La fenêtre suivante s'affiche :



Vous devez juste indiquer :

1. Le répertoire où est installé Half-Life.
2. Le nom du mod (pour Counter c'est "cstrike").
3. Le nom de l'image source qui contient votre tag :
    - ClipBoard : l'image se trouve dans le presse-papiers.
    - File on Hard Drive : une image bmp, jpg, ou png se trouvant sur votre disque dur.
    - Let me edit : pour créer vous-même l'image avec Wally.

Vous devez respecter certaines règles pour créer un tag : la taille de l'image doit être divisible par 16, et il ne doit pas y avoir plus de 10752 pixels en tout.

Télécharger des textures

Si vous n'avez pas le courage de créer vous-même des textures (et je vous comprends car je suis moi-même un très mauvais dessinateur :p), vous avez la possibilité de télécharger des textures sur Internet.

Les sites qui vous proposent des milliers de textures pour votre map sont rares, mais ils existent. Voici ceux que j'ai pu dénicher :

- http://www.mapping-area.com/ : un site français, créé par des Zér0s qui comme vous ont appris à mapper sur ce site ! Les Webmasters se sont rencontrés sur le Site du Zér0 et ont décidé de monter "Mapping Area", un site de ressources pour mappeurs qui propose gratuitement des textures (et nombreuses autres choses !).
- http://www.planethalflife.com/studio/ : nombreuses textures de très bonne qualité ! A noter aussi un tutorial pour vous aider à devenir un "textureur" professionnel !
- http://www.planethalflife.com/wadfather/ : sans aucun doute un des plus gros sites sur les textures. Allez le voir pour vous faire une idée. Il propose des fichiers wads et des sprites non seulement pour Half-Life, mais aussi pour Quake II et Quake III, Unreal Tournament etc...

Si vous en connaissez d'autres, n'hésitez pas à m'en faire part sur le forum. En attendant, je suis certain que vous trouverez votre bonheur sur ces sites tant ils sont riches en textures

Vous pouvez télécharger des textures de 2 manières différentes :

- En téléchargeant le wad : un wad est un fichier qui contient plusieurs textures. Si vous téléchargez un wad, vous n'aurez pratiquement rien à faire. Placez-le juste dans le dossier du mod pour lequel vous faites votre map, puis configurez Worldcraft dans les options pour qu'il prenne en compte ce wad, ou bien intégrez les textures de ce wad dans votre wad perso à partir de Wally (menu Wizard / WAD Merge, diponible uniquement dans Wally 1.55b)
- En téléchargeant texture par texture : c'est une méthode plus longue et moins pratique qui consiste à récupérer les jpeg des textures des sites. Vous devez enregistrer le jpeg sur votre disque puis l'intégrer à votre fichier wad à partir de Wally (menu Package / Add Images)

N'oubliez surtout pas de distribuer le wad avec votre map sinon les autres joueurs ne pourront pas la lancer !

Vous savez tout sur les textures. Absolument tout !
Maintenant il nous reste quelques autres petits trucs sympas à apprendre sur les bases de Worldcraft, et après on pourra attaquer la section 2 !

Les prefabs

A force de créer des blocs, vous devez maintenant commencer à vous répéter un peu... et vous êtes parfois obligés de répéter les mêmes gestes 100 fois pour une même map !
Par exemple, créer un lampadaire ou une voiture prend beaucoup de temps, surtout que vous devez paufiner à chaque fois les blocs pour que ça ne soit pas trop moche. C'est long et c'est dur !

Heureusement il y a les prefabs ! Ce que c'est ? Nous allons le voir tout de suite

Présentation

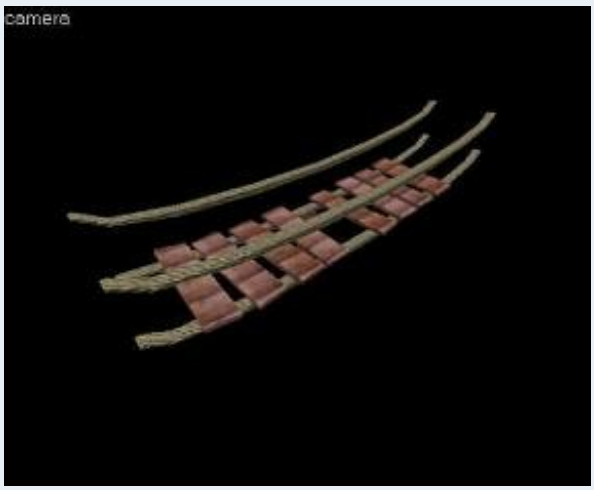
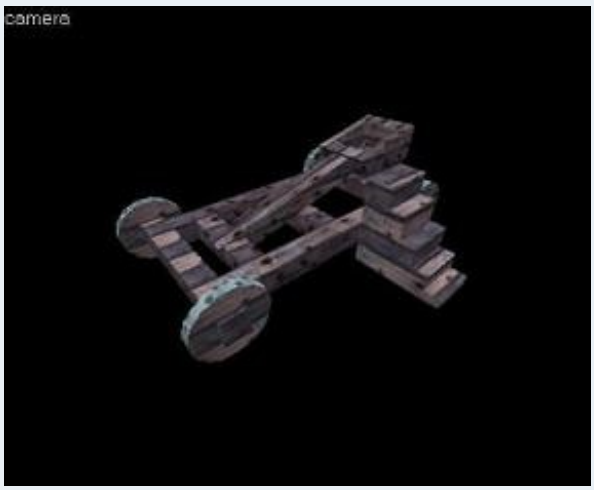
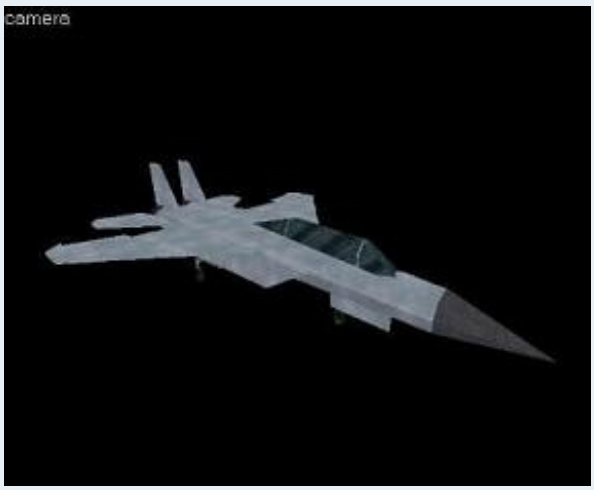
Voici un nouveau mot de vocabulaire à retenir désormais : **prefab**.

Mais qu'est-ce qu'un prefab ?

Un prefab est en fait un groupe d'objets préfabriqués que vous pouvez insérer d'un seul clic et sans effort dans votre map
Par exemple, si vous voulez mettre un camion dans votre map et que vous n'avez pas le courage de le créer de toutes pièces, vous insérez un prefab de camion et le tour est joué ! Il vous aura fallu 5 secondes, et ça c'est du temps de gagné pour le reste du mapping !

- Avantage : vous pouvez maintenant mettre des objets complexes dans votre map sans trop vous casser la tête à les créer.
- Inconvénients : il ne faut surtout pas avoir la même flême que certains mappeurs qui se mettent à insérer des prefabs un peu partout et n'importe quand. Leur map perd toute originalité et au bout du compte elle n'est pas si géniale que ça. Il faut aussi y mettre du vôtre en mapping !

Voici quelques exemples de prefabs que vous pouvez insérer en un éclair :

	
Un palmier	Un pont
	
Une catapulte	Un F-15

Sympa non ?

Bon je vois que vous brûlez d'impatience, alors je vais vous expliquer comment faire pour en insérer vous aussi dans votre map

Insérer un prefab

Pour insérer un prefab, il faut déjà en avoir en stock ! Heureusement, Worldcraft vous en livre quelques-uns... pas supers ni très variés, mais bon c'est déjà ça.

Les prefabs sont regroupés dans des catégories de prefabs. On peut avoir par exemple une catégorie "Salle de bains" où vous pourrez trouver des prefabs de toilettes, douches etc...

Pour lister les catégories de prefabs disponibles, rendez-vous dans la zone où vous choisissez habituellement la forme de vos blocs. On vous propose une catégorie, qui était jusqu'ici à "Primitives" : c'était les formes basiques des blocs. Faites dérouler ce menu pour afficher toutes les catégories de prefabs disponibles.

Je choisis par exemple "computers" (ordinateurs). Dans la zone "Objects" en-dessous, vous avez la liste des prefabs disponibles pour cette catégorie. Je vais par exemple prendre "Green Computer 2, east".
Ensuite, il me suffit de cliquer sur le bouton "Insert original prefab" juste en-dessous et le prefab sera placé dans votre map ! A vous ensuite de le positionner à l'endroit que vous voulez.

Un vrai jeu d'enfant Et voilà le résultat sous Worldcraft :

Bon OK, c'est un vieil ordinateur et il est tout cassé... encore un héritage de Gordon Freeman ça

A noter que les prefabs contiennent parfois des informations supplémentaires dans leur nom. Par exemple, ici le "Green Computer 2" a une orientation vers l'est (East). Vous avez le même prefab orienté différemment pour West, South et North

Bien entendu, vous pouvez toujours faire une rotation vous-même par la suite si vous le désirez.

Etant donné que le prefab est ressorti sous forme de groupe d'objets, il vous suffit de cliquer sur un de ses blocs pour le sélectionner en entier. Vous pouvez le supprimer en appuyant sur "Suppr", comme pour un bloc normal.

Voilà, à part ça il n'y a rien d'autre à dire tellement c'est simple...

Créer un prefab

Si les prefabs de Worldcraft vous paraissent un peu limités, vous avez la possibilité de créer les vôtres ! Comme ça, vous passez un peu de temps une fois à créer un bon prefab et après il vous sera très facile de le réutiliser.

Le meilleur moyen serait de vous montrer un exemple simple. Imaginons que je veuille créer un prefab de caisse ouverte, pour

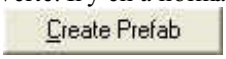
que le joueur puisse rentrer dedans. Voici ce que je fais étape par étape :

1. Je crée un bloc avec une texture de caisse.
2. J'aligne éventuellement la texture de caisse avec ce bloc grâce au bouton "Texture lock".
3. Je fais un hollowing sur ce bloc en laissant une épaisseur de 2 unités pour les "murs".
4. J'aligne les textures à l'intérieur du bloc.
5. J'active "Ignore Groups" pour pouvoir sélectionner une face.
6. Je supprime cette face.

Voilà le résultat sous Worldcraft :

Cette caisse ouverte a l'air innoffensive mais j'y ai passé quelques minutes, surtout pour l'alignement des textures qui n'était pas évident.

Bref, je n'ai pas envie de recommencer ce travail plusieurs fois, alors je vais en faire un prefab !

1. Je sélectionne les blocs qui constituent cette caisse ouverte. Il y en a normalement 5.
2. Je clique sur le bouton "Create prefab" en bas à droite. 

3. La fenêtre suivante s'affiche alors :

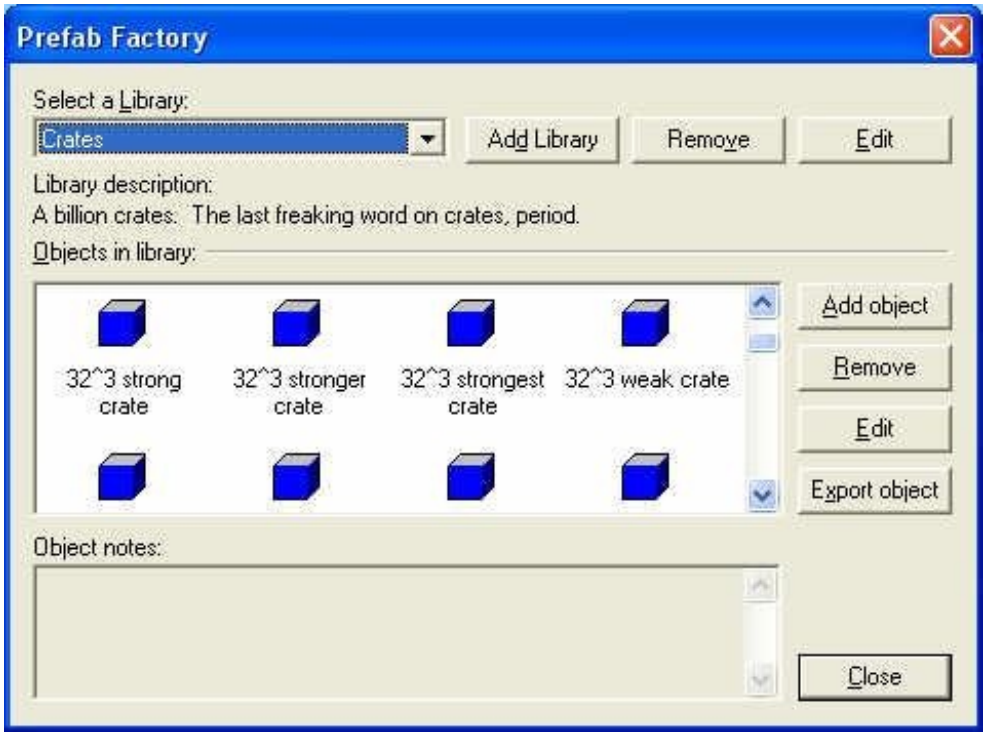
Je dois tout d'abord entrer en haut le nom du prefab, puis une brève description de ce prefab, et enfin la catégorie dans laquelle il faut le placer. Je choisis "Crates", car c'est la catégorie de Worldcraft pour les caisses.

Si votre prefab ne rentre dans aucune catégorie et que vous voulez créer la vôtre, nous allons voir comment procéder plus bas.

Hop ! Un petit clic sur OK, et le prefab est créé ! Vous pouvez maintenant l'insérer quand bon vous semble !

Bibliothèque de prefabs

Worldcraft vous propose une fenêtre dans laquelle vous pouvez gérer et administrer facilement vos prefabs. Pour l'ouvrir, allez dans le menu "Tools / Prefab Factory...".
Voici à quoi elle ressemble :



Son utilisation est simple, je ne vais donc pas vous décrire les boutons un par un.
Vous avez la possibilité de créer votre propre catégorie (library), et d'y insérer ou supprimer des prefabs. Tout simple !

Télécharger des prefabs

Enfin, il se peut que vous manquiez d'idées de prefabs à créer, ou que vous ne soyez pas encore suffisamment bons pour créer des prefabs dignes de ce nom.

Dans ce cas, sachez que le Net regorge de prefabs à n'en plus pouvoir ! Vous y trouverez vraiment de tout : des bons, des mauvais... A vous de faire le tri, parce que vous ne vous servirez vraiment pas de tous !
Mais bon, ça reste une manière simple et rapide de garnir sa bibliothèque de prefabs alors pourquoi pas ?

Je vous laisse quelques liens parmi les sites les plus connus pour les prefabs :

- http://www.mapping-area.com/ : un site français, créé par des Zér0s qui comme vous ont appris à mapper sur ce site ! Les Webmasters se sont rencontrés sur le Site du Zér0 et ont décidé de monter "Mapping Area", un site de ressources pour mappeurs qui propose gratuitement des préfabs (et nombreuses autres choses !).
- http://www.planethalflife.com/radiation/ : peu de prefabs, mais généralement de bonne qualité.
- http://www.planethalflife.com/prefablab/ : un autre bon endroit pour pêcher des prefabs...
- http://prefabs.gamedesign.net/ : beaucoup de prefabs ! Mais pas tous de bonne qualité...

Bonne pêche !

Faites très attention lorsque vous insérez des prefabs que vous avez téléchargé... En effet, nombreux sont ceux qui risquent de faire planter votre map lors de la compilation ! Ce n'est heureusement pas le cas de tous, mais méfiez-vous !


On ne sait jamais si le prefab est de bonne qualité ou pas.

Les catégories de prefabs sont enregistrées dans le répertoire de Worldcraft, dans des fichiers au format *.ol.
Malheureusement, la plupart du temps les sites de prefabs ne vous fournissent pas des *.ol mais des *.map : ce sont des fichiers de maps qui ne contiennent que le prefab. Dans ce cas c'est moins pratique : il vous faudra faire du copier-coller pour pouvoir intégrer le prefab à votre map.
Voilà un chapitre qu'il était tout simple ! Grâce à lui vous allez je l'espère constituer vos propres librairies de prefabs et gagner ainsi beaucoup de temps de mapping !

Décor naturel (A : Générer un terrain)

Gensurf est un logiciel qui permet de créer des terrains et autres formes cabossés sous Hammer. Il n'est vraiment pas recommander d'utiliser ce logiciel car il peut faire ramer votre map sous le jeu. Je vous conseille d'oublier cette partie et de passer à la suivante!

Installer GenSurf

GenSurf est un petit programme conçu pour créer des terrains aléatoires. En effet, si jamais un jour vous décidez de faire des maps à l'extérieur, comme c'est très fréquemment le cas pour Counter-Strike, vous chercherez peut-être à créer des terrains naturels et réalistes... Et comme un terrain naturel est rarement plat comme les blocs de Worldcraft, ça risque de pas faire très joli si vous ne faites pas attention

Heureusement, GenSurf est là pour vous aider ! N'attendez pas, cliquez sur le lien ci-dessous pour le télécharger :

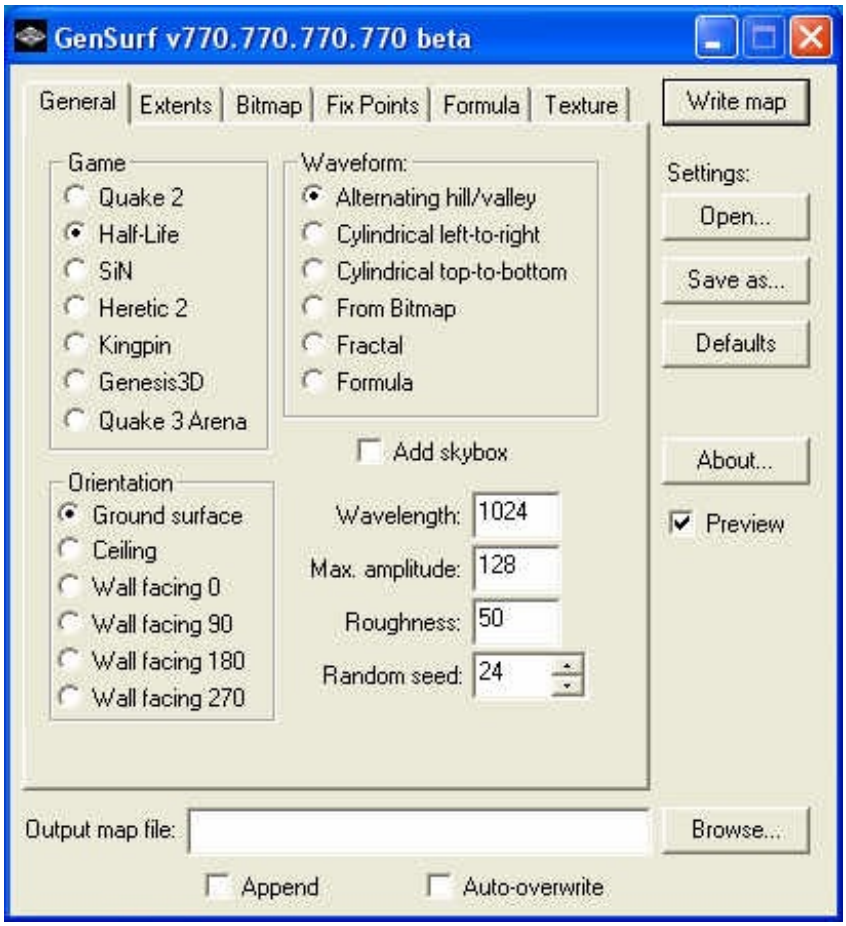
GenSurf.zip (125 Ko)

Il n'y a pas vraiment d'installation à faire : dézippez simplement le programme de GenSurf dans un dossier (celui où Worldcraft est installé par exemple). Pensez à créer un raccourci dans le menu Démarrer ou sur le Bureau pour pouvoir y accéder facilement.

Voilà, maintenant il ne vous reste plus qu'à lancer GenSurf...

Configuration de base

Il y a 2 ou 3 petites choses à configurer avant de pouvoir se servir de GenSurf. Lorsque vous démarrez le programme la première fois, il risque de vous faire un peu peur : des boutons d'option de partout, des onglets, des numéros...



En fait, il n'est pas si compliqué que ça, à condition que vous sachiez ce que vous voulez faire.

Commencez dans un premiez temps par sélectionner "Half-Life" dans la zone Game. Comme vous pouvez le voir, GenSurf fonctionne aussi pour de nombreux jeux basés sur les moteurs de Quake II et Quake III (ce qui est normal car Half-Life est basé sur le moteur de Quake II je vous le rappelle).

Ensuite, choisissez l'orientation de votre terrain. La plupart du temps, vous ferez un terrain de sol : "Ground Surface", mais si vous le désirez GenSurf peut aussi vous générer des plafonds et des façades aléatoires (ce que l'on fait rarement !).

Pensez aussi à cocher la case "Preview" tout à droite. GenSurf affichera alors une fenêtre très pratique qui vous permettra d'avoir un aperçu du terrain que vous générez.

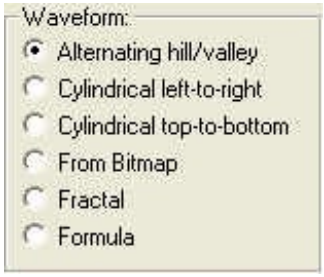
Dans l'onglet "Texture", profitez-en pour indiquer le nom de la texture à utiliser pour le sol. Par défaut, c'est OUT_GRND1 (une texture d'herbe), mais vous pouvez mettre ce que vous voulez : de la neige, du sable etc...

Voilà, le principal est configuré. Quant au reste, ce sont des options que vous aurez peut-être besoin de modifier à chaque fois. Nous allons en voir la plupart dans le prochain paragraphe où nous verrons comment générer un terrain aléatoire avec GenSurf (c'est bien pour ça qu'on est là, non ?)

Génération aléatoire

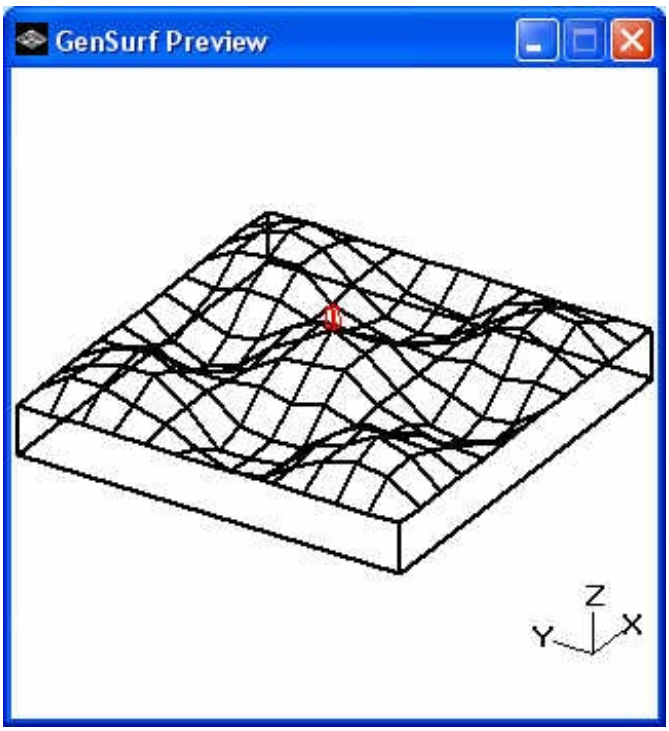
A partir de là c'est du sérieux. On va donner des indications à GenSurf pour qu'il nous génère un terrain.

Pour commencer, on doit choisir la façon de le terrain va être généré. Vous devez choisir une des options de la section "Waveform" :

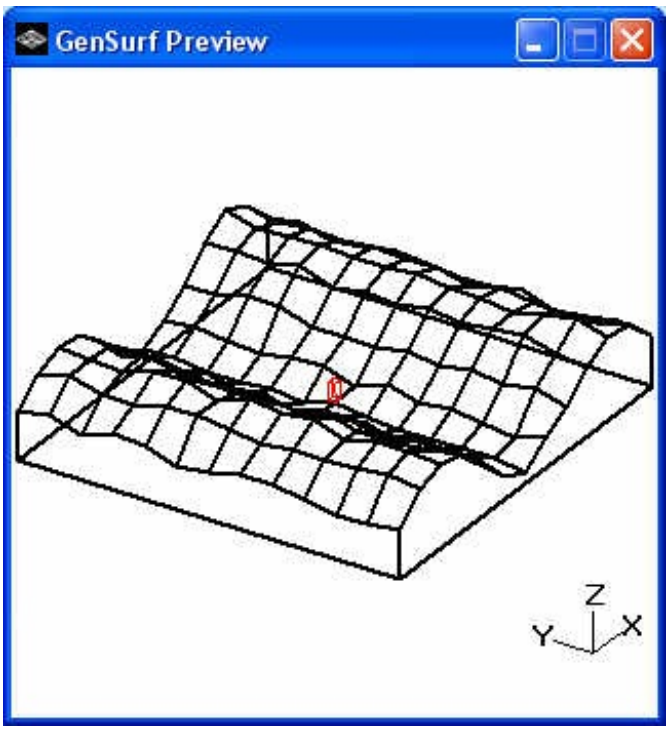


Choisissez judicieusement quelle option vous allez cocher :

- Alternating hill/valley : le terrain sera constitué de petites collines et de vallées (en alternance). Voici à quoi ressemble un tel terrain en général :

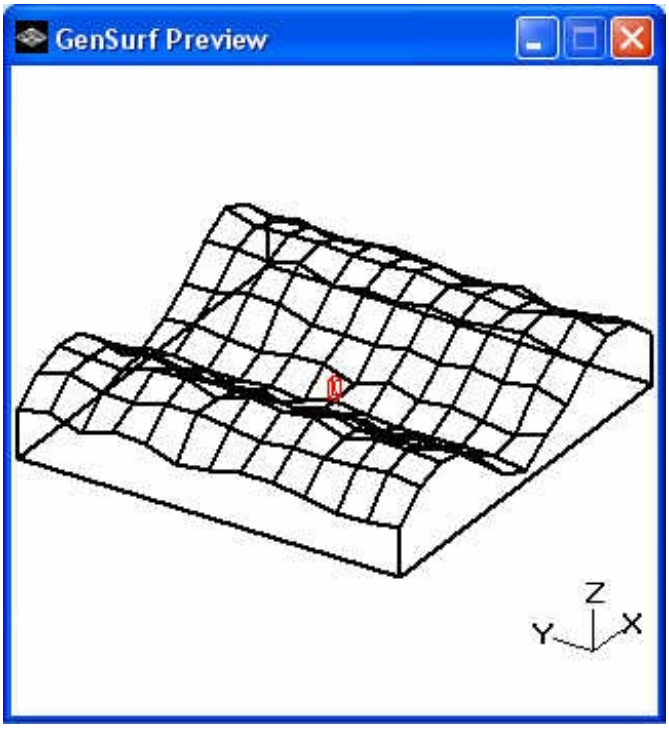


  Bref, c'est un terrain bossu et très accidenté, idéal pour simuler une zone de combats dans Day of Defeat par exemple.
- Cylindrical left-to-right : une succession de cylindres de la gauche vers la droite. Une variante assez intéressante quoique peu naturelle.

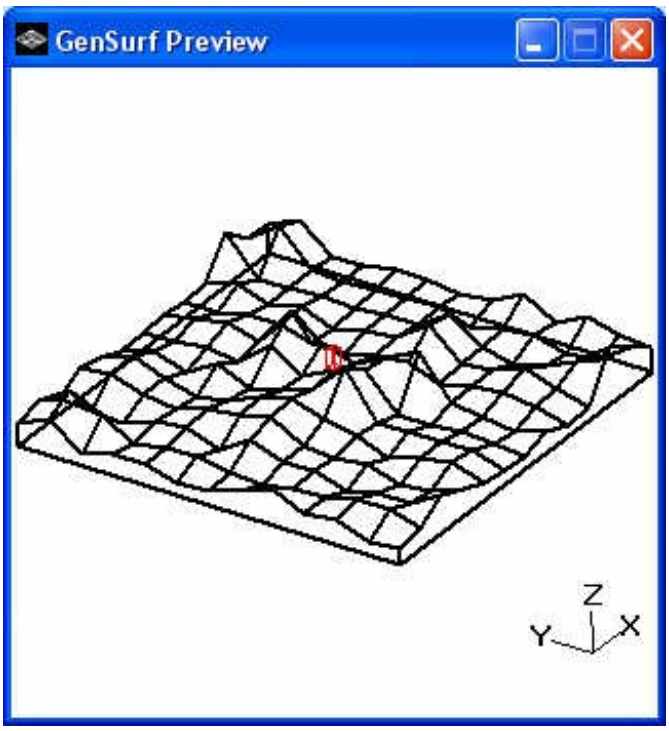


Enfin, au moins ça fait faire un peu de sport aux joueurs avec toutes ces pentes à gravir

- Cylindrical top-to-bottom : idem, mais dans le sens du haut vers le bas :

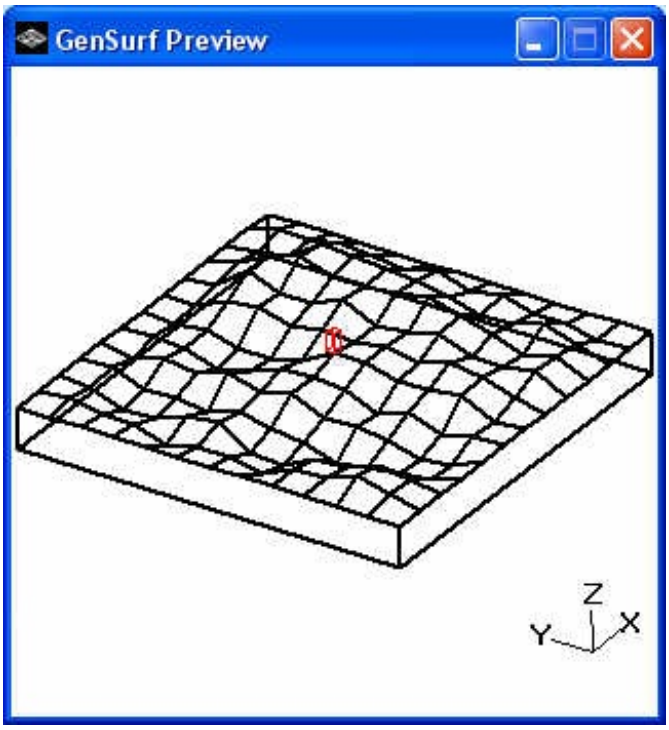


- rom Bitmap : voilà une variante très très particulière, mais intéressante. Je vous conseille de la tester... ne serait-ce que pour l'intérêt des programmeurs qui ont développé GenSurf. Voici un aperçu basé sur l'image Rhododendron.bmp qui se trouve dans le dossier de Windows :



Pour ceux qui ont lu le cours sur les Images Numériques, vous ne devriez pas avoir trop de mal à comprendre ceci : GenSurf se base ici sur la palette de couleurs de l'image bitmap pour donner une forme au terrain qui dépendra alors de l'image. Voilà d'ailleurs pourquoi ça ne marche pas pour les bitmaps 24 bits : il n'y a pas de palette !
Quant aux autres, ne cherchez pas à comprendre : essayez et vous verrez N'oubliez pas d'indiquer le nom du bitmap dans l'onglet "Bitmap".

- Fractal : voilà un mode plus simple à utiliser quand même et moins tordu. Lui, il se base sur les fractales. Ca impose pas mal de calculs au PC, mais bon en général ça ne se voit même pas !
  C'est le mode idéal que je vous conseille d'utiliser pour générer un terrain standard. Voici un exemple de terrain en "Fractal" :



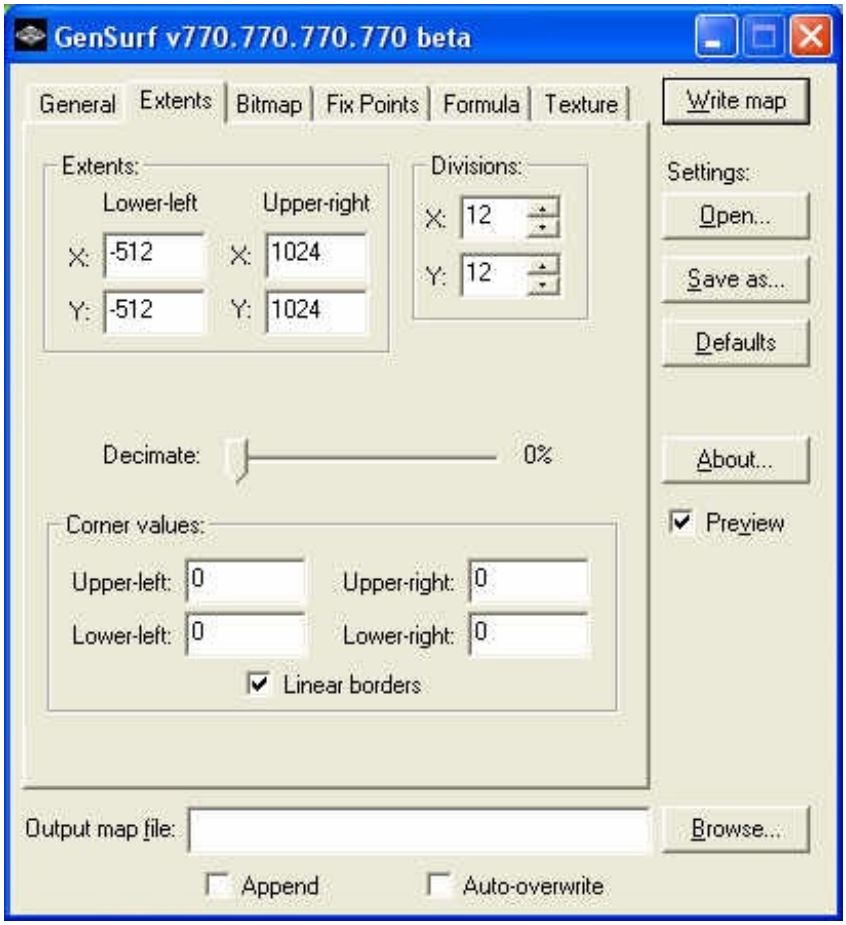
Dans ce mode, je vous conseille de modifier les valeurs Roughness et Random Seed en bas à droite de l'onglet "Général".
Roughness détermine l'écart de hauteur entre les blocs. Plus cette valeur est élevée, plus vous avez de chances de former une colline ou un gros trou
Random Seed est une valeur qui permet de rendre le terrain aléatoire. Essayez de rentrer n'importe quel nombre et le

terrain changera. En bref, changez cette valeur jusqu'à ce que le terrain vous convienne.

- Formula : dernière méthode que GenSurf peut utiliser pour gérer l'aléatoire : le formulaire. Il s'agit en réalité d'un système basé sur les feuilles de calcul de Excel (d'ailleurs vous devez avoir installé Excel), puis sur des algorithmes complexes liant les cellules l'une à l'autre. Qu'est-ce qu'ils sont allés nous inventer là ? J'en sais rien, mais moi je ne m'en sers pas en tout cas

Vous pouvez aussi modifier les valeurs Wavelength et MaxAmplitude lorsque vous êtes dans l'un des 3 premiers modes. Wavelength détermine la largeur des collines et MaxAmplitude la hauteur maximale entre le haut d'une colline et le bas d'un fossé.

Vous avez la possibilité de configurer d'autres détails pour ce terrain. Rendez-vous cette fois dans l'onglet "Extents" (mode étendu) :



En haut à gauche, vous pouvez indiquer la taille de votre terrain. Si vous trouvez qu'il est trop petit ou trop gros, alors entrez ici la taille qui vous convient.

"Divisions" correspond au nombres de divisions de blocs sur l'axe des X et des Y. Plus il y a de divisions, plus il y aura de blocs et plus le terrain sera affiné (plus joli). Attention à ne pas exagérer ces valeurs car vous risqueriez de mettre trop de blocs à la fois et de faire par la suite "ramer" votre map !

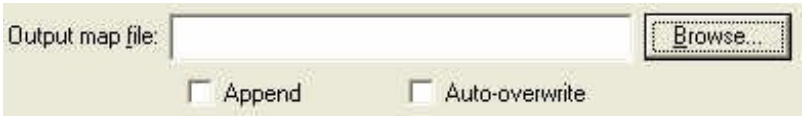
"Decimate" vous permet de faire d'autres divisions en diagonale des blocs. Si vous mettez une valeur trop élevé le terrain sera tout vide ! Moi je ne me sers pas de cette fonction car elle n'a que des inconvénients à mon goût.

Enfin, "Corner Values" vous permet de modifier la hauteur des bords du terrain...
Si vous décochez "Linear borders" alors le terrain ne sera plus aligné sur les bords (laissez coché, c'est une fonction pratique !).

Récupérer le terrain sous Worldcraft

Maintenant que vous avez enfin trouvé le terrain qui vous plaît, il est temps de le récupérer dans votre belle map

Pour ce faire, penchez-vous sur la ligne toute en bas du programme : "Output map file".



Cliquez sur "Browse" pour choisir où enregistrer la map. En effet, GenSurf va créer un fichier au format *.map contenant uniquement le terrain.

Dans la fenêtre Browse, il y a un bouton "Ouvrir", mais il faut en fait comprendre "Enregistrer". Vous n'avez pas à ouvrir de fichier de map, c'est GenSurf qui va le créer ! C'est certainement une erreur du programmeur...
Tapez donc le nom du fichier qui va contenir le terrain : "terrain.map" par exemple.

Cliquez ensuite sur le bouton "Write map" en haut à droite et le tour est joué ! GenSurf vous affiche un message vous indiquant que tout s'est bien déroulé :

Il vous indique par ailleurs le nombre de blocs dont il s'est servi pour créer ce terrain.

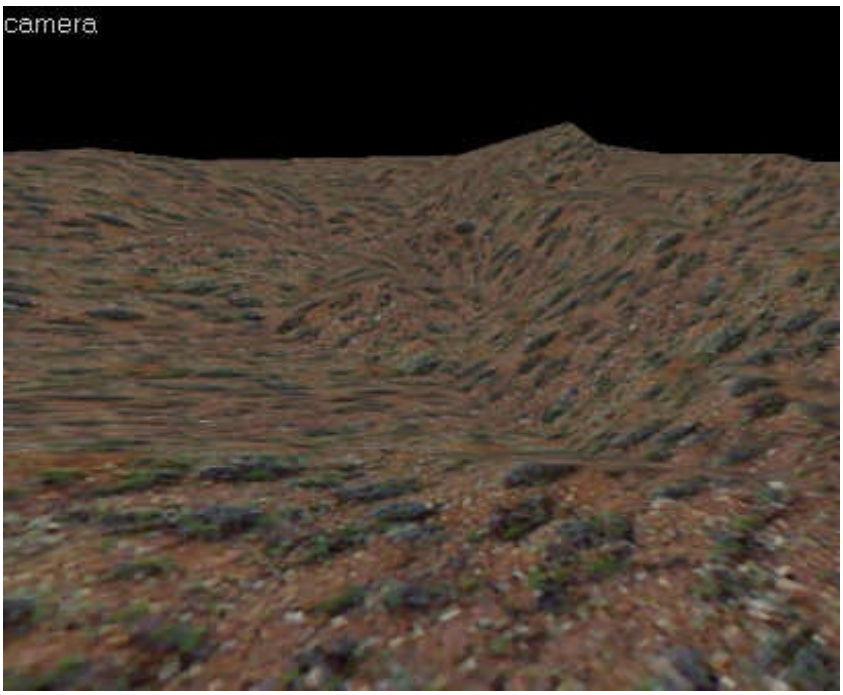
Pour intégrer le terrain à votre map, faites comme ceci :

- Ouvrez la map du terrain sous Worldcraft
- Sélectionnez le terrain
- Copiez le terrain (CTRL + C)
- Ouvrez votre vraie map
- Collez le terrain (CTRL + V)

Et voilà ! Le terrain est maintenant dans votre map !

Pour les sceptiques qui douteraient encore de la fiabilité de GenSurf, je vous rassure : GenSurf ne plante pas. Vous ne risquez pas d'avoir une erreur à la compilation par sa faute. Le terrain généré est tout simplement parfait

Voici à quoi ressemble le terrain une fois sous Worldcraft :



Bluffant non ? Imaginez le temps que vous auriez dû passer si vous aviez fait ce terrain vous-même lol. Ce terrain est le fruit de nombreux calculs aléatoires de votre ordinateur qui s'est chargé ensuite de placer des centaines de blocs en modifiant leurs vertices, de manière à produire ce superbe terrain que vous voyez là.

En clair, quand vous voulez mettre un joli terrain dans votre map, pensez à GenSurf !
Nous savons maintenant générer des terrains aléatoires... mais si on cherche à en créer un précis ? Si on veut modeler son décor naturel à sa façon ?
C'est là que Terrain Generator entre en jeu. Nous allons l'étudier dans le prochain chapitre.

Décor naturel (B : Modeler un terrain)

De même que Gensurf, Terrain Generator est un logiciel qui permet de créer des terrains et autres formes cabossés sous Hammer. Mais cette fois c'est vous qui décidé de leurs formes! Il n'est vraiment pas recommandé non plus d'utiliser ce logiciel car il peut faire ramer votre map sous le jeu. Je vous conseille d'oublier cette partie et de passer dès à présent aux entités!

Installer Terrain Generator

Terrain Generator est un programme plus lourd que GenSurf, par conséquent le programme est assez gros (3 Mo), mais il en vaut vraiment le coup !

TerrainGenerator_v2.2.0.zip (3 Mo)

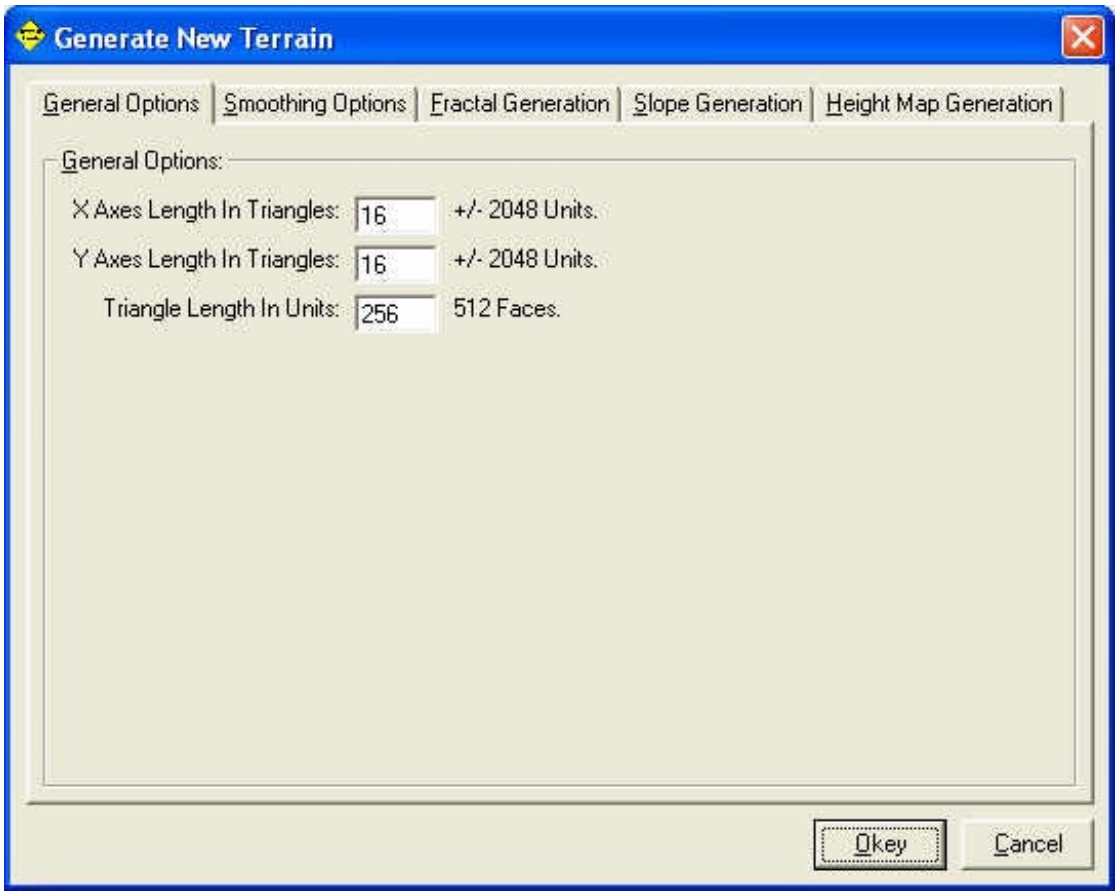
Pour l'installer, dézippez tous les fichiers dans un dossier temporaire et lancez "setup.exe". Une fois l'installation terminée, vous pourrez supprimer les fichiers du dossier temporaire.

Le programme d'installation va vous créer une icône dans le menu démarrer pour lancer le programme. Allez-y, lancez-le n'ayez pas peur

Préparer un nouveau terrain

Lorsque vous démarrez le programme, il a l'air un peu vide... ce n'est qu'une impression je vous rassure.

Pour commencer, on va demander à Terrain Generator de nous préparer un terrain vierge, qu'on pourra modifier, diviser, tordre, torturer à souhait
Le menu File / New va vous ouvrir la boîte de dialogue suivante :



Et là vous vous dites : "Argh !!! Encore un programme compliqué !". Eh bien non, pas forcément... En fait, seul le premier onglet (General Options) est obligatoire à renseigner.
Tous les autres onglets permettent de... générer aléatoirement le terrain de base, à l'aide de fractales, de bitmaps etc... Exactement comme GenSurf, mais en encore plus compliqué

Vous n'avez absolument pas besoin de toutes ces fonctions pour apprécier la puissance de TerrainGenerator. Renseignez les champs suivants et ce sera amplement suffisant :

- X Axes Length In Triangles : nombre de triangles (blocs) pour un côté du terrain.

- YAxes Length In Triangles : nombre de triangles (blocs) pour l'autre côté du terrain.

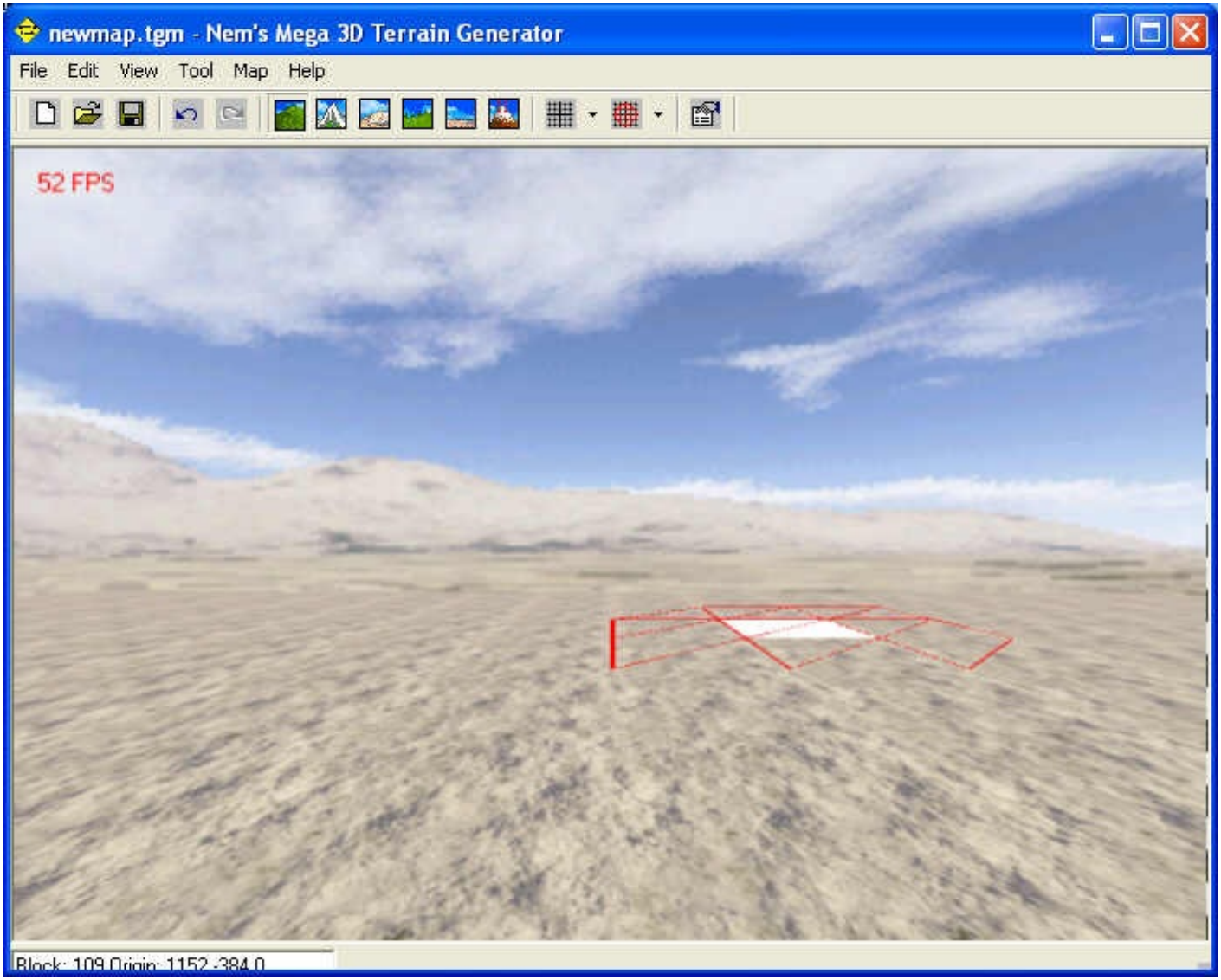
  Vous pouvez aller jusqu'à 2048 unités, mais vous verrez que 16 c'est vraiment suffisant. Faites très attention à ne pas beaucoup augmenter ces valeurs, sinon le nombre de blocs dans votre map va augmenter en même temps que vos r_speeds

- Triangle Length In Units : longueur du triangle en unités de Worldcraft.

Mais, comme vous allez bientôt vous en rendre compte, on modifie rarement ces valeurs. On garde le terrain par défaut, c'est-à-dire qu'on clique immédiatement sur "Okey" sans se préoccuper du reste.

Les outils du modelage

Après avoir cliqué sur "Okey", le terrain vierge s'affiche :

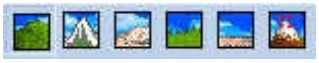


Au centre de l'écran, vous avez un terrain plat prêt à être modelé.

Il y a ici un sol plat, avec une texture un peu désertique, et un fond de type sky (rappelant là encore le désert). Si votre map n'a rien à voir avec ce décor, vous pouvez modifier la texture utilisée dans le menu Map / Properties.
Quoiqu'il en soit, ce n'est pas très grave et ne vous préoccupez pas du sky, on peut changer tout ça dans Worldcraft. Tout ce qui nous intéresse ici, c'est la forme à donner au terrain.

Primo, il faut savoir se déplacer dans ce logiciel. Et ce n'est pas pareil que dans Worldcraft, voyez-vous même :

- Bouton droit de la souris : en maintenant enfoncé le bouton droit tout en bougeant la souris, vous regarderez dans une direction différente.
- Flèches du clavier (Haut Bas Gauche Droite) : pour se déplacer dans le niveau. A noter que les touches Z, S, Q et D marchent aussi.

 Justement, les outils à votre disposition sont situés dans la barre d'outils du haut.

Il y en a 6 comme vous pouvez le voir. Nous allons les étudier un par un, de gauche à droite :

- Raise / Lower Tool : c'est l'outil par défaut, le plus important et le plus utile ! Comme vous pouvez le voir, lorsque vous passez la souris sur le terrain, il y a un cercle rouge. C'est la zone actuellement sélectionnée, que vous pouvez modifier. A l'aide du bouton gauche de la souris, vous pouvez faire monter ou baisser le terrain à cet endroit. Essayez, vous verrez c'est très facile !
- Mountain / Valley Tool : cet outil, à condition de bien s'en servir, permet de créer automatiquement une montagne ou une vallée au milieu de votre terrain. Il permet aussi de ramener tout le terrain à la même hauteur.
- Smooth Tool : très pratique, il permet de "lisser" la partie sélectionnée du terrain... ce qui permet de donner un aspect plus réaliste à votre terrain.
- Random Noise : modifie aléatoirement la partie sélectionnée du terrain. On appelle ça "créer du bruit" (noise). Cela vous permet de rendre le terrain un peu plus "naturel" à cet endroit.
- Flatten Tool : aplatit la partie sélectionnée du terrain.
- Touch Up Tool : un peu plus délicat cet outil. Il vous permet de modifier la position d'une vertice (et d'une seule). La vertice sélectionnée est indiquée par un petit cube. Pour changer de vertice sur le bloc sélectionné, utilisez la molette de la souris. Cet outil est le plus précis de tous, mais je vous conseille de ne l'utiliser qu'à la fin pour les "finitions".

Notez que la flèche de la souris est en décalage par rapport au cercle rouge au sol. Ce n'est rien de grave mais il faut s'y habituer. Faites comme si la flèche de la souris n'était pas là, seul le cercle rouge compte.

A noter que le menu Map permet de faire des choses intéressantes :

"Smoothing Pass" lisse tout le terrain, comme le fait l'outil "Smooth Tool", mais cette fois sur toute la surface du terrain.
"Random Noise Pass" ajoute du bruit dans tout le terrain, comme l'outil "Random Noise". A utiliser tout au début pour avoir un terrain déjà naturel.
"Scaling Pass" fait un Scale sur tout le terrain (même technique que dans Worldcraft, appliquée sur les textures).

Récupérer le terrain sous Worldcraft

Une fois le terrain manipulé sous Terrain Generator, il est temps de le récupérer dans Worldcraft.

Pour commencer, allez faire un tour dans le menu File / Export / Export Options. Cette boîte de dialogue contient des informations importantes :

A gauche, vous pouvez choisir un "style" d'exportation. Laissez "Best fit style", ça ira très bien.
En haut à droite, je vous conseille fortement de décocher les cases "Add Hint Brushes" et "Add Sky Brushes". Les hint brush n'auraient pas beaucoup d'effet sur votre map, si ce n'est de la rendre peu visible sous Worldcraft, et le sky génèrerait ici un sky box, ce qui n'est pas recommandé là non plus.
Enfin, en bas à droite vous pouvez faire en sorte que ce terrain soit du sol (Flatten Bottom) ou du toit (Flatten Top, idéal pour le toit d'une grotte par exemple).

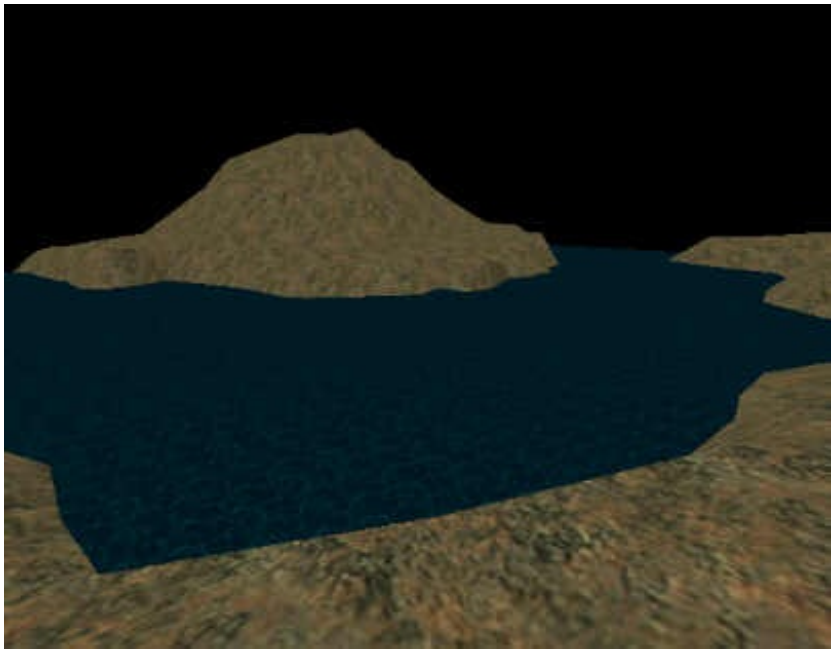
Pour exporter votre terrain, allez dans le menu File / Export / Export To .map. Vous choisirez ensuite où vous voulez enregistrer votre map.

Allez maintenant dans Worldcraft, menu File / Open, et en bas de la fenêtre qui s'ouvre choisissez Fichiers de type : "Game maps (*.map)". Ouvrez votre map.

Si vous avez une map déjà en cours de création ailleurs, voici ce qu'il faut faire pour placer ce terrain dans votre map :

1. Sélectionnez tous les blocs du terrain en créant un large rectangle dans les vues 2D à l'aide du "Selection Tool".
2. Copiez ce terrain (Ctrl + C).
3. Ouvrez votre "vraie" map en cours de création.
4. Faites "Coller" dedans (Ctrl + V).
5. Vous pouvez aussi en profiter, tant que le terrain est sélectionné, pour utiliser l'outil "Apply Current Texture" afin de modifier la texture du terrain.

Moi j'ai juste rajouté un peu d'eau, et regardez ce que ça donne en un minimum d'efforts :



Fait en 5 minutes à tout casser
Rien à dire, vos maps ne seront jamais plus comme avant !

Je vous laisse imaginer toutes les possibilités : un fort protégé sur une montagne, un terrain après des bombardements dans Day of Defeat, une grotte souterraine etc...
Voici comment fonctionne Terrain Generator!
Désormais, il est temps de s'attaquer aux entités!

Partie 2 : Entités et effets spéciaux

(Développpez vos connaissances grâce aux entités
C'est la partie la plus importante et la plus longue)

Les entités

Rappelez-vous : on distingue deux grandes familles d'objets dans Worldcraft :

- Les blocs, que nous avons suffisamment étudiés...
- ... et les entités : c'est le gros morceau qui nous reste à voir.

Autant vous le dire de suite : il y a du pain sur la planche !
Allez on y va ! (et avec le sourire)

Entités-point & entités-bloc

Très important !

On distingue 2 sortes d'entités :

- Les entités-point
- Les entités-bloc

Les entités-point

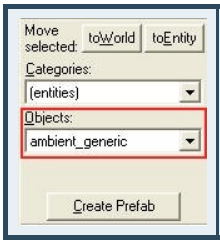
Nous commencerons à nous intéresser aux entités-point.

Oui, mais c'est quoi une entité-point ?

Analysons ce terme. Il est constitué de 2 mots : **entité** et **point**.

Comme je vous l'ai déjà expliqué, les **entités** font partie des maps au même titre que les blocs. Elles donnent d'autres informations très utiles, comme la lumière, les spots, les armes, les munitions, les ennemis, les explosions, les particules, les sprites etc etc...
Le mot point signifie que le position de l'entité est définie par un point (dans l'espace). Une entité-point n'a donc pas vraiment de taille réelle, elle a juste une position bien précise.


Pour créer une entité-point, il faut utiliser ce bouton de la barre d'outils de gauche. C'est le bouton "Entity Tool".

	Une fois activé, il faut choisir quelle entité-point on va créer. On utilisera pour cela le même menu déroulant que lorsqu'on choisissait une forme de bloc (en bas à droite).

Notez que selon le fichier FGD que vous utlisez, la liste des entités ne sera pas forcément la même. Pour l'instant, je vais commenter quelques entités communes à tous les mods.

Choisissez par exemple l'entité "light" (lumière). On pourra cependant changer le type d'entité par la suite.

Placez votre souris dans une vue 2D. Le curseur se transforme en une croix accompagnée d'une hache. Cliquez à l'endroit où vous voulez.
Ajustez ensuite la position de votre entité-point dans les autres vues 2D (maintenez pour cela le bouton gauche de la souris enfoncé).
Enfin, pour valider l'entité, tapez "Entrée".

Vous devriez repérer dans la vue 3D un dessin d'ampoule (à condition d'être en mode "3D Textured Polygons") . C'est l'entité que vous venez de créer. Vous pouvez la sélectionner avec le "Selection Tool" :  .

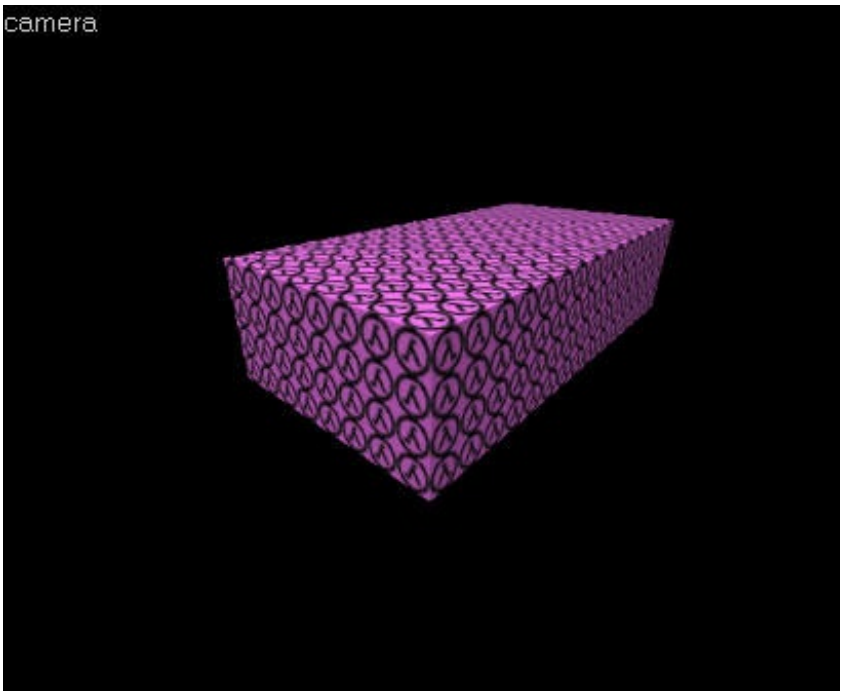
Vous devriez voir ceci :

Les entités-bloc

A la différence des entités-point, les **entités-bloc** ont une position et une forme précise. Elles couvrent un espace bien déterminé.

Au départ, une entité-bloc n'est rien d'autre qu'un simple bloc. Pour le transformer en entité, il faut le sélectionner et cliquer sur le bouton "ToEntity".A tout moment, vous avez la possibilité de remettre le bloc comme il était avant, en cliquant sur "ToWorld".

Généralement, l'entité-bloc n'est pas visible. Dans ce cas, pour se repérer, le mappeur a l'habitude de lui appliquer la texture "AAATRIGGER", comme vous pouvez le voir ci dessous :

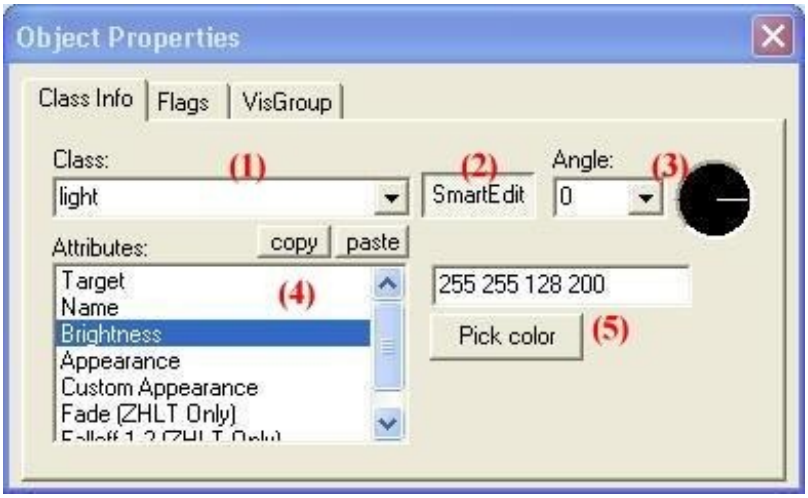


Sinon, si le bloc est visible, vous pouvez lui appliquer la texture de votre choix sans problème.

Mis à part ces petits détails, une entité-bloc et une entité-point se ressemblent. D'ailleurs, l'une et l'autre possèdent une fenêtre de propriétés comme nous allons le voir maintenant.

Les propriétés d'une entité

Ce qu'il y a de plus important dans une entité, c'est sa partie "Propriétés". Pour l'afficher, sélectionnez l'entité et faites la combinaison clavier Alt + Entrée (ou cliquez avec le bouton droit de la souris et sélectionnez "Properties").
La fenêtre suivante apparaît :



Il faut bien repérer les 3 onglets :

- Class Info
- Flags
- VisGroup

Le plus important est "Class Info". Nous allons donc l'analyser attentivement. Reportez-vous à l'image ci-dessus pour savoir à quoi correspondent les numéros.

1. Class : dans le menu déroulant, vous avez la possibilité de modifier le type de l'entité, comme je vous l'avais promis. Pour l'instant, laissez "light".
2. SmartEdit : ce bouton devra toujours rester enfoncé (sauf cas exceptionnel que je vous signalerai). Il permet d'activer/désactiver le mode "simple" d'édition d'entité. Alors, comme c'est déjà assez compliqué comme ça, n'allez pas vous prendre encore plus la tête
3. Angle : cela permet de définir l'orientation de l'entité (on ne s'en sert pas dans tous les cas). Il faut distinguer 2 parties :
   - A gauche, vous pouvez taper manuellement une valeur (en degrés) ou sélectionner "Up" (Haut) ou "Down" (Bas).
   - A droite, une sorte de radar vous permet d'avoir un meilleur aperçu de l'angle que vous choisissez.
4. Attributes : c'est maintenant que ça devient intéressant. La parties "Attributes" vous affiche tous les attributs de l'entité. En effet, toute entité possède des attributs qui lui sont propres.

   En ce qui concerne les 2 autres attributs que vous pouvez voir sur la capture d'écran, ils n'apparaissent que si vous utilisez le fichier FGD avancé de Counter-Strike (celui que je vous ai fait télécharger). Ils n'ont pas vraiment d'importance pour le moment...
5. Valeur de l'attribut : à droite s'affiche la valeur de l'attribut que vous avez sélectionné. Vous avez la possibilité de la modifier. Selon le cas, on tape une valeur numérique (12345...), alphanumérique (abcde12345...), ou bien on fait un choix dans un menu déroulant (ce qui est plus sympa).

Il nous reste à voir encore 2 onglets : "Flags" et "VisGroup".

L'onglet "Flags" est assez important puisqu'il regroupe d'autres propriétés de l'entité (les drapeaux) que l'on appelle couramment en langage de mappeur... les flags ;). Il s'agit d'une liste de cases à cocher, que vous pouvez cocher/décocher comme bon vous semble.
Ici, "light" possède un seul flag : "Initially Dark", qui permet de faire en sorte que la lumière soit éteinte au départ (jusqu'à ce qu'on l'allume avec un bouton).

Notez que certaines entités ne possèdent pas de flags.

En ce qui concerne l'onglet "VisGroup", on l'utilisera quelques fois mais pas très souvent.
Il sert en gros à affecter une même fonction à plusieurs entités-bloc (oui, ça ne marche que pour ce type d'entité).
Je vous en reparlerai en temps voulu.

Le gros travail, donc, consiste à décrire la fonction de chaque entité et de ses attributs. Vu le nombre d'entités et d'attributs, y'en a pour un moment...

Alors, voilà à quoi j'ai pensé :
Plutôt que de vous décrire chaque entité une par une, je vais créer des mini-chapitres par thème. Par exemple, le thème de la

lumière regroupera l'entité "light", "light_environment" et "spot" ; celui de Counter-Strike les entités spécifiques à Counter ("info_hostage_rescue", "info_vip_start", "info_bomb_target" etc...)...

Cette technique a un avantage et un défaut :

- L'avantage : si vous débutez vraiment en mapping et que vous cherchez à faire un effet précis, ou à suivre un ordre pour lire ce cours, alors cette technique est appropriée.
- Le défaut : si vous voulez savoir exactement à quoi sert une entité bien précise, vous aurez du mal à vous y retrouver. C'est pourquoi je créerai plus tard un chapitre en annexe, listant toutes les entités par ordre alphabétique : ce sera plus pratique pour ceux qui maîtrisent déjà bien Worldcraft.

Nous étudierons donc les entités par thème dans les chapitres suivants 

Position de départ d'un joueur

La première des entités que nous devrions voir, c'est celle-là.
Elle permet de définir la position de départ du joueur... parce qu'il faut bien qu'il commence quelque part non ?

Erreur très grave : il ne faut jamais oublier de mettre un départ au joueur. Sans départ, le joueur sera placé aux coordonnées (0,0,0) de la map, c'est-à-dire au centre.
Et si il n'y a rien au centre (du vide), préparez-vous à en voir de toutes les couleurs !!!

Il y a plusieurs types de départ (faut croire qu'un seul c'est pas assez) :

- Solo
- Coopération
- Multijoueur

Nous verrons par la suite comment cela fonctionne pour Counter-Strike (là encore c'est différent !).

Départ dans une aventure solo

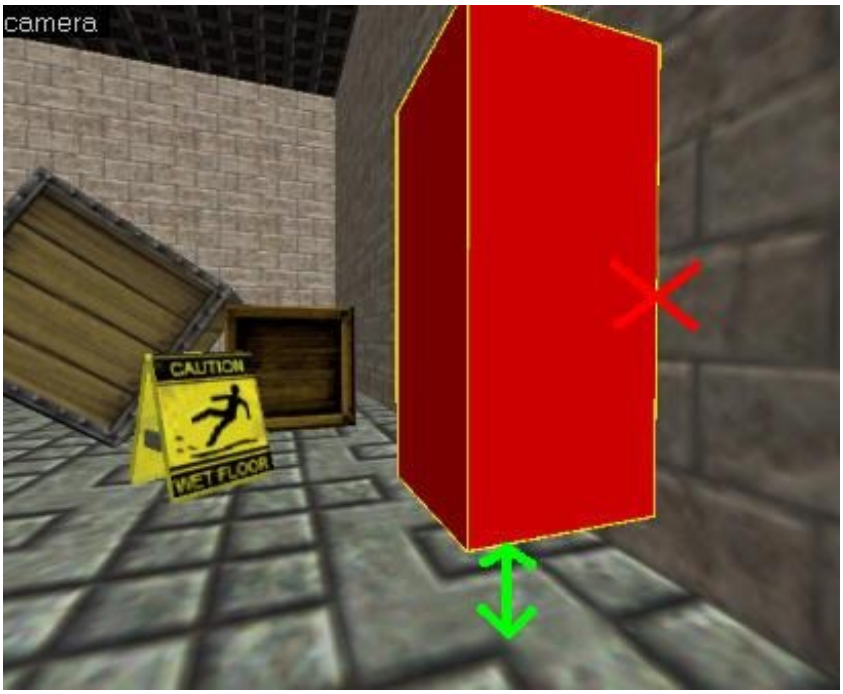
Entité concernée : info_player_start
Type d'entité : entité-point
Difficulté : très facile

L'entité ***info_player_start*** marque le point de départ du joueur dans une map solo (pour un seul joueur). Il ne doit y en avoir qu'une seule dans votre map !

Si vous utilisez le FGD de Half-Life, l'entité est représentée par un cube vert fluo bien immonde. Il représente l'espace que couvre votre joueur au moment où il entre dans la map.

Lorsque vous placez une entité de départ (que ce soit en solo ou en multijoueurs), veillez à ce qu'elle ne touche pas de bloc. En effet, si elle est en contact direct avec un bloc, le joueur restera coincé avant même d'avoir pu remuer un doigt !

Appuyons-nous sur cet exemple, qui nous montre ce qu'il faut faire et ne pas faire :



L'entité est bien surélevée par rapport au sol, ce qui évite qu'elle soit en contact direct avec un bloc. Le joueur tombera un peu mais ne se fera pas mal.
Par contre, l'entité est collée au mur : ce n'est pas bon car le joueur restera collé à ce mur. Il faudrait l'écarter un peu plus, et on n'aurait pas de problème !

Passons maintenant aux propriétés (attributs) de cette entité... Héhé, elle n'en a pas !

Vous pouvez par contre définir son orientation (à l'aide du radar). C'est très important : c'est la direction dans laquelle le joueur regardera lorsqu'il arrivera dans la map. Ca ferait mauvaise impression de lui montrer un mur dès qu'il commence à jouer, non ? ;o)

Quand je dis que l'entité n'a pas de propriété, c'est un peu faux quelque part : il y a toujours une propriété d'orientation. D'ailleurs, si vous utilisez le FGD de Counter-Strike, vous devriez voir un attribut "Pitch Yaw Roll (Y Z X)", où vous pouvez définir manuellement l'orientation.

Cette entité possède un seul flag : "Not In Deathmatch". Il a très peu d'intérêt : il permet de faire disparaître l'entité si la map est jouée en multijoueurs. Mais comme votre map est SOIT pour 1 joueur, SOIT pour plusieurs joueurs, il ne risque pas d'y avoir de confusion...

Départ dans une map multijoueur

Entité concernée : info_player_deathmatch
Type d'entité : entité-point
Difficulté : très facile

Cette entité est destinée à une map multijoueurs (deathmatch = match à mort). Elle est valable pour Half-Life et beaucoup de mods... sauf Counter-Strike, que nous verrons plus bas.

Vous devez mettre plusieurs ***info_player_deathmatch*** (autant qu'il y aura de joueurs). S'il y a plus de joueurs que de départs, certains mourront avant même d'avoir commencé ! Méfiez-vous !

Cette fois, il y a des propriétés. 2 propriétés :

- Target : c'est le nom de l'entité qui est activée lorsqu'un joueur démarre. Ne vous en préoccupez pas pour le moment, ce n'est pas très important.
- Master : là c'est encore plus compliqué ! C'est la condition pour que "Target" soit activé. Nous en reparlerons plus tard car c'est vraiment pas facile...

Il y a un flag, on se demande ce qu'il fait là : "Not in deathmatch". L'entité disparaît en deathmatch... euh c'est un peu bidon non ?

Voilà, jusqu'ici ça reste très simple. C'est valable pour HL et de nombreux mods... Mais pour Counter-Strike, c'est un peu différent. C'est ce que nous allons voir maintenant.

Le cas de Counter-Strike

Entités concernées : info_player_start et info_player_deathmatch
Type d'entité : entité-point
Difficulté : très facile

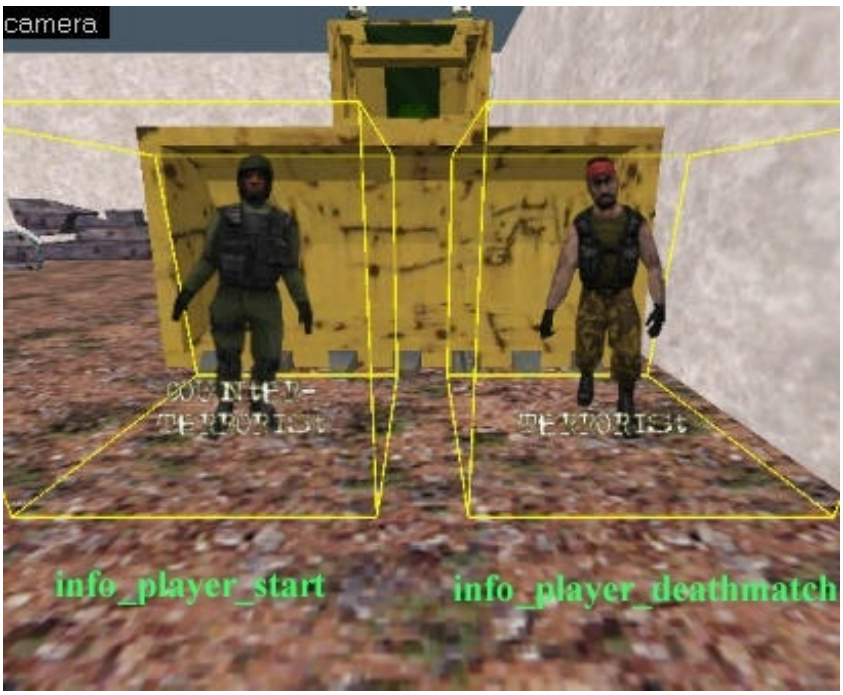
Bien, si vous avez l'intention de faire une map pour Counter-Strike, vous devez oublier tout ce que je viens de vous apprendre plus haut... C'est totalement différent.
Je vous rassure, ça reste très très simple !!!

Voici comment cela fonctionne :

- info_player_start correspond au départ d'un *anti-terroriste*
- info_player_deathmatch au départ d'un *terroriste*

Et c'est tout ! Vous devez utiliser, bien entendu, le FGD de Counter-Strike que je vous ai donné dans le premier chapitre. Sinon, les entités n'apparaissent pas correctement.

C'est clair ou je vous fais un dessin ? Bon, comme vous êtes sur le site du Zér0, je vous offre le dessin en cadeau au cas où vous auriez pas compris :



Vérifiez que vous mettez AUTANT de Terroristes que d'Anti-terroristes. Si l'auto-team balance est activé sur le serveur et qu'il n'y a pas suffisamment de places, des joueurs mourront dès le début de la partie ! C'est un peu embêtant non ?

Voilà, c'est tout ce qu'il fallait dire. Avec ça vous savez tout sur les entités de départ. Profitez-en, ce sont les entités les plus simples !
Voilà, c'est fini pour ce thème !

Lumière !

Votre map est maintenant jouable puisque vous avez appris à créer un départ de joueur.
Mais si vous l'avez testée, vous vous êtes rendus compte qu'il faisait tout noir ! Eh oui, si on ne met pas un minimum de lumière, vos niveaux sont plongés dans l'obscurité...

Notre objectif : allumer la lumière !

Une lampe

Entité concernée : light
Type d'entité : entité-point
Difficulté : facile

Bien. Parmi les différentes lumières dont on dispose, *light* est de loin la plus utilisée. Elle a ses avantages et ses défauts.

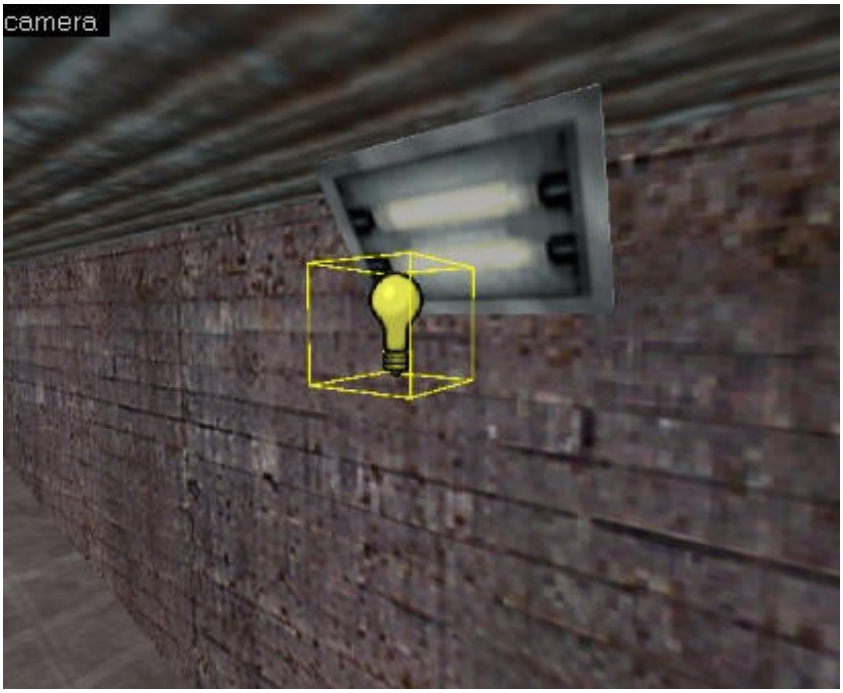
A quoi correspond l'entité *light* ?

Il s'agit d'une petite source de lumière. Seule, elle suffit à éclairer toute une pièce.
En général, on l'assimile à une lampe (que ce soit une lampe de bureau, une ampoule etc...).

Bien entendu, il faut créer un bloc à côté ayant une texture de lumière, pour qu'on aie l'impression que cette lumière provienne de quelque part (croyez-moi, ça fait bizarre une lumière qui surgit du néant).

Il n'y a pas que les lampes qui produisent de la lumière dans les maps. Il est conseillé de varier les sources lumineuses : on peut par exemple placer une entité *light* à côté de la fenêtre d'une maison. Personne ne trouvera choquant que la lumière émerge d'une fenêtre.

Voici comment on pourrait présenter un ensemble bloc/light :



On distingue 2 choses différentes : le bloc ayant la texture d'un néon, et l'entité *light* à côté qui crée la lumière.

Nous allons voir maintenant les attributs de cette entité :

- Target : quand l'entité est activée (lorsqu'on l'a appelée), on peut appeler une autre entité. Il suffit d'écrire son nom ici. Ce n'est pas très intéressant pour le moment, vous apprendrez à appeler une entité plus tard.
- Name : c'est le nom de votre lumière. Vous n'êtes pas obligés de lui en donner un. C'est utile si on veut pouvoir allumer et

éteindre la lumière lorsqu'on appuie sur un bouton, comme vous apprendrez à le faire.

- Brightness : c'est la couleur de votre lumière. Par défaut, la lumière est jaune (classique). Mais si vous cliquez sur "Pick Color", vous pourrez en choisir une autre, plus adéquate avec la texture de lumière que vous avez choisie. Par exemple, prenez une couleur blanche si le néon est blanc (c'est logique non ?).
- Appearance : vous permet de donner un effet spécial à votre lumière. Aidez-vous de la liste ci-dessous pour choisir l'effet qui vous plaît :
  - Normal : c'est l'apparence que l'on utilise la plupart du temps. La lumière reste allumée.
  - Fluorescent Flicker : utilisé pour lumière cassée, généralement un néon. Celle-ci clignote aléatoirement.
  - Slow, strong pulse : la lumière s'éteint et se rallume progressivement.
  - Slow pulse, noblack : c'est pareil, mais la lumière ne s'éteint pas aussi longtemps.
  - Gentle pulse : la lumière clignote mais reste allumée.
  - Flicker A : clignote assez rapidement.
  - Flicker B : la lumière s'éteint brusquement, et se rallume progressivement.
  - Candle A : cet effet est utile lorsque la lumière provient d'une bougie. Elle s'éteint et se rallume de temps en temps, comme si le vent agissait sur la bougie.
  - Candle B : idem, mais dans un autre style.
  - Candle C : idem, c'est un autre effet, histoire de varier...
  - Fast strob : effet stroboscope garanti. La lumière clignote très rapidement. J'espère que vous n'êtes pas épilleptiques !
  - Slow strob : pareil, mais ça clignote moins rapidement. Heureusement pour nos yeux ! Ce style permet de mettre une réelle ambiance dans votre map car la lumière ne s'allume que par intermitences... et s'il y a des ennemis dans la salle le joueur va stresser comme un fou, je vous le garantis !

N'abusez pas trop de cette technique, elle pourrait faire ramer les petits PC, ou, pire, faire planter la compilation : "Too many direct light styles on a face". La plupart des lumières sont de type "Normal", mais rien ne vous empêche une fantaisie de temps en temps 

Pour voir vous-même ces effets, téléchargez cette map d'exemple :

lights.zip (79 Ko)

C'est un tout petit niveau que vous pouvez démarrer sous Half-Life ou Counter-Strike. Il est constitué de longs couloirs (pour éviter d'avoir l'erreur "Too many direct light styles on a face") avec un effet de lampe différent à chaque fois.

- CustomAppearance : plus compliqué, cela vous permet d'utiliser un de vos propres effets. Franchement, je crois que vous n'en aurez jamais besoin...

Si vous avez d'autres propriétés (pour ceux qui utilisent le FGD avancé de Counter Strike), ne vous en préoccupez pas. Vous n'en tirerez pas grand chose...

En ce qui concerne les flags, il n'y en a qu'un seul :

- Initially dark : si c'est coché, la lumière est éteinte au départ. C'est utile si on se sert d'un bouton pour l'allumer (vous apprendrez à le faire plus tard, ne vous inquiétez pas)

Voilàà, vous pouvez maintenant illuminer vos niveaux avec les entités *light*, vous savez tout ce qu'il faut dessus.

Le soleil

Entité concernée : light_environment
Type d'entité : entité-point
Difficulté : facile

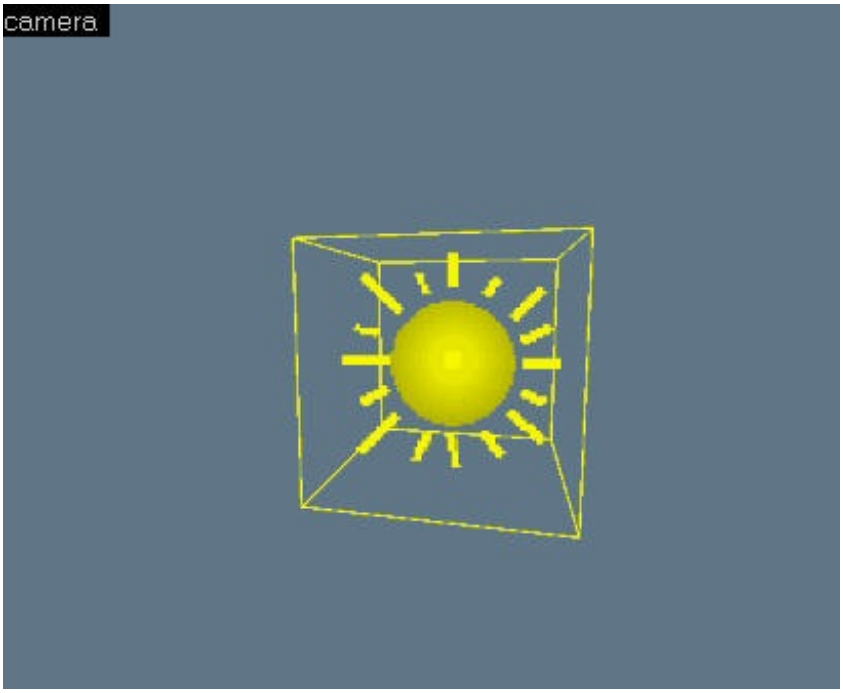
Cette lumière concerne uniquement les niveaux d'extérieur, c'est-à-dire les maps qui contiennent des skys (ciel).

Eh bien oui, on peut aussi se servir de la lumière du soleil dans Worldcraft. C'est très pratique pour éclairer une grande partie de votre niveau d'un seul coup. Cependant, la compilation risque de durer assez longtemps, cette lumière étant bien plus longue à calculer...

Voici le fonctionnement : vous placez une (et une seule) entité *light_environment* quelque part dans votre map. L'emplacement n'a strictement aucune importance. Toutes les faces recouvertes de la texture "sky" émettront de la lumière : ce sont elles qui font le soleil.

Rappelez vous ! Il ne doit y avoir qu'un seul *light_environment* dans votre map !

Si vous utilisez le FGD avancé de Counter-Strike, l'entité est représentée par un dessin de soleil. Sinon, cela reste une ampoule comme pour *light* (attention aux confusions !).
Voici l'entité telle qu'elle apparaît dans une map Counter-Strike :



Passons maintenant aux attributs de cette entité. Il y en a moins que pour light :

- Pitch Yaw Roll (Y Z X) : c'est l'orientation du soleil (d'où il provient dans votre map). Est, Ouest, Nord, Sud... Utilisez le radar pour plus de simplicité.
- Pitch : c'est la position du soleil (en degrés), en hauteur cette fois.

  Vers midi, le pitch est de -90° (le soleil vient d'en haut).

  En général, on met une valeur comprise entre 0° et -90° : ainsi le soleil provient du haut.

  Enfin, si vous entrez une valeur positive (exemple : 40°), le soleil provient du bas... Dans un monde extraterrestre peut-être ?
- Brightness : couleur de la lumière du soleil. Bon là si vous mettez une lumière verte, on va vraiment se croire chez E.T... On s'amuse rarement à changer cet attribut.

Toutefois, si votre niveau se déroule la nuit, essayez de mettre la couleur noire. Si si, ça marche ! Ca éclaire tout en donnant l'impression de jouer durant la nuit...

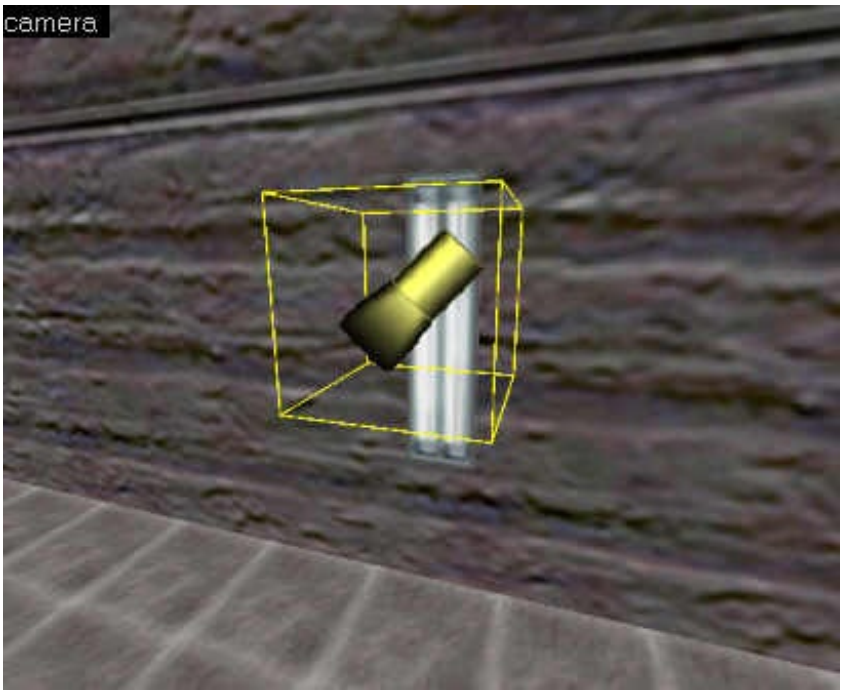
C'est tout, il n'y a pas de flags pour cette entité (vous ne voudriez tout de même pas éteindre le soleil en appuyant sur un bouton ???).

Les spots

Entité concernée : light_spot
Type d'entité : entité-point
Difficulté : assez difficile...

Ce type de lumière ressemble à *light*. Mais ici, la lumière est dirigée vers un point (et non dans toutes les directions comme avec *light*).

Sous Worldcraft, l'entité est représentée par une ampoule. Toutefois, si vous utilisez le FGD avancé de Counter que je vous ai fait télécharger, vous devriez voir ceci :



Si vous débutez vraiment, je vous déconseille d'utiliser de telles lumières. Leur intérêt est plutôt limité. Mais bon, si vous y tenez... Commençons par voir les attributs de cette entité :

- Name : nom de la lumière. Cela ne sert que si vous avez l'intention d'associer cette lumière à un bouton pour l'allumer et l'éteindre. Vous verrez comment faire cela dans un prochain chapitre...
- Target : lorsque l'entité est appelée, vous avez la possibilité d'en appeler une autre. Il suffit de mettre son nom ici. C'est un peu difficile pour le moment, vous devriez laisser ça de côté...
- Pitch Yaw Roll (Y Z X) : orientation de la lumière. Comme pour *light_environment*, utilisez le radar en haut à droite : c'est plus simple.
- Inner (bright) angle : c'est l'angle d'ouverture de la lumière. En effet, ce type de lumière est un cône dont vous pouvez définir la taille.
- Outer (fading) angle : c'est le second angle. Il correspond à l'angle de fermeture de la lumière.
- Pitch : bon je vous refais pas mon speech... C'est comme pour *light_environment* : cela permet d'orienter la lumière en hauteur.
- Brightness : c'est la couleur de la lumière. Cliquez sur "Pick Color" pour en choisir une. Généralement, la lumière est jaune.
- Is Sky : cette lumière peut remplacer la lumière du soleil (*light_environment*) : vous avez ainsi plus de possibilités de paramétrage. Mais est-ce vraiment utile ?
    - No: ce n'est pas la lumière du soleil. C'est la valeur que l'on met en général.
    - Yes : cela correspond à la lumière du soleil. Dans ce cas, pensez à ne pas mettre 2 entités de ce type dans votre map...

- Appearance : c'est l'effet de la lumière, comme nous avons vu pour *light*.
    - Normal : c'est l'apparence que l'on utilise la plupart du temps. La lumière reste allumée.
    - Fluorescent Flicker : utilisé pour lumière cassée, généralement un néon. Celle-ci clignote aléatoirement.
    - Slow, strong pulse : la lumière s'éteint et se rallume progressivement.
    - Slow pulse, noblack : c'est pareil, mais la lumière ne s'éteint pas aussi longtemps.
    - Gentle pulse : la lumière clignote mais reste allumée.
    - Flicker A : clignote assez rapidement.
    - Flicker B : la lumière s'éteint brusquement, et se rallume progressivement.
    - Candle A : cet effet est utile lorsque la lumière provient d'une bougie. Elle s'éteint et se rallume de temps en temps, comme si le vent agissait sur la bougie.
    - Candle B : idem, mais dans un autre style.
    - Candle C : idem, c'est un autre effet, histoire de varier...
    - Fast strob : effet stroboscope garanti. La lumière clignote très rapidement. J'espère que vous n'êtes pas épilleptiques !
    - Slow strob : pareil, mais ça clignote moins rapidement. Heureusement pour nos yeux !

- CustomAppearance : pour un effet de lumière perso. Très rarement utilisé... et puis quel intérêt ?

Cette entité possède un flag, comme pour *light* :

- Initially dark : la lumière est éteinte au départ. On doit pouvoir l'allumer en appuyant sur un bouton, sinon elle restera toujours éteinte et ça n'a aucun intérêt...

Les lueurs

Entité concernée : env_glow
Type d'entité : entité-point
Difficulté : facile

Utilisée en général en même temps que les lumières, les lueurs permettent de donner facilement un véritable effet dans votre map. Elles embellissent les lampes. On les utilise principalement à l'extérieur, parce que c'est là que le joueur s'attend à voir des lueurs.

Concrètement, si vous voulez faire apparaître une lueur autour d'un lampe, vous devez créer la lampe, mettre votre entité *light*, puis mettre un env_glow juste à côté.

Lorsque vous jouez, l'effet ressemble à ceci :

Mettez tout simplement un *env_glow* où vous le désirez dans votre map, et éditez ensuite ses propriétés si nécessaire :

- Name : nom de l'objet. Pratiquement inutilisé pour cette entité.
- Render FX, Render Mode, FX Amount et FX Color : permettent de modifier la transparence de l'objet. Pour plus d'informations, lisez le chapitre sur les murs (paragraphe sur la transparence).
  Sachez simplement que pour cette entité, il vous faudra mettre Render Mode = Glow si vous voulez modifier la transparence.
- Sprite Name : nom du sprite à utiliser pour la lueur. Par défaut, c'est glow01.spr.
  Lisez le chapitre sur les sprites pour en savoir plus.
- Scale : c'est la taille de la lueur. 1 correspond à la taille normale. Si vous augmentez cette valeur, la lueur sera plus grande mais aussi beaucoup plus pixellisée...

En temps normal, vous ne toucherez même pas aux propriétés de cette entité tellement elle est simple à utiliser.

Donc, si vous voulez donner une meilleure apparence à vos lampes, pensez aux *env_glow* !
Du bon boulot ! On avance à grands pas dans notre étude des entités.
Bonne nouvelle : le plus intéressant est à venir, mais aussi le plus difficile hélas...

Eh oui, ça commence à devenir une habitude : tout ce qui est bien est difficile à faire. Allez, courage ! Vous deviendrez bientôt des mappeurs professionnels à faire pâlir les gars de chez Valve

Les déclencheurs (A : simples)

En voilà un thème qu'il est important !

A la fin de la lecture de ce thème, vous manierez à la perfection les déclenchements dans Half-Life et Counter-Strike. Vous saurez comment gérer les évènements, allumer une lumière grâce à un bouton etc..

Ce que vous allez apprendre va vous sembler un peu abstrait. Il va falloir que vous assimiliez pas mal de trucs sans savoir à quoi ça sert. Jouez le jeu : vous verrez après que vous n'aurez pas appris tout ça pour rien, bien au contraire

Appels d'entités

Nous allons tout d'abord étudier quelques généralités. En effet, on peut retrouver certains attributs dans la plupart des entités. La liste d'attributs que je vais vous donner est donc valable pour beaucoup d'entités que vous étudierez dans les autres thèmes.

Ce que vous allez apprendre est très important pour la suite, alors redoublez de concentration

Name

L'attribut "Name" est très courant. C'est le nom de l'entité. Par défaut, l'entité n'a pas de nom, ce qui ne pose pas de problème étant donné qu'on n'est pas obligé de lui en donner un.

Pourquoi lui donner un nom alors ?

La réponse est simple : pour l'appeler. Beaucoup d'entités ont la possibilité d'être appelées à distance pour déclencher une action à distance. Cette action varie en fonction de l'entité appelée.
Je prends un exemple que nous venons de voir : la lumière. Si on appelle une entité *light*, celle-ci s'éteint si elle était allumée, et vice-versa : elle s'allume si elle était éteinte.

Les possibilités sont très nombreuses : on peut casser un objet, déclencher une explosion etc... Vous saurez faire cela lorsque vous connaîtrez les entités appropriées, bien sûr

Dernier détail : il n'est pas interdit de donner le même nom à plusieurs entités. Ces entités seront appelées en même temps, ce qui peut s'avérer pratique.

Global Entity Name

C'est la même chose que pour "Name", sauf que certaines entités ont la possibilité de fonctionner sur plusieurs niveaux (dans le cas d'une aventure solo qui se déroule sur plusieurs maps).

Il faut lui donner un nom global, qu'on pourra réutiliser dans une autre map. Par exemple, si une porte est bloquée dans la map A, et que pour l'ouvrir il faut appuyer sur un bouton dans la map B, on donnera un nom global à la porte. On pourra ainsi l'ouvrir à partir de la map B.

Target

Lorsqu'une entité est appelée, elle a la possibilité d'en appeler une autre. On peut ainsi créer une chaîne d'appels d'entités...

Bon, en clair cet attribut sert à appeler une entité. Vous devez simplement rentrer son nom (attribut "Name" de l'entité à appeler).

"Target" signifie "Cible" : c'est l'attribut le plus important de cette section puisque c'est lui qui commande l'appel d'entité.

Delay before trigger

C'est le temps en secondes que va patienter Half-Life avant d'appeler l'entité (avec "Target"). Par exemple, si vous voulez attendre 4 secondes avant d'éteindre la lumière, rentrez la valeur "4".

Le séparateur décimal est le point (.) : il remplace la virgule. Pour patienter 3 secondes et demi, il faut donc rentrer "3.5".

Kill target

Un peu spécial : cet attribut permet de détruire une entité. Il fonctionne comme "Target", mais il permet de retirer l'entité du jeu et non pas de l'appeler.

La disparition de l'entité est un peu brutale, alors évitez que l'entité disparaisse sous les yeux du joueur ou alors il risque de se croire victime de phénomènes paranormaux

Master

Là ça devient compliqué : vous devez indiquer dans le champ le nom d'un "multisource". Cela permet de verrouiller une entité tant que plusieurs autres entités n'ont pas été activées. On aura l'occasion d'en reparler

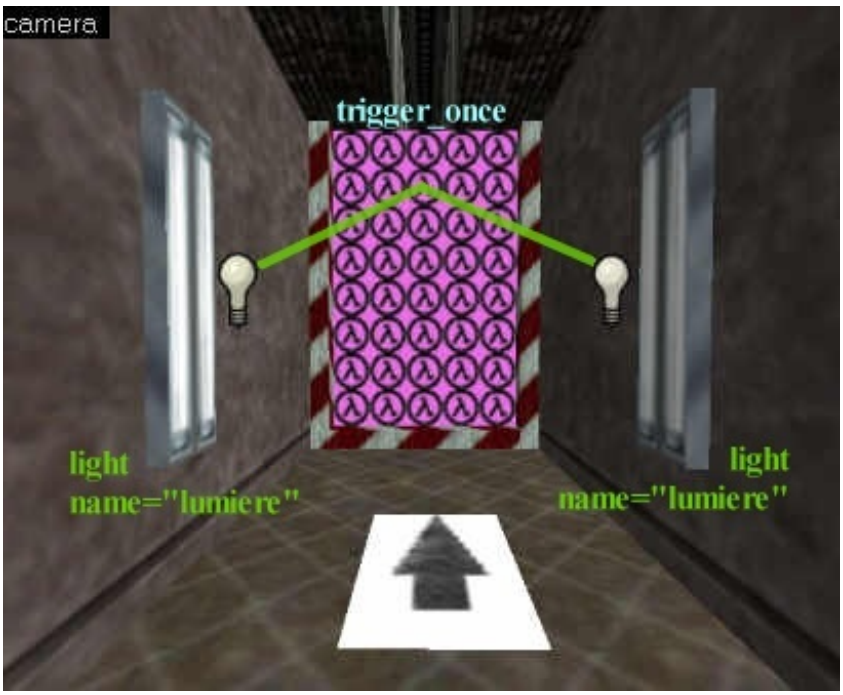
Au premier passage dans une zone

Entité concernée : trigger_once
Type d'entité : entité-bloc
Difficulté : assez facile

Pour appeler une entité, on doit se servir d'un évènement. Le passage du joueur dans une zone en est un. Par exemple, si le joueur rentre dans une salle la lumière s'allume automatiquement !
Pour utiliser cette entité, transformez un bloc en entité-bloc (avec la texture AAATRIGGER car il n'est pas visible quand on joue).

Le *trigger_once* permet d'appeler une entité la première fois que le joueur rentre dans une zone. La seconde fois, il ne se passera plus rien.

Par exemple, le joueur marche dans un couloir et traverse une sorte de porte. J'ai placé un *trigger_once* (invisible) au niveau de la porte et deux entités *light* éteintes (flag "Initially dark"). L'attribut "Target" du *trigger_once* pointe vers l'entité nommée "lumiere". Il y a deux entités *light* nommées "lumiere" : elles seront donc allumées en même temps.



Si le joueur repasse dans la zone du *trigger_once*, les lumières ne s'éteindront pas puisque le *trigger_once* ne fonctionne qu'une seule fois.

Pour ce qui est des attributs, vous les connaissez tous pour la plupart. Vous les avez vu plus haut. Il en reste toutefois 2 qui peuvent être intéressants :

- Sound style : vous pouvez jouer un son lorsque le joueur rentre dans cette zone. On met généralement un petit bip que l'on utilise conjointement avec l'affichage d'un message (voir ci-dessous).Pour plus d'infos sur les sons dans Half-Life, reportez-vous au thème sur les sons.
- Message (set sound too!) : on peut rentrer un petit texte à afficher sur l'écran du joueur (c'est plutôt rare). "set sound too!" signifie qu'un son est joué lorsque le message s'affiche.

Quant aux flags (eh oui il ne faut pas les oublier), les voici :

- Monsters : coché, le trigger sera aussi activé si un autre personnage du jeu rentre dedans (scientifiques, ennemis...).
- No clients : l'entité n'affectera aucun joueur.
- Pushables : le trigger est activé par les objets que l'on peut pousser, comme les caisses (func_pushable).

A chaque passage dans une zone

Entité concernée : trigger_multiple
Type d'entité : entité-bloc
Difficulté : assez facile

Si vous savez utiliser les *trigger_once*, alors vous n'aurez aucun problème avec les *trigger_multiple*.
C'est exactement la même chose sauf que l'action est répétée à chaque passage dans la zone.

Je ne vais pas par conséquent tout réexpliquer, tout a été dit plus haut.
Reste cependant un nouvel attribut :

- Delay before reset : c'est le temps en secondes avant que l'on puisse réutiliser le *trigger_multiple*.Par exemple, si on met "4", et qu'on traverse le trigger une première fois, il ne se passera rien pendant 4 secondes si on retraverse la zone.

Après plusieurs passages dans une zone

Entité concernée : trigger_counter
Type d'entité : entité-bloc
Difficulté : assez facile

C'est encore très simple. Ca fonctionne comme les autres trigger, mais il faut passer plusieurs fois dans la zone pour que l'entité soit appelée.

- Count before activation : nombre de passages nécessaires pour que l'entité décrite dans "Target" soit appelée.

Au chargement de la map

Entité concernée : trigger_auto
Type d'entité : entité-point
Difficulté : moyen

Cette entité se déclenche au chargement de la map. Elle permet d'appeler des entités dès que le jeu commence. Cela peut s'avérer parfois très pratique !

Cette entité ne fonctionne que si vous mettez son attribut "Trigger State = On".

Voici tous les attributs de cette entité :

- Target : nom de l'entité à appeler.
- Delay before trigger : temps en secondes avant l'appel de l'entité.
- KillTarget : nom d'une entité à retirer du jeu.
- Global State to Read : nom de la variable globale à lire à partir d'une entité *env_global*. Une variable globale est une information stockée en mémoire que l'on peut lire sur plusieurs maps différentes (si vous faites une aventure solo sur plusieurs maps par exemple).
- Trigger State : état de la variable globale.
  - Off : désactivée (par défaut). Dans ce cas, l'entité *trigger_auto* ne se déclenchera pas au chargement de la map.
  - On : activée. **Vous devez mettre cette valeur pour que l'entité fonctionne**.

- Toggle : dépend de l'état de la variable globale lue dans "Global State to Read". Si la valeur est à "Off", alors elle devient "On", et vice-versa.

A noter aussi un Flag qui pourrait vous être parfois utile :

- Remove On Fire : une fois que l'entité a été déclenchée, le *trigger_auto* est retiré du jeu.

Les boutons simples

Entité concernée : button_target
Type d'entité : entité-bloc
Difficulté : plutôt facile

Les boutons sont le moyen le plus simple (et le plus courant) qu'on utilise pour appeler une entité. Le joueur doit appuyer sur la touche "Utiliser" pour appuyer sur le bouton.

Il existe plusieurs types de boutons. Nous étudierons ici le cas le plus simple : le *button_target*. Voici les étapes à suivre dans l'ordre pour créer un bouton :

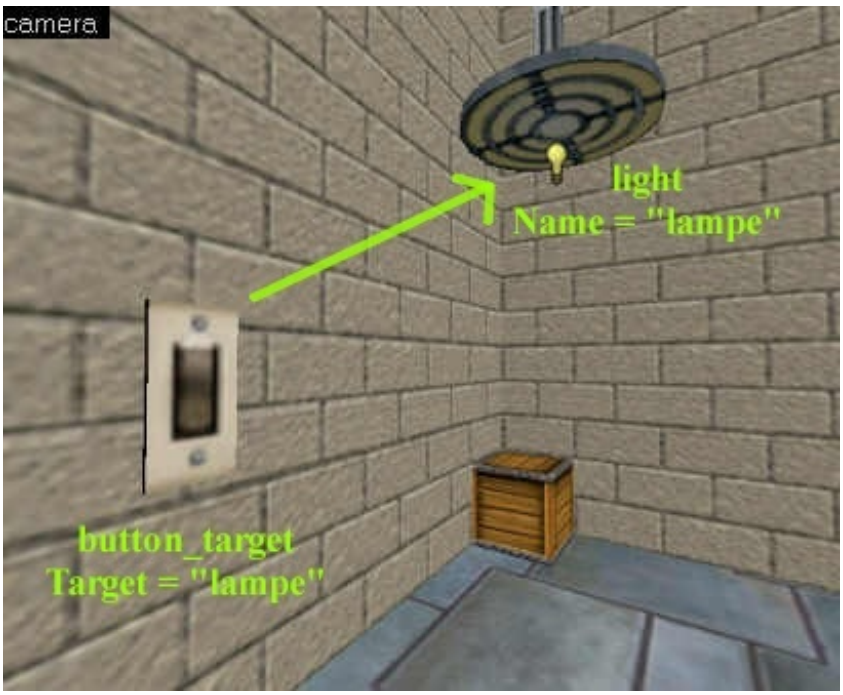
1. Choisissez pour commencer une texture de bouton. Je vous rappelle que celles-ci commencent en général par +A ou +0 (elles changent en fonction de l'état du bouton).
2. Créez un bloc ayant la même taille que la texture. S'il n'est pas correctement aligné, désactivez le "Texture Lock" , déplacez le bloc jusqu'à l'alignement, puis réactivez le "Texture Lock".
3. Transformez le bloc en entité-bloc de type *button_target*.
4. Modifiez enfin ses attributs pour appeler l'entité que vous voulez. Seul "Target" devrait vous intéresser.

Cette entité possède d'autres attributs que vous ne connaissez pas (les FX). Je ne vous en parle pas pour le moment car ils n'ont pas d'intérêt avec les boutons. Ils servent à rendre l'entité transparente, ou partiellement transparente.

Passons aux flags de cette entité...

- Use Activates : généralement, ce flag est coché. Il permet d'obliger le joueur à appuyer sur la touche "Utiliser" pour activer le bouton.
  Si le flag est décoché, on pourra alors activer le bouton en... tirant dessus 
- Start On : le bouton est à l'état "Utilisé" ("On") lorsqu'on démarre la map. C'est plutôt rare, à vous de voir si ça convient dans votre cas ou pas.

Bon, pour bien terminer voici un petit exemple avec un interrupteur de lumière :



Quand même, ne me dites pas que c'est dur le mapping !

Les boutons translatifs

Entité concernée : func_button
Type d'entité : entité-bloc
Difficulté : moyen

Ces boutons sont plus utilisés car ils permettent l'accès à un plus grand nombre de paramétrages que les *button_target*.
On peut par exemple faire déplacer le bouton ou jouer un son lorsqu'on l'active.

- Pour un bouton immobile : cochez le flag "Don't move".
- Pour un bouton translatif : réglez d'abord l'orientation du déplacement avec le radar en haut à droite. Sachez que la valeur 0° correspond à la direction vers la droite dans la vue "top (x/y)".
  Réglez ensuite ses attributs, comme "Lip", "delay before reset (-1 stay)" etc...

Là, histoire de bien tout récapituler, je vais vous dresser la liste de tous ses attributs :

- Global Entity Name : c'est le nom global de cette entité. Comme nous l'avons vu plus haut, il est utilisé seulement si vous faites une aventure solo qui se déroule sur plusieurs maps différentes. On peut ainsi activer un objet d'une map A à partir d'une autre map B.
- Name : nom de l'entité pour qu'on puisse l'appeler avec "Target". Pour un bouton, ça n'a pas vraiment d'intérêt...
- Target : l'attribut le plus important. Vous devez indiquer ici le nom de l'entité à appeler.
- Render FX : permet de définir la manière dont l'objet renvoit la lumière. On modifie rarement cet attribut.
- Render Mode : permet de gérer les effets de transparence. C'est plutôt rare pour un bouton, donc je ne vous en parle pas de suite
- FX Amount : à associer avec "Render Mode". Il définit en fait le niveau de transparence de l'objet.
- FX Color (R G B) : c'est la couleur de l'effet. Là encore, on en reparlera quand on traitera plus en détail de la transparence.
- Speed : c'est la vitesse à laquelle le bouton se déplace lorsqu'il est activé.
- Health (shootable if > 0) : par défaut, la valeur est 0 (le bouton ne peut pas être détruit).
  Mais si vous voulez pouvoir vous acharner comme un gros bourrin dessus, alors modifiez cette valeur. Pour vous donner une idée, sachez que 100 correspond à la vie d'un joueur.
  Lorsque le bouton a perdu toute sa vie, l'entité est appelée.
- Lip : quand le bouton se déplace, la longueur du mouvement correspond à la taille de ce bouton. Si vous voulez qu'il se déplace encore plus, mettez un nombre supérieur à 1.
  Par exemple, si Lip = "5", alors le bouton se déplacera de 5 fois sa longueur.
- Master : ça peut vous être utile pour votre bouton... Les entités *multisource* permettent de verrouiller l'appel d'une entité tant que plusieurs boutons n'ont pas été activés. Vous devez entrer ici le nom du *multisource* qui gère cela.
- Sounds : son qui sera joué lorsqu'on appuiera sur le bouton.
- delay before reset (-1 stay) : si le bouton se déplace, il peut revenir à sa position initiale au bout d'un certain laps de

temps.
Pour "3", il reviendra au bout de 3 secondes.
Pour "-1", il restera bloqué et ne reviendra jamais...

- Delay before trigger : temps en secondes avant l'appel d'entité.
- Locked Sound, Unlocked Sound, Locked Sentence, Unlocked Sentence : je les mets tous dans le même paquet parce qu'ils ne servent que dans un cas particulier : une map multi pour Half-Life en mode "HL TeamPlay". Vous ne devriez pas avoir à vous en servir d'ici tôt
- Minimum light level : niveau minimum de luminosité. Si j'étais vous, je ne modifierais pas cet attribut car vous pourriez facilement faire planter le compilateur hlrad.exe

Reste à voir les flags (importants pour cette entité) :

- Don't move : coché, le bouton ne se déplacera pas lorsqu'il sera enclenché. Très utilisé car beaucoup de boutons sont immobiles.
- Toggle : le principe du toggle est le suivant : si le bouton est à l'état actif, il devient inactif. Et réciproquement bien entendu On coche souvent ce flag.
- Sparks : le bouton émettra des étincelles. C'est un joli effet, il faut bien l'avouer...
- Touch Activates : si on touche le bouton, alors il s'active tout seul. Le joueur n'a pas besoin d'appuyer sur "Utiliser" pour l'enclencher.

Les boutons n'ont plus de secrets pour vous maintenant

Les boutons rotatifs

Entité concernée : func_rot_button
Type d'entité : entité-bloc
Difficulté : assez difficile

Aïe aïe aïe ! Comme tout ce qui est rotatif, c'est assez dur à paramétrer quand on n'a pas l'habitude... Au début, on est assez dérouté par ce principe de rotation, alors qu'en fait ce n'est pas si dur que ça.

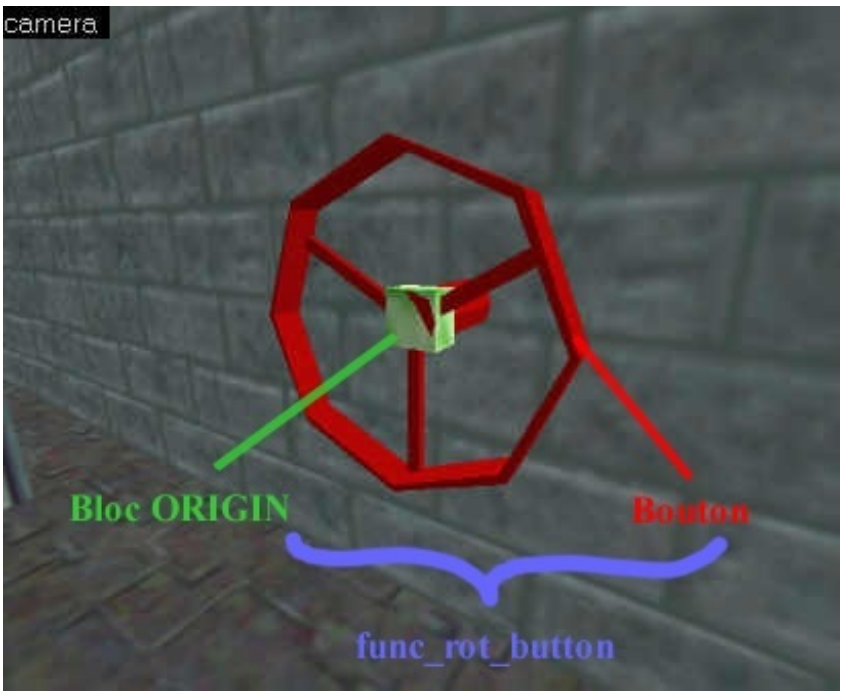
Si vous débutez en mapping, vous devriez laisser tomber les boutons rotatifs pour le moment. Vous vous ferez la main lorsque vous apprendrez à créer des portes rotatives.

Bon, pour ceux qui sont déterminés, suivez-moi

A quoi sert le bloc recouvert de la texture ORIGIN ?

Il définit l'axe de rotation de l'objet. Sans lui, Worldcraft ne sait pas comment faire tourner votre bouton.
Comment savoir quel est l'axe de rotation de votre objet ? C'est bien là l'éternelle question que tout le monde s'est posée. Où dois-je placer ce $%$##%$#%# de bloc ORIGIN ?

En fait, c'est très simple (une fois que l'on a compris, comme d'habitude ;). L'axe de rotation est la partie autour de laquelle votre objet tourne. Cette partie reste pratiquement immobile.
Un exemple avec une vanne :
Un bouton rotatif est une entité constituée de 2 blocs : le bouton en lui-même et un bloc recouvert de la texture ORIGIN :



Comme vous pouvez le voir, le bloc ORIGIN (en vert) est situé sur l'axe de rotation, la seule partie qui ne se déplace pas. Le reste (en rouge) est une association de blocs qui tourneront autour du bloc ORIGIN. Bien entendu, le bloc ORIGIN est invisible lorsqu'on joue.

L'association de ces deux éléments doit être convertie en *func_rot_button*. Sélectionnez le bloc ORIGIN et le reste de la vanne puis cliquez sur "To Entity". Choisissez enfin l'entité *func_rot_button*.

En ce qui concerne les attributs, on retrouve pas mals de choses qu'on connaît. Voici les nouveaux attributs qui peuvent vous intéresser :

- Targetted object : c'est en fait l'équivalent de "Target".
- ChangeTarget Name : permet de changer de cible après une première activation.
- Distance (deg) : indiquez la distance parcourue par le bloc en degrés. Par exemple, si vous mettez "360", le bloc effectuera un tour complet sur lui-même.

Les flags, et puis après on arrête c'est promis

- Not solid : le bloc n'est pas solide, on peut passer à travers.
- Reverse Dir : le mouvement se fait en sens inverse (sens contraire des aiguilles d'une montre).
- Toggle : on l'a vu pour *func_button*.
- X Axis : le bloc tournera suivant l'axe des X (si rien n'est coché, c'est l'axe Z qui est utilisé).
- Y Axis : pareil mais pour l'axe des Y.
- Touch Activates : le bouton est activé dès qu'on le touche.

Nous avons donc vu pratiquement tous les moyens que l'on avait pour appeler une entité... mais pas tous !

Mais pour l'instant, vous ne pouvez appliquer ce que vous avez appris qu'avec les entités "light". Lorsque vous découvrirez de nouvelles entités, vous verrez qu'il est possible de les appeler elles aussi : par exemple, si on appelle un funk_breakable, l'objet se casse à distance !

Rassurez-vous, je vous signalerai à chaque fois ce qui se passe lorsqu'on appelle l'entité.

Allez, on ne s'arrête pas en si bon chemin... On a encore du boulot

Les déclencheurs (B : complexes)

Nous avons vu jusqu'ici la partie "simple" des déclencheurs. Il en reste encore quelques-uns plus complexes mais très intéressants. Grâce à eux, vous saurez faire tout ce que vous voudrez pour appeler une entité.

Appels d'entités dans un ordre prédéfini

Entité concernée : multi_manager
Type d'entité : entité-point
Difficulté : moyen

Imaginons que le joueur entre dans un couloir tout noir. Il y a 3 lampes mais elles sont éteintes. Il appuie sur l'interrupteur, et là, les lampes s'allument l'une après l'autre à une seconde d'intervalle.

Ce genre de chose, vous n'auriez pas pu le faire si vous aviez donné le même nom aux 3 entités light. En effet, elles se seraient allumées en même temps, et ici on veut qu'elles s'allument l'une après l'autre.

C'est l'entité *multi_manager* va nous permettre de réaliser cet effet. Elle permet d'appeler des entités dans un ordre précis en laissant un intervalle de X secondes entre chaque appel. C'est donc vous qui définissez quand telle entité doit être appelée.

Son utilisation est assez particulière. Commencez par intégrer une entité-point de type *multi_manager* et ouvrez ses propriétés. Comme vous pouvez le voir, cette entité ne possède qu'un attribut : "Name", que vous connaissez bien.

Donnez donc un nom au *multi_manager*. Dans notre cas, on va lui donner le nom "allumerlampes".
L'interrupteur est un bouton qui pointe vers le *multi_manager*. Son attribut "Target" a donc la valeur "allumerlampes".

Et maintenant, comment je fais pour dire quelles entités le multi_manager doit appeler ?

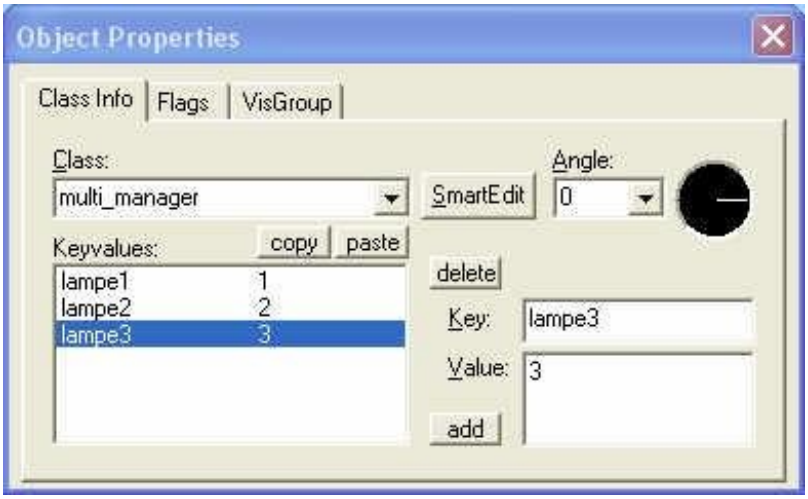
C'est là que c'est un peu particulier. Ouvrez les propriétés du *multi_manager*, et désactivez le bouton "Smart Edit". Ce bouton permet normalement d'avoir accès à l'interface "simple" pour les propriétés d'entités. Ce que vous voyez maintenant est l'interface complexe

Cliquez sur le petit bouton "add" en bas pour rajouter une nouvelle clé (attribut). Une petite fenêtre s'affiche et vous demande le nom de la clé et sa valeur. Remplissez comme ceci :

- Key : entrez ici le nom de l'entité à appeler.
- Value : entrez ici au bout de combien de temps cette entité doit être appelée. Entrez une valeur en secondes et n'oubliez pas d'utiliser le point comme séparateur décimal (ex : "1.5").

Répétez ceci autant de fois que vous voulez appeler d'entités.

Moi, mes lampes s'appelaient "lampe1", "lampe2" et "lampe3" et je voulais les appeler à une seconde d'intervalle à chaque fois. Voici le résultat que je vois sous Worldcraft :



Cette entité possède un seul Flag : "multithreated". S'il est coché, alors le *multi_manager* peut être appelé plusieurs fois en même temps par des joueurs différents (en multijoueurs seulement). Il peut alors par exemple s'exécuter 3 fois en même temps si 3 joueurs l'ont activé alors qu'il était déjà activé
Bon, c'est pas évident à expliquer mais en fait c'est très simple. Sachez que ce flag est quand même assez rarement coché mais il

peut servir !

Verrouiller une entité

Entité concernée : multisource
Type d'entité : entité-point
Difficulté : assez difficile

Voilà le scénario : une grosse porte blindée est protégée et ne peut pas être ouverte avant que le joueur ait appuyé sur 2 boutons. Ce genre de choses arrive souvent dans l'aventure de Half-Life : Gordon doit par exemple réactiver l'oxygène, le feu, les moteurs... avant de pouvoir lancer une explosion qui déchire tout !

Bon, dans notre cas il s'agit donc d'une porte bloquée tant que 2 boutons n'ont pas été activés. Voici ce que vous devez faire dans l'ordre :

1. Intégrez un *multisource* dans votre map (il sera invisible quand vous jouerez).
2. Donnez-lui un nom dans son attribut "Name", par exemple ici : "bloqueur". Seul cet attribut nous servira ici. Les 2 autres, "Target" et "Global State Master", servent respectivement lorsque vous utilisez un *game_team_master*, et lorsque vous faites une partie solo qui se déroule sur plusieurs maps.
3. Occupez-vous maintenant des boutons. Leur attribut "Target" doit pointer sur le *multisource*. On va donc ici mettre Target = "bloqueur", et ce pour les 2 boutons.

   
   Attention ! Les boutons se désactivent par défaut au bout de quelques secondes, et ils ne sont alors plus considérés comme activés ! Je vous recommande donc fortement de mettre la valeur "-1" à leur attribut "Delay before reset".
4. Dans le champ "Master" de la porte, mettez le nom du *multisource*. Ici : Master = "bloqueur".

Et voilà ! Que se passera-t-il ? La porte refusera de s'ouvrir tant que les 2 boutons n'auront pas été activés !


Notez que lorsque les 2 boutons auront été activés, la porte ne s'ouvrira pas toute seule. Mais elle ne sera plus bloquée : le joueur pourra alors l'ouvrir lorsqu'il s'en approchera...

Relais d'appels

Entité concernée : trigger_relay
Type d'entité : entité-point
Difficulté : assez facile

Le *trigger_relay* est une entité un peu oubliée, parce que rarement utile. Toutefois, il m'a semblé important de vous en parler rapidement pour que vous sachiez au moins que ça existe.

Que fait-elle ? Elle sert de relai, comme son nom l'indique, entre un déclencheur et sa cible.

Par exemple, un bouton permet d'allumer une lumière... plutôt que de faire pointer le bouton vers l'entité *light*, vous le faites pointer vers le *trigger_relay* qui appelera à son tour l'entité *light*.


??? Mais c'est complètement débile ? A quoi ça sert ?

Lol, oui c'est vrai on a l'impression que ça revient au même. Mais grâce à cette entité et à sa propriété "KillTarget", vous pourrez supprimer une entité du jeu qui ne sert plus à rien avant d'appeler la lumière.
Bon, c'est un peu limité je le reconnais, mais il fallait que vous sachiez que ça existe.

A noter que l'attribut "Trigger State" ne sert que lorsque vous utilisez une variable globale (dans une partie solo qui se déroule sur plusieurs maps). Il définit l'état de cette variable.
Quant au flag "Remove on fire", s'il est coché alors l'entité *trigger_relay* sera retirée du jeu après avoir été utilisée.

Changer la cible d'une entité

Entité concernée : trigger_changetarget
Type d'entité : entité-point
Difficulté : assez facile

Voilà une autre entité spéciale, mais celle-ci est bien plus intéressante. Comme son nom l'indique, elle permet de changer la cible (Target) d'une entité, et ce en cours de partie.

Pour s'en servir, il faut placer d'abord cette entité quelque part dans la map (logique). Editez ensuite les attributs suivants :

- Name : nom de l'entité *trigger_changetarget*. Lorsqu'elle sera appelée, elle changera la cible d'une autre entité.
- Target : nom de l'entité donc vous devez modifier la cible.
- New Target : indiquez ici la nouvelle cible à donner pour cette entité.

Cette entité vous sera certainement utile à un moment, donc ne l'oubliez pas !
Voiiilàà... Je crois qu'avec tout ça on aura fait le tour des appels d'entités dans Worldcraft lol 

Il faut dire que c'est quelque chose de très important et d'incontournable en mapping, et avec ce que je vous ai appris vous pouvez vous vanter de tout savoir dessus.

Les murs

Là, vous vous dites certainement que vous n'avez rien compris au mapping . Après tout, c'est vrai, pourquoi existerait-il des entités de murs alors que les blocs sont censés le faire ?

Rassurez-vous, il y a toujours une explication à tout. Par exemple, comment rendre vos murs transparents, ou partiellement transparents ? Il faut obligatoirement les transformer en entités.

La transparence

La transparence est exploitable sur les entités-bloc visibles, notamment les murs. Elle vous permet de voir à travers vos blocs, que ce soit sur une partie du bloc ou l'ensemble du bloc...

Pour pouvoir rendre votre entité-bloc transparente, on se sert des attributs d'effets (FX). Ils sont au nombre de 4. On les retrouve sur toutes les entités qui peuvent être rendues transparentes :

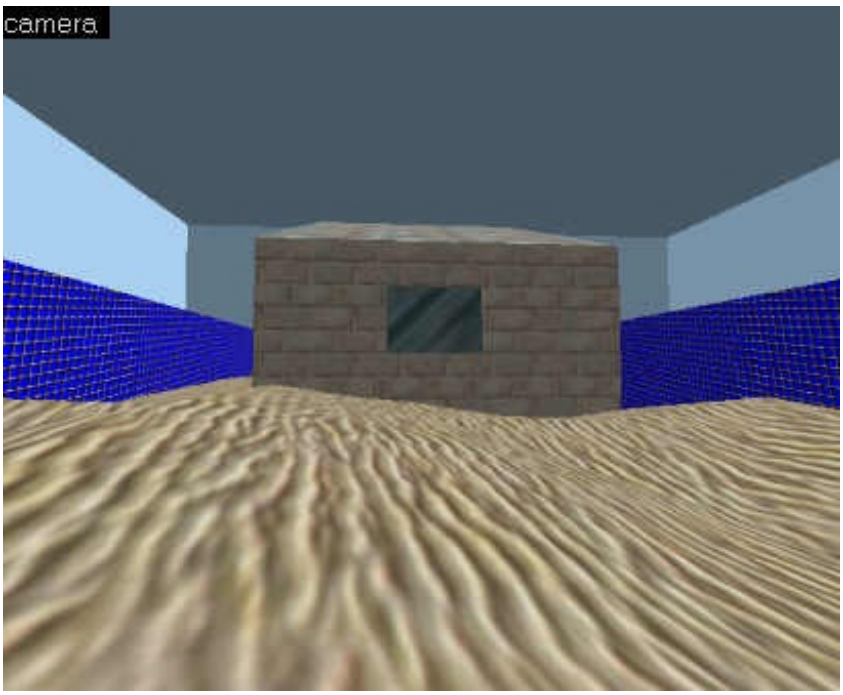
- Render FX : permet de changer la manière dont la lumière est réfléchie sur l'objet. Cela ne fonctionne que si l'entité est transparente. On peut donc faire "clignoter" le bloc ou bien le faire disparaître progressivement etc...
- Render Mode : c'est le plus important. C'est lui qui détermine le type de transparence.
  - Normal : aucune transparence (par défaut).
  - Color : transparence basée sur la couleur. Vous devrez définir cette couleur avec l'attribut "FX Color (R G B)".
  - Texture : transparence basée sur la texture du bloc. C'est une des transparences qui donne le meilleur effet à mon goût
  - Glow : cette transparence n'est utlisée que pour les entités *env_glow*. Elle sert donc assez rarement. Vous ne devez pas l'utiliser pour une autre entité que celle-là.
  - Solid : permet d'obtenir une transparence partielle de l'objet. Je m'explique : les textures commençant par "{" contiennent du bleu (c'est le cas des échelles et des grilles notamment). Ce bleu peut être rendu transparent avec cette valeur, à condition de mettre une valeur supérieure à 0 à "FX Amount (1 - 255)".
  - Additive : enfin, cette transparence est basée sur la luminosité de l'objet. Cela fonctionne comme pour "Texture", mais généralement l'objet est plus lumineux (et la transparence est moins jolie aussi).
- FX Amount (1 -255) : puissance de l'effet de transparence. Doit obligatoirement être supérieur à 0 pour que l'objet apparaisse.
  1 = transparence maximale.
  128 = transparence moyenne.
  255 = transparence minimale (opaque).
- FX Color (R G B) : couleur de l'effet. On l'utilise pour une transparence de type "Color" ou "Glow". Pour les autres, ça ne sert à rien.

C'est l'occasion idéale pour faire un test... On va utiliser les 2 types de transparence les plus courants : Texture (ou Additive) et Solid.
Le premier sert à rendre tout l'objet transparent. La plupart du temps, cette transparence sert pour les vitres.
Le second permet de faire disparaître une partie du bloc pour le rendre partiellement transparent. C'est plus fréquent que "Texture". On voit des transparences de ce types pour les échelles, les arbres, les grilles etc...

Je vais donc vous montrer comment rendre une vitre et un grillage transparents. Pour cela, j'ai créé une sorte de désert, avec un petite maison entourée d'un grillage. La maison comporte une vitre, et j'ai créé un fauteuil à l'intérieur.
Comme vous pouvez le voir, ça n'a rien de très compliqué

- La vitre est un bloc recouvert d'une texture de vitre (GLASS). Je l'ai transformée en entité-bloc de type func_wall (voir plus bas). Je lui ai mis les attributs :
  Render Mode = Texture
  FX Amount = 100
- Le grillage est un autre bloc recouvert d'une texture faite pour être partiellement transparente (commençant par "{"). Le bleu qui apparaît sous Worldcraft peut être rendu transparent si on transforme le bloc en entité-bloc de type *func_wall*, et qu'on lui applique les attributs suivants :
  Render Mode = Solid
  FX Amount = 255

Voici ce que l'on voit sous Worldcraft :



Lorsqu'on joue sur la map, les effets de transparence apparaissent :

Bel effet non ? On peut ainsi voir ce qu'il y a derrière la vitre, et le grillage est un vrai grillage !

Notez que la vitre est bien transparente, mais elle ne peut pas être cassée. Si on veut pouvoir l'exploser en tirant dessus comme un bourrin, il faut se servir de l'entité func_breakable que l'on étudiera plus tard.

Voilà, avec ça vous savez tout sur la transparence et vous êtes parés pour l'appliquer à tous types de blocs : échelles, grilles, vitres etc...

Les murs simples

Entité concernée : func_wall
Type d'entité : entité-bloc
Difficulté : facile

Une entité de mur ??? Et les blocs, ça sert à quoi alors ?

Bon, c'est l'heure des explications... Si je viens de vous montrer commment fonctionne la transparence, ce n'est pas sans raison. Ce sont des attributs qui ne se retrouve que dans les entités-bloc : il faut donc absolument transformer son bloc en entité pour pouvoir le rendre transparent.
L'utilisation du *func_wall* présente donc 2 avantages :

- La possibilité d'appliquer des effets spéciaux (transparence entre autres).
- Eviter de multiplier le nombre de polygones. A la compilation, si votre bloc touche un autre bloc, celui-ci sera scindé en plusieurs blocs, un peu comme si vous faisiez un "Carve". Sauf que vous n'avez rien demandé !
  Lorsque vous jouerez, il y aura encore plus de polygones à l'écran et les petits PC vont mal le supporter... à moins que vous ne transformiez le bloc en entité.

Cette entité a un flag : "Not in deathmatch", qui supprime le bloc dans une partie multijoueurs.

A part ça il n'y à rien à dire. Vous savez comment rendre le bloc transparent, vous n'avez qu'à le transformer en *func_wall*, à modifier ses attributs et le tour est joué !

Les murs toggle

Entité concernée : func_wall_toggle
Type d'entité : entité-bloc
Difficulté : facile

Bigre ! Il y a donc plusieurs types de murs ? C'est vrai, pourquoi faire simple quand on peut faire compliqué ? Allez expliquer ça aux programmeurs de Half-Life, j'y suis pour rien ;o)

Bon, il y a quand même une différence notable entre les *func_wall* et les *func_wall_toggle*.
Cette entité a la possibilité d'être appelée. Eh oui, c'est la seule différence. Mettez-lui un nom et appelez-la. Que se passe-t-il si on appelle cette entité ? Elle disparaît ! Carrément sous les yeux du joueur (mais en général on évite qu'il voit ça directement, ça ferait bizarre...).

Que signifie toggle ?

Ah ! Une définition ! Une objet **Toggle** se modifie de manière différente à chaque fois qu'il est appelé.
Pour *func_wall_toggle*, si le mur est visible il disparaît ; et si le mur est invisible il apparaît. Cela dépend donc de l'état de l'entité.
Pareil pour *light*, qui est une entité de type toggle : si la lumière est allumée, elle s'éteint ; et si elle est éteinte elle s'allume.

Cette entité a un nouvel attribut : "Starts invisible". S'il est coché, le mur sera invisible au début. Sinon, il sera visible. Il ne tient qu'à vous par la suite de modifier son état (visible/invisible).

Les murs illusoires

Entité concernée : func_illusionary
Type d'entité : entité-bloc
Difficulté : facile

Parmi les différents types de murs qui existent, en voici un nouveau : le *func_illusionary*. Ce mur est visible aux yeux du joueur... mais si vous le touchez, vous vous apercevrez que vous passez à travers. Tout le monde passe à travers : vous, les ennemis, les scientifiques et même vos balles !

C'est pour cela qu'on l'appelle le mur illusoire. Il est assez peu utilisé, mais ça peut servir (si vous avez envie de perturber le joueur, c'est gagné à tous les coups :).

Voici un nouvel attribut, propre à cette entité :

- Contents : modifie le contenu du bloc.
  - Empty : le bloc est totalement vide, on passe à travers comme s'il n'existait pas.
  - Volumetric Light : on passe toujours à travers, mais avec un peu plus de mal. Idéal pour simuler le passage dans une zone de végétation (bloque un peu le joueur mais le laisse passer quand même).

Voilà une bonne chose de faite. Vous savez maintenant traiter la transparence et l'appliquer à des entités-bloc de murs.
On progresse, on progresse... 

Monter et descendre

L'objectif de ce thème est de vous apprendre à créer des étages dans vos niveaux et à créer des échelles, escaliers, ascenseurs... pour s'y rendre.

Les mappeurs qui débutent ont tendance à oublier que Worldcraft permet de créer des maps en 3 dimensions. Qu'est-ce que je veux dire par là ? Qu'ils n'exploitent pas tout l'espace disponible pour la map. En effet, si on se contente de faire une map plate (sur un étage), elle sera peut-être grande mais pas très variée, alors qu'elle pourrait être déclinée sur plusieurs étages.

Complexifiez vos maps ! Créez des immeubles à 6 étages, des égouts, des souterrains etc etc... L'intérêt tactique n'en sera que meilleur

Bon, je ne vous explique pas comment créer un étage :-p Vous avez juste besoin de créer un second bloc, qui fera à la fois office de plafond pour le niveau 1, et de sol pour le niveau 2.
Pour ce chapitre, j'ai créé une salle à deux étages. Le joueur commence en bas, et a envie d'un petit rafraîchissement de l'étage du dessus.

Pour monter au second étage, on a plusieurs solutions. Nous allons commencer par l'escalier.

Les escaliers

Entité concernée : aucune
Type d'entité : aucun
Difficulté : facile

Ca vous laisse perplexe ? Pourtant, c'est vrai : un escalier n'est pas une entité.

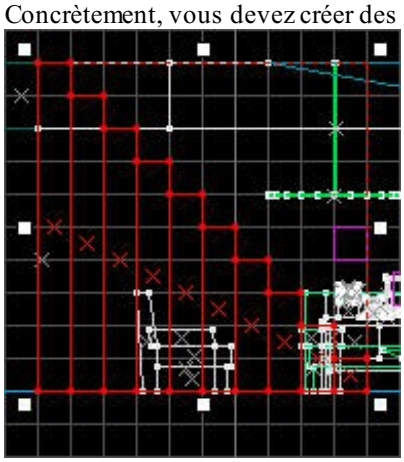
Bon nombre de débutants (moi notamment) se creusent la tête pour créer un escalier. Lorsqu'ils demandent à quelqu'un de plus expert comment faire, celui-ci leur répond souvent : "Ben, il suffit de créer un escalier !". Et c'est vrai en plus

Si je vous parle des escaliers dans un thème d'entité, c'est parce que je sais que c'est là que vous irez chercher en premier.
Bon trève de bavardages, on va créer ce %$#$%!# d'escalier

De quoi est constitué un escalier ?

Un escalier est en fait une succession de blocs de plus en plus grands. Chaque bloc correspond à une marche. Si les blocs sont proches les uns des autres et que la hauteur qui les sépare n'est pas trop importante, alors le joueur monte de l'un à l'autre naturellement, sans avoir à sauter.

Concrètement, vous devez créer des blocs et les superposer comme ceci :



C'est la vue de côté. On peut donc voir ici comment sont positionnés les blocs pour former l'escalier.

La vue 3D donnera le résultat suivant :

Notez que j'ai auparavant fait un petit carving dans le toit pour créer un "trou", afin que le joueur puisse passer

Lorsque le joueur arrivera à l'escalier, il "montera" naturellement, comme si de rien n'était.

Le joueur peut monter un escalier tant que les blocs (marches) ne sont pas espacées de plus de 8 unités de hauteur. Au-delà, c'est un obstacle le joueur ne montera pas.
Sur cet exemple, c'est déjà très raide. Essayez de faire des marches assez peu espacées pour être tranquilles.

Entraînez-vous à créer des escaliers vous aussi. Ca paraît un peu bizarre au début, mais c'est en fait très simple vous verrez

Les échelles

Entité concernée : func_ladder et func_wall
Type d'entité : entité-bloc
Difficulté : facile

Seconde possibilité pour monter à l'étage : créer une échelle. Les échelles sont des éléments très courants dans les maps pour Counter et Half-Life. En effet, elles permettent de monter rapidement une hauteur importante, tout en occupant un minimum d'espace.
C'est donc pour cela que les mappeurs préfèrent en général les échelles aux escaliers (qu'il ne faut pas oublier toutefois).

De plus, la création des échelles est vraiment simple (comparé à ce qui nous attend plus bas pour les ascenseurs), à condition de bien suivre dans l'ordre la démarche que je vais vous donner.

Pour créer une échelle :

1. Créer un premier bloc avec une texture d'échelle ("ladder").
2. Appliquer à ce bloc l'entité func_wall, pour le rendre transparent. Les échelles contiennent du bleu dans leur texture, que l'on peut faire disparaître comme nous l'avons appris dans le chapitre précédent.
   Mettre les valeurs suivantes à ces attributs :

   Render Mode = Solid
   FX Amount (1 - 255) = 255

3. Créer un second bloc ayant la même taille que l'échelle, mais cette fois avec la texture AAATRIGGER. Placez-le juste devant le premier bloc.
4. Appliquez à ce nouveau bloc l'entité *func_ladder*. Vous n'avez pas besoin de renseigner son attribut ("Name") qui ne sert a rien. Laissez tel quel.

C'est tout ! L'échelle est prête !
Au cas où vous n'auriez pas bien compris, le site du Zér0 vous fait le dessin en bonus

Testez votre map et vous verrez que ça marche parfaitement !

Comment ça fonctionne ce charabia ???

Oui, ça doit vous paraître bizarre. Il est important que vous compreniez ce que je viens de vous montrer...

Pour créer une échelle, on se sert de 2 blocs :

- *func_wall* est un simple bloc transparent. Il ne sert qu'à montrer la présence d'une échelle, mais il n'a pas la fonction d'une échelle. Si vous vous approchez d'un tel bloc, vous verrez que vous ne pourrez pas monter.
- *func_ladder* a la fonction d'une échelle. C'est lui qui indique au moteur de Half-Life qu'il faut faire monter le joueur.

Le func_ladder est invisible (on lui applique la texture AAATRIGGER pour se rappeler dans Worldcraft que ce bloc est invisible). On l'associe toujours avec un bloc visible : si on ne le faisait pas, il y aurait une échelle invisible et le joueur monterait sans savoir pourquoi

Elémentaire, non ?

Bon, si vous avez mal compris je vous conseille de relire et de prendre des forces avant d'attaquer la suite. Les escaliers et les échelles, c'était du pipeau par rapport à ce qui vous attend

Les ascenseurs automatiques

Entité concernée : func_plat
Type d'entité : entité-bloc
Difficulté : moyen

Pour monter à l'étage, on peut aussi se servir des ascenseurs. Ils sont plutôt pratiques si vous voulez que plusieurs personnes montent en même temps (toute une team). Et puis, ça met de l'ambiance dans la map !

Le problème, c'est que faire un ascenseur c'est pas vraiment facile. Déjà, on a le choix entre 2 types d'ascenseurs :

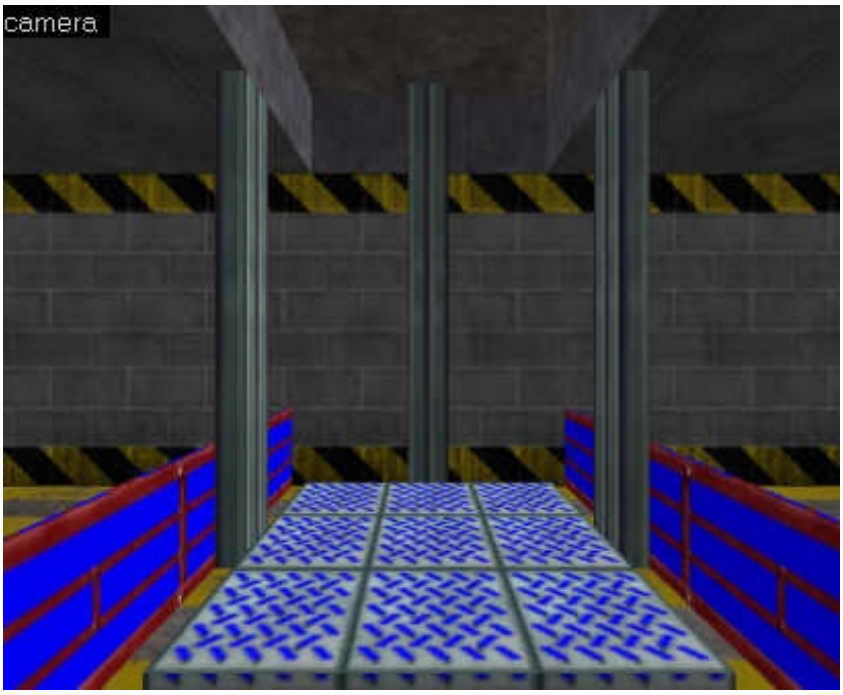
- Les ascenseurs automatiques : ils se mettent en marche dès que vous êtes dedans.
- Les ascenseurs manuels : il faut appuyer sur un bouton pour les mettre en route (plus sympa mais un peu plus difficile).

On commence donc par les ascenseurs automatiques.

Dans un premier temps, vous devez créer l'architecture de votre ascenseur. Ca peut être un simple bloc, ou un ensemble de blocs (porte d'entrée, acoudoir etc...). Pour commencer, on va faire quelque chose de simple, avec un seul bloc, mais vous verrez qu'on peut obtenir un bien meilleur effet avec un ascenseur constitué de plusieurs blocs.

Pour créer un ascenseur, utilisez des textures du groupe ELEV par exemple, comme {ELEV_FLR.

Voici par exemple un ascenseur très simple. Notez que seul le bloc en bas sert d'ascenseur (entité *func_plat*), le reste n'est que du décor.



Après avoir sélectionné le bloc et cliqué sur "ToEntity", choisissez l'entité *func_plat* et renseignez ses attributs. Voici les attributs qui sont susceptibles de vous intéresser :

- Render FX, Render Mode, FX Amount et FX Color : pour la transparence (voir le thème d'entité approprié). Ici par exemple, on en a besoin pour rendre le bleu de la texture transparent.
- Move Sound : choisissez le son qui sera joué lorsque l'ascenseur sera mis en marche.
- Stop Sound : son qui sera joué lors de l'arrêt de l'ascenseur. Choisissez un son du même groupe de préférence (tech, freight, squeek...).
- Sound Volume : volume sonore. Mettez un valeur entre 0 et 1 (1 étant la valeur maximale).

  Le séparateur décimal est le point (.). Au lieu d'écrire 0,85 vous devrez taper 0.85.
- Travel altitude (can be negative) : très important, c'est la hauteur (en unités de Worldcraft) dont montera ou descendra votre ascenseur.

  Si le bloc se trouve initialement en bas et que vous voulez qu'il monte, vous devez mettre un nombre négatif tel que -100.
- Speed : vitesse de mouvement de l'ascenseur. Ne rentrez pas une valeur trop élevée ou vous risquez de faire bugger Half-Life...

Par ailleurs, cette entité possède 1 seul flag, "Toggle", qui permet d'éviter que l'ascenseur retourne automatiquement à son position de départ.

Testez votre map : lorsque vous vous approcherez de l'ascenseur, il se mettra en marche tout seul.

Les ascenseurs manuels

Entité concernée : func_door et func_button
Type d'entité : entité-bloc
Difficulté : moyen

Pour créer un ascenseur manuel, vous avez besoin de 2 entités. Créez dans un premier temps le bouton qu'il faudra actionner pour mettre en marche l'ascenseur (*func_button*). Ensuite, créez votre ascenseur (généralement constitué de plusieurs blocs), sélectionnez ces blocs et transformez-les en entité *func_door*.

Dans le *func_button*, renseignez l'attribut Target, on inscrivant le nom de l'ascenseur. Par exemple :

Target = "mon_ascenseur"

Ensuite, vous devez renseignez les attributs de l'ascenseur (*func_door*). Voici les plus utiles :

- Name : nom de l'ascenseur. Donnez le même nom que celui que vous avez écrit pour le bouton. Ici, on aurait :

  Name = "mon_ascenseur"
- Speed : vitesse de déplacement de l'ascenseur.
- Move Sound : son joué lors du déplacement.
- Stop Sound : son joué à l'arrêt de l'ascenseur.
- Delay before close, -1 stay open : temps en secondes avant que l'ascenseur ne revienne àa sa position de départ. Mettez -1 si vous voulez que l'ascenseur ne revienne pas tout seul à sa position de départ.
- Lip : hauteur du déplacement. Mettez obligatoirement une valeur négative.

Il y a d'autres attributs, mais pour la plupart ils ne sont pas intéressants si on fait un ascenseur.

Ensuite, vous devez modifier l'angle de déplacement (en haut à droite de la fenêtre). Sélectionnez "Up" pour que l'ascenseur monte, et "Down" pour qu'il descende.

Enfin voici les flags à cocher (ou à ne pas cocher) :

- Starts Open : l'ascenseur commence dans sa position finale. Ca permet de faire l'inverse, en quelque sorte. Ne le faites pas, ça va encore plus vous compliquer la vie 
- Don't link : aucune idée :(. Si quelqu'un sait ce que c'est...
- Passable : on peut passer à travers l'ascenseur. Il n'est pas matériel. Ca n'a aucun intérêt pour l'ascenseur.
- Toggle : coché, ce flag permet de faire en sorte que l'ascenseur monte s'il est en bas, et qu'il descende s'il est en haut. Utile si vous avez mis la valeur -1 à "Delay before close".
- Use only : cochez ce flag ! Ainsi, l'ascenseur ne démarrera que si on a appuyé sur le bouton (c'est ce qu'on voulait faire non ?).
- Monster Can't : les monstres ne peuvent pas se servir de l'ascenseur (dans le cas d'une aventure solo bien entendu).
- Not in deathmatch : l'ascenseur n'apparaîtra pas dans une partie multijoueur. On coche très rarement ce flag...

Les ascenseurs rotatifs

Entité concernée : func_platrot
Type d'entité : entité-bloc
Difficulté : assez difficile

Ce type d'ascenseur fait monter le joueur tout en effectuant une rotation. C'est un peu tordu et assez difficile à réaliser, de plus on n'en voit pas beaucoup dans la campagne de Half-Life...

Si vous tenez à en mettre un dans votre map, alors créez l'ascenseur puis ajoutez un bloc ORIGIN au niveau de l'axe de rotation de l'ascenseur (généralement au centre).
Sélectionnez l'ascenseur et le bloc ORIGIN , et convertissez-les en entité *func_platrot*.

Vous connaissez maintenant la plupart des attributs d'un ascenseur, cependant il y en a quelques nouveaux :

- Speed of rotation : c'est la vitesse à laquelle l'ascenseur tourne sur lui-même tout en montant (je vous raconte pas les calculs que doit faire le moteur de Half-Life pour concilier tout ça
- Spin amount : distance de rotation en degrés (ex : 180).

Les flags vous permettront notamment de changer le plan de la rotation (qui est par défaut de Z, mais vous pouvez mettre X ou Y).
Voilà, à partir de maintenant vous allez pouvoir créer des étages dans vos niveaux, des escaliers, des ascenseurs, des échelles... Votre niveau sera bien meilleur, je vous l'assure 

Les portes

Eléments indispensables à toute map, les portes permettent de rajouter de l'ambience à vos maps (si si je vous l'assure !). Et puis, avouez que les maps qui ne possèdent pas de portes sont quand même rares !

Sachez que vous pouvez créer 2 types de portes :

- Les portes translatives : simples à mettre en place, mais bon pas très réalistes...
- Les portes rotatives : ce sont les meilleures, mais vous aurez un peu de mal au début à comprendre leur fonctionnement. Rassurez-vous, je ferai de mon mieux pour que vous ne sentiez même pas la difficulté

Le dernier paragraphe sur les portes vitrées est à part, car il ne s'agit pas d'une entité à part entière, mais plutôt d'un "truc" de mappeur, qui vous sera utile si vous voulez faire ce type de portes.

Les portes translatives

Entité concernée : func_door
Type d'entité : entité-bloc
Difficulté : assez facile

Les portes translatives sont les premières que nous étudierons car ce sont les plus simples à mettre en place. Lorsque vous saurez créer une porte translative, vous n'aurez pas grand chose à apprendre pour faire une porte rotative.

Mais qu'est-ce qu'une porte translative ?

C'est une porte qui s'ouvre en se délaçant sur un côté... un peu comme les portes qu'on peut rencontrer dans les complexes technologiques thermo-nucléaires qui étudient les matériaux anormaux

Par exemple, la porte ci-dessous s'ouvre du bas vers le haut : c'est une porte translative.

Les portes translatives peuvent s'ouvrir du bas vers le haut comme ici, mais aussi de gauche à droite, ou de droite à gauche

Autant vous dire qu'on n'en voit pas tous les jours... Mais dans Half-Life ça passait inaperçu. Ceci dit, si vous faites une map pour Counter-Strike, ça risque de pas faire très réaliste : je vous conseille d'utiliser une porte rotative dans ce cas.

Pour créer ce type de porte, vous aurez recours à l'entité-bloc *func_door* :

1. Créez un bloc avec une texture de porte (groupes DOOR et DR), ayant la même taille que cette texture de porte.
2. Sélectionnez le bloc.
3. Cliquez sur le bouton "To Entity".
4. Dans le menu déroulant "Class", choisissez l'entité *func_door*.

Simple non ? Encore vous reste-t-il à configurer ses paramètres pour la faire marcher... Je vais donc vous lister ici tous les attributs nouveaux qui peuvent nous être utiles... Les autres attributs dont je ne parlerai pas, nous les avons déjà vu pour la plupart dans les précédents chapitres.

- Angle : très important ! Vous devez définir l'angle d'ouverture à l'aide du petit radar en haut à droite de la fenêtre. Selon l'angle choisi, la porte ne s'ouvrira pas de la même manière !

  
  Si vous voulez que la porte s'ouvre verticalement, cliquez sur le menu déroulant en-dessous de "Angle" et choisissez "Up" ou "Down".
- Name : si vous voulez pouvoir déclencher l'ouverture de la porte en appuyant sur un bouton, donnez-lui un nom ici (voyez le chapitre sur les déclencheurs pour plus de détails).
- Speed : vitesse d'ouverture de la porte. Réduisez-la ou augmentez-la selon vos besoins.
- Master : très compliqué à utiliser. On reverra ça lorsqu'on étudiera le *multisource*.
  Cette entité vous permettra de verrouiller l'ouverture de la porte tant que plusieurs conditions n'ont pas été remplies (activation de plusieurs boutons par exemple).
- Move Sound : son qui sera joué lorsque la porte s'ouvrira.
- Stop Sound : son qui sera joué lorsque la porte aura fini de s'ouvrir.
- delay before close, -1 stay open : c'est le temps en secondes qui va s'écouler avant que la porte ne se referme toute seule. Si vous voulez qu'elle reste ouverte, mettez la valeur -1.
- Lip : par défaut, la porte se déplace d'autant d'unités que sa largeur (ou hauteur selon votre cas). Si vous voulez que la porte s'ouvre plus ou moins que cela, mettez une valeur différente de 0 à Lip (en unités de Worldcraft).
- Damage inflicted when blocked : si vous voulez que le joueur puisse se faire mal s'il se coince les doigts dans la porte, mettez ici le nombre de points de vie à lui retirer 
- Health (shoot open) : si vous voulez que l'on puisse ouvrir la porte en lui tirant dessus, mettez ici le nombre de points de vie qu'elle possède.
  Lorsqu'elle sera "tuée", elle s'ouvrira enfin 

Voici les flags disponibles :

- Starts Open : la porte commencera dans sa position ouverte.
- Passable : on traverse la porte sans être bloqué.
- Toggle : ce flag n'a pas grand intérêt ici.
- Use Only : cochez ce flag si vous voulez qu'on ne puisse ouvrir la porte qu'en appuyant sur la touche "Utiliser".
- Monsters Can't : les monstres ne pourront pas ouvrir la porte tous seuls.
- Not in deathmatch : l'entité ne sera pas présente dans une map multijoueur (ce qui est complètement inutile pour une porte).


Très important à savoir : l'entité func_door ne sert pas qu'à créer des portes.
On peut aussi s'en servir pour simuler le déplacement d'un objet, comme par exemple une voiture qui avance, un avion qui décolle etc...

Les portes rotatives

Entité concernée : func_door_rotating
Type d'entité : entité-bloc
Difficulté : moyen

Plus difficile à créer, mais aussi beaucoup plus sympathiques : voici les portes rotatives !

Respectez bien cet ordre pour créer une telle porte, sinon vous allez pas vous en sortir lol 

1. Créez un bloc avec une texture de porte, comme ceci :

2. Créez un bloc avec la texture ORIGIN (se trouvant dans halflife.wad). Ce bloc doit être placé au niveau de l'axe de rotation de la porte !

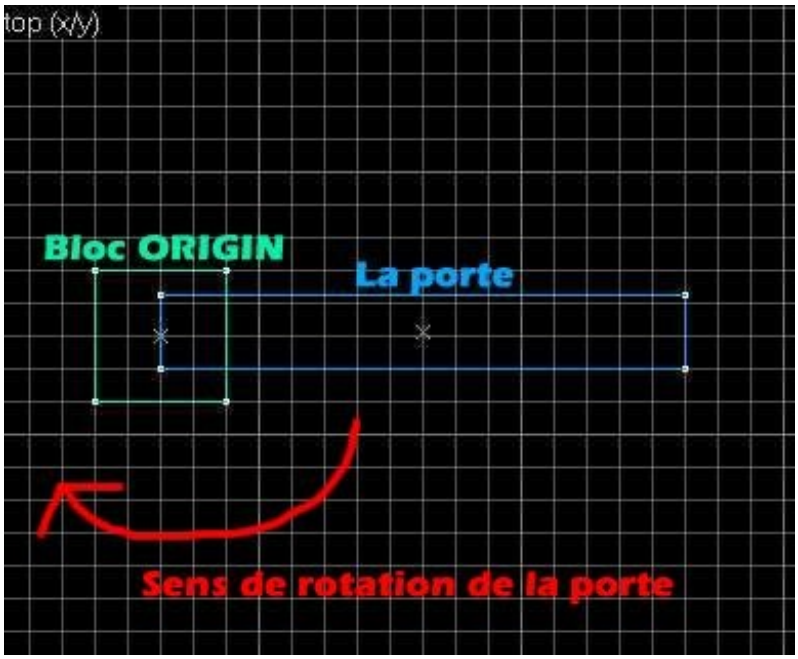
Voici la texture ORIGIN :

Vous devez savoir manier le bloc ORIGIN, c'est extrêmement important !!! On s'en resservira plus tard lorsqu'on créera des entités rotatives.

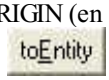
Mais qu'est-ce que l'axe de rotation de la porte ?

C'est l'endroit autour duquel la porte tourne. C'est en fait là que la rotation s'effectue.
Dans le cas d'une porte, la bloc ORIGIN doit être placé comme ceci :

Grâce à lui, Half-Life saura comment faire tourner le bloc.
Voyez par exemple comment ça se passe sur la vue de dessus (Top) :



Si vous avez compris ce petit schéma, alors vous avez tout compris

3. Sélectionnez la porte et le bloc ORIGIN (en maitenant la touche Ctrl enfoncée pour une multisélection).
4. Cliquez sur le bouton "To entity" 

5. Choisissez l'entité nommée *func_door_roating* dans la liste.
6. Enfin, éditez ses propriétés et ce sera bon

Nous avons déjà vu plus haut les propriétés d'une porte, je ne vais donc pas les répéter ici. Il y a seulement un nouvel attribut à connaître :

- Distance (deg) : c'est l'angle d'ouverture de la porte en degrés. Par défaut, la porte s'ouvre de 90°. Vous pouvez augmenter cette valeur ou la diminuer, mais ça n'a que peu d'intérêt (par exemple, si la porte s'ouvre de 10° le joueur ne pourra pas passer !)

Cette entité possède des nouveaux flags très intéressants :

- Reverse Dir : la porte s'ouvrira en sens inverse.
- One-way : on ne peut ouvrir la porte que dans un sens.
- X axis : par défaut, la porte s'ouvre selon l'axe des Z (vue Top). Mais vous pouvez la faire tourner selon l'axe des X si ça vous chante (c'est très rare quand même).
- Y axis : idem, mais pour l'axe des Y.

Quant aux autres flags, on les a déjà vu plus haut

Les portes momentanées

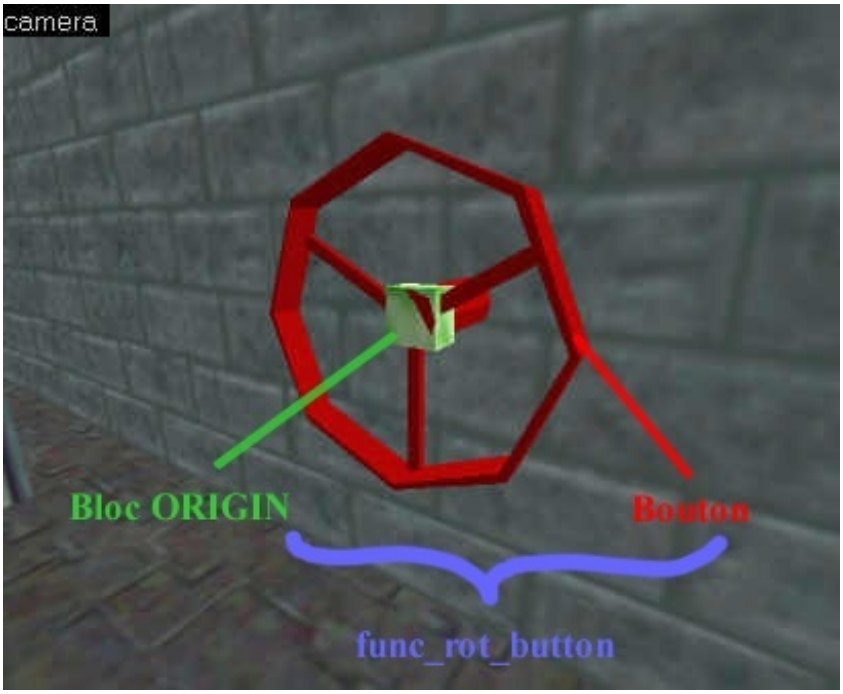
Entités concernées : momentary_door et momentary_rot_button
Type d'entité : entité-bloc
Difficulté : moyen

Les portes momentanées sont un peu spéciales. Ce sont des portes translatives qui doivent obligatoirement être ouvertes avec un bouton rotatif appelé *momentary_rot_button*.

Que se passe-t-il concrètement ? Le joueur doit actionner un bouton rotatif (une vanne par exemple), et laisser la touche "Utiliser" enfoncée pour que le bouton tourne jusqu'au bout. Plus le joueur fait tourner le bouton, plus la porte s'ouvre. Et dés qu'il arrête d'actionner le bouton, hop ! La porte se referme lentement...
Voilà de quoi assurer du suspense dans votre map pour le joueur qui doit vite se précipiter vers la porte avant qu'elle ne se referme

Le bouton

Pour commencer, créez le *momentary_rot_button* comme pour n'importe quel bouton rotatif (je vous l'ai appris dans un précédent chapitre). Voici par exemple une vanne bien moche dont je suis très fier et que je vous ai déjà présenté pour le *func_rot_button* (j'y ai passé du temps !) :



Sur ce dessin que je vous avais déjà présenté, il est écrit *func_rot_button*, mais je vous rassure : c'est exactement la même chose pour un *momentary_rot_button*

N'oubliez pas d'ajouter un bloc ORIGIN au *momentary_rot_button* pour indiquer l'axe de rotation du bouton, c'est très important !

Voici les nouveaux attributs de cette entité (les autres vous les avez déjà vu et revus dans les chapitres précédents) :

- Targetted object : indiquez ici le nom de la *momentary_door* à ouvrir.
- Auto-return speed : si le bouton doit revenir à sa position initiale, indiquez ici à quelle vitesse il doit revenir. Plus il revient vite et plus la porte se refermera vite ! Le joueur aura intérêt à se dépêcher !
- _minlight : niveau de luminosité minimum de l'objet (information pour HLrad). Si vous voulez que votre bouton soit un peu visible même s'il fait noir, mettez une valeur supérieure à 0.

Voici les flags particuliers à cette entité :

- Door Hack : permet de faire tourner plusieurs *momentary_rot_button* en même temps s'ils ont le même nom.
- Not useable : le bouton ne peut pas être utilisé.
- Auto-return : le bouton revient automatiquement à sa position initiale (et la porte en même temps). Je vous recommande de cocher ce flag, ça met un peu de piment dans l'action

La porte

Maintenant, créez une porte (de préférence une grosse porte blindée qui s'ouvre et se ferme lentement ;). Transformez-la en *momentary_door*.
Pensez à lui donner un nom (Name), pour que le bouton puisse l'appeler lorsqu'il est actionné.

Les attributs sont les mêmes que pour une porte translative, donc vous saurez remplir correctement ses propriétés vous-mêmes.

Je vous recommande d'infliger des dommages au joueur s'il se fait coincer par la porte (Damage inflicted when blocked), parce qu'en général c'est ce qu'on fait pour ce genre de portes.

Vous n'avez maintenant plus qu'à tester votre oeuvre : il faudra actionner un moment le bouton pour ouvrir complètement la porte et se précipiter vers elle avant qu'elle ne se referme !
Notez qu'il n'est pas obligatoire de faire refermer la porte, mais personnellement je trouve que c'est plus rigolo de faire ça avec les

momentary_door

Le truc des portes vitrées

Une porte vitrée est une porte au milieu de laquelle il y a une vitre. Vous pouvez donc voir ce qui se passe de l'autre côté de la porte.

C'est facile à imaginer, mais sous Worldcraft c'est (en théorie) impossible à faire.
En effet, si vous appliquez un effet de transparence à votre *func_door* ou à votre *func_door_rotating*, toute votre porte sera transparente !

Comment faire alors pour rendre uniquement la vitre transparente ?

Il va falloir ruser... et se servir d'un déclencheur.

Commencez par créer la porte, puis la vitre (j'ai fait un petit Carving pour mettre la vitre dans la porte) :

1. Sélectionnez tout d'abord la porte sans la vitre.
2. Affectez-lui l'entité *func_door_rotating* (ou *func_door*) et donnez-lui un nom (par exemple "portevitree").
   Cochez le flag "Use only".
   Editez ses autre propriétés.
3. Affectez l'entité *func_door_rotating* (ou *func_door*) à la vitre, et donnez-lui le même nom que la porte ("portevitree").
   Affectez-lui les attributs de transparence que vous voulez. Cochez le flag "Use only".
   Editez ses autres propriétés de la même manière que la porte. Les propriétés doivent être exactement les mêmes, sinon ça ne marchera pas (je pense surtout à la vitesse d'ouverture "speed").
4. Maintenant, placez un *trigger_multiple* devant la porte, comme vous avez appris à le faire.
5. Dans le champ "Target", mettez le nom de votre porte ("portevitree").

Le principe est le suivant : le joueur arrive devant la porte, et traverse le *trigger_multiple*. Celui-ci déclenche alors l'ouverture simultanée de deux portes ayant le même nom : la porte et la vitre.
Si c'est bien fait le joueur n'y voit que du feu !!!

Il est important de cocher le flag "Use only" pour s'assurer que l'on ne pourra ouvrir la porte qu'en passant par le *trigger_multiple*.

Voilà le résultat sous Worldcraft :

J'ai un peu exagéré la distance entre le *trigger_multiple* et la porte. En réalité, vous devriez carrément coller le *trigger_multiple* à la porte, pour être sûr que le joueur passera dedans.

Waterworld

Le titre de ce chapitre doit vous paraître un peu bizarre, mais c'est le meilleur que j'ai trouvé pour dire que j'allais parler de l'eau Les cinéphiles se rappelleront peut-être d'un vieux film du même titre (avec Kevin Costner) qui n'avait pas bien marché d'ailleurs...

Bon, mis à part cette petite parenthèse, on ne va pas s'égarer : on parle toujours bel et bien de Worldcraft
Et plus précisément, on va tenter de répondre à la question suivante :

Comment mettre de l'eau dans une map ?

Je vous rassure tout de suite, c'est extrêmement simple !

L'eau

Entité concernée : func_water
Type d'entité : entité-bloc
Difficulté : facile

Puisque c'est simple, on ne va pas s'y attarder longtemps.

Pour mettre de l'eau, commencez par créer un bloc avec une texture d'eau (elle doit impérativement commencer par un "!"). C'est dans cette zone qu'il y aura de l'eau.

Le fichier de textures "liquids.wad" contient de nombreuses textures d'eau !

Comme c'est une entité-bloc, vous savez ce que vous devez faire : cliquez sur le bouton "ToEntity" puis sélectionnez l'entité *func_water*.

Vous n'avez plus qu'à éditer ses propriétés que vous connaissez pour la plupart.

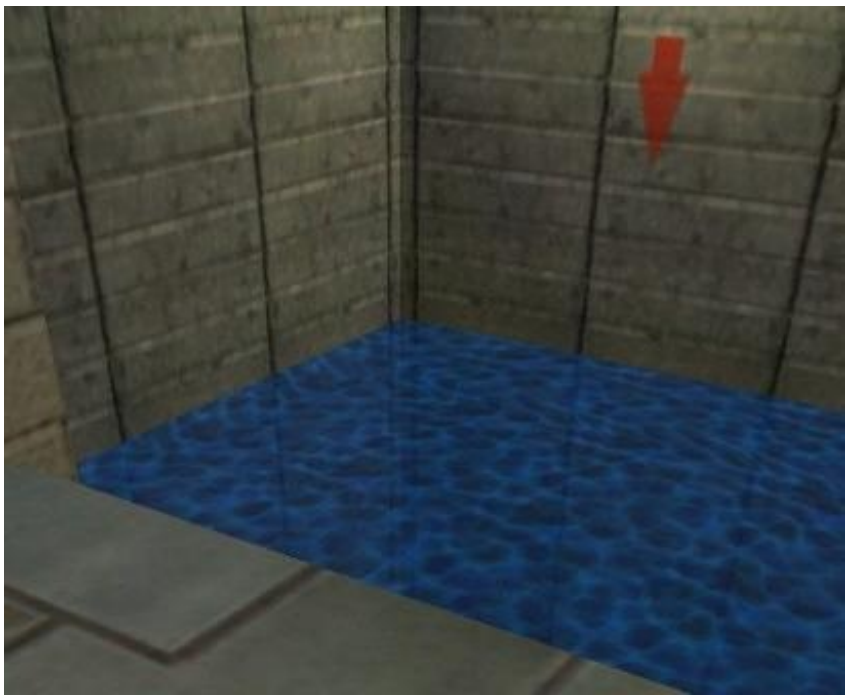
Pensez à définir sa transparence, sinon votre eau ne sera pas très jolie (mettez par exemple : *Render Mode* = *Additive*, *FX Amount* = *128*)

Ceci dit, cette entité possède 2 nouvelles propriétés :

- Contents : par défaut, le bloc contient de l'eau... mais on peut faire en sorte qu'il contienne autre chose :
  - Water : de l'eau normale, comme on l'aime
  - Slime : ce sont des déchets toxiques, et il vaut mieux éviter de nager dedans !
  - Lava : de la lave... enfin, un truc qui brûle et qui fait très mal.
- Wave Height : c'est la hauteur des remous que font les vagues. Si vous voulez créer une mer agitée en pleine tempête, mettez une valeur supérieure à 0. Sinon, laissez 0 pour que l'eau soit calme.

Si vous voulez que la valeur de "Wave Height" soit la même par défaut pour toutes les entités *func_water*, allez dans le menu Map/Map Properties, et modifiez la valeur de "Default Wave Height".

Voilà le résultat :



Les plus perspicaces auront certainement noté que cette entité possède toutes les propriétés et tous les flags d'une *func_door* ! Ce n'est pas par hasard : il est possible de faire monter ou baisser le niveau de l'eau, en se servant de ce que vous avez appris pour les portes translatives.

Imaginez par exemple une map avec des égoûts, avec une valve à la sortie... si un joueur tourne la valve, l'eau des égoûts monte, monte inexorablement... jusqu'à noyer tous les joueurs qui se trouvaient dans les égoûts !
Eh oui, il faut avoir de l'imagination en mapping

Les bulles

Entité concernée : env_bubbles
Type d'entité : entité-bloc
Difficulté : facile

Sympathiques en milieu aquatique ou dans des tuyaux, vous en avez certainement vu dans Half-Life... mais il est vrai que l'on a pas beaucoup utilisé cette entité, et peut-être que certains ne savaient pas que l'on pouvait faire apparaître des bulles dans une map.
C'est tout à votre avantage : le joueur se laisse vite impressionner par un effet comme celui-là, et pourtant cette entité est très simple à mettre en place !

Pour ceux qui auraient encore des trous de mémoire, voici à quoi ressemblent les bulles sous l'eau :

Créez un bloc avec une texture AAATRIGGER, recouvrant toute la zone où vous voulez qu'il y aie des bulles.

Transformez le bloc en entité, comme vous savez le faire (bouton "ToEntity") et choisissez l'entité *env_bubbles*.

Cette entité possède 4 attributs :

- Angle (radar en haut à droite) : servez-vous du radar ou de la liste "Angle" pour déterminer dans quelle direction vont les bulles. Mettez "Up" par exemple si les bulles vont vers le haut (c'est le plus courant).
- Name : nom de l'entité.
- Bubble density : c'est la densité des bulles. Plus cette valeur est élevée, plus il y a de bulles à la fois dans un groupe.
- Bubble frequency : fréquence d'apparition des bulles. Les bulles apparaissent de temps en temps, et par groupes. Plus cette valeur est élevée, plus il y aura de groupes de bulles.
- Speed of Current : vitesse de déplacement des bulles.

Cette entité peut être activée par un trigger ou un bouton : il suffit de l'appeler par son nom. Si vous voulez qu'il n'y aie pas de bulles au départ, cochez le flag "Start Off".
Vous voilà maintenant aptes à créer des niveaux sous-marins !

Déclencher l'Apocalypse

Des fois, ça nous démange : on a envie de tout faire péter. Half-Life a tout prévu pour nous rassasier :

- Les étincelles : pour annoncer que quelque chose cloche et que ça risque de sauter.
- Les tremblements de terre : d'abord une légère secousse, puis un tremblement de plus en plus important qui nous annonce qu'il vaudrait mieux ne pas moisir là longtemps
- Les explosions : aïe aïe aïe, ça commence à chauffer par là...
- Les bombardements : l'apothéose. Avec ça vous êtes sûr que le joueur est mort

Bref, vous avez à votre disposition tout un arsenal d'entités pour créer des effets spéciaux à faire pâlir les plus gros producteurs hollywoodiens !
Voilà en gros ce qu'on va étudier dans ce chapitre...

Les étincelles

Entité concernée : env_spark
Type d'entité : entité-point
Difficulté : très facile

On ne peut pas rêver plus simple : placez une entité-point *env_spark* là où vous voulez qu'il y aie des étincelles. Vous n'avez rien à faire d'autre, c'est prêt

Voici ce que vous pourrez voir dans votre map :

Petit détail technique : il faut placer le centre de l'entité à l'endroit d'où doivent jaillir les étincelles... sinon les étincelles sortiraient de nulle part !

A noter que vous avez la possibilité d'appeler cette entité, ce qui explique la présence des flags "Toggle" et "Start ON", ainsi que celle de l'attribut "Name" que vous connaissez bien.
Cette entité possède un nouvel attribut :

- Max Delay : la durée pendant laquelle il y aura des étincelles. Laissez "0" pour que ça ne s'arrête jamais.

D'autres entités comme les *func_button* possèdent un flag "Sparks" qui produit automatiquement des étincelles sur l'objet. C'est plus rapide encore à mettre en place !

Les tremblements de terre

Entité concernée : env_shake
Type d'entité : entité-point
Difficulté : assez facile

L'entité-point *env_shake* permet de produire des tremblements de l'écran de toutes sortes : ça va de la petite secousse à peine perceptible au tremblement de terre post-apocalyptique d'une fin du monde, en passant par un choc puissant dû à une collision.
Bref, vous pouvez tout faire

Placez l'entité *env_shake* au centre du tremblement de terre (aussi appelé "épicentre" en géologie, mais je ne veux pas vous rappeler de mauvais souvenirs ;).
Si vous voulez que le tremblement se fasse ressentir dans tout le niveau, cochez le flag "GlobalShake".

Voici les attributs de cette entité (vous avez tout intérêt à les connaître par coeur) :

- Name : nom de l'entité, pour que vous puissez l'appeler avec un trigger. Je pense que vous voulez tous que le tremblement s'effectue à un moment précis et non pas au début de la map !
- Amplitude 0-16 : c'est la puissance de la secousse, allant de 0 à 16 (aucun rapport avec l'échelle de Richter je présume...)
- Effect Radius : c'est le rayon d'effet de la secousse, autour de l'épicentre... Si vous avez coché le flag "GlobalShake", alors cet attribut ne sert plus à rien.
- Duration (seconds) : durée du tremblement en secondes.

- 0.1 = jerk, 255.0 = rumble : c'est la fréquence des secousses. Je m'explique : un tremblement de terre est constitué de plusieurs secousses par seconde.
  Si vous mettez une valeur élevée (255.0), alors votre écran va beaucoup trembler.
  Si vous mettez une valeur faible (0.1), il n'y aura qu'une seule secousse, mais très puissante (pour simuler un choc par exemple).

Les explosions

Entité concernée : env_explosion
Type d'entité : entité-point
Difficulté : facile

Maintenant, rien de tel qu'une bonne petite explosion pour effrayer le joueur, le blesser, voire le tuer si vous êtes pas gentils
Voilà à quoi ressemble une explosion (pour ceux qui n'en auraient jamais vu :

Le fonctionnement reste simple : placez un *env_explosion* là où vous voulez qu'il y aie une explosion. Editez ensuite ses propriétés :

- Name : nom de l'explosion pour l'appeler avec un trigger.
- Magnitude : puissance de l'explosion. Une magnitude de 100 correspond à peu près à la puissance produite par une grenade.

Cette entité possède peu de propriétés mais de nombreux flags :

- No Damage : l'explosion ne blessera pas le joueur... elle servira donc juste à mettre un peu d'ambiance.
- Repeatable : c'est un flag important. Il permet de faire en sorte que l'explosion puisse se produire plusieurs fois... S'il n'est pas coché, il n'y aura qu'une seule et unique explosion.
- No Fireball : pas de flammes. C'est pour les esprits sensibles
- No Smoke : pas de fumée (prévention contre le tabagisme ?)
- No Decal : l'explosion ne laissera pas de traces (les traces sont des decals).
- No Sparks : pas d'étincelles (ça pourrait faire peur au joueur).

Bon, on a surtout l'impression que les flags permettent de retirer tous les éléments de l'explosion : pas de flammes, pas de fumées, pas d'étincelles, pas de bobos etc etc... Où est l'intérêt d'avoir une explosion si ce n'est pour tout faire flamber, fumer, étinceller et, bien entendu, blesser le joueur ? Je vous le demande...
L'ère des explosions propres est arrivée !

Les bombardements

Entité concernée : func_mortar_field
Type d'entité : entité-bloc
Difficulté : moyen, voire difficile selon le cas

Pour être sûr d'achever le joueur, je vous conseille de vous servir des bombardements. On entend d'abord des bruits d'avions, puis le largage des bombes, puis tout explose et la terre tremble. Une entité radicale !

Commencez par créer un bloc avec la texture AAATRIGGER comme on a souvent l'habitude de faire avec les entités-bloc. Convertissez ce bloc en entité avec le bouton "ToEntity", et choisissez l'entité nommée *func_mortar_field*

Il y a plusieurs moyens de se servir de cette entité, en fonction de l'utilisation que vous faites de ses attributs :

- Name : nom de l'entité. Vous allez forcément devoir l'appeler (avec un bouton ou un trigger) pour déclencher le bombardement à un moment précis.
- Spread Radius : puissance des bombes. Plus une bombe est puissante, plus son rayon d'action est important (et plus vous avez de chances de mourir dans l'explosion ;o)
- Repeat Count : nombre de bombes larguées.
- Targeting : cet attribut est très important, vous ne devez pas le négliger. Il peut prendre 3 valeurs :

    - Random : les bombes sont larguées aléatoirement dans la zone où se situe le *func_mortar_field*.
    - Activator : les bombes tomberont précisément à l'endroit où l'entité a été appelée, c'est à dire soit dans la zone du trigger que le joueur a traversée, soit à proximité du bouton où il a eu le malheur d'appuyer.
      En utilisant cette valeur, vous êtes de toute manière sûrs que le joueur va se prendre un bel obus dans la tronche !!!
    - Table : c'est le plus difficile à utiliser... Si vous vous souvenez bien, à un moment dans Half-Life le joueur doit détruire un très gros alien et il dispose d'un tableau de commandes grâce auquel il peut indiquer où les bombes doivent être larguées. Ainsi, s'il se débrouillait bien, il pouvait faire larguer les bombes sur le méchant monstre afin de le détruire définitivement.
      Je ne saurais vous expliquer comment refaire la même chose dans votre map, mais quoiqu'il en soit c'est pas évident à faire fonctionner...
- X Controller : sert uniquement si vous utilisez la valeur "Table". Vous devez indiquer ici le nom de l'entité qui dirige le point de largage des bombes au niveau de l'axe des X.
- Y Controller : idem pour l'axe des Y.

Aucun flag n'est disponible pour cette entité... Enfin, elle est plutôt facile à utiliser du temps que vous ne vous servez pas d'une table de contrôle pour diriger les bombardements. Dans ce cas, bonjour la galère 

Lol, si avec tout ça vous n'arrivez pas à donner une ambiance explosive à votre map, je peux rien faire de plus pour vous 

Casser et pousser

Grâce à ce chapitre court et simple, vous saurez créer des objets cassables et poussables, très très fréquents dans une map. On peut dire que ce sont des éléments incontournables !

Vous verrez aussi comment faire pour pousser le joueur (lorsqu'il est dans un conduit de ventilation par exemple, il faut simuler le vent qui le pousse).

Objets cassables

Entité concernée : func_breakable
Type d'entité : entité-bloc
Difficulté : facile

Il existe de nombreux objets que l'on peut casser dans Half-Life : caisses, vitres, ordinateurs etc...
Je pense que vous les connaissez bien, alors je ne vais pas m'attarder à vous expliquer ce que c'est. Voyons plutôt comment rendre un objet cassable :

Créez le bloc que l'on pourra casser. N'oubliez surtout pas de lui donner la texture appropriée (vitre, caisse etc...). Transformez-le en entité-bloc de type *func_breakable*.

Ouvrez ensuite ses propriétés. Voici la liste complète de ses attributs :

- Name : nom de l'objet. Si vous appelez un *func_breakable*, à l'aide d'un trigger, alors il se cassera tout seul.
- Global Entity Name : nom global de l'objet, si vous faites une aventure solo sur plusieurs maps.
- Target on Break : nom de la cible à déclencher lorsque l'objet est cassé.
- Strength : résistance de l'objet. Par défaut, la valeur est 1, c'est-à-dire qu'un simple coup casse l'objet. Si vous voulez que l'objet soit plus résistant, alors mettez une valeur plus élevée (100 correspond à la vie d'un joueur).
- Material Type : très important ! Cela définit le type de matériel qui est cassé, et donc l'apparence qu'auront ses débris.
  - Glass : vitre.
  - Wood : bois.
  - Metal : blocs métalliques.
  - Flesh : euh, ça c'est des débris d'alien bien dégueullasses...
  - Cinder Block : blocs de ciment (si on casse un mur).
  - Ceiling Tile : morceaux de plafond (ces morceaux sont fins).
  - Computer : débris d'ordinateur. On peut voir des cartes PCI et des disques durs s'envoler... euh, vous n'oseriez tout de même pas casser un ordinateur ?
  - Unbreakable Glass : voilà ce que j'appelle un truc débile : "Vitre incassable". Si vous mettez cette valeur, alors l'objet sera une vitre incassable... alors pourquoi avoir mis ça dans un *func_breakable* ? Voilà un bel exemple d'illogisme !
  - Rocks : morceaux de roches des montagnes rocheuses que vous connaissez bien depuis Half-Life. Je me demande avec quel arme le joueur pourra casser cette roche... un bazooka ? ;o)
- Gibs direction : indique la direction dans laquelle partiront les débris
  - Random : les débris partiront dans tous les sens (par défaut).
  - Relative to attack : les débris partiront dans la direction opposée à l'attaque. C'est peut-être plus réaliste, mais le joueur ne le remarque pas forcément. Ca impose d'ailleurs un peu plus de calculs à l'ordinateur.
- Delay before fire : temps en secondes avant de déclencher la cible définie dans "Target on break".
- Gib model : si vous voulez utiliser vos propres débris, indiquez ici le fichier *.mdl à utiliser.
- Spawn On Break : utilisé généralement avec les caisses. Si l'objet est cassé, cela permet de libérer un bonus (qui se trouvait à l'intérieur de la caisse).
  Par défaut, aucun bonus n'apparaît ; mais si vous voulez en offrir un au joueur, alors faites votre choix dans la liste (qui est plutôt longue). Servez-vous du chapitre sur les armes et les munitions pour savoir quoi choisir.
  Bien entendu, cela n'est valable que pour Half-Life
- Explode Magnitude (0=none) : par défaut, l'objet n'explose pas lorsqu'il est cassé (valeur 0). Si vous voulez qu'une explosion se déclenche lorsque l'objet est cassé, alors mettez une valeur supérieure à 0 correspondant à la puissance de l'explosion. 100 = puissance d'une grenade.
- Render FX, Render Mode, FX Amount et FX Color : voir le chapitre sur la transparence pour savoir comment les utiliser.
- Minimum light level : niveau de luminosité minimum de l'objet, si vous voulez que celui-ci reste visible même dans le noir.

Après cela, voici les flags que vous pouvez cocher :

- Only Trigger : on ne peut casser l'objet que par l'intermédiaire d'un trigger ou d'un bouton. Dans ce cas, veillez à bien donner un nom à l'entité pour qu'on puisse l'appeler.
- Touch : explose lorsqu'on le touche.
- Pressure : explose lorsqu'on l'utilise.
- Instant Crowbar : quelle que soit la résistance de l'objet (Strength), un seul coup de pied de biche suffit à le casser.

Ouf ! Voilà je crois n'avoir rien oublié

De toute façon, un func_breakable reste toujours simple à utiliser, et on ne se sert pas à chaque fois de tous ses attributs !

Objets poussables

Entité concernée : func_pushable
Type d'entité : entité-bloc
Difficulté : facile

Un *func_pushable* est normalement très simple à utiliser :
Créez l'objet poussable (généralement un caisse), et transformez-le en entité-bloc *func_pushable*.

Normalement, cela suffit amplement à intégrer un objet poussable dans sa map... à moins que vous ne vouliez que l'objet soit poussable ET cassable.
Dans ce cas, regardez les attributs : vous retrouverez les mêmes que ceux des *func_breakable* que nous avons étudié plus haut !

Inutile donc de vous réexpliquer comment les utiliser, vous trouverez vous-mêmes les attributs à remplir.

Pensez à cocher le flag "Breakable" pour indiquer que cet objet est poussable et cassable à la fois.

Cette entité possède toutefois 3 nouveaux attributs :

- Hull Size : indiquez la taille de l'objet. Cet attribut n'est pas très important mais permet à Half-Life de déterminer de quelle manière l'objet doit être poussé. Il n'est vraiment pas utile de modifier la valeur par défaut.
  - Point size : par défaut. La taille est calculée à partir du point de contact entre le joueur et l'objet.
  - Player size : taille du joueur.
  - Big size : grande taille.
  - Player duck : taille du joueur accroupi.
- Friction (0-400) : ben c'est la friction ;o) Concrètement, plus il y a de friction, plus l'objet glissera facilement. Réciproquement, moins il y a de friction, plus le joueur va avoir du mal à pousser l'objet.
- Buoyancy : flottabilité. Si l'objet est poussé dans l'eau, alors il va plus ou moins flotter. A vous de déterminer la partie de l'objet qui restera émergée de l'eau.

Quant au flag "Breakable", je vous en ai déjà parlé plus haut. S'il est coché, alors l'objet sera cassable et poussable à la fois.

Effets sonores

Un mappeur ne se contente pas de créer un monde d'images, il peut aussi mettre du son dans sa map ! Eh oui, il ne faut pas l'oublier, surtout qu'un bon fond sonore met très facilement de l'ambiance dans votre map !

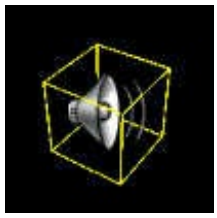
Alors, allons-y, et montez le son !

Jouer du son

Entité concernée : ambient_generic
Type d'entité : entité-point
Difficulté : assez facile

Placez pour commencer une entité-point *ambient_generic* dans votre map. Cette entité doit être placée à l'endroit où vous voulez qu'on entende le son.

Sous Worldcraft, cette entité ressemble à ceci :



Vous pouvez ensuite ouvrir la fenêtre des propriétés, notamment pour indiquer le son qui sera joué !

En ce qui concerne les fichiers sonores, ils sont tous en *.wav et situés dans le dossier Half-Life\NomDuMod\sound\
Surtout, n'oubliez pas de distribuer le son avec votre map si les joueurs n'ont pas ce fichier !

- Name : nom de l'entité, pour que vous puissiez l'appeler avec un trigger.
- WAV Name : très important, il ne faut pas oublier de le renseigner. C'est le chemin d'accès au fichier WAV que Half-Life devra jouer.
  Choisissez bien votre WAV, écoutez-les un peu tous pour savoir lequel prendre.
- Volume (10 = loudest) : volume sonore. 10 correspond au volume le plus élevé. Je vous recommande fortement de mettre cette valeur, parce que sinon on n'entendra pas grand chose !
- Dynamic Presets : vous pouvez choisir un preset pour modifier la façon dont le son sera joué.

  Le terme "dynamic" signifie ici que la modification du son se fera en temps réel lorsque vous jouerez sur la map. Concrètement, cela veut dire que les petites configs risquent de ramer un peu, voire beaucoup si vous avez déjà des r_speeds élevées !
- Start Volume : si vous voulez que le volume sonore monte ou descende petit à petit, mettez ici la valeur du volume initial.
- Fade in time (0-100) : pour réaliser un "fade in". Cela permet d'introduire lentement le son. Indiquez le temps pendans lequel ce "fade in" s'effectuera.
- Fade out time (0-100) : idem pour un "fade out", c'est-à-dire pour que le son disparaisse lentement à la fin (lorsqu'il a été joué).
- Pitch (> 100 = higher) : pour modifier le pitch du son. Si vous ne savez pas ce que c'est, vous n'avez qu'à essayer : vous verrez, ça déforme complètement le son !
  Mettez une valeur inférieure à 100 pour un pitch plus faible, et une valeur supérieure à 100 pour un pitch plus élevé.
- Start Pitch : valeur du pitch initial si vous voulez que le pitch évolue au cours du temps.
- Spin up time (0-100) : modifie le temps pendant lequel un spin up du son s'effectuera.
- Spin down time (0-100) : idem pour un spin down.
- Attributs commençant par LFO : LFO signifie Low Frequency Oscillator. C'est une fonction spéciale de Half-Life pour modifier le son (comme si on ne pouvait pas le modifier avec tous ces attributs !).
  A moins d'être vraiment très calés en musique électronique, aucun joueur ne connaît ces fonctions ni même ne les utilise. Vous prenez pas la tête avec tout ça, croyez-moi
- Incremental spin up count : valeur de l'incrément (un nombre qui s'ajoute à lui-même plusieurs fois) qui augmente le spin up à chaque fois. Il faut avoir rempli l'attribut "Spin up time" pour que ça marche !

Entre nous, je n'utilise pas toutes ces fonctions-là je vous rassure
A vrai dire, c'est bien trop compliqué pour rien, et vous n'aurez certainement jamais besoin de toutes ces fonctionnalités. Pensez simplement à remplir le champ "WAV Name", ce sera amplement suffisant

Passons maintenant aux flags de cette entité (ils sont plus intéressants que les attributs) :

- Play everywhere : le son sera joué partout dans la map. La place de l'*ambient_generic* n'a alors plus aucune importance.
- Small Radius : si "Play everywhere" n'est pas coché, le son ne s'entendra que dans un rayon plus ou moins grand autour de l'*ambient_generic*.
  Cochez ce flag pour que ce rayon soit petit (on n'entendra le son que très près de l'*ambient_generic*).
- Medium Radius : idem, mais cette fois la taille du rayon où le son s'entendra est moyen.
- Large Radius : idem, mais le son sera joué dans un rayon bien plus large encore.
- Start Silent : le son est inactif au lancement du niveau, il faudra alors l'activer avec un trigger pour qu'on l'entende.
- Is Not Looped : le son ne sera pas joué en boucle. Cochez ce flag si votre son n'est pas un son d'ambiance que l'on entend pendant toute la partie.

Pour finir, j'ajouterai qu'il ne faut pas trop abuser des effets sonores en multijoueur parce que le son sera alors rechargé par tous les PC toutes les X minutes, ce qui risque de se faire ressentir sur les petites configs.

Modifier l'environnement sonore

Entité concernée : env_sound
Type d'entité : entité-point
Difficulté : assez facile

Contrairement à ce que tout le monde est tenté de croire, cette entité ne produit aucun son. En fait, elle modifie la façon dont tous les sons d'une certaine zone seront émis.

Késako ? Prenons un exemple concret : la map cs_siege de Counter-Strike. Les anti-terroristes commencent dans une zone très large ou on entend un écho important. Eh bien, c'est un *env_sound* qui crée cet écho.
Vous remarquerez d'ailleurs que TOUS les sons sont modifiés : tirs de balles, commandes de la radio, bruits de pas etc...

Placez un *env_sound* à l'endroit où vous voulez que les sons soient modifiés. Il n'y a aucun flag, vous n'avez que 2 attributs à renseigner :

- Radius : rayon en unités de Worldcraft dans lequel les joueurs verront tous leurs sons modifiés.
- Room Type : type de pièce. Indiquez ici quelle est la modification à apporter.
  Vous avez un large choix : caverne, tunnel, métal, sous l'eau etc...

Important à savoir : lorsqu'un joueur quitte le rayon d'action d'un *env_sound*, la façon dont les sons sont émis ne revient pas à sa valeur initiale ! Il faudra remettre un *env_sound* plus loin avec Room Type = "Normal (off)", pour que tout rentre dans l'ordre 

Musique du CD

Entité concernée : trigger_cdaudio (Half-Life uniquement)
Type d'entité : entité-bloc
Difficulté : très facile

Et dire que j'ai fait toute la campagne d'Half-Life sans le CD ! Je ne savais pas que Gordon écoutait de la musique lorsqu'il réglait ses comptes avec les aliens, mais apparemment il existe vraiment une plage audio sur le CD d'Half-Life !
Oui oui je vous rassure, je n'ai pas une version crackée de Half-Life 

Bon, plus sérieusement : comme tous les trigger, cette entité est une entité-bloc, invisible lorsqu'on joue. Commencez donc par créer un bloc avec la texture AAATRIGGER puis cliquez sur "ToEntity" comme vous savez si bien le faire mainteneant 

Choisissez l'entité *trigger_cdaudio* (disponible uniquement pour une map Half-Life). Cette entité possède 2 attributs :

- Name : nom de l'entité. Inutile dans ce cas, parce qu'on n'a pas besoin de l'appeler. En effet, dès que le joueur traversera le trigger, la musique sera lancée.
- Track # : plage du CD Audio à jouer. Vous avez le choix entre 30 plages, je vous conseille de les écouter pour savoir laquelle choisir.
  Pour que la musique s'arrête, il faudra que le joueur retraverse un *trigger_cdaudio* avec
  Track # = "Stop"

Si le joueur n'a pas de CD dans son lecteur, le jeu ne plantera pas (heureusement !).
Je vous conseille de vous servir de cette fonction uniquement pour une campagne solo de Half-Life.

Notez qu'il existe une entité-point qui fait exactement pareil : *target_cdaudio*. Son fonctionnement est le même, mais je préfère me servir d'une entité-bloc dans ce cas plutôt que d'une entité-point. Je pense que vous aussi ;o)

Les hauts-parleurs

Entité concernée : speaker (Half-Life uniquement)

Type d'entité : entité-point
Difficulté : facile

Qu'est-ce qu'un haut-parleur ?

C'est encore une entité qui n'existe que dans la campagne solo de Half-Life. Souvenez-vous. Vous entendiez de temps en temps des hauts-parleurs qui prononçaient des phrases gaies et rassurantes du type : "Ordre à toutes les unités d'abattre à vue le dénommé Gordon Freeman".
Eh bien un *speaker*, c'est ça !

Cette entité produit normalement un son aléatoire toutes les X minutes. Voici ses attributs :

- Name : nom de l'entité si vous voulez l'appeler.
- Announcement Presets : si vous désirez choisir vous-même le son à jouer. Sinon, laissez la valeur "None" et le speaker jouera un son au hasard.
- Sentence Group Name : pour ceux qui désirent afficher des phrases sur l'écran du joueur (il faut se baser sur le fichier sentences.txt). En temps normal vous ne vous servirez pas de cette fonction.
- Volume (10 = loudest) : volume sonore, 10 correspondant au volume le plus élevé.

A noter la présence du flag "Start Silent" qui désactive le haut-parleur jusqu'à ce qu'il soit activé par un trigger.

En bref c'est une entité sympa en mode solo Half-Life, mais sinon son intérêt est vraiment très limité...
Maintenant, vous allez pouvoir mettre du son dans vos maps ! Pensez surtout au son de fond, y'a vraiment rien de tel pour mettre l'ambiance dans votre map (et ça change tout !).

Rotations

Les objets rotatifs sont un peu particuliers dans Half-Life à cause de la texture ORIGIN. Ce chapitre sera l'occasion pour nous de faire le point sur les rotations et de voir 2 entités qui l'utilisent.

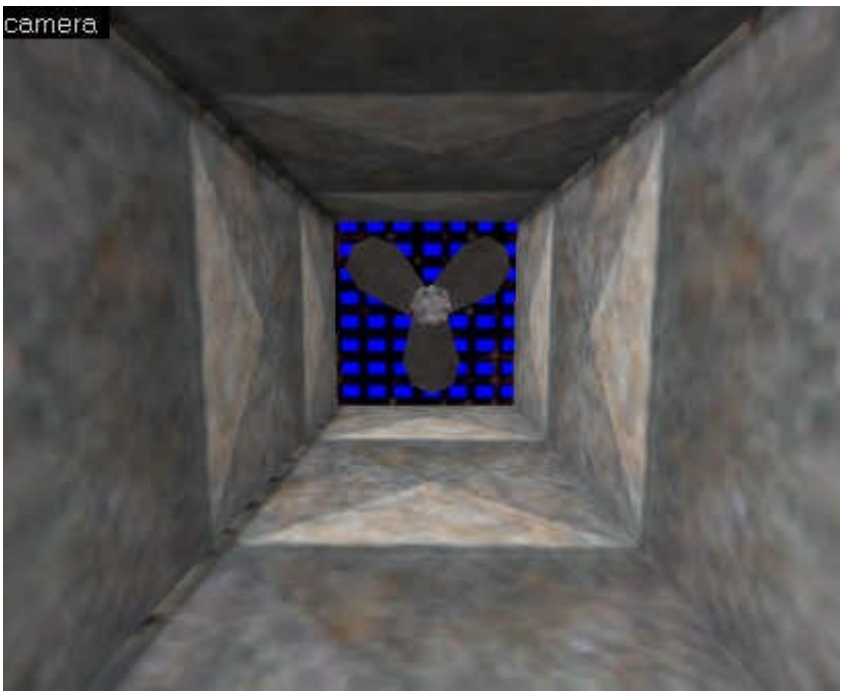
Il existe bien plus d'entités rotatives que cela, mais je n'en parle pas ici. J'ai préféré les garder pour les chapitres qui vont le mieux avec (exemple : l'ascenseur rotatif dans le thème sur les ascenseurs).
Ces 2 entités-là ne rentrent dans aucun thème particulier, voilà donc pourquoi je les ai mises dans ce chapitre.

Objets rotatifs

Entité concernée : func_rotating
Type d'entité : entité-bloc
Difficulté : moyen

Parmi les objets que l'on rend rotatifs, il y a surtout des hélices, que ce soient des hélices de ventilateurs géants ou bien des hélices d'hélicoptère.

Dans un premier temps, il vous faut créer l'objet que vous allez rendre rotatif. Je vais par exemple créer un petit ventilateur :

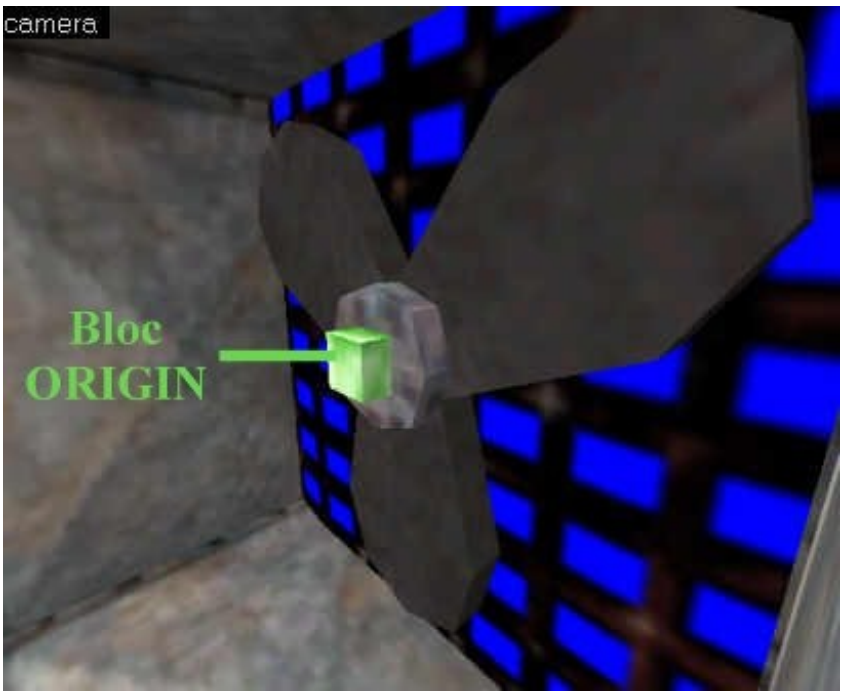


Avec un peu d'expérience du Morphing Tool, vous devriez pouvoir faire pareil que moi (et certainement mieux lol)

Il va maintenant falloir placer un bloc spécial, entièrement recouvert de la texture ORIGIN. Cette texture (représentant un visage vert) est particulière car elle n'est pas visible quand on joue (heureusement), et elle indique au moteur de Half-Life le centre de rotation de l'objet.

Qu'est-ce que le centre de rotation de l'objet ?

C'est l'endroit autour duquel il tourne. En pratique, c'est l'endroit qui se déplace le moins.
Sur mon exemple, les hélices tournent autour d'un centre (au milieu). C'est cet endroit qui est le centre de rotation de l'objet. C'est là qu'il va falloir poser notre bloc ORIGIN :



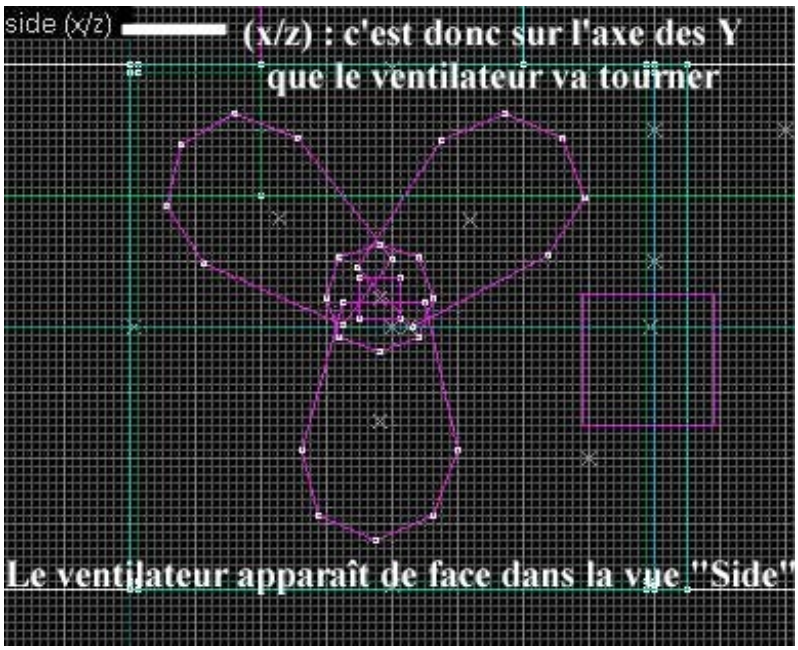
A vous d'adapter ce cas à votre situation, mais vous ne devriez pas avoir trop de mal maintenant que je vous ai montré où j'ai placé mon bloc ORIGIN...

Bien, maintenant sélectionnez votre objet ET le bloc Origin, puis cliquez sur le bouton "ToEntity".
Choisissez l'entité *func_rotating* et éditez maintenant ses propriétés :

- Name : nom de l'entité si celle-ci est déclenchée par un trigger.
- Render FX, Render Mode, FX Amount et FX Color : voir le chapitre sur la transparence pour savoir comment les utiliser.
- Pitch Yaw Roll : c'est l'orientation de l'entité. Vous pouvez la modifier aussi grâce au radar en haut à droite. Normalement, vous ne devriez pas avoir besoin de modifier ces valeurs pour cette entité.
- Light Flags et Light Origin : ces champs n'apparaissent pas toujours, cela dépend du FGD que vous possédez. Ils permettent de modifier les informations de lumières pour cette entité. Vous n'en aurez pas besoin ici.
- Rotation Speed : vitesse de rotation de l'objet. Pour un ventilateur comme le mien, je vous recommande de mettre au moins 500, sinon plus, parce qu'en-dessous on a l'impression que c'est à l'arrêt
- Volume (10 = loudest) : le ventilateur peut produire du son et remplacer ainsi l'entité *ambient_generic*. Mettez ici le volume du son (valeur de 1 à 10). Je vous recommande de mettre 10 pour qu'on entende quelque chose
- Friction (0 - 100%) : ce pourcentage correspond à la vitesse à laquelle l'entité va freiner (le flag "Acc/Dcc" doit être coché).
- Fan Sounds : son que va produire le ventilateur (en anglais : fan = ventilateur). Laissez "No Sound" si vous ne voulez pas que l'entité produise du son.
- WAV Name : nom du fichier WAV à jouer lors de la rotation. Utilisez cet attribut uniquement si les sons de "Fan Sounds" ne vous conviennent pas.
- _minlight : niveau de luminosité minimum. Pour que l'entité soit visible même s'il fait noir ;)) C'est un attribut très rarement utilisé.
- X Y Z - Move here after lighting : je ne me sers pas trop de cet attribut. C'est la position initiale de l'entité (en coordonnées X Y et Z) après que la pièce ait été éclairée. Je n'en vois pas vraiment l'intérêt...
- Damage inflicted when blocked : si le joueur est coincé par une des hélices, alors il perdra le nombre de points de vie indiqués ici.

Voyons voir maintenant les flags de cette entité. Ils sont nombreux :

- Start ON : l'objet est actif dès le début du niveau. Cochez ce flag si vous voulez que l'objet tourne sans avoir été appelé par un trigger auparavant. C'est assez fréquent, alors si votre ventilateur ne tourne pas vous saurez pourquoi ;))
- Reverse Direction : l'objet tournera en sens inverse.
- X Axis : l'objet tournera selon l'axe des X (au lieu de Z par défaut).
- Pour savoir quel axe utiliser, j'ai une astuce. Regardez dans vos vues 2D sous Worldcraft : dans l'une d'entre elles la ventilateur apparaît de face.



En haut à gauche, deux lettres appraissent (ici X et Z). La lettre qui manque (Y) est l'axe sur lequel l'engin va tourner. C'est tordu mais ça marche

- Y axis : l'objet tournera sur l'axe des Y (au lieu de Z).
- Acc / Dcc : l'objet accélèrera et décélèrera doucement. Idéal pour montrer que le ventilateur se met en place ou est en train de s'arrêter.
- Fan Pain : aïe bobo ! Si ce flag est coché, le joueur sera repoussé par le ventilateur s'il s'en approche de trop près. De plus, il va perdre des points de vie très rapidement (voire se faire hacher s'il persiste à vouloir passer ;))
- Not Solid : l'entité n'est pas matérielle. Le joueur peut alors passer à travers comme si elle n'existait pas.
- Small Radius : comme on l'avait vu pour le thème d'entités sur le son, ce flag fait en sorte que le son soit entendu dans un faible rayon.
- Medium Radius : idem pour un rayon moyen.
- Large Radius : idem pour un large rayon.

Voilà, testez votre niveau et vous verrez l'objet s'animer ! Idéal pour mettre un peu de vie dans une map !

Si votre objet ne s'anime pas, vérifiez que vous avez bien coché le flag "Start ON", et que vous avez donné une vitesse à la rotation de l'objet.

Objets pendants

Entité concernée : func_pendulum
Type d'entité : entité-bloc
Difficulté : moyen

Pas évidents à réaliser... et pourtant ça vaut le coup je vous assure ! Les objets qui pendent sont très rares dans les maps, peu de monde les connaît ! Alors créez la surprise en montrant au joueur pour la première fois une fonctionnalité du moteur de Half-Life qu'il ne connaissait pas !

Il va nous falloir un exemple pour que vous compreniez bien le fonctionnement d'une telle entité.
Je veux par exemple faire pendre une corde :

D'autres applications sont possibles : un pendule, un objet qui vient de se détacher du plafond etc...

Bref, je transforme cette corde en entité *func_pendulum*, puis j'édite ses propriétés :

- Global Entity Name : nom global de l'entité si celle-ci est utilisée sur plusieurs niveaux.
- Name : nom de l'entité.
- Render FX, Render Mode, FX Amount et FX Color : pour en savoir plus, lire le chapitre sur la transparence.
- Speed : vitesse de déplacement de l'objet.
- Distance (deg) : angle de rotation en degrés. Cette entité doit tourner en décrivant un certain angle (généralement inférieur à 90°) qu'il faut préciser ici.
- Damping (0-1000) : imaginez que la corde fasse une rotation à une vitesse de 100 tout le temps. Il va y avoir quelque chose qui va clocher : en effet, lorsque la corde a décrit un mouvement, elle doit freiner son mouvement (sinon elle s'arrêterait brusquement et ce serait perturbant car pas naturel).
  En bref, mettez ici une valeur entre 0 et 1000 qui indiquera si l'entité doit plus ou moins freiner son mouvement entre chaque rotation.
- Damage inflicted when blocked : si le joueur est coincé par cette entité, alors il perdra le nombre de points de vie indiqués ici.
- _minlight : niveau de luminosité minimum, pour que l'entité reste visible même dans le noir 
- Pitch Yaw Roll (Y Z X) : orientation de l'entité, que vous pouvez modifier aussi grâce au radar en haut à droite de la fenêtre. Il peut être intéressant de modifier l'orientation pour cette entité, à vous de voir...

Les flags sont les suivants :

- Start ON : l'entité est active dès le lancement du niveau. Sinon, il faudra l'activer avec un trigger.
- Passable : on peut traverser cette entité, elle n'est pas matérielle. Le joueur comme les ennemis passeront donc à travers comme si elle n'existait pas.
- Auto-return : l'entité revient à chaque fois à sa position initiale. Il paraît que cette fonctionnalité est un peu buggée d'après Valve, moi j'ai testé et effectivement il arrive parfois que l'objet s'arrête brusquement.
- X Axis : comme je vous l'ai expliqué plus haut, si vous voulez que l'entité tourne sur l'axe des X au lieu de Z, alors cochez ce flag.
- Y Axis : idem pour l'axe des Y.
- Not in Deathmatch : l'entité n'apparaîtra pas dans une partie multijoueurs.

Il va peut-être vous falloir tester votre map plusieurs fois avant d'obtenir l'effet désiré. Surveillez notamment les attributs Speed, Distance et Damping... ce sont eux qui sont principalement responsables de la rotation de votre objet.
Je reconnais que les rotations sont un peu difficiles à manier au début, mais vous verrez que, petit à petit, vous saurez placer rapidement le bloc ORIGIN. Les problèmes avec le centre de rotation ne seront alors plus qu'un lointain (et mauvais) souvenir.

Environnement dangereux

Pour rajouter un peu de difficulté dans vos niveaux, sachez qu'il existe certaines entités bien sympas qui vont nous aider.
Au programme, nous allons voir comment créer une zone glissante, une zone ventilée, et enfin nous défierons les lois de la gravité !

Rien que ça !

Zone glissante

Entité concernée : func_friction
Type d'entité : entité-bloc
Difficulté : facile

Comment simuler qu'une zone est glissante, comme c'est le cas sur de la glace ou sur un sol bien ciré ?
Il faut modifier l'adhérence du joueur avec le sol !

Pour cela, vous allez devoir créer une entité-bloc *func_friction*.
Avant tout, vous devez noter 2 choses importantes concernant cette entité :

1. Le bloc est visible lorsqu'on joue, appliquez donc une texture correcte et non pas AAATRIGGER.
2. Lorsque le joueur est entré en contact avec le *func_friction*, l'adhérence est modifiée pour toujours ! Concrètement, cela veut dire que si le joueur sort de la zone glissante, il glissera toujours ! Pour remédier à cela, il vous faudra donc "entourer" la zone glissante d'autres *func_friction* qui rétabliront les valeurs initiales de la friction.

Une fois que votre bloc est correctement créé, que vous l'avez transformé en entité *func_friction*, il vous restera à modifier ses propriétés. Sur ce point-là je vous rassure, cette entité est très simple :

- Render FX, Render Mode, FX Amount et FX Color : voir le chapitre sur la transparence pour plus d'informations.
- Percentage of standard (0 - 100) : cette valeur définit l'adhérence du terrain. Mettez une valeur entre 0 et 100, 15 étant la valeur par défaut. Plus cette valeur est élevée, plus le joueur "glissera".

On ne fait pas plus simple

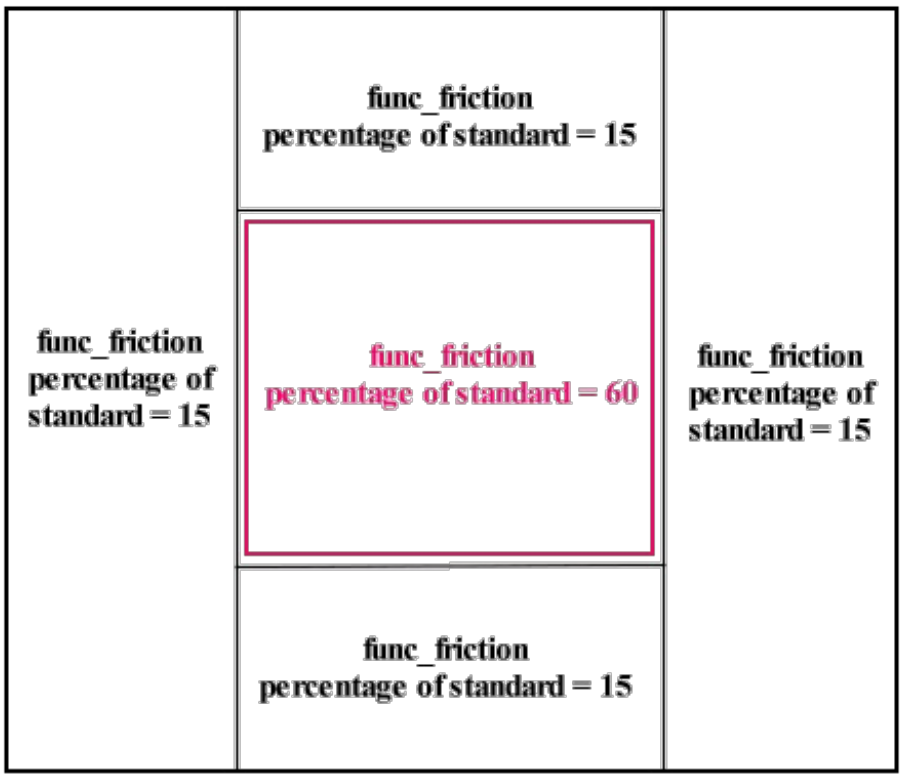
Cette entité possède un flag vraiment dépourvu d'intérêt et que vous connaissez bien : "Not in deathmatch". Je rappelle à ceux qui l'auraient oublié que ce flag supprime l'entité dans une partie multijoueurs.

Voilà, vous allez voir maintenant le joueur glisser, ce qui est un facteur important pour une partie solo, et puis en multijoueurs c'est tellement fun

Oui, mais après que le joueur ait traversé le *func_friction*, il garde la même adhérence ?!

Relisez ce que je vous ai dit plus haut
Il vous faudra "entourer" le func_friction d'autres blocs *func_friction* avec les valeurs par défaut, c'est à dire "Percentage of standard = 15".

Il vous faut un schéma ? Le voici :



J'espère que ce schéma vous aura éclairci les idées

Zone ventilée

Entité concernée : trigger_push
Type d'entité : entité-bloc
Difficulté : assez facile

Vous connaissez certainement les zones ventilées : dans Half-Life, Gordon se retrouve plusieurs fois pris au piège dans des ventilateurs géants qui le poussent avec une force phénoménale.

Eh bien, les *trigger_push* vont nous permettre de faire cela ! Concrètement, ils "poussent" le joueur dans une direction précise.

Cette entité est invisible, donc vous pouvez lui appliquer la texture AAATRIGGER. L'entité doit couvrir toute la zone où le vent doit pousser le joueur.

Après, ouvrez la fenêtre d'édition des propriétés comme vous en avez l'habitude. Voici les différents attributs disponibles :

- Target : cible à déclencher lorsque l'entité est activée.
- Name : nom de l'entité (pour pouvoir l'activer par un trigger).
- Kill target : nom d'une entité à retirer du jeu.
- Target Path : chemin vers la cible.
- Master : utilisé pour un multisource ou un game_team_master.
- Sound style : son à produire lors de l'activation (vous avez le choix entre "No sound" et "No sound" lol ;).
- Delay before trigger : temps en secondes avant de déclencher la cible.
- Message (set sound too!) : message à afficher à l'écran du joueur (basé sur le fichier sentences.txt et titles.txt). Rarement utilisé dans ce cas.
- Speed of push : voilà enfin l'entité la plus intéressante. C'est la vitesse à laquelle le joueur est poussé, en unité de Worldcraft par seconde.

A noter la présence de 3 flags :

- Once only : le joueur ne sera poussé qu'une fois.
- Start off : l'entité est inactive au début du niveau. Il faudra donc l'activer par un trigger pour qu'elle se mette en marche.
- Pushables : les entités *func_pushables* sont elles aussi affectées par le vent.

Comme en général le vent provient d'une hélice, pensez à créer un *func_rotating* comme vous l'avez appris... Au moins, le joueur

comprendra pourquoi il est poussé (sinon il va rien capter à votre map lol)

Modifier la gravité

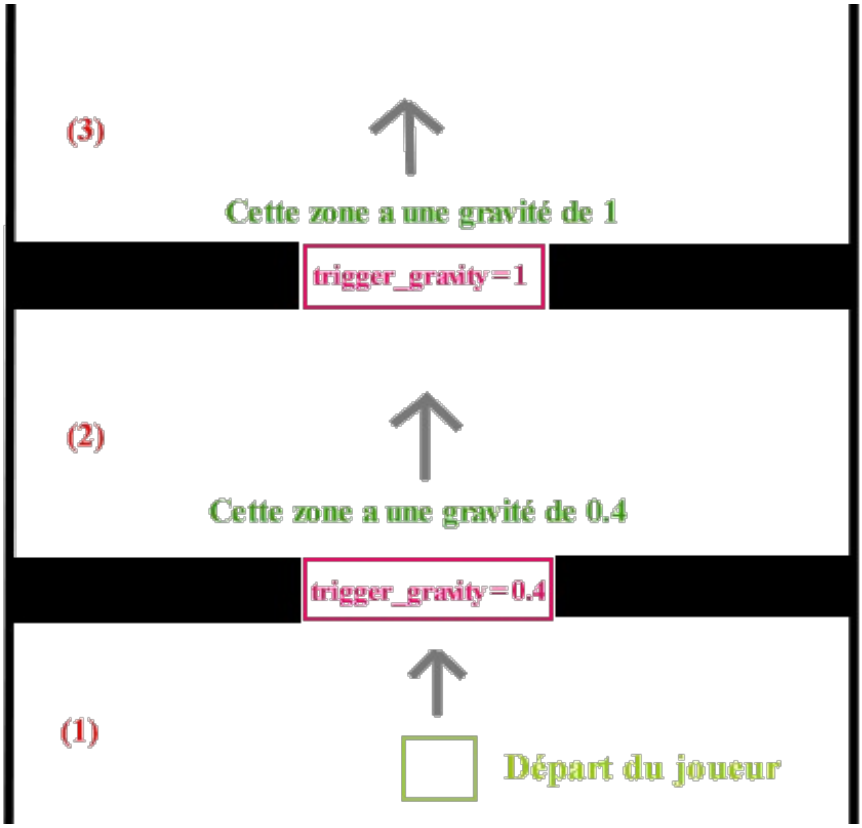
Entité concernée : trigger_gravity
Type d'entité : entité-bloc
Difficulté : facile

Ca peut être marrant parfois de modifier la gravité dans une map... pour le fun, mais aussi parfois vous en avez vraiment besoin. C'est le cas notamment si vous faites une map dans l'espace ou dans les mondes de Xen.

Vous devez commencer par créer un bloc avec la texture AAATRIGGER (en effet, celui-ci sera invisible).
Cependant, il ne faut pas croire que la gravité ne sera modifiée que dans la zone couverte par le bloc...

Dès que le joueur rentre dans le *trigger_gravity*, sa gravité est modifiée pour toujours, même s'il sort du *trigger_gravity*.
Pour rétablir la gravité normale, vous devrez donc refaire passer le joueur pas un autre *trigger_gravity* avec la valeur Gravity = 1.

C'est assez particulier, alors je vous ai fait un petit schéma explicatif pour que ça soit plus clair :



1. Le joueur est dans une zone normale, et il approche du *trigger_gravity*.
2. Le joueur a traversé le *trigger_gravity* et dans toute la zone (2) il possède une gravité de 0.4.
3. Le joueur repasse dans un *trigger_gravity* qui rétablit la gravité normale. La zone 3 est donc comme la zone (1).

Une fois qu'on a compris le principe, ça coule tout seul

Bon, je parle je parle mais je ne vous ai toujours pas présenté les attributs de cette entité.
Eh bien en fait, les attributs sont strictement les mêmes que ceux que nous avons vu plus haut pour les *trigger_push*. Seule une chose diffère, et c'est la plus importante :

- Gravity (0-1) : valeur de la gravité. 1 correspond à une gravité normale. Mettez une valeur inférieure à 1 pour que la gravité soit plus faible.

Cette entité possède 3 flags :

- Monsters : les monstres sont affectés eux aussi par la gravité.
- No Clients : aucun joueur ne sera affecté par cette entité (alors à quoi bon l'utiliser ? lol, y'a des trucs bizarres dans Worldcraft quand même)
- Pushables : les *func_pushable* peuvent activer le *trigger_gravity*.

Faites donc très attention lorsque vous placez des trigger_gravity dans votre map : si vous les placez mal, le joueur risque d'avoir une gravité trop faible pendant toute la map alors que ce n'était pas prévu !
Avec tout cela, le joueur n'a qu'à bien se tenir !
Si vous faites une partie solo, mettez quelques monstres dans ces zones dangereuses, et vous verrez que le joueur va sacrément peiner lol 

Paranormal

Bigre ! Ai-je bien entendu "paranormal" ? Il existe donc des entités paranormales dans Half-Life ?

En effet En fait, j'ai réuni dans ce chapitre toutes les entités liées aux mondes de Xen, les mondes extra-terrestres que l'on aperçoit à la fin de l'aventure d'Half-Life.
Ces entités sont faciles à utiliser, et elles produisent de beaux effets dans une map !

Autant vous le dire de suite : ces entités ne sont valables que si vous faites une map Half-Life, parce qu'on n'a pas l'habitude de voir ce genre de trucs dans Counter-Strike ou encore Day of defeat !

Particules

Entité concernée : env_funnel
Type d'entité : entité-point
Difficulté : très très facile

Oui, c'est le moins que l'on puisse dire : cette entité est extrêmement simple à utiliser... du coup elle n'est pas très paramétrable, ce qui est assez dommage.

Bon, de quoi s'agit-il ? C'est un effet de particules qui semblent être aspirées en un point. On peut voir cet effet au début de la campagne de Half-Life lorsque Gordon active la machine infernale

Pour ceux qui auraient besoin d'un screenshot, le voici :

Avouez que c'est assez joli L'animation est en effet fort sympathique !

Pour vous en servir, c'est très simple : placez une entité *env_funnel* à l'endroit où vous désirez que les particules "s'engouffrent". Ensuite, donnez un nom à cette entité grâce à son attribut "Name" et faites en sorte qu'elle soit appelée par un trigger.

Il est impératif d'utiliser un trigger pour déclencher l'animation. Celle-ci n'est pas automatiquement active au lancement du niveau !

Et voilou, vous aurez une superbe animation dans votre map !

A noter que cette entité possède un seul flag, assez sympa :

- Reverse : l'animation sera exécutée à l'envers. Les particules seront donc éparpillées (c'est très joli aussi

Je regrette simplement qu'on ne puisse pas changer la couleur des particules, enfin c'est comme ça...

Téléportation

Entités concernées : trigger_teleport et info_teleport_destination
Type d'entités : respectivement entité-bloc et entité-point
Difficulté : facile

Bien pratique parfois : la téléportation ! Au moins, le joueur peut se déplacer en un éclair dans votre map !

Son fonctionnement est en fait assez simple. Il y a 2 entités :

- trigger_teleport : départ du téléporteur.
- info_teleport_destination : destination du téléporteur.

Etape 1 : Le *trigger_teleport* est une entité-bloc invisible. Vous devez donc créer un bloc avec la texture AAATRIGGER, que vous transformerez en *trigger_teleport*.
Lorsque le joueur entrera dans la zone du trigger, il sera téléporté.

Vous n'aurez à remplir qu'une seule propriété du *trigger_teleport* :

- Target : nom de l'*info_teleport_destination* vers lequel le joueur sera téléporté.

Les flags "Monsters" et "Pushables" permettent respectivement de faire en sorte que les monstres et les entités poussables soient eux aussi téléportés s'ils rentrent dans la zone. Quant à "No Clients", vous ne vous en servirez jamais, car elle empêche l'entité de fonctionner

Etape 2 : Créez maintenant une entité-point *info_teleport_destination* là où vous voulez que le joueur atterrisse. Placez l'entité au niveau du sol, et légèrement surélevée pour que le joueur ne soit pas bloqué.
Là encore, vous ne devez éditer qu'une propriété :

- Name : nom de l'entité, qui doit correspondre au champ "Target" du *trigger_teleport*.

Comme vous pouvez le voir, c'est vraiment simple. On peut créer une téléportation en quelques secondes !

A noter que le téléporteur est ici invisible. Pour que le joueur voit qu'il a affaire à un téléporteur, il vous faudra faire comme dans Half-Life : placer un sprite (comme nous apprendrons à le faire) qui représente une sorte de porte de téléportation.

Ejection du joueur

Entité concernée : trigger_monsterjump (Half-Life uniquement)
Type d'entité : entité-bloc
Difficulté : assez facile

Cette entité propulse le joueur selon une vitesse, une direction et une hauteur que vous définissez. Dans l'aventure de Half-Life, cette entité est utilisée dans les mondes de Xen par exemple, lorsque le joueur subit une sorte de soufflerie au-dessus d'un nid d'extraterrestres
Mais vous pouvez trouver de nombreux autres usages à cette entité : trampoline, ventilation etc etc...

Vous devez créer un bloc avec la texture AAATRIGGER. C'est la zone dans laquelle le joueur subira la propulsion. Bien entendu, ce bloc sera invisible lorsque vous jouerez...

Trasnformez le bloc en entité *trigger_monsterjump* puis éditez ses propriétés :

- Angle : servez-vous du radar en haut à droite pour définir dans quelle direction le joueur sera propulsé.
- Master : ne sert qu'avec un *game_team_master*, dans le cas d'une partie Half-Life Teamplay. Vous ne devriez donc pas vous en servir
- Jump Speed : vitesse à laquelle le joueur est propulsé.
- Jump Height : hauteur à laquelle le joueur est propulsé.

Aucun flag pour cette entité, donc pas de prise de tête supplémentaire
Comme vous pouvez le voir, cette entité est relativement simple à utiliser et peut se révéler très pratique !

Xen & Compagnie

Entités concernées : toutes les entités commençant par "xen_" (Half-Life uniquement)
Type d'entités : entités-point
Difficulté : très facile

Il existe 6 entités commençant par "xen_". Je les ai toutes regroupées ici car leur fonctionnement et leur utilité restent les mêmes : ce sont des entités décoratives.

Bien entendu, ces entités n'existent que sous Half-Life... et vous ne devriez les mettre que dans les mondes extra-terrestres (Xen).

Pour placer une de ces entités, activez le "Entity Tool"  , et sélectionnez l'entité Xen de votre choix. Pour placer l'entité dans votre map, je vous recommande de vous servir de la vue 3D : cliquez au niveau du sol là où vous voulez placer l'entité.

Ces entités doivent être collées au sol. Cela se fait automatiquement si vous placez l'entité comme je vous ai dit : en cliquant dans la vue 3D sur le sol.

Ces entités possèdent des propriétés que vous connaissez maintenant tous : nom, cible, attributs de transparence etc... Vous n'aurez pas à vous en servir, donc n'y touchez pas

Reste maintenant à vous présenter ces 6 entités, pour que vous puissiez choisir celles que vous allez mettre dans votre map :

	**xen_hair** C'est un brin d'herbe. Il est assez petit, sombre, et il bouge comme s'il y avait du vent... ou un esprit qui le dirige ?! Brrr...Quoiqu'il en soit, vous devriez placer plusieurs entités de ce type à la fois. Et ne vous inquiétez pas : l'herbe ne mord pas On passe à travers sans se blesser.
	**xen_plantlight** Une gentile petite plante lumineuse. Oui, elle produit de la lumière !Seulement voilà, elle est un peu peureuse : dès qu'on s'approche d'elle, elle se rétracte et la lumière s'éteint.
	**xen_spore_small** Une petite plante pas méchante... celle-ci est suffisamment petite pour qu'on puisse sauter dessus (pas évident quand même) et accéder à des endroits cachés de vos niveaux
	**xen_spore_medium** C'est la même entité, mais un peu plus grande... Bah ouais, y'en a de toutes tailles !
	**xen_spore_large** Oulah, celle-là par contre elle est immense ! Pour avoir une idée de sa taille, vous n'avez qu'à en placer une sous Worldcraft : la taille du cube qui représente l'entité vous donne de précieuses indications !
	**xen_tree** Un arbre, oui, mais sans feuilles... Il a d'ailleurs l'air menaçant, et c'est bien vrai : il fait mal ! Faut éviter de s'approcher de lui, ou il risque de vous écraser comme une chaussette (25 points de vie de dégâts).

J'espère que vous trouverez ce petit tableau pratique... en tout cas, grâce à lui vous n'aurez pas à tester votre map pour savoir à quoi ressemblent ces entités

Ces entités sont en fait des "models" car elles sont actives (on en reparlera). Je tiens juste à vous mettre en garde : mettre des models contribue à augmenter les "epolys" (voir chapitre sur les r_speeds), ce qui risque de faire ramer votre map ! N'abusez donc pas trop de ces entités !

Voilà, vous savez tout maintenant sur les entités extra-terrestres ! Vous allez pouvoir créer des mondes de Xen impressionnants !

Blesser et soigner

C'est un chapitre un peu sadomaso, je reconnais
Nous allons voir comment on peut torturer le joueur dans une map HL (dans une map CS ce genre de manipulations est possible mais pas conseillé).

Nous verrons dans un premier temps comment blesser le joueur, puis comment soigner ses petits bobos. Enfin, pour ajouter une touche de réalisme à la scène, nous verrons comment générer du sang dans votre map !

Blesser

Entité concernée : trigger_hurt
Type d'entité : entité-bloc
Difficulté : assez facile

Cette entité est une entité-bloc invisible, c'est-à-dire que vous allez devoir créer un bloc avec la texture AAATRIGGER. Dans toute la zone couverte par ce bloc, le joueur subira des dégâts. Placez donc le bloc à l'endroit désiré et transformez-le en entité *trigger_hurt*.

Il n'y a rien de bien compliqué en ce qui concerne ses propriétés. Les 2 plus importantes à retenir sont "Damage" et "Damage Type".

Voici la liste des propriétés disponibles :

- Name : nom de l'entité. Utile si vous souhaitez appeler cette entité afin qu'elle soit déclenchée, et non pas active dès le début de la map.
- Target : nom d'une entité à appeler.
- Master : indiquez le nom d'un *multisource* ou *game_team_master*. On se sert de cet attribut pour infliger des dégâts uniquement à une équipe précise, mais c'est un peu compliqué à utiliser...
- Damage : dégâts infligés au joueur par seconde. Si vous mettez "5", le joueur perdra 5 points de vie par seconde tant qu'il sera dans la zone du *trigger_hurt*.
- Delay before trigger : délai en secondes avant de déclencher la cible définie dans "Target".
- Damage Type : c'est là qu'on a l'embarras du choix ! En effet, vous devez sélectionner ici le type de dégâts infligés au joueur : sa réaction ne sera pas la même s'il est paralysé, brûlé, gelé etc etc...

  J'ai testé tous ces types de dégâts, et je peux vous dire que la plupart n'ont pas vraiment d'intérêt : ils se ressemblent tous. Je ne vous décrirai donc que ceux qui sont vraiment intéressants :
  - GENERIC : ça c'est le bobo de base. Vraiment rien de spécial à dire : ça fait mal et pis c'est tout. Retenez que "GENERIC" est employé la plupart du temps pour définir quelque chose par défaut.
  - CRUSH
  - BULLET
  - SLASH
  - BURN : brûle le joueur. S'il sort de la zone, il continuera à cramer un petit moment eh eh ;)Voici l'icône qui apparaît en bas à droite de l'écran lorsque le joueur brûle.
  - FREEZE : gêle le joueur. Là il prend froid et se trouve un peu ralenti.Cette icône apparaît lorsque le joueur prend froid...
  - FALL : à utiliser lorsque le joueur tombe de très haut !
  - BLAST
  - CLUB
  - SHOCK : bzzz bzzz... Si le joueur est censé prendre une décharge de 100 000 volts dans les dents, c'est SHOCK qu'il faut utiliser.Un éclair représente la foudre qui s'abat impitoyablement sur Freeman
  - SONIC
  - ENERGYBEAM
  - DROWN : si le joueur vient à manquer d'air, utilisez DROWN. A noter que ce dégât s'effectue automatiquement si le joueur reste trop longtemps sous l'eau.O2 : ça évoque ce qu'il manque au joueur... De l'air !!!
  - PARALYSE : sert à paralyser un peu le joueur.
  - NERVEGAS : le joueur respire un gaz dangereux qui lui fait perdre de la vie.Malheureusement, il n'existe pas de masque à oxygène dans Half-Life pour se protéger de ça... Il va donc falloir sortir vite fait de la zone de gaz si le joueur ne veut pas y laisser ses os !
  - POISON : ben c'est du poison Cette fois, si le joueur sort de la zone du trigger_hurt, il continuera quand même à subir des dégâts pendant un certain temps.C'est aussi le symbole utilisé pour dire que quelque chose est dangereux (hazardous).
  - RADIATION : immanquable. Dans tout jeu qui se respecte, il y a forcément des radiations issues de dépôts nucléaires, c'est un minimum Le joueur entend un crissement (krss krss) de plus en plus persistant lorsqu'il s'approche de la zone contaminée.Si vous ne connaissez pas ce symbole, c'est que vous n'avez jamais joué à Half-Life !

- DROWNRECOVER
- CHEMICAL : si le joueur piétine (ou nage) dans des substances chimiques et nocives, c'est ce dégât qu'il faut appliquer. A noter que l'on retrouve le même effet si on met Contents=Slime à un func_water.
  Mais pourquoi laisse-t-on traîner des substances chimiques dans les maps de Half-Life !?? Il le font exprès ou quoi ??? 
- SLOWBURN : le comble du sadomasochisme. Là, vous brûlez le joueur à petit feu, histoire qu'il ressente encore plus sa douleur...
- SLOWFREEZE : idem pour geler lentement le joueur... Perso, je préfère le brûler à petit feu, mais bon chacun ses goûts lol 

Faites votre choix ! Vous voyez maintenant que Half-Life a tout prévu pour faire varier les plaisirs au joueur... enfin, si on peut appeler ça plaisir !

Bon, c'est pas encore fini, il nous faut voir les Flags de cette entité :

- Target Once : intéressant et pratique. L'entité ne blessera qu'une seule fois, ce qui évite que le joueur perde toute sa vie...
- Start Off : l'entité ne fait pas mal au joueur avant d'être déclenchée. En clair, au début de la map elle n'est pas active. On l'utilisera parfois de paire avec "Target Once".
- No clients : n'affectera aucun joueur.
- FireClientOnly : la cible de "Target" ne sera appelée que lorsqu'un joueur aura été blessé.
- TouchClientOnly : seuls les joueurs seront blessés (pas les monstres donc).

Voilà maintenant vous savez tout ce qu'il faut pour gratiner le joueur !!!

Soigner

Entités concernées : item_healthkit, env_beverage, item_battery / func_healthcharger, func_recharge
Type d'entités : entités-point / entités-bloc
Difficulté : assez facile

Dans Half-Life, on ne fait pas que se blesser... heureusement ! On peut aussi soigner ses blessures, ou réparer sa combinaison de protection 

Nous verrons d'abord comment ça fonctionne avec des entités-point, puis avec des entités-bloc.

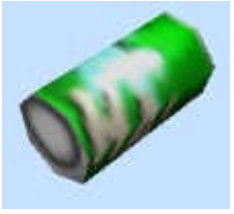
Le fonctionnement est en effet totalement différent : les entités-point se ramassent et donnent instantanément quelques points de vie, tandis que les entités-bloc s'utilisent et remontent lentement la vie (mais on peut gagner plus de vie grâce à elles).

Trousse de secours

La trousse de secours se trouve au sol. Lorsque le joueur passe par là et la ramasse, il reprend instantanément 20 points de vie. Pratique, rapide et efficace 

Le fonctionnement de cette entité-point est très simple : il vous suffit de poser un *item_healthkit* où vous le désirez dans votre map. Il est inutile d'éditer ses propriétés car elles sont inintéressantes.
Placer une telle entité ne prend donc que quelques secondes 

La canette



Vous ne le saviez pas ? Une canette redonne 1 point de vie ! C'est pratique et sympa de se servir au distributeur de boissons pour remonter sa vie 

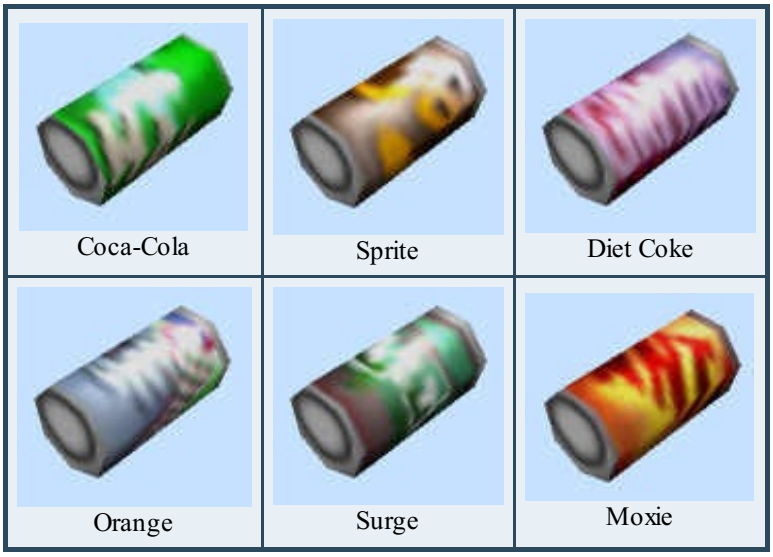
Vous devez placer une entité-point env_beverage, généralement à côté d'un distributeur de boissons. Voilà comment ça fonctionne la plupart du temps :

J'ai placé un *env_beverage* là où les canettes sortent normalement. Le distributeur est un bouton qui, lorsqu'il est utilisé, appelle une canette.
Je vous recommande de cocher le flag "Touch Activates" du *func_button* : ainsi une canette sortira automatiquement lorsque le joueur s'approchera du distributeur.

Voici les 3 attributs disponibles pour cette entité :

- Name : nom de l'entité. Il est important de lui donner un nom pour qu'on puisse faire apparaître une canette lorsqu'on le désire.
- Capacity : nombre de canettes pouvant être générées. Par exemple, si vous mettez "20", le distributeur pourra sortir 20 canettes (20 points de vie), et après il sera hors-service.

  Ce nombre est très important, car si vous ne précisez rien le joueur pourra remonter sa vie tant qu'il voudra jusqu'à 100% ! Vous ne voudriez quand même pas que ce soit aussi facile pour lui non ?
- Beverage Type : hi hi... Bon ok, ça sert à rien, c'est con, mais c'est amusant Vous pouvez définir ici le type de boisson qui va sortir du distrib' (choisissez votre marque préférée !).



A noter que "Random" fait apparaître une canette différente à chaque fois au hasard.

La batterie

Elle recharge immédiatement votre armure de 15%. N'hésitez pas à en poser 2 ou 3, parce que 15% c'est peu...

Le fonctionnement reste strictement le même que pour la trousse de secours. Cette fois, c'est l'entité *item_battery* qu'il faut poser au sol. C'est tout !

Borne de secours

Ceci est une borne de secours. Le joueur doit s'approcher d'elle et laisser appuyée la touche "Utiliser" jusqu'à ce que la borne soit épuisée. Cette borne peut redonner au maximum 50 points de vie.

Il faut ici créer un bloc avec la texture +0MEDKIT, qui indique que la borne est en service. Lorsqu'elle sera épuisée, la borne aura automatiquement la texture +AMEDKIT (rappelez-vous le cours sur les textures !).

Transformez ensuite le bloc en entité-bloc *func_healthcharger*. C'est tout, il n'y a pas de propriétés intéressantes à éditer...

Ne vous prenez pas la tête à créer ce bloc : Worldcraft vous livre un prefab tout prêt à être inséré dans votre map ! Celui-ci se trouve dans la catégorie "Usable Objects" et s'appelle "medkit". C'est déjà une entité *func_healthcharger*, donc tout est prêt vous n'avez rien à toucher :o)

Borne de recharge

Voilà une borne de recharge pour votre armure.
Elle peut remonter de 30% votre armure.

La texture à employer ici est +0RECHARGE. De même, celle-ci se transformera en +ARECHARGE lorsqu'elle sera vide.

Il faut cette fois transformer le bloc en entité-bloc de type *func_recharge*, et vous avez tout bon !

Cette fois encore, il existe un prefab tout prêt livré avec Worldcraft. Il se trouve dans la catégorie "Usable Objects" et s'appelle "recharger".

Du sang !

Entité concernée : env_blood
Type d'entité : entité-point
Difficulté : assez facile

Cette entité rentre parfaitement dans le thème des blessures. Et il faut dire que c'est assez sympa : elle permet de générer du sang dans votre map !

On l'utilisera par exemple dans une map solo, pour le scénario (si votre ami est en train de se faire massacrer par un alien par exemple). Mais on peut aussi générer ce sang à partir du joueur (lorsqu'il passe dans un *trigger_hurt* !).

Placez une entité-point *env_blood* à l'endroit désiré, et éditez ses propriétés :

- Angle : la direction de l'entité est ici très importante. Servez-vous du radar en haut à droite pour indiquer dans quelle

direction le sang doit partir.
- Name : nom de l'entité. Important car cette entité est normalement déclenchée par un trigger...
- Blood Color : couleur du sang. Bon je sais qu'on est censé avoir le sang rouge, mais n'oubliez pas que dans Half-Life il y a aussi des aliens !
  - Red (Human) : sang rouge, pour un humain.
  - Yellow (Alien) : sang jaune, pour un alien.
- Amount of blood (damage to simulate) : quantité de sang. Plus la valeur est élevée, plus il y aura de sang (n'en faites pas trop sinon ça va devenir gore )

Les flags sont aussi importants que les attributs, alors ne les oubliez pas !

- Random Direction : le sang partira dans une direction aléatoire. Dans ce cas, la valeur de l'"angle" est ignorée.
- Blood Stream : envoit des particules de sang (c'est pas mal).
- On Player : le sang proviendra du joueur et non pas de l'entité. Dans ce cas, la position de l'entité n'a plus aucun intérêt, puisque le sang vient du joueur...
- Spray Decals : idéal et conseillé, ce flag permet de compléter l'effet en appliquant un decal de sang sur le mur. On s'y croirait presque lol 


Ne passez pas à côté des flags de cette entité : ils sont très très importants !

Bien, maintenant vous pourrez générer du sang où vous semble comme bon vous semble... Y'a des entités sympas quand même dans Half-Life, vous trouvez pas ? 

Avec tout ça, vous allez pouvoir jongler avec la vie du joueur : un coup vous le blessez, un coup vous le soignez etc etc... Mmh, mais ne le prenez pas pour un Tamagoshi, parce qu'il risque de pas apprécier de voir qu'on joue impunément avec sa vie 

Moyens de transport

Si vous faites une grande map, vous aurez peut-être envie d'offrir un moyen de transport au joueur (c'est tellement pratique !). Bon, le problème c'est qu'il n'y a pas 36 véhicules différents disponibles dans Half-Life. Voilà ceux que j'ai répertoriés :

- Les trains
- Les voitures (uniquement sous Counter-Strike)
- Les tapis roulants

Voilà, c'est peu mais vous verrez qu'avec ça vous pouvez faire pas mal de choses, étant donné que c'est à vous de modéliser votre engin (camion, voiture de course, petit train etc etc...). Laissez libre cours à votre imagination !

Bon, par contre ces entités sont un peu difficiles à utiliser, mais si vous suivez bien ce que je vous dis dans l'ordre, vous ne devriez sentir aucune douleur

Les trains

Entités concernées : func_tracktrain, func_traincontrols, path_track
Type d'entité : toutes des entités-bloc, sauf le path_track (entité-point)
Difficulté : plutôt difficile

Les trains sont un peu difficiles à configurer, en effet. En fait, il existe plusieurs sortes de trains. Nous verrons ici le plus standard de tous. Pour les autres trains, je vous en parlerai brièvement mais sachez déjà que le fonctionnement est quasiment identique.

Le mot "train" ne doit pas vous rebuter. Ok c'est un train comme celui que l'on rencontre dans Half-Life, mais vous n'êtes pas obligés de faire le même véhicule ! Vous pouvez créer un véhicule avec la forme que vous voulez, aussi complexe que vous le désirez !

La particularité du train, c'est qu'il doit suivre un chemin bien précis (des rails par exemple). Le joueur ne peut donc pas aller où il veut dans la map.
Si vous faites une map pour Counter-Strike, nous verrons qu'il est possible de créer une voiture "libre" que l'on peut conduire où l'on veut !

En fait, la première chose à faire serait de créer le train avec des blocs. Commencez donc par ça ! A vous de voir la forme que vous voulez lui donner.
En ce qui me concerne, je vais en faire un assez simplet comme dans Half-Life, mais libre à vous de faire ce que vous voulez !

Voilà ce que ça donne en 2 minutes (c'est la chose la plus immonde que j'aie jamais faite) :

Horrible non ?
Bon peu importe, c'est pour l'exemple.

Ceci constitue donc la charpente de votre train. Ne faites pas comme moi : appliquez-vous, pour que ça ait plus l'air d'un train que d'un Lego®

Et si je veux mettre une autre entité qui se déplace en même temps que le train, comme une mitraillette ?

Hélas, ne cherchez pas à le faire, c'est impossible ! Half-Life ne permet pas de déplacer d'autres entités en même temps que le train.
Ceux qui font du mapping pour Day of Defeat ont par contre la possibilité de le faire, grâce au système "Spirit" inclus dans Dod (j'en reparlerai dans les entités de Dod).

Bon, que faire ensuite ? Il faut créer le bloc ORIGIN de votre train. Eh oui, ces fameux blocs ORIGIN vous poursuivent jusqu'ici !
Je vous rappelle pour info qu'ils permettent à Half-Life de connaître le centre de gravité de votre objet.
Ici, il faudra placer le bloc ORIGIN le plus au centre possible de votre train, comme ceci :

Sélectionnez ensuite TOUS les blocs que vous avez utilisés pour le train, dont le bloc ORIGIN. Cliquez sur "toEntity" et choisissez l'entité *func_tracktrain*.

Voici les propriétés intéressantes à retenir :

- Name : ultra-important, c'est le nom de votre train (ex : "montrain"). Ce nom permettra à Half-Life d'identifier votre train.
- Global Entity Name : si votre train doit fonctionner sur plusieurs maps (pour une aventure solo), donnez ici un nom à votre train. Ce nom doit être unique dans toutes les maps.
- First stop target : nom du premier arrêt (c'est le nom d'une entité path_track). C'est à partir de là que ça se corse, je vous expliquerai le fonctionnement plus bas. Pour l'instant, mettez "path1".
- Sound : son produit par le train en déplacement.
- Distance between the wheels : distance en unités de Worldcraft séparant les roues avant et arrière de votre train. Half-Life saura ainsi faire tourner le train.
  Ne vous prenez pas la tête et mettez la longueur de votre train : ça marchera sans problème !

  Pour connaître la longueur de votre train, sélectionnez-le et lisez ce qui est écrit en bas de l'écran, dans la barre d'informations.
  Exemple : "176w 256l 82h". Le train a donc une longueur de 256 unités !
- Height above track : distance en hauteur séparant le centre du train (son bloc ORIGIN) et les rails. Si vous avez bien placé votre train sur les rails, alors mettez une valeur de 0.
- Initial speed : vitesse initiale du train (en unités de Worldcraft par seconde). Si vous mettez une valeur supérieure à 0, alors on aura l'impression que le train accélère petit à petit, jusqu'à sa vitesse maximale.
- Speed (units per second) : vitesse du train, en unités de Worldcraft par seconde.
- Damage on crush : si quelqu'un se fait écraser par le train, indiquez ici le nombre de points de vie qu'il perdra (0 = pas de bobo).
- Volume (10 = loudest) : volume du son. Mettez une valeur entre 0 et 10, sachant que 10 est la valeur la plus élevée.
- Bank angle on turns : angle de rotation du train lors d'un virage. Les virages ne sont pas très difficiles à mettre en place, et souvent si on laisse la valeur de 0 ça marche très bien !

Cette entité possède 3 flags :

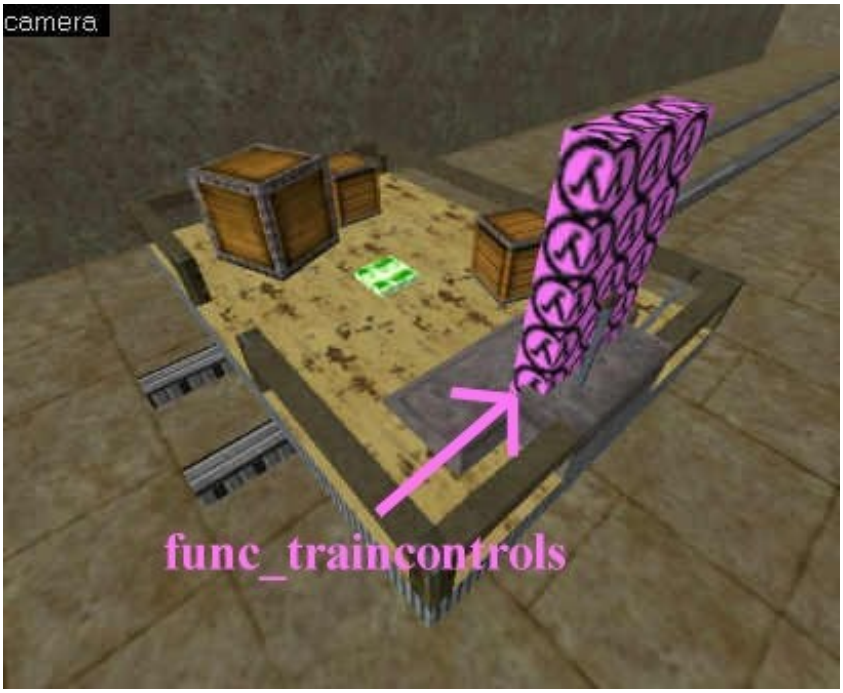
- No Pitch (X-rot) : il n'y aura pas de tangage du train (ne cochez pas, ça fait plus réaliste comme ça
- No user control : le joueur ne peut pas prendre le contrôle du train. Dans ce cas, le train est activé tout seul par un trigger.
- Passable : le train traversera les joueurs et les entités. Ce sera un train "fantôme" : on peut le voir mais pas le toucher. Il est rare que l'on se serve de ce flag quand même

Ensuite (car hélas on n'a pas encore fini), il va falloir orienter le train. Parce qu'il y a 90% de chances que votre train soit dans le mauvais sens lorsque vous jouerez sur votre map, même si vous croyez l'avoir mis dans le bon sens !

Pour ce faire, sélectionnez le train et allez dans le menu Tools/Transform (Ctrl + M). Faites faire à votre train une rotation sur l'axe des Z (90 °, 180 ° ou 270 °).
Chaque cas est particulier donc je ne peux pas vous dire de quelle valeur faire tourner votre train. Faites plusieurs essais jusqu'à ce que le train soit dans le bon sens, c'est tout

Ensuite, il va falloir créer un point où l'on pourra contrôler votre train (le faire avancer, reculer etc...). Ce sera le point de commande de votre train.
Créez un bloc AAATRIGGER, que vous transformerez en entité *func_traincontrols*. Placez ce bloc là où le joueur est censé prendre le contrôle du train.
Cette entité possède juste un attribut : "Train Name". Indiquez-y le nom de votre train.

Ce qui doit donner :



Vous remarquerez que le train a changé d'orientation, pour que dans Half-Life il soit correctement positionné.
Les contrôles du train sont situés juste devant les manettes de gaz (logique).

Le parcours du train

Maintenant que le train est en place, il vous reste à définir son trajet.
En effet je vous le rappelle, un train doit suivre un parcours bien précis (comme des rails). Eh bien c'est le moment de s'en occuper !

Pour indiquer le parcours, on place à plusieurs endroits des entités-point nommées *path_track*. Le train traversera chaque *path_track* dans l'ordre que vous aurez défini.

Pour commencez, placez un premier path_track juste devant le bloc ORIGIN, à la même hauteur que celui-ci :

Le *path_track* est en violet, et se trouve à la même hauteur que le bloc ORIGIN.
Je vous rappelle que le champ "First stop target" des propriétés du train doit avoir la valeur "path1".

Il va maintenant falloir éditer les propriétés du path_track :

- Name : nom du path_track. Il est TRES IMPORTANT de nommer ses path_track intelligemment, parce qu'il va y en avoir plusieurs. Je vous recommande par exemple d'appeler le premier "path1", le second "path2", le troisième "path3" etc...
- Next stop target : nom du prochain *path_track* où le train doit aller. Ca lui indique sa prochaine destination.

Les 2 entités ci-dessus suffisent amplement pour faire fonctionner les *path_track*. Voici toutefois la description des autres attributs :

- Fire on pass : nom d'une entité à déclencher lors du passage du train.
- Branch path : point de passage secondaire (si votre train utilise des changements de voies).
- Fire on dead end : nom d'une entité à déclencher si ce path_track est situé à la fin de la voie (par exemple une voix qui dit : "Terminus, tout le monde descend !" 
- New train speed : nouvelle vitesse du train, si vous voulez qu'il accélère ou ralentisse à partir de là.

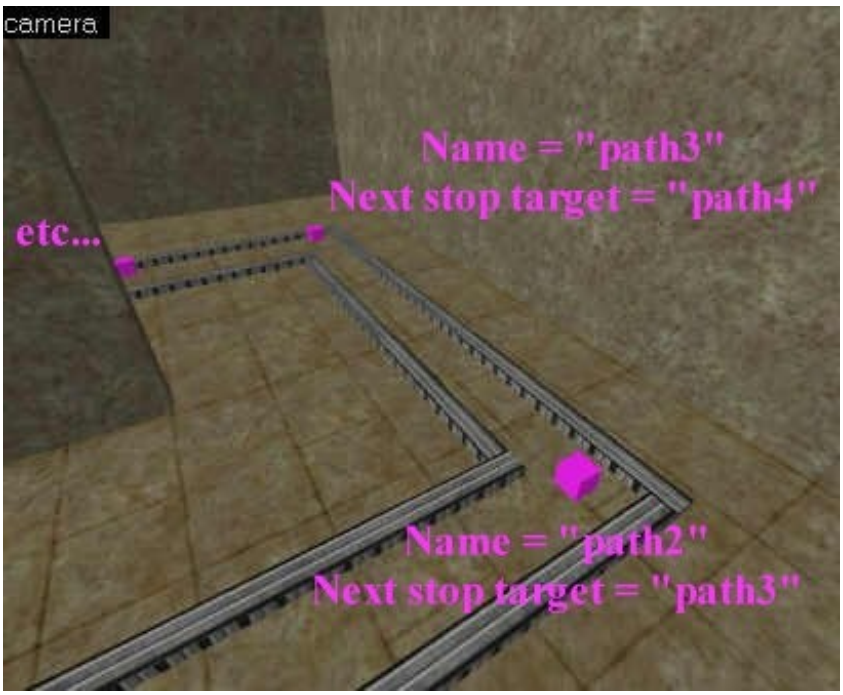
Les flags sont les suivants :

- Disabled : le train ne peut pas se rendre à ce point tant que l'entité n'a pas été activée.
- Fire once : l'entité est désactivée une fois que le train est passé par là.
- Branch reverse : inversion des changements de voies.
- Disable train : le train s'arrête lorsqu'il arrive à ce point.

Voilà, vous devrez placer des *path_track* à tous les endroits où votre train doit tourner. Surtout, veillez bien à ce que toutes ces entités soient à la même hauteur !
Voici le fonctionnement des *path_track* :

- Premier path_track : Name = "path1", Next stop target = "path2"
- Deuxième path_track : Name = "path2", Next stop target = "path3"
- Troisième path_track : Name = "path3", Next stop target = "path4"
- etc etc...

Enfin, voici un dernier petit exemple pour illustrer le tout :



Comme vous pouvez le voir, je n'ai placé des *path_track* que lorsque c'était nécessaire (aux virages). Il est inutile de mettre 36 points de passages au milieu d'une ligne droite : 2 suffisent, 1 à chaque bout !

Autres entités utiles

Bon, je crois vous avoir pratiquement tout dit sur les trains, je ne vais donc pas m'y attarder éternellement. Toutefois, je vous indique quand même qu'il existe d'autres entités qui peuvent vous être utiles :

- *func_trackchange* : changement de voie pour un train, pour le faire monter d'un étage par exemple.
- *func_trackautochange* : changement automatique de voie pour un train. C'est la même chose qu'un ascenseur rotatif en fait.
- *func_train* : pourquoi je n'ai pas parlé de cette entité en premier ? Parce que c'est la même chose pratiquement que le func_tracktrain, à part que ça utilise des path_corner au lieu des path_track. Mieux vaut utiliser un func_tracktrain à mon goût, enfin bon c'est vous qui voyez

Pour conclure, je dirais qu'un train est un peu compliqué à mettre en place la première fois, mais que le jeu en vaut la chandelle. Après, c'est 100% de régal : on peut fragger des aliens sur son train en fonçant dans le tas ! BOURRIIIINN !!!!!!

Les voitures de CS

Entités concernées : func_vehicle, func_vehiclecontrols
Type d'entités : entités-bloc
Difficulté : assez difficile

Les voitures ont été introduites avec Counter-Strike, et par conséquent elles ne fonctionnent que sous ce mod. En d'autres termes, il faut faire une map pour CS pour pouvoir profiter des voitures !

Mais qu'est-ce que j'appelle une voiture ? Si vous avez lu mon explication sur les trains (juste au-dessus), vous avez retenu une chose assez gênante : le train est obligé de suivre un rail, un chemin précis.
Une voiture fonctionne pratiquement comme un train, sauf que cette fois... vous pouvez aller où vous voulez sur votre map !

Certains se souviennent peut-être du tank qui était dans cs_siege dans les premières versions de Counter-Strike... eh bien c'était ça ! Une voiture !
En plus, il y avait de la place dans le tank pour que d'autres potes en profitent : vous au volant, les autres assis derrière et attendant votre go ! Je vous laisse imaginer les possibilités !

Notez que les développeurs de CS semblent avoir abandonné les voitures. En clair, vous pouvez toujours faire des maps avec des voitures, mais vous ne verrez aucune map officielle avec des voitures.

Bon ! Commençons

Pour mon exemple, je vais recréer l'univers de la route 66. Avec des blocs donc, je crée une route américaine, et je mets du sky autour (il ne s'agit pas de faire une vraie map, donc on va pas se prendre la tête).
Ensuite, et c'est le plus important : je crée... la moto ! Une Harley-Davidson, ça va de soi

Rappelez-vous que vous n'êtes pas obligés de faire une "voiture". Ici par exemple je fais un moto, mais vous pouvez faire un tank, un bus, une ambulance, une trotinette à moteur...

Passez un temps important à modéliser votre engin, pour que ça ait l'air de quelque chose de potable. Voici ma Harley :

Deuxième étape : le bloc ORIGIN. Là encore, vous allez devoir placer un bloc ayant la texture ORIGIN au centre de gravité de votre voiture.
En gros, ça correspond au milieu de votre voiture (bien au centre). Voyez où je l'ai placé sur ma Harley :

Sélectionnez le tout (voiture + bloc ORIGIN), et cliquez sur "toEntity". Dans la fenêtre qui s'ouvre, choisissez l'entité *func_vehicle*.

Les propriétés à éditer son quasiment les même que pour un train :

- Name : c'est le nom de votre voiture. Exemple : "moto".
- First stop target : normalement c'est pour les path_track, mais ne vous en servez pas ici. Indiquez juste le nom de votre

voiture ça suffira ! Par exemple, je mettrai ici "moto".
- Sound : son produit par la voiture en déplacement.
- Length of vehicle : longueur de la voiture en unités de Worldcraft.

Pour connaître la longueur de votre voiture, sélectionnez-la et lisez ce qui est écrit en bas de l'écran, dans la barre d'informations.
Exemple : "44w 128l 54h". La moto a donc une longueur de 128 unités, et une largeur de 44 unités !

- Width of vehicle : largeur de la voiture en unités de Worldcraft. Vous savez désormais retrouver la largeur de votre voiture, je viens de vous l'expliquer
- Height above track : distance en hauteur séparant le centre de la voiture (son bloc ORIGIN) et les path_track. Vous pouvez mettre 0, étant donné que nous allons placer la voiture et les path_track à la même hauteur.
- Initial speed : vitesse initiale de la voiture (en unités de Worldcraft par seconde).
- Speed (units per second) : vitesse de la voiture, en unités de Worldcraft par seconde. Je vous conseille d'augmenter la valeur par défaut, parce que c'est assez lent sinon. Mettez au moins 200...
- Damage on crush : si quelqu'un se fait écraser par la voiture, indiquez ici le nombre de points de vie qu'il perdra (0 = pas de bobo).

Attention : il semble que la voiture écrase indifféremment ennemis, amis et hotages !!! Vérifiez ce qui se passe si un de vos alliés passe sous les roues

- Volume (10 = loudest) : volume du son. Mettez une valeur entre 0 et 10, sachant que 10 est la valeur la plus élevée.
- Bank angle on turns : angle de rotation de la voiture lors d'un virage. Laissez la valeur de 0, ça marche très bien !

Cette entité possède 3 flags :

- No Pitch (X-rot) : il n'y aura pas de tangage de la voiture.
- No user control : le joueur ne peut pas prendre le contrôle de la voiture.
- Passable : la voiture devient "fantôme". C'est-à-dire qu'elle traverse tous les joueurs sans exception, et sans les blesser.

Etape suivante : orientation de la voiture. Ici, il semble que ma moto est dans le bon sens de la route... pourtant, allez savoir pourquoi, lorsque vous jouerez Half-Life va vous la placer dans n'importe quel autre sens que celui que vous voulez.
Si jamais c'est le cas, alors c'est très simple : allez dans le menu Tools/Transform (Ctrl + M), et faites une rotation de 90° sur l'axe des Z. Testez votre map, et si ce n'est pas bon recommencez l'opération : rotation de 90° etc... Jusqu'à ce que la voiture soit enfin dans le bon sens. Ouf !

Bien, maintenant il faut placer le point de commande de votre véhicule. En clair, vous allez dire à Half-Life où se trouve le volant de votre voiture (ici le guidon de ma moto).
Créez un bloc avec la texture AAATRIGGER là où vous voulez qu'on puisse prendre contrôle du véhicule, et transformez-le en entité *func_vehiclecontrols*.
Dans sa propriété "Vehicle Name", mettez le nom de la voiture qu'il commande.

Vous pouvez voir deux nouvelles choses ci-dessus : l'orientation de la voiture a changé (ainsi elle sera placée dans le sens de la route lorsque je jouerai), et le bloc AAATRIGGER est placé devant le guidon pour que je puisse contrôler ma Harley.

Oufff ! C'est tout bon, on s'arrête là. Vous avez une voiture qui marche !!! Et moi, je vais aller faire un petit tour sur ma moto...

Les sons joués sont vraiment sympas, surtout quand on freine brusquement
Pour faire avancer votre moto, vous devez laisser enfoncée la toucher "Avancer" (celle que vous utilisez lorsque vous faites marcher votre personnage tout droit).

Des heures de délire en perspective
Les voitures peuvent modifier complètement la stratégie d'une map et peuvent être très intéressantes, le tout est d'être arrivé jusqu'au bout de ce chapitre vivant

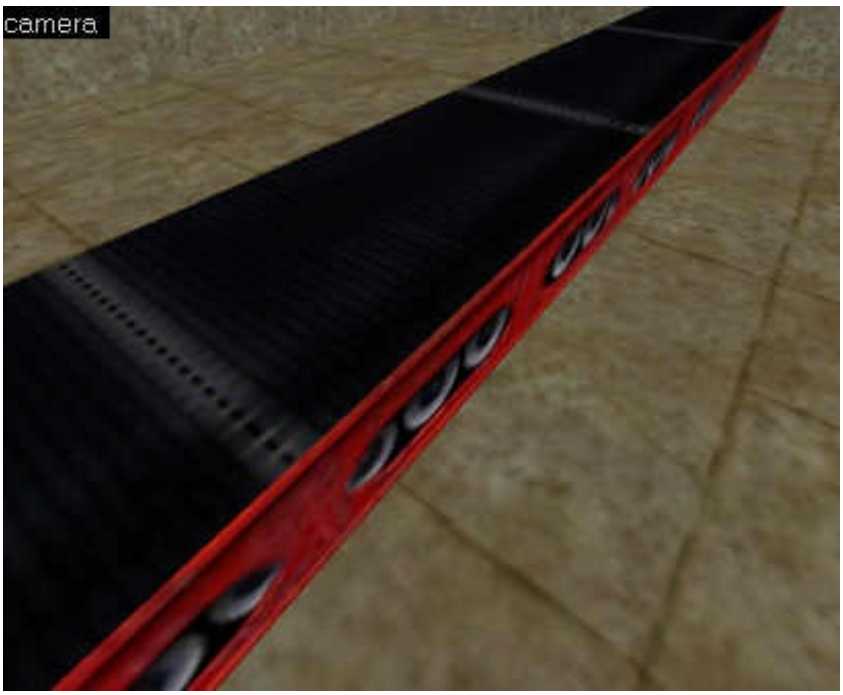
Les tapis roulants

Entité concernée : func_conveyor
Type d'entité : entité-bloc
Difficulté : assez facile

J'ai décidé de terminer ce chapitre sur les moyens de transport par... les tapis roulants ! Pour deux raisons : d'abord parce que c'est un moyen de transport, même si ce n'est pas très pratique, et ensuite pour vous permettre de souffler après un chapitre assez difficile, je le reconnais.
Donc, ne vous en faites pas, les tapis roulants sont très faciles à mettre en place, comparés aux trains ou aux voitures

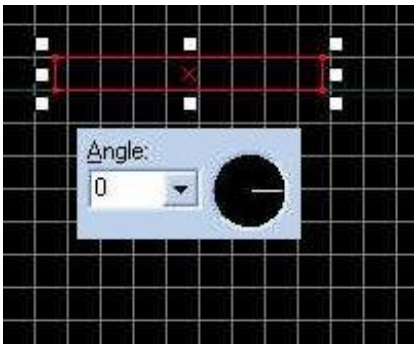
Même si cette entité est faite au départ pour les tapis roulants, vous pouvez sans problème vous en servir pour simuler un cours d'eau !

Pour commencer, il va falloir créer le tapis roulant. Là encore, rien de plus facile ! Il faut juste savoir que les textures appropriées se nomment C2A4_CONV1, C2A4_CONV2 et SCROLL_CONV3.
Voici ce que vous devriez voir si vous vous êtes bien débrouillés :



Transformez ce bloc en entité *func_conveyor*.
La première chose très importante, c'est l'orientation de l'entité. Cela vous permettra de définir dans quel sens le tapis roulant doit vous déplacer.
L'orientation est tellement importante que je vais vous expliquer exactement comment ça fonctionne. C'est valable pour l'orientation de toutes les entités, alors écoutez bien.

Il faut regarder la vue Top, et en même temps le "radar" de la fenêtre des propriétés. Comment orienter le radar pour que notre entité soit orientée vers le haut de la vue Top ?
Eh bien il suffit d'orienter le radar vers le haut !
Si on superpose ces deux éléments, on obtient ceci :



Ici, notre tapis roulant (vu de haut) se déplacera vers la droite. Si on voulait qu'il se déplace vers la gauche, il faudrait orienter le radar dans l'autre sens ! C'est tout bête

Bon, maintenant que vous savez cela, vous devriez éditer les propriétés du *func_conveyor*. Mais bon, comme vous les connaissez toutes et qu'ici elles n'ont vraiment aucun intérêt (même Name ne sert pas), je ne vais pas vous en parler... ah si, il y en a quand même une qui peut être utile :

- Conveyor Speed : c'est la vitesse du tapis roulant, en unités de Worldcraft par seconde. La vitesse de 100 proposée par défaut est tout à fait correcte pour les tapis roulant.

En revanche, je trouve les 2 flags intéressants :

- No Push : l'entité ne déplacera pas le joueur. Par contre, elle continuera à simuler un mouvement. Par exemple : le tapis roulant continuera à avancer, mais si vous marchez dessus vous n'avancerez pas avec lui).
- Not Solid : le tapis roulant est visible mais pas solide. On passe donc à travers. Toutefois, il continue à pousser le joueur comme si de rien n'était.

C'était un chapitre assez difficile, mais que voulez-vous, il faut bien un peu de difficulté quand on commence à faire des choses intéressantes.

Bon je vous laisse, y'a ma Harley qui m'attend 

Comme au cinéma

Vous voulez jouer à l'apprenti cinéaste ? Ca tombe bien, Half-Life a tout prévu : caméras, effets de fondu etc... Tout pour commencer une carrière de metteur en scène quoi

Les caméras

Entités concernées : trigger_camera, info_target et path_corner
Type d'entités : entités-point
Difficulté : moyen

Elles sont vraiment pratiques dans une map multijoueurs Half-Life ou Counter-Strike, et dans un mod solo elles rajoutent un effet d'ambiance car on peut alors suivre le joueur à la troisième personne.
De qui s'agit-il ? Des caméras bien entendu !

Les développeurs de Counter-Strike nous encouragent d'ailleurs à mettre des caméras dans nos maps. Alors, vous savez ce qu'il vous reste à faire...

Pour intégrer une caméra dans sa map, il faut placer une entité-point *trigger_camera* à l'endroit voulu. Ceux qui utilisent le FGD de Counter-Strike devraient voir ceci (sinon c'est un gros cube bien moche) :

Notez qu'il n'y a aucune différence entre les caméras de Half-Life et celles de Counter-Strike.

Il existe 2 types de caméras :

- Les caméras fixes : lorsque vous regardez une telle caméra, celle-ci reste fixe et vise toujours le même endroit. Elle est assez simple à configurer.
- Les caméras mouvantes : ce type de caméra se déplace en suivant un parcours précis. Idéal notamment pour simuler une caméra du surveillance. Seul problème : elle est un peu compliquée à mettre en place.

Les caméras fixes

Placez un *trigger_camera* à l'endroit où vous voulez mettre votre caméra. Une caméra est invisible pour le joueur, c'est juste une indication sur sa position. Si vous voulez "dessiner" une caméra pour plus de réalisme, il va falloir la créer vous-même avec des blocs !
Pour l'instant on se contentera d'une caméra invisible, car en général peu de mappeurs prennent la peine de représenter la caméra.

Placez ensuite une entité *info_target* à l'endroit où la caméra doit viser. C'est cette entité (invisible elle aussi) que la caméra regardera.

Schématiquement ça donne ceci :

Le schéma vous dit tout : le *trigger_camera* prendra Target = "nomdelacible", et *info_target* prendra Name = "nomdelacible". Cela suffit pour faire une caméra qui fonctionne ! Pensez toutefois à mettre un nom (Name) à votre caméra pour pouvoir l'appeler à l'aide d'un bouton ou d'un trigger (sinon vous ne pourrez jamais vous en servir pendant le jeu !).

Voici les quelques nouveaux attributs à connaître :

- Hold Time : temps pendant lequel le joueur regardera la caméra (en secondes). Je vous conseille de mettre moins de 10 secondes, parce que sinon ça fait un peu trop long à attendre pour le joueur.
- Initial Speed : vitesse initale de déplacement de la caméra. A utiliser pour les caméras mouvantes.
- Acceleration units/sec² : accélération de la caméra (en unités par seconde au carré). Euh, l'unité est un peu complexe, alors si vous n'êtes pas sûr de vous n'essayez pas de toucher à cet attribut
- Stop Deceleration units/sec² : décélération de la caméra, lors de son arrêt (en unités par seconde au carré)

3 flags vraiment intéressants sont à noter :

- Start At Player : la caméra part du joueur pour se rendre vers la cible (on a l'impression de "sortir" des yeux du joueur, ça donne un assez bon effet). A utiliser seulement si la caméra est assez proche du joueur.
- Follow Player : la cible de la caméra devient le joueur (en clair, l'attribut Target ne sert plus à rien). La caméra suivra donc vos déplacements.
- Freeze Player : le joueur ne peut pas bouger lorsqu'il regarde la caméra (c'est souvent utile !).

Les caméras mouvantes

Il y a plusieurs méthodes pour se servir de caméras mouvantes. Dans tous les cas, l'objectif est de faire en sorte que la caméra ne soit pas fixe : elle doit bouger, soit autour d'un objet, soit changer de cible.

Si vous devez changer de cible, ce qui est assez simple à mettre en place en fait, utilisez l'attribut "Path_corner", qui indiquera le nom du premier *path_corner* à regarder. Faites ensuite une boucle avec les *path_corner* qui doivent s'appeler entre eux, exactement comme vous l'avez appris pour le fonctionnement des trains de Half-Life.
Je ne vous refait pas l'explication ici, ce serait trop long, alors que tout est expliqué en détail dans le chapitre sur les véhicules.

Autre technique : elle consiste à lier la caméra avec un *func_train*. Vous devez mettre pour cible à la caméra le nom du train, de telle sorte que la caméra suive le mouvement de ce train. Bien entendu, vous devez savoir configurer un train auparavant.
Le train peut être un vrai train modélisé, mais vous pouvez aussi "tricher" : créez un petit bloc, transformez-le en train, et faites en sorte que le train soit invisible. Ainsi, il y aura un petit bloc invisible qui se baladera dans votre map, pendant que la caméra suivra son mouvement !

Comme vous pouvez le voir, tout ceci est très lié avec les trains. Alors apprenez avant à vous servir des trains, et vous verrez alors que faire une caméra mouvante n'est pas tellement compliqué
Si toutefois vous n'y arrivez pas, ne vous bloquez pas là-dessus. Il est tellement plus simple de faire une caméra fixe ! En plus, la plupart des mappeurs mettent des caméras fixes dans leurs maps, et c'est le plus souvent amplement suffisant ! Faites plusieurs caméras si nécessaire...

Les caméras de la mort

Entités concernées : info_intermission et info_target
Type d'entités : entités-point
Difficulté : facile (si on sait se servir d'une caméra fixe)

Sous ce nom bizarre de caméra de la mort, se cache en fait un type de caméra peu connu fait pour le mode multijoueurs seulement. Et je ne vous cacherai pas que, même si je ne l'ai pas testé sous Counter-Strike, c'est une caméra qui fonctionnera surtout sous Half-Life Multijoueurs

Le fonctionnement est on ne peut plus simple : placez un *info_intermission*, comme si c'était la caméra. Placez ensuite un *info_target* comme vous l'avez fait pour la caméra fixe.
Remplissez ensuite les attributs de ces entités (il n'y a rien de plus simple puisque chacune de ces entités n'a qu'un seul attribut !).
Si vous n'y arrivez pas, c'est que vous n'avez pas lu le fonctionnement des caméras fixes ci-dessus. Relisez-le plus attentivement, parce que vous n'avez aucune excuse : c'est très très très simple

Inutile de faire un bouton pour appeler ce type de caméra. Celle-ci sera déclenchée automatiquement 6 secondes après la mort du joueur si celui-ci ne fait rien.
Enfin bon, reconnaissons que ce type d'entité a peu d'intérêt, parce qu'en multijoueurs Half-Life on respawn tout de suite pour refoncer comme un bourrin dans la bataille. Une vraie boucherie quoi Mais si vous êtes un mappeur attentionné, alors placez quelques-unes de ces entités dans votre map pour les joueurs qui voudraient attendre un peu avant de retourner au combat (quelles mauviettes !).

Effet de fondu

Entités concernées : env_fade
Type d'entités : entité-point
Difficulté : assez facile

Voilà une entité qui permet de créer de bons effets de fondu assez facilement, grâce à un procédé de décoloration dernière génération qui nécessite des cartes graphiques très très poussées ! (pour ceux qui n'auraient pas compris je rigole, lol : l'effet est sympa, mais n'importe quel PC peut le faire !)
Si vous mettez un *env_fade* dans votre map (assorti avec un trigger pour le déclencher, pensez à bien renseigner les attributs Name et Target comme vous avez appris à le faire), vous pourrez en quelques clics faire des effets qui surprendront le joueur !

Regardez plutôt les attributs de cette entité, vous allez en apprendre beaucoup ! J'ai volontairement enlevé l'attribut "Name" : si vous vous demandez à quoi il sert, il serait grand temps de revoir les premiers chapitres de cette Partie II du cours

- Duration (seconds) : temps que dure la décoloration (en secondes). Par défaut, c'est une transition de 2 secondes, mais vous pouvez mettre plus si vous voulez faire durer le plaisir
- Hold Fade (seconds) : temps pendant lequel l'écran restera décoloré (en secondes). A ne pas confondre avec "Duration", qui indique le temps que met l'écran à se décolorer !
- Fade Alpha : puissance de la décoloration (valeur entre 0 et 255, 0 = rien, 255 = totale). En général on met 255, car avec cette valeur l'écran devient complètement opaque (on ne voit plus le jeu, on a un écran d'une seule et même couleur !).
- Fade Color (R G B) : c'est la couleur vers laquelle l'écran va fondre. Indiquez les composantes Rouge Vert Bleu, ou bien servez-vous du bouton "Pick color", c'est plus simple ! Par défaut, c'est 0 0 0, c'est à dire noir complet. Mais si vous voulez que l'écran vire au rouge sang, mettez 255 0 0 !

Grâce aux flags vous pouvez avoir plusieurs variantes de cette entité :

- Fade From : produit exactement l'effet inverse. C'est-à-dire que l'écran part de la couleur opaque et revient progressivement dans le jeu.
- Modulate : un flag à tester absolument ! Au lieu de passer vers un écran opaque, le joueur sera toujours dans le jeu, mais la couleur indiquée de "Fade Color" apparaîtra comme dominante à l'écran ! Un effet ex-tra par exemple si vous mettez une

couleur rouge : le joueur évoluera dans un monde tout rouge, comme s'il avait été mortellement touché.
- Activator Only : seul celui qui aura activé le *env_fade* subira la décoloration. On ne s'en servira donc que dans une partie multijoueurs, pour éviter que tout le monde voie son écran virer au rouge parce qu'un blaireau a appuyé sur un bouton sans le faire exprès...

Afficher du texte

Entités concernées : env_message
Type d'entités : entité-point
Difficulté : assez facile

Cette entité sera en réalité utilisée si vous faites votre propre mod. Si vous vous en servez sur un mod déjà existant vous risqueriez d'avoir des problèmes, car cette entité a besoin d'un fichier appelé "titles.txt" pour fonctionner.
Quoi je parle chinois ? Bon, ouvrez le fichier Half-Life\valve\titles.txt. Vous allez comprendre... Descendez un peu, au début il n'y a que des paramètres. Par exemple, voici un extrait de ce fichier :

```
CR28
{
SUJET :
Gordon Freeman
Homme, 27 ans
}

CR29
{
FORMATION :
Diplômé en Physique Théorique
Institut de Technologie du Massachusetts
}

CR30
{
POSTE :
Chercheur
}

CR31
{
AFFECTATION :
Laboratoire des MatériauxAnormaux
}
```

Ca ne vous rappelle rien ???
Eh oui ! C'est le texte qui s'affiche lorsque vous démarrez une partie solo de Half-Life, tout au début du jeu ! Ce fichier contient donc tous les messages qui apparaissent à l'écran du joueur.
Chaque partie fonctionne comme ceci :

```
NomDuTexte
{
Texte à afficher...
}
```

A vous donc de créer ce fichier "titles.txt" dans le répertoire de votre mod. Je vous conseille de vous baser sur celui de Half-Life pour voir comment il fonctionne. De préférence, numérotez chaque message comme ils l'ont fait pour Half-Life (CR01, CR02, CR03...)

A quoi sert notre entité *env_message* dans l'histoire ? Permettez son déclenchement à l'aide d'un trigger pour commencer...
Utilisez l'attribut "Message Name" pour indiquer le nom du message à afficher (par exemple CR08). Voilà c'est tout ! Tout les paramètres (couleur, durée d'affichage...) sont à indiquer dans le fichier titles.txt.
A noter que *env_message* peut aussi déclencher un petit son, comme le ferait un *ambient_generic*.

Vous comprenez maintenant l'importance cruciale du fichier titles.txt pour celui qui veut créer son mod solo, avec un scénario à lui et tout et tout
Pourquoi ne pas créer une grande aventure interactive en 3D grâce à Half-Life ? Vous connaissez maintenant tout ce qu'il faut savoir : caméra, effets de fondu, textes pour le scénario etc etc...

Armes et munitions

Je crois qu'il est temps de faire une pause... Je vais vous concocter un chapitre simple, qui prend pas la tête, et grâce auquel vous allez pouvoir bourriner un peu dans votre map !

Nous verrons comment poser au sol des armes et des munitions dans Half-Life, puis comment poser des armes au sol avec Counter-Strike. Bien entendu, il n'est pas possible de récupérer des munitions au sol sous Counter, mais déjà avoir la possibilité de donner une arme c'est pas si mal !

Armes de Half-Life

Entités concernées : toutes les entités commençant par "weapon_"
Type d'entité : entités-point
Difficulté : très facile

Tous ceux qui ont déjà joué à Half-Life doivent connaître à peu près par coeur les armes du jeu, leurs qualités, leurs défauts etc... Je ne vais pas faire un descriptif détaillé de chaque arme, vous en savez peut-être plus que moi.
Tout ce que je vais faire, c'est vous montrer comment on pose une arme sous Worldcraft, et quelles sont les armes disponibles.

Vous devrez poser des armes au sol dans une map solo Half-Life (au fur et à mesure de l'aventure, le joueur récupère des armes de plus en plus puissantes), ou bien dans une map multijoueurs Half-Life (là c'est à vous de disposer comme il faut les armes, en rendant bien sûr les meilleures un peu difficiles d'accès 

Bon, comment fait-on ? Je vois que vous êtes impatients lol ! Je vous rassure, c'est très très simple !
Dans votre map Half-Life sous Worldcraft, activez le "Entity Tool" et regardez les entités commençant par "weapon_". Ce sont les armes que nous cherchions, tout simplement !
Sous Worldcraft, l'arme est représentée par un petit cube. Pensez à le surélever en hauteur, de sorte à ce que l'arme ne reste pas "bloquée" au sol. Je vous rassure : si votre arme est placée au-dessus du sol, elle ne va pas rester là en lévitation ! Elle tombera toute seule au début de la partie (et en général ça ne se voit pas).

Toutes ces entités ont les mêmes attributs et flags. Je vous le dis de suite : je n'y ai jamais touché et je n'y toucherai pas ! Ils n'ont quasiment aucun intérêt, je dirais même il n'y a pas du tout d'intérêt ! Contentez-vous de poser votre entité et le tour est joué !


Ok merci, mais comment je vais faire pour reconnaître les armes par leur nom ???

Ah, évidemment si vous ne connaissez pas les noms des armes... Eh bien écoutez, tout ce que je peux vous proposer, c'est un screenshot de chaque arme, avec son nom à côté. Comme ça, vous pourrez choisir celle que vous voulez intégrer dans votre map.
Allez, hop ! Voilà le tableau pour vous aider !

Nom	Aperçu	Description
weapon_crowbar		Voici le pied de biche, l'arme favorite de Gordon Freeman. Fonctionne aussi bien sur les caisses que sur les crânes !
weapon_9mmhandgun		Le flingue de base... c'est pas avec ça que vous pourrez fragger du monde, mais ça peut servir parfois : quand on est à court de munitions, ou quand on veut tirer sous l'eau (vous ne saviez pas qu'on pouvait tirer sous l'eau avec ?)
weapon_357		Plus puissant que le précédent, ce flingue fait trop penser au far-west dans un monde gouverné par des armes aux munitions radioactives... A n'utiliser que si vous avez l'âme d'un cow-boy !
weapon_shotgun		A n'utiliser qu'en combat rapproché. Normalement, au bout quelques tirs l'ennemi est suffisamment criblé de balles, vous pouvez le laisser mourir tranquillement et passer à quelqu'un d'autre 
weapon_9mmAR		Une arme très polyvalente, que l'on peut sortir dans quasiment toutes les situations. A noter que si vous êtes à court de munitions, vous pouvez toujours balancer une grenade ou deux, et là en théorie ça fait du ménage...
weapon_crossbow		Une arbalète... Fonctionne sous l'eau avec zoom intégré. A ne pas oublier si vous partez à la pêche au pirhanas !
weapon_hornetgun		Une arme alien peu puissante, mais on ne lui en veut pas car les munitions sont illimités, et les missiles sont guidés !

weapon_snark		Mon arme préférée Ce sont de zolies petites bêbêtes qui se ruent sur la première personne qu'elles voient pour la dévorer toute crue ! Ca met de l'ambiance dans une map multijoueurs, je peux vous le garantir !ps : évitez de rester près des bestioles après les avoir lancés, elles sont capables de se retourner contre vous !
weapon_handgrenade		Dégoupillez-la, lancez-la et comptez jusqu'à 5. Normalement si tout va bien, ça devrait faire boum...
weapon_tripmine		Une mine laser : posez-la et éloignez-vous rapidement. Si quelqu'un passe par là, il en aura pour son compte
weapon_satchel		Arme explosive en deux temps : tout d'abord vous placez l'explosif au sol, et vous attendez que quelqu'un passe par là. Normalement, il va se rapprocher de l'engin pour voir ce que c'est exactement, et juste au moment où il se rend compte du piège : BOUM ! Vous appuyez sur le détonateur.Arme pour sadiques uniquement.
weapon_rpg		Le bon vieux lance-roquette, toujours aussi efficace contre les durs à cuire, mais fonctionne aussi sur les hélicoptères et avions.Le fabriquant recommande de ne pas s'en servir en combat rapproché, il paraît que des utilisateurs s'en sont plaint !
weapon_gauss		Une arme expérimentale, donc en version bêta... Si vous l'utilisez, c'est à vos risques et périls ! Mais vu comme elle est puissante, on prend le risque quand même ^^Attention à ne pas la charger trop longtemps, ou l'engin explosera dans vos mains !
weapon_egon		Encore une arme expérimentale... mais alors celle-là c'est l'arme ultime ! Vos ennemis n'auront pas le temps de riposter, ils auront déjà été aspirés par cet engin ou bien désintégrés atome par atome.Très gourmand en munitions, à n'utiliser que si vous voulez faire du nettoyage sans laisser de traces (autres que la cervelle de vos ennemis écrabouillée sur les murs)

Il faudrait rajouter l'entité *weaponbox*, qui contient des tas d'armes et munitions au hasard.

Si avec tout cet arsenal vous n'arrivez pas à vous en sortir, je ne peux plus rien pour vous !

Les munitions de Half-Life

Entité concernée : entités commençant par "ammo_"
Type d'entité : entités-point
Difficulté : très facile

Je vous ai décrit les armes de Half-Life, je ne vais pas faire pareil pour les munitions... pour la simple et bonne raison que vous saurez retrouver les munitions qui correspondent aux armes que vous posez !

Ce sont toutes les entités commençant par "ammo_". Vous trouverez facilement les correspondances avec les armes. Par exemple, *ammo_rpgclip* sont des munitions pour l'entité *weapon_rpg* (le lance-roquettes !).
Vous ne devriez pas avoir de difficultés à poser des munitions si vous savez poser des armes !

Encore une fois, rappelez-vous de poser les entités légèrement au-dessus du sol pour éviter d'éventuels problèmes.

Confisquer les armes

Entité concernée : player_weaponstrip
Type d'entité : entité-bloc
Difficulté : très facile

Bon, là c'est trop facile, le joueur est armé jusqu'aux dents et détruit tout sur son passage. Hop ! Confisqué !
Ca ne vous est jamais arrivé dans l'aventure de Half-Life. A un moment où tout allait pour le mieux, on se fait embarquer par les marines et on se réveille... sans aucune arme !

C'est ce que cette entité de sadique permet de faire : elle retire toutes les armes et munitions du joueur. Elle lui laisse juste la combinaison de protection.
Pour l'activer, utilisez un trigger, et appelez cette entité (elle n'a que le champ "Name" qui sert à lui donner un nom).

Ben voilà c'est tout, mais c'est une entité que je n'utiliserais pas plus d'une fois par jeu, parce que sinon le joueur risquerait de finir par péter les plombs

Les armes de Counter-Strike

Entité concernée : armoury_entity
Type d'entité : entité-point

Difficulté : très facile

Dans Counter-Strike, on a l'habitude d'acheter ses armes avant de partir au combat. C'est ce qui se passe dans toutes les maps officielles.
Toutefois, si vous faites une map de training, pour votre team par exemple, vous aurez peut-être envie de poser des armes au sol... comme c'est le cas dans la célèbre *awp_map* (la map la plus moche et la plus mal foutue que l'univers ait connu !)
Vous avez le droit de poser des armes au sol, mais retenez bien ceci : si vous voulez faire une vraie map, il va falloir oublier les "sp_", "awp_" et compagnie. Une vraie map est travaillée, suffisamment grande (pas trop quand même), et a amplement plus de chances d'être reconnue et jouée sur Internet.

Je ne vous cacherai pas que je désapprouve l'utilisation des armes au sol, à moins que ce ne soit une map pour vous et vos potes. Si vous hésitiez à en poser dans votre map, alors je vous encourage à oublier cela et à vous lancer dans un vrai projet ! Votre map n'en sera que meilleure à tous points de vue !

Bien, voilà comment ça fonctionne. C'est un peu différent que pour Half-Life, comme on l'a vu plus haut. Il n'y a qu'une seule entité à poser : *armoury_entity*. Là encore, ne collez pas cette entité au sol, surélevez-la un peu : l'arme tombera proprement au sol au début de la map.

Regardez maintenant les propriétés de cette entité. Il y en a 2 :

- Item : nom de l'arme à poser. Comme vous êtes des férus de Counter-Strike, je pense que vous saurez reconnaître les noms des armes sans problème 
  A noter que l'on peut mettre des kevlar, mais pas des lunettes de vision nocturne.
- Count : c'est le nombre d'armes à poser. Les armes seront toutes placées au même endroit, empilées les unes sur les autres si vous préférez. En temps normal, on préfère mettre une arme par entité, mais rien ne vous empêche de mettre plusieurs fois la même arme au même endroit !

Voilà c'est tout, il n'y a rien de bien sorcier. Je ne vois pas ce que je pourrais rajouter... ah si : pour bien vous faire la main avec le mapping Counter-Strike, voici ce qu'il ne faut PAS faire :

- Mettre des armes au sol
- Faire une speedmap

Si vous aviez l'intention de faire cela, alors oubliez tout ! Votre map n'aura aucun intérêt, et sera très rapidment oubliée (en quelques heures).
En revanche, si vous faites un plan de votre map, que vous réfléchissez à son organisation et que vous la faites suffisamment grande, là vous avez toutes les chances d'être reconnu et de faire des maps de plus en plus belles ! 
N'oubliez jamais de faire le plein d'armes et munitions avant de partir au combat, parce qu'à mains nues c'est généralement plus difficile 

Sprites et models

Ce chapitre, lorsque vous commencez à bien vous débrouiller en mapping, est extrêment important.
Nous allons apprendre à nous servir de nouvelles entités, qui sont carrément de nouveaux éléments dans une map. Nous allons aussi pour commencer étudier le fonctionnement du "package" de Half-Life, ce qui sera utile si vous désirez créer votre propre mod (et même si vous ne faites que des maps, c'est quelque chose d'important à connaître !).

Comme il s'agit de quelque chose de totalement nouveau, je vous demande de redoubler d'attention. Eh non, vous n'aviez pas encore tout vu sur les possibilités du moteur de Half-Life !

Le package

Nous allons en fait nous poser une question essentielle : **mais où est passé Half-Life ???**
Peut-être le savez-vous déjà, le jeu de Half-Life se trouve dans le sous-dossier "valve" (exactement comme s'il s'agissait d'un mod appelé "valve"). Et même si vous ne connaissez pas bien le fonctionnement d'un mod, vous allez vite vous rendre compte que les dossiers sont le plus souvent... vides !!!

Les maps de l'aventure de Half-Life par exemple : où sont-elles ? Normalement vous devriez les trouver dans le sous-dossier "maps", mais celui-ci est vide, ou presque. Alors !? Où sont passés tous les fichiers ???

Bon, on arrête là pour les questions, et on passe aux réponses

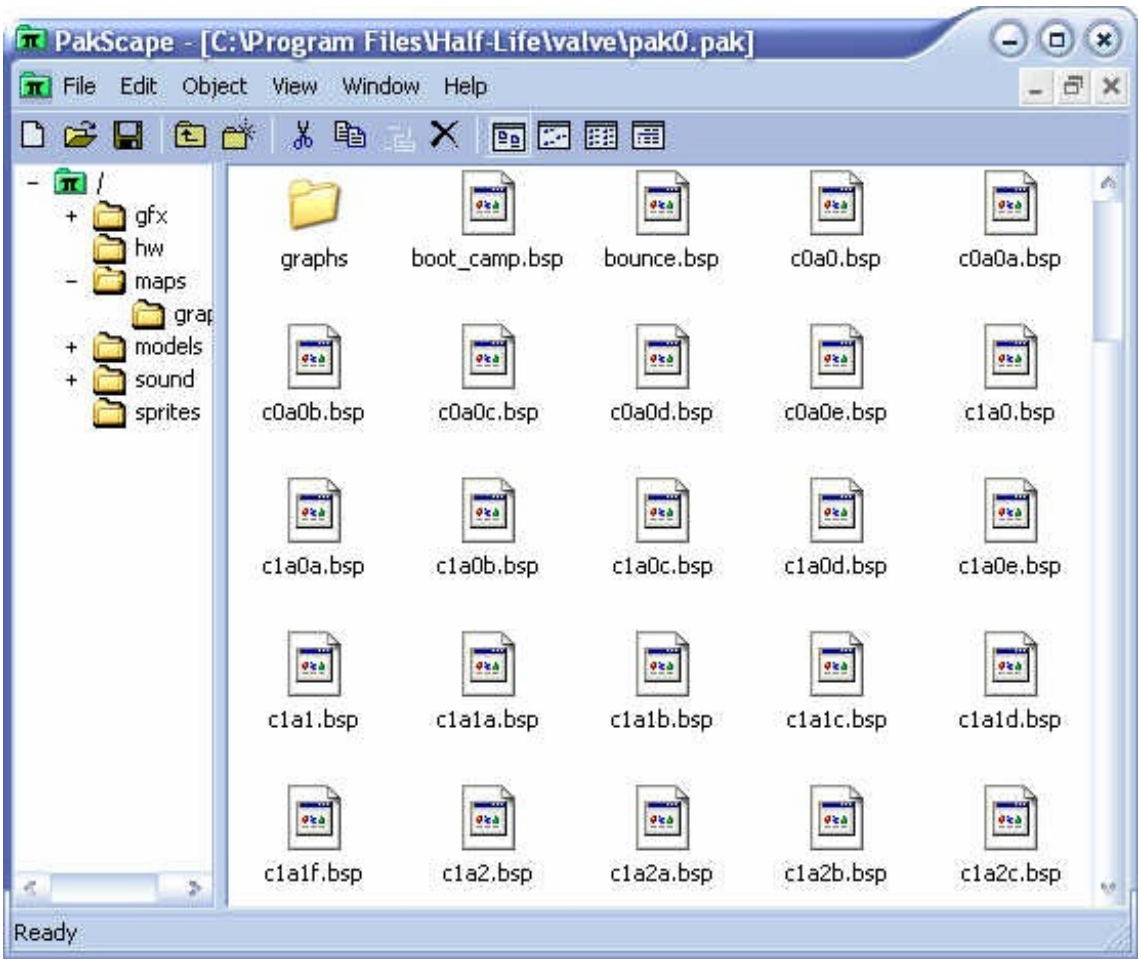
Allez dans le dossier "valve", et recherchez un gros fichier... il doit s'agir normalement de pak0.pak. Ne trouvez-vous pas que 290 Mo ça fait un peu gros pour un fichier ?
Rhaa, on aimerait bien l'ouvrir pour voir ce qu'il y a dedans... mais Windows ne connaît pas ce format, pas plus que Winzip, Winrar etc etc... Le programme permettant de lire ce type de fichier (.pak), vous ne l'avez sûrement pas. Je vais donc vous le donner

PakScape.zip (105 Ko)

Il s'appelle donc PakScape. Copiez-le dans le répertoire où Hammer est installé par exemple, et créez un raccourci dans le menu Démarrer. Il ne nécessite pas d'installation.

Grâce à lui, vous allez pouvoir ouvrir ce mystérieux fichier pak0.pak. Allez dans le menu File / Open, et indiquez où se trouve le fichier pak0.pak.
PakScape le charge, et vous présente son contenu sous forme de fichiers et dossiers. Eh oui, les fichiers pak sont en fait des fichiers compressés, qui dans notre cas sont utilisés pour stocker les fichiers principaux de Half-Life. Pourquoi les laisser dans un fichier compressé ? Pour que ça prenne moins de place sur le disque, tout simplement

Allez par exemple dans le dossier "maps", et là vous verrez la quantité de maps utilisées dans l'aventure de Half-Life !



L'intérêt de PakScape, c'est qu'il vous permet de "récupérer" certains fichiers, c'est-à-dire de les extraire du pak, afin que vous puissiez les lire, comprendre leur fonctionnement etc...
Vous pouvez même vous en servir pour créer votre propre pak, si vous réalisez un mod. En plus, comme le joueur standard ne connaît pas PakScape, vous protégez vos données (maps, fichiers de données etc...). Rien de tel pour rester discret !

Si je vous en parle, ce n'est pas tellement pour les fichiers de maps. Vous pouvez toujours essayer de décompiler un ou deux bsp, mais je vous préviens ça sera l'anarchie dans Worldcraft
En fait, nous allons nous pencher sur 2 éléments importants du moteur de Half-Life : les sprites et les models. D'ailleurs, vous noterez qu'il y a des dossiers du même nom de le pak. Les fichiers sprites portent l'extension .spr, et les fichiers de models .mdl.

Les sprites

Entité concernée : cycler_sprite
Type d'entité : entité-point
Difficulté : moyen

Nous aimerions maintenant voir à quoi ressemblent ces fameux sprites, portant l'extension .spr. Oui mais voilà, là encore nous n'avons pas de logiciel capable de les ouvrir (hormis Half-Life lui-même, mais ce serait peu pratique).
Bon, eh bien je crois que vous allez devoir vous farcir un autre download

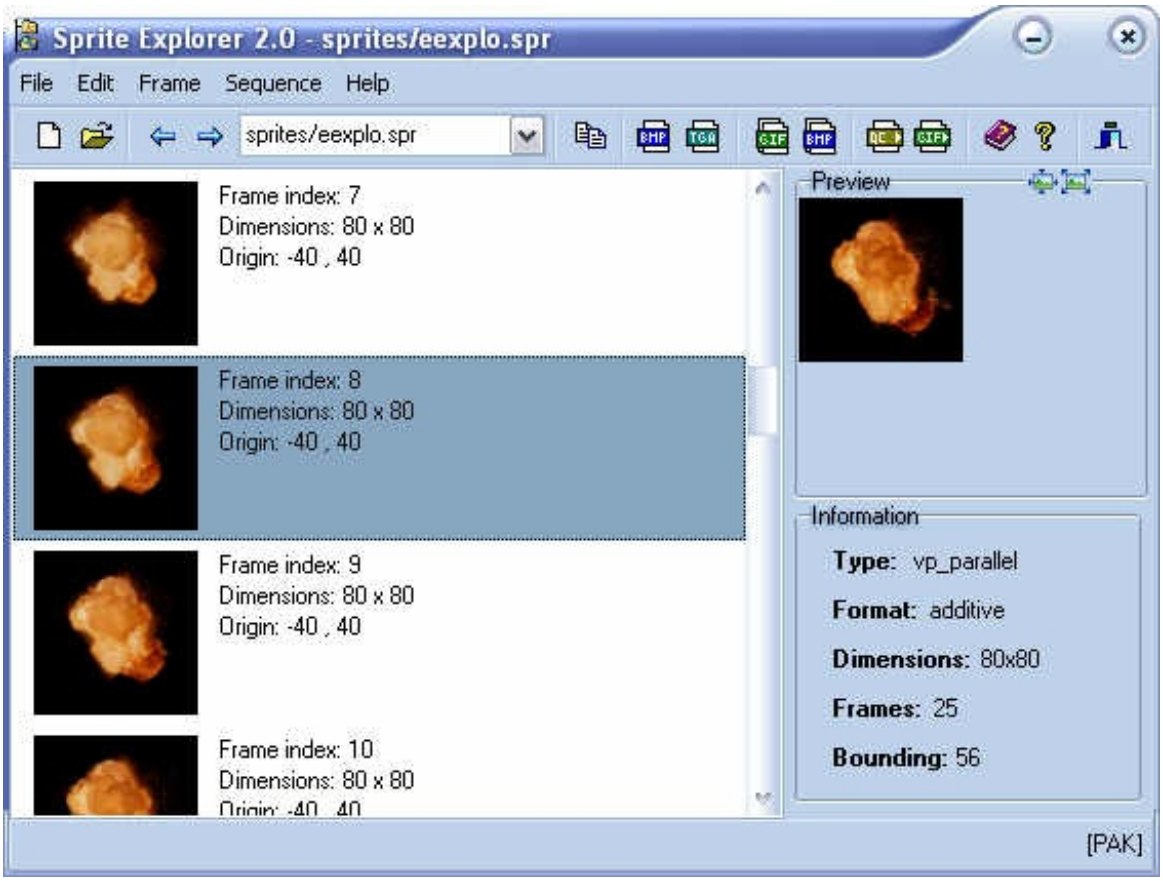
SpriteExplorer_v2.0.zip (864 Ko)

Installez le programme à l'aide du fichier setup.exe.
L'installation est propre, et vous verrez que le programme l'est aussi, puisqu'il vous permet non seulement de visualiser des sprites, mais aussi de les exporter dans d'autres formats, ou bien encore de créer les vôtres !
C'est un outil précieux que je vous confie, ne le perdez pas

Vous avez plusieurs possibilités pour ouvrir un sprite. Soit vous double-cliquez sur l'icône dans PakScape, soit vous extrayez le fichier hors de pak et vous l'ouvrez avec SpriteExplorer, soit enfin vous allez dans le menu "File / Open PAK file..." et vous sélectionnez le sprites que vous souhaitez ouvrir.

Nous allons pour commencer ouvrir eexplo.spr. Ouvrez-le en même temps que moi, et regardez ce que vous voyez. Une explosion ! Eh non vous ne rêvez pas, vous avez sous les yeux une superbe explosion comme on peut en voir dans Half-Life !
Ainsi les sprites sont, à première vue, une sorte de Gif animés... Vous pouvez d'ailleurs voir à gauche chaque image de

l'animation.
Notez que tous les sprites ne sont pas animés, donc certains sont constitués d'une seule image.



Mais quand utilise-t-on les sprites dans Half-Life ?

Tout le temps ! Et je dis bien : tout le temps. Vous voulez des exemples de sprites ? L'affichage de votre vie restante est un sprite, celui de l'explosion d'une grenade en est un autre, mais il ne faut pas oublier les étincelles, les vapeurs, les bulles d'eau etc etc... Ce sont tous des sprites !
Même dans Counter-Strike, on utilise de nombreux sprites (par exemple le menu pour acheter des armes).

Concrètement, les sprites sont des animations en 2D. Et j'insiste : en 2D seulement. Ca peut paraître bizarre de pense que les explosions ne sont que de vulgaires trompe-l'oeil, mais c'est bel et bien le cas ! Moi aussi je m'étais fait avoir, imaginant que les explosions étaient faites en 3D... Mais ce n'est pas le cas, preuve que le moteur de Half-Life commence à prendre vraiment de l'âge !
Les sprites peuvent être rendus transparents très facilement, tout comme les Gifs. Il suffit d'indiquer la "couleur transparente", que le moteur de Half-Life fera disparaître pendant le jeu.

Je vous laisse le soin de découvrir tous les sprites disponibles, peut-être en trouverez-vous certains que vous aurez envie de mettre dans votre map... Si c'est le cas, il est temps de vous expliquer comment faire.

Placez une entité-point *cycler_sprite* dans votre niveau. Vous avez 2 propriétés intéressantes :

- Sprite : chemin vers le sprite à utiliser. Je vous conseille d'extraire le sprite du pak avant, pour pouvoir indiquer sa position facilement sous Worldcraft. Vous pouvez ensuite supprimer le sprite que vous venez d'extraire sans souci. Half-Life saura le retrouver dans le pak.
- Frames per second : vitesse d'animation du sprite (laissez par défaut c'est bon comme ça). Si vous mettez 0, le sprite ne s'animera pas.

Voilà, à vous de bien placer le *cylcer_sprite* dans votre map, et surtout de trouver (ou créer) le sprite de vos rêves qui donnera une toute autre allure à votre map. Un conseil : ne répétez à personne que ce sont des animations en 2D ou bien tout le monde va finir par trouver ça moche, alors qu'en fait l'effet est excellent sous Half-Life (on n'y voit que du feu !)

Les models

Bien plus dur à manipuler (et encore plus à créer), les models sont un autre élément du moteur de Half-Life. Mais eux sont plus

évolués que les sprites.
Ces fichiers .mdl contiennent en fait de véritables personnages en 3D très détaillés, avec toutes leurs animations possibles et imaginables. En clair, après la 2D, on passe à l'étude de la 3D

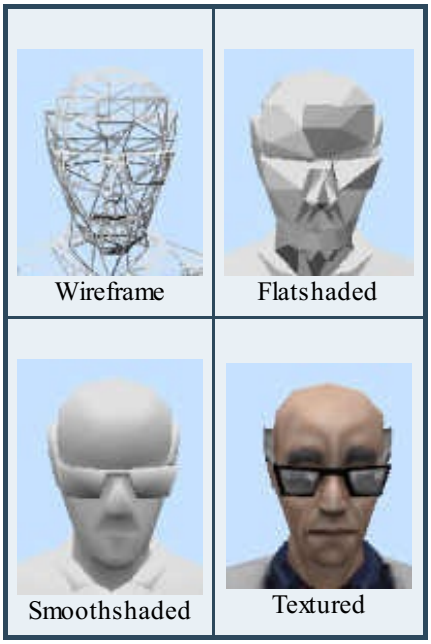
Où sont les models ? Les ennemis sont des models (vous remarquerez qu'ils sont plus détaillés qu'on ne pourrait le faire avec des blocs), les scientifiques aussi, les barney, les joueurs... mais aussi les armes et munitions au sol !

Je crois que nous allons nous poser la même question pour la troisième fois : comment ouvrir les models ? Lol, vous allez devoir faire un 3ème téléchargement. Mais n'hésitez pas à prendre ce programme, il vaut vraiment le coup d'oeil et vous allez en apprendre long sur les techniques de la 3D !

HL_Model_Viewer.zip (112 Ko)

Une fois le programme installé et lancé, allez dans le menu "File / Open PAK File...". Le programme va extraire tout seul les models que vous voulez visualiser.
Le contenu du pak apparaît à gauche. Rendez-vous dans le dossier "models", et ouvrez pour commencer "scientist.mdl".

Regardez-le, il est pas beau notre scientifique là comme ça ?
Notez que vous pouvez le visualiser de plusieurs manières (Wireframe, Textured etc...). Je vous conseille le Textured pour y voir plus clair, mais il peut être intéressant d'analyser le mouvement du personnage en mode Wireframe. Voyez ce que ça donne en fonction du mode sélectionné :



Vous connaissiez 3 de ces modes d'affichage dans Worldcraft. Le nouveau (Smoothshaded) est propre aux models : c'est une technique qui permet de lisser les polygones pour un rendu encore plus réaliste (et il est clair qu'entre Flatshaded et Smoothshaded y'a pas photo !)

En cochant un peu toutes les cases dans le premier onglet, vous verrez que les models sont des éléments complexes. Ils possèdent un squelette (bones), des hit boxes (selon l'endroit où on tire les dégâts seront différents)...

Comme c'est un objet en 3D, vous pouvez faire bouger la caméra autour sans problème ! Voici les commandes :

- Bouton gauche de la souris : maintenez le bouton gauche de la souris pour tourner autour du personnage.
- Bouton droit de la souris : maintenez le bouton droit de la souris pour zommer ou dézoomer.
- Shift + Bouton gauche de la souris : maintenez le bouton gauche de la souris en même temps que la touche Shift (en-dessous de majuscule) pour déplacer le personnage sur l'écran.

Continuons notre découverte... Allons dans l'onglet Sequence. Je crois que c'est là le plus impressionnant : chaque sequence que vous pouvez sélectionner est une animation que le personnage peut faire !
Pour l'instant, le scientifique marche (walk). Mettez la séquence "fear1". Impressionnant non ? Voilà un scientifique courageux dans toute sa splendeur 

Vous pouvez tester toutes les animations, vous allez voir qu'il y en a une quantité astronomique ! Eh oui, le travail de Valve ne consistait pas seulement à mapper, mais aussi à modeler et à skinner. Tiens, à propos de modeler, sachez que HL Model Viewer ne permet pas de modifier les models. A dire vrai, je ne connais aucun programme gratuit permettant de le faire. J'ai pu tester Milkshape 3D (un des plus connus) pendant 30 jours, et je peux vous dire que non seulement c'est très difficile, mais en plus on peut y passer des semaines à créer un model avec ses animations !

Quand on voit tous les models que Valve a réalisé, ça laisse songeur... Enfin bref, je vous laisse ouvrir quelques autres models pour que vous puissiez vous faire une idée. Notez que certains models ne sont pas animés.

Nous avons vu le principal sur les models, le reste je vous laisse le soin de le découvrir vous-même en cliquant un peu partout 
J'espère que vous avez pris plaisir à lire ce chapitre, car il contient une foule de nouveautés toutes aussi intéressantes et enrichissantes. Nous allons justement nous resservir de ce que nous venons d'apprendre dans le prochain chapitre...

L'IA : Intelligence Artificielle

Allons bon... vous ne saviez pas que les marines disposaient d'une intelligence (très) artificielle ? ;o)
Ne nous moquons pas toutefois, parce qu'en activant le niveau difficile ils deviennent assez coriaces, et réagissent assez bien à ce que nous faisons. Certes, aujourd'hui d'autres jeux existent qui prennent en charge une IA encore plus développée, mais vous verrez que celle de Half-Life est déjà assez complète et que l'on peut faire bien des choses...

Nous verrons donc comment ajouter des monstres dans une map (c'est facile vous verrez), puis comment les faire bouger et exécuter certaines actions (ce qui est un peu plus dur...).

Ajouter des monstres

Entités concernées : entités commençant par "monster_"
Type d'entité : entités-point
Difficulté : assez facile

Vous verrez : le terme "monstre" est trompeur. Voici ce que l'on trouve dans les entités "monster_" :

- Des aliens et des marines : oui, eux ce sont des monstres, des vrais, des méchants, qui ne cherchent qu'à vous découper en petits morceaux.
- Des objets explosifs : ce sont par exemple les mines lasers (*monster_tripmine*). Celles-ci, contrairement à *weapon_tripmine*, sont déjà posées à l'angle d'un couloir et attendent patiemment le joueur... ce sont donc des ennemis.
- Des personnages alliés : scientifiques, barneys etc... Ils ne veulent pas votre mal (en théorie) et cherchent à vous aider. Utiles pour créer un scénario et mettre de l'ambiance !

Comme vous pouvez le voir, les "monstres" sont variés, et ils sont très nombreux ! Je vous laisse le soin de les découvrir et de les placer dans les niveaux à votre guise.
Comme le joueur, surélevez un peu l'entité du sol pour éviter que le monstre ne reste coincé au départ...

Voici 2 propriétés propres aux monstres que vous devez connaître. Ce sont des triggers un peu particuliers :

- TriggerTarget : nom de la cible à déclencher lorsque la condition de "Trigger Condition" est remplie (voir ci-dessous).
- Trigger Condition : indiquez une condition qui déclenchera le trigger. Par exemple, si vous mettez "50% Health Remaining", une entité sera appelée lorsque le monstre n'aura plus que la moitié de sa vie (pourquoi pas un tremblement de terre pour mettre de l'ambiance )
  - No trigger : rien ne se passera, l'entité n'appellera aucune cible (par défaut).
  - See Player, Mad at Player : lorsque le monstre a vu le joueur et l'a en ligne de mire. Bref, il est en train de lui foncer dessus 
  - Take Damage : lorsque le joueur a blessé le monstre.
  - 50% Health Remaining : lorsque le monstre n'a plus que 50% de sa vie.
  - Death : lorsque le monstre est mort.
  - Hear World : lorsque le monstre arrive dans la partie.
  - Hear Player : lorsque le monstre entend le joueur arriver.
  - Hear Combat : lorsque le monstre entend le joueur combattre d'autres monstres.
  - See Player Unconditionnal : lorsque le monstre voit le joueur.
  - See Player, Not in Combat : lorsque le monstre voit le joueur, mais seulement s'il n'est pas déjà en train de combattre.

Au niveau des flags, on a généralement ceux-ci :

- WaitTillSeen : attend de voir le joueur avant de commencer à bouger.
- Gag : empêche le monstre de parler (idéal pour faire taire un scientifique).
- MonsterClip : le monstre sera bloqué par l'entité *func_monsterclip*, qui fonctionne comme la texture CLIP, sauf que celle-là concerne uniquement les monstres. Le joueur ne sera donc pas bloqué par elle.
- Prisoner : le monstre restera immobile, il ne réagira à rien.
- WaitForScript : attend un script pour agir.
- Pre-Disaster : dans l'histoire de Half-Life, il y a deux moments. Le premier, au tout début, c'est quand tout va bien, que tout est calme... avant le désastre. Le second, ben c'est après l'expérience de Gordon qui a tourné au cauchemar. Si vous cochez cette case, les monstres (en particulier les scientifiques et les barneys), réagiront comme si tout allait bien (première période donc). Sinon, ils parleront entre eux des aliens, des explosions et de toute cette joyeuse apocalypse 
- Fade Corpse : le corps disparaît une fois que le monstre est mort.

Mais suffit-il de placer un monstre pour qu'il fonctionne dans votre niveau ?
Eh bien non ! Vous allez voir que l'on doit au moins leur donner des nodes...

Utilisation des nodes

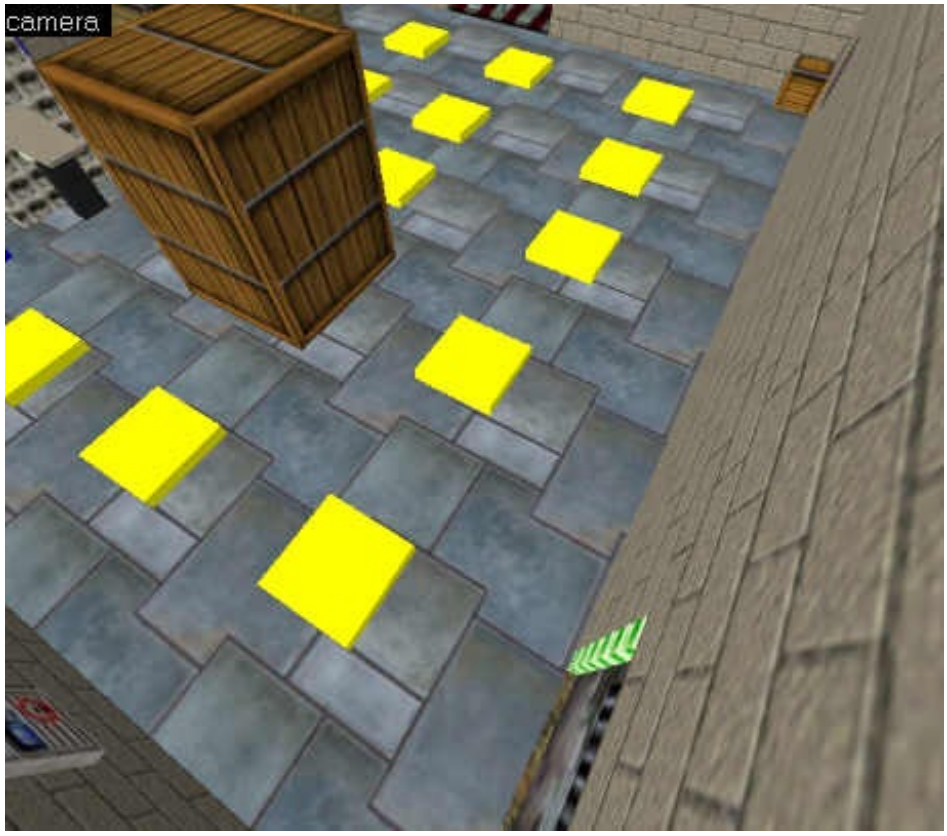
Entités concernées : info_node et info_node_air
Type d'entité : entités-point
Difficulté : très facile

Imaginez que les *info_node* sont comme des waypoints pour les bots. Les monstres ont besoin de savoir où ils peuvent aller dans votre niveau... ce qui évitera donc qu'ils se prennent bêtement un mur !

Le code de Half-Life se charge de dire au monstre d'attaquer, riposter, fuir etc... Vous, vous devez juste indiquer au monstre où il peut aller. C'est tout

Ne placez les *info_node* que lorsque votre map est terminée. En effet, si vous la modifiez après, vous aurez peut-être besoin de déplacer les *info_node*, et dans ce cas bonjour la galère !

Les *info_node* ressemblent sous Worldcraft à de grands rectangles jaunes. Il faut les placer tout près du sol, à une distance raisonnable les uns des autres. Evitez aussi de les placer trop près des murs si le monstre est grand, car il pourrait se coincer dedans, intelligent comme il est



Tout ce que vous devez retenir, c'est qu'un monstre va d'un node à un autre, sans se soucier de savoir s'il y a quelque chose au mileu. Veillez donc à ce que rien ne gêne le passage du monstre entre 2 nodes !

Lorsque vous lancez la map, un message "Node graph out of date. Rebuilding..." apparaît, et fait travailler votre ordinateur quelques secondes. Que se passe-t-il ? Half-Life génère un fichier nomdevotremap.nod dans le dossier maps/graphs. Celui-ci contient les coordonnées liant les nodes de votre map, très importantes pour que les monstres sachent où se déplacer... Pensez à livrer ce fichier avec votre map pour éviter que les joueurs ne soient bloqués quelques secondes avant le lancement de la partie !

Animation des monstres

Entité concernée : scripted_sequence
Type d'entité : entité-point
Difficulté : difficile

Animer un monstre est un peu plus compliqué. Si vous voulez qu'il effectue un mouvement précis, il va falloir vous servir de l'entité *scripted_sequence*.

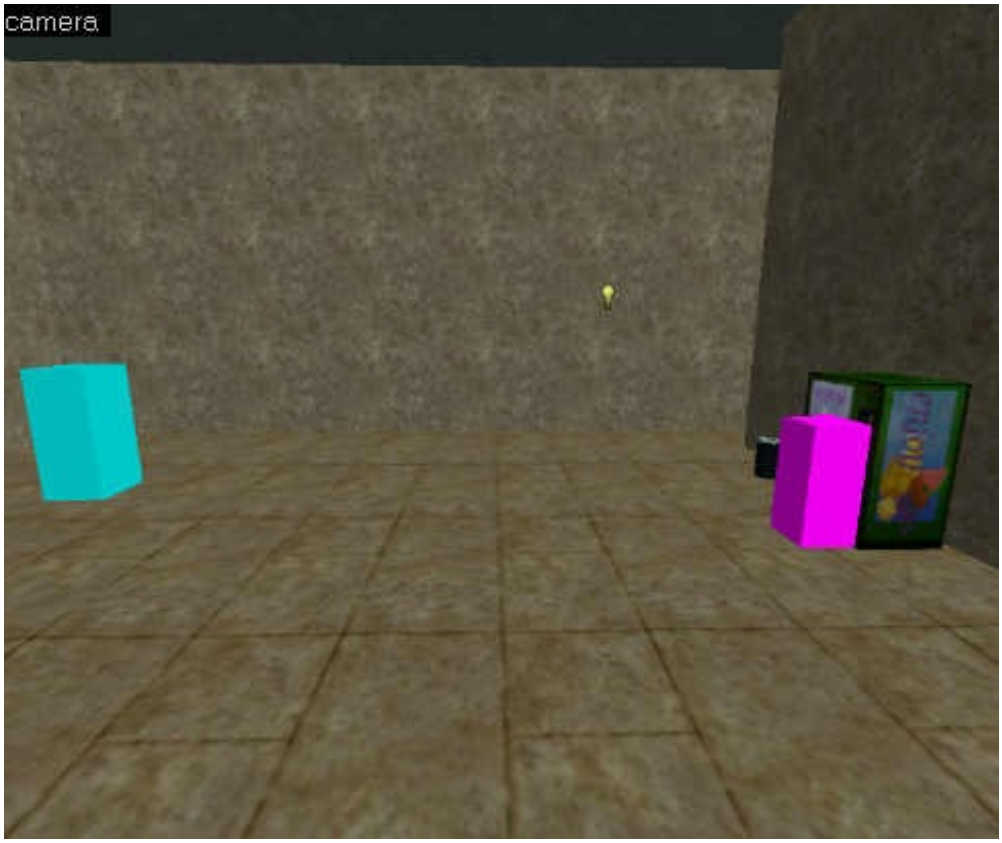
A mon avis, le meilleur moyen de vous faire comprendre comment ça fonctionne, c'est un exemple. Je vous propose cette petite

histoire : un scientifique est déshydraté, il va au distributeur de boissons, mais manque de chance sa pièce reste coincée et il se met à frapper la machine comme un forcené.
Bon, c'est pas vraiment un scénario digne de Spielberg, mais c'est un bon début pour commencer notre carrière de cinéma Si vous arrivez à réaliser cette scène, vous saurez tout faire (il suffit d'adapter le décor). Suivez bien ce que je vous dis !

Commencez par créer une salle, que vous décorerez à votre guise (pour ma part je m'amuserai pas à faire des décors ;). Placez à un bout de la salle un *monster_scientist* (un scientifique), et à l'autre bout de la salle un distributeur de boissons (textures GENERIC 106). Placez une entité *scripted_sequence* juste devant le distrib'.

Vous devez placer le *scripted_sequence* juste au niveau du sol (au pixel près). Sinon, ça ne marchera pas. Par ailleurs, pensez à "coller" cette entité devant le distributeur, parce que s'il y a un écart on ne comprendra pas dans quoi le scientifique met sa pièce

Voici ce que vous devriez voir :



A gauche, le scientifique, à droite le distributeur avec le *scripted_sequence* collé juste devant.
Notez que rien ne doit gêner le passage du scientifique au milieu !

Commencez par donner un nom (name) au scientifique. Nous l'appellerons "chercheur" dans notre exemple.
C'est tout ce que vous aviez à faire pour le scientifique.

Passons maintenant au *scripted_sequence*. Voici les propriétés utiles que vous devez modifier pour la plupart :

- Angle du radar : modifiez l'angle du radar en haut à droite de manière à ce que cette entité soit dirigée vers le distributeur. Sinon, le scientifique risque d'effectuer son animation du mauvais côté !
- Name : nom de l'animation. Dans notre exemple, nous appellerons cette animation "boire".
- Target Monster : nom du monstre qui va effectuer l'action. Ici, il s'agit de "chercheur".
- Action Animation : c'est l'action que va effectuer le monstre. Il faut choisir la bonne, sachant qu'il doit y en avoir quelques centaines rien pour les scientifiques !

  Mais comment je trouve le nom de l'animation ?

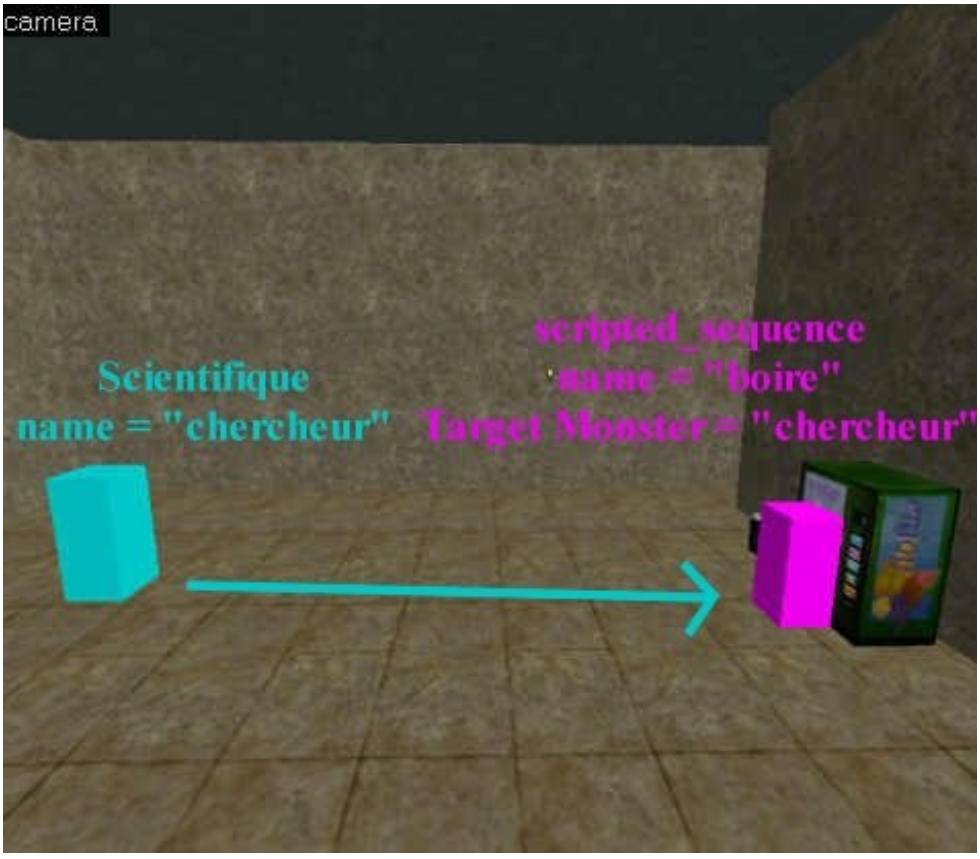
  Pourquoi croyez-vous que je vous ai parlé de Half-Life Model Viewer dans les précédents chapitres ? Lancez ce programme, ouvrez le pak de Half-Life, et sélectionnez "models/scientist.mdl". C'est le model du scientifique.
  Allez dans l'onglet "Sequence", et là choisissez la séquence qui vous plaît Dans notre cas, la séquence qui nous intéresse s'appelle "buysoda", c'est donc cela que nous allons écrire dans la champ de l'attribut "Action Animation".
- Idle Animation : ce n'est pas vital de remplir ce champ. C'est le nom de l'animation que le monstre effectuera lorsqu'il n'aura plus rien à faire. Il sera donc au stade "idle", et la répétera autant de fois que nécessaire, avant qu'il ne soit

perturbé.
- Move to Position : cela détermine de quelle manière le monstre se rend au *scripted_sequence*.
  - No : par défaut. Le monstre ne se rendra pas au *scripted_sequence* pour effectuer l'action. Vous pouvez donc placer cette entité où vous le voulez : le monstre restera à la place où il est pour faire son animation.
  - Walk : le monstre marchera en direction du *scripted_sequence*. C'est cette valeur que nous allons mettre pour notre exemple.
  - Run : le monstre courra en direction du *scripted_sequence*.
  - Instantaneous : le monstre se retrouve instantanément comme par magie à l'emplacement du *scripted_sequence*. C'est un peu une téléportation quoi
  - No - Turn to Face : le monstre ne bougera pas de là où il est, mais il regardera en direction du *scripted_sequence*.

Au niveau des flags, on a :

- Repeatable : cette animation pourra être déclenchée plusieurs fois.
- Leave Corpse : si le monstre meurt durant son animation, son corps disparaît.
- No Interruptions : rien n'empêchera le monstre d'effectuer son action jusqu'à la fin, même si le joueur lui envoie 3 roquettes dans la face
- Override AI : le code de Half-Life ignorera l'intelligence artificielle du personnage. Il effectuera son action quoiqu'il arrive.
- No Script Move : ne replace pas le personnage à la fin du script.

Schématiquement ça donne ça :



Pour activer l'animation, il va falloir utiliser un trigger. L'entité à activer est le *scripted_sequence*.
Placez donc un *trigger_once* (avec Target = "boire") pas trop loin dans la map : dès que le joueur rentrera dedans, le scientifique ira boire un coup... enfin, il essaiera du moins

Hop ! Je suis allé vous faire un petit screenshot sous Half-Life pour vous prouver que tout cela marche :

Pas mal hein ?

Ce n'est pas évident à comprendre du premier coup, mais je vous conseille de persévérer. Une fois qu'on a compris le fonctionnement des *scripted_sequence*, on ne les lâche plus

Comme vous pouvez le voir, l'intelligence artificielle de Half-Life est pleine de ressources cachées ! Je parie que bon nombre d'entre vous auront envie d'essayer de mapper pour Half-Life solo après avoir lu ce chapitre... et pourquoi pas, de développer un mod si vous vous en sentez le courage, et si vous trouvez suffisamment de personnes pour vous aider !

Fin de niveau

Quoi ? Déjà fini ?
Même si Half-Life permet de créer des cartes assez grandes, il y a des limites à tout. Et pourtant, Valve a trouvé une astuce (qui n'a rien d'extraordinaire), qui consiste à lier 2 niveaux entre eux. En clair, quand on arrive au bout d'une map, un message "Chargement..." s'affiche, et en quelques secondes on se retrouve exactement au même endroit, mais sur une autre map : cela nous permet de continuer l'aventure !
Tous les niveaux de Half-Life fonctionnent donc comme cela : ils sont reliés entre eux grâce à une technique que je vais vous enseigner.

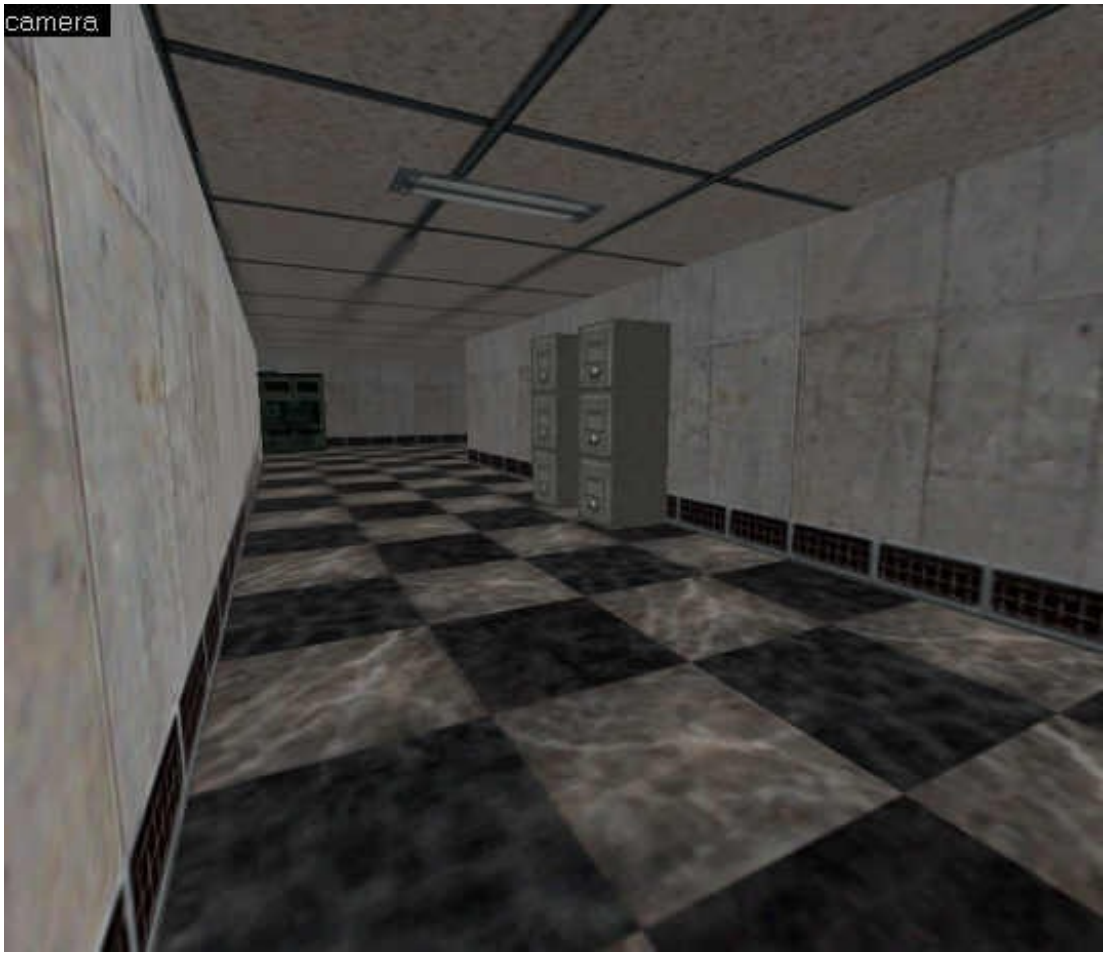
Par ailleurs, nous verrons aussi comment mettre fin à une partie... solo, bien entendu. Ce chapitre ne s'adresse donc qu'à ceux qui font une aventure Half-Life, ou qui souhaitent créer une aventure solo pour leur propre mod...

Transition entre 2 niveaux

Entités concernées : info_landmark et trigger_changelevel
Type d'entités : entité-point et entité-bloc
Difficulté : moyen

Tout d'abord, il faut savoir que la zone de transition entre 2 niveaux s'effectue toujours dans un couloir, pratiquement vide de préférence. Ce que je vous demande donc de faire sur votre map, c'est une zone pratiquement vide de tout décor : on ne doit pas voir de "salle" derrière ni devant, il ne doit y avoir pratiquement aucune visibilité. Juste quelques murs.

Voici par exemple mon couloir de transition :



J'ai placé mon angle de vue à l'endroit où se trouve le joueur lorsqu'il arrive. Il vient donc de derrière et compte aller à l'autre bout du couloir... sauf que ce couloir est spécial, parce que c'est à cet endroit que Half-Life va changer de map (ni vu, ni connu, ou presque ;).

Commencez par placer d'abord une entité-point *info_landmark*. Mettez-la à peu près au niveau du sol. Donnez-lui un nom dans ses propriétés ("landmark" par exemple).
Ensuite, un peu plus loin, créez un bloc avec la texture AAATRIGGER, que vous transformerez en entité-bloc *trigger_changelevel* (il s'agit ni plus ni moins d'un nouveau type de trigger que vous ne connaissiez par encore). Editez ses propriétés, en indiquant le nom de la nouvelle map (la map 2) dans "New map name", ainsi que celui de l'*info_landmark* dans

"Landmark name" (nous l'avions appelé "landmark" par exemple).

Ca devrait donner :

Comme vous pouvez le voir, il est très important de mettre l'*info_landmark* **AVANT** le trigger.

Le joueur n'est pas obligé de "rentrer" dans l'*info_landmark*. Tout ce que je vous demande, c'est de le mettre avant le trigger.
Quant au trigger, faites en sorte que le joueur ne puisse pas le rater, car là il doit rentrer dedans. Sur mon exemple, il fait toute la largeur du couloir...

Une fois que c'est fait, sélectionnez cette portion de couloirs (un peu avant et un peu après), et faites Copier (Ctrl + C).
Ouvrez la nouvelle map, et faites Coller (Ctrl + V).
Normalement il doit y avoir des trous (de gros leaks) puisque vos couloirs ne sont pas fermés. Pensez donc à les fermer dans un premier temps pour éviter un Leak.

C'est là qu'il ne faut pas se tromper... Vous devez déplacer UNIQUEMENT le *trigger_changelevel*. Rien d'autre, et surtout pas l'*info_landmark*, ni même la position des murs. Tout doit être exactement à la même place, vous ne devez rien modifier dans la zone de transition.
Décalez le trigger_changelevel dans le couloir, de sorte cette fois à ce qu'il se trouve avant l'*info_landmark*. Modifiez dans ses propriétés le nom de la map à charger (la première map 1 cette fois).
Ainsi, si vous vous êtes bien débrouillé, lorsque le joueur voudra faire demi-tour et revenir dans la map 1, il passera d'abord par l'entité *info_landmark*, puis par le trigger.

Surtout rappelez-vous : ne touchez PAS à l'*info_landmark* ! Si vous voyez de gros bugs lorsque la transition s'effectue, vous saurez pourquoi : il ne fallait pas toucher cette entité, ni même la position des murs !

Et voilà qui est fait ! Votre transition s'effectuera en douceur pour le joueur, et vous pourrez créer une looongue aventure (une succession de maps) comme dans Half-Life !

Terminer la partie

Entités concernées : trigger_endsection
Type d'entités : entité-bloc
Difficulté : facile

Je ne vais pas faire long, car cette entité est un trigger (et vous êtes censés savoir vous en servir), et elle est très simple à

configurer.
Sa fonction ? Elle met fin à la partie et retourne au menu principal. C'est donc à la fin de l'aventure que vous devrez vous servir de cette entité.

Lorsque le joueur traverse le trigger, la partie est donc arrêtée. Au niveau des attributs, "Section" vous propose de faire 3 fois la même chose ^^. Ne vous prenez pas la tête, ces fonctions sont identiques... Evitez toutefois de mettre _oem_end_demo, parce que celui-là en plus charge une page HTML qui ne sert que pour Half-Life : Uplink Demo.
Le chapitre était court, mais nécessaire ! Désormais, vous avez toutes les clés en main pour créer une aventure aussi longue que Half-Life (ou même plus longue encore )

Counter-Strike

Nous allons voir dans ce chapitre la plupart des entités spécifiques à Counter-Strike, en fonction du type de map que vous allez faire (CS, DE, AS ou ES).

Ce chapitre ne vous apprend pas à créer une map Counter-Strike ! Il faut avoir lu le reste du cours pour savoir créer une map ! Ici nous ne faisons qu'étudier les particularités du mapping Counter-Strike.

Je vous rappelle que vous n'avez pas à indiquer à Worldcraft quel type de map vous faites. C'est Counter-Strike qui déterminera les objectifs de la map en fonction des entités qu'il y a (exemple : s'il y a des otages alors il saura que c'est une map de sauvetage d'otages).

Maps CS

Le type de map CS (Sauvetage d'otages) est le tout premier qui soit apparu avec Counter-Strike. Le but pour les Counters est de ramener des otages à un point de sauvetage (Rescue).

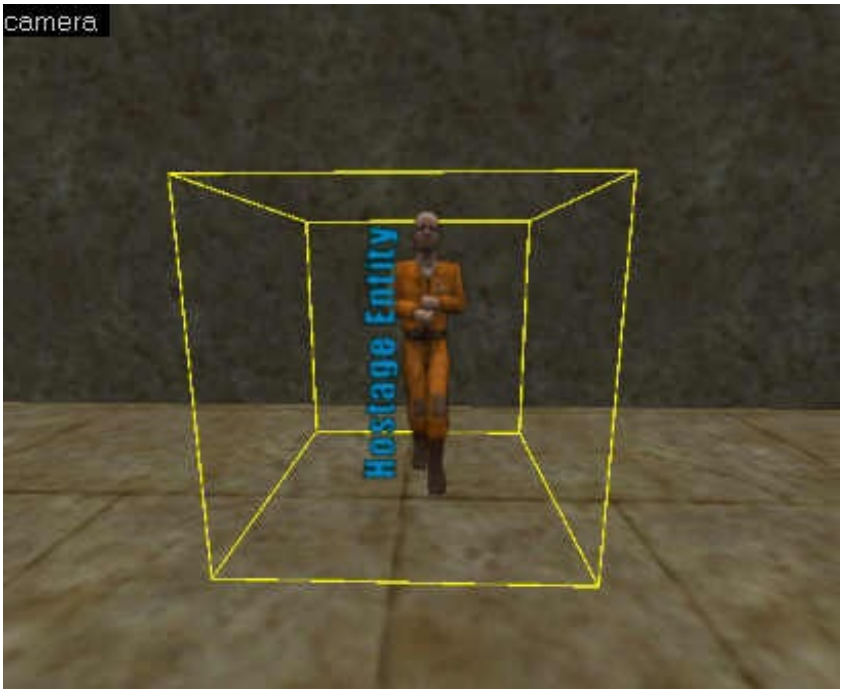
Cela fait qu'on se servira de 2 entités différentes :

- L'entité représentant un otage
- L'entité indiquant le point de sauvetage (Rescue)

Les otages

Entité concernée : hostage_entity
Type d'entité : entité-point
Difficulté : facile

Un *hostage_entity* fonctionne exactement comme le départ d'un joueur. Pensez donc à le surélever un peu du sol pour éviter qu'il ne reste coincé !



Placez autant de *hostage_entity* qu'il ne doit y avoir d'otages. Généralement, il y a 4 otages dans une map CS.

Vous pouvez éditer ses propriétés si vous le désirez mais ce n'est pas obligatoire...

- Pitch Yaw Roll (X Y Z) : sert à l'orientation de l'entité. Utilisez plutôt le radar en haut à droite de la fenêtre pour indiquer dans quelle direction l'otage doit "regarder".
- Model : vous pouvez choisir un modèle d'otage custom si vous en avez créé un. Je vous recommande de ne pas modifier le modèle de l'otage.

- Skin : choisissez le skin de l'otage :
  - Orange Suit Worker : un travailleur dans une zolie tenue orange. Utile pour les maps se déroulant dans des chantiers par exemple...

  - Shirt & Tie : un civil dans une tenue de boulot (cravate livrée en série). Personnellement, je trouve que ce skin est meilleur car on peut le mettre dans la plupart des maps se déroulant dans la vie réelle.

Voilà, les otages sont maintenant disposés dans votre map !

Les points de sauvetage

Entités concernées : info_hostage_rescue ou func_hostage_rescue
Type d'entité : respectivement entité-point ou entité-bloc
Difficulté : très facile

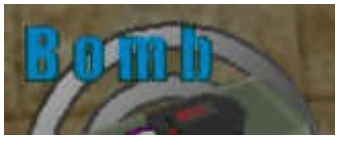
Comme vous pouvez le voir, vous pouvez vous servir de 2 entités pour indiquer le point de sauvetage : une entité-point ou une entité-bloc. L'entité-point est très simple à utiliser : il suffit de la mettre là où il y a un point de sauvetage.
Pour l'entité-bloc c'est pareil, sauf que là vous devrez créer un bloc avec la texture AAATRIGGER comme vous avez appris à le faire. Cette technique est plus précise que l'entité-point.

Il n'est pas interdit de mettre plusieurs points de sauvetage si votre map est assez complexe.

Il n'y a vraiment rien à ajouter, c'est une entité tellement simple à utiliser...

Maps DE

Entités concernées : info_bomb_target ou func_bomb_target
Type d'entité : respectivement entité-point ou entité-bloc
Difficulté : très facile



Le type DE est le deuxième apparu avec Counter-Strike, et c'est aujourd'hui sans conteste l'un des plus populaires, avec des maps comme de_dust...

Dans une map DE, on ne se servira que d'une seule entité : celle qui indique le point de

pose de la bombe. Counter-Strike donnera alors la bombe à l'un des terros au hasard lorsque vous jouerez.

Là encore, vous avez le choix entre 2 entités : l'entité-point ou l'entité-bloc. Il vous suffit de placer l'entité à l'endroit désiré.

Les créateurs de Counter-Strike précisent que l'on peut mettre au choix 1 ou 2 points de pose de la bombe. Pas plus

Cette fois, je vous recommande de vous servir de l'entité-bloc. Créez un bloc avec la texture AAATRIGGER et faites-lui couvrir toute la zone où l'on pourra placer la bombe. Transformez-le en entité *func_bomb_target*.
Non seulement cette technique est plus précise, mais en plus cette entité possède un attribut très intéressant :

- Target (when bomb blows) : cible à déclencher lorsque la bombe aura explosé. Cela vous permet par exemple d'appeler un *multimanager* qui à son tour déclenchera d'autres explosions, cassera des objets etc etc... Vous pourrez même jouer un son si vous le désirez !

Maps AS

Les maps AS sont rares, et c'est assez dommage car c'est pourtant un type de map intéressant à jouer et à mapper. Il ne tient qu'à vous de produire plus de maps AS

Le principe est le suivant : les Counters doivent escorter un VIP jusqu'à un point de sauvetage (Rescue), et les Terros feront tout pour assassiner le VIP.

Ici, on a besoin de 2 entités :

- Le départ du VIP.
- Le point de sauvetage du VIP.

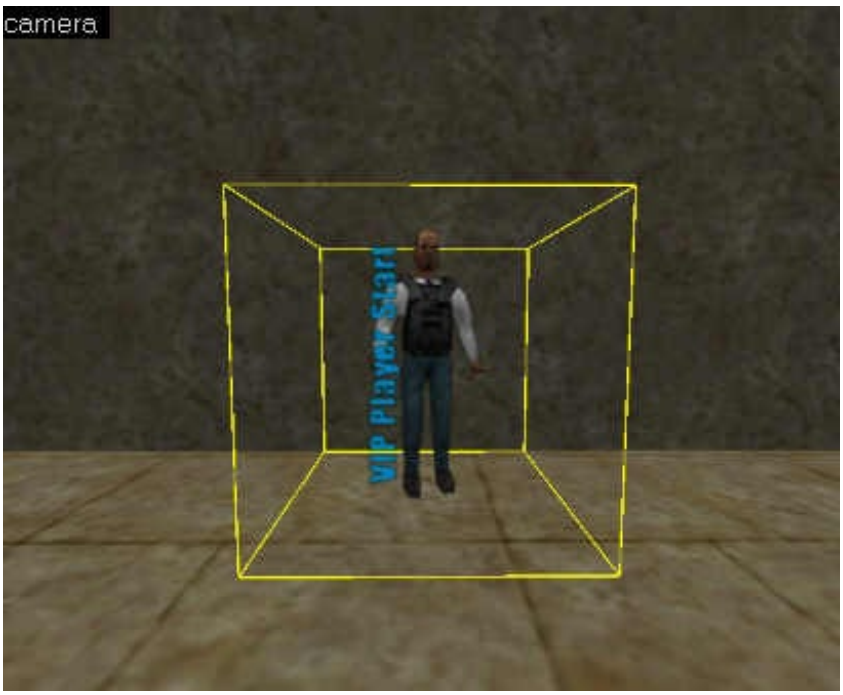
Départ du VIP

Entités concernées : info_vip_start
Type d'entité : entité-point
Difficulté : très facile

Le départ du VIP est indiqué grâce à l'entité *info_vip_start*. Elle fonctionne comme toutes les entités de départ de joueur.

Vous ne devez mettre qu'un seul VIP, et il doit commencer au même endroit que les Counters !

Sous Worldcraft, le VIP ressemble à ceci :



Rappelez-vous de surélever un peu le VIP pour qu'il ne reste pas coincé au sol, et tout ira bien

Point de sauvetage du VIP

Entités concernées : func_vip_safetyzone
Type d'entité : entité-bloc
Difficulté : ultra facile

Cette fois, vous ne pouvez utiliser qu'une entité-bloc pour déterminer le point de sauvetage du VIP.

Créez un bloc avec la texture AAATRIGGER à l'endroit où le VIP pourra s'échapper (et donc faire gagner les Counters). Cliquez sur "To Entity" et choisissez l'entité *func_vip_safetyzone*.

Et voilà qui est fait

Maps ES

Entités concernées : func_escapezone
Type d'entité : entité-bloc
Difficulté : ultra facile

Ca, c'est ce qu'on peut appeler un type de map oublié. En effet, aucune map ES n'est livrée dans la version officielle de Counter-Strike, et je parie que bon nombre d'entre vous ne savaient pas que ça existait ! Pourtant, ce type de map est vraiment intéressant et il mérite de revoir le jour. Il ne tient qu'à vous de créer des maps ES !

Le but pour les terros est de s'échapper (d'une prison par exemple, ou bien de passer la frontière), et les Counters doivent les en empêcher (c'est-à-dire tous les tuer jusqu'au dernier ;). Au bout de 5 rounds il me semble, les rôles sont inversés : c'est au tour des Counters de s'échapper !

En tant que mappeur, votre rôle sera d'indiquer le (ou les) points où on peut s'échapper.
Pour ce faire, créez un bloc avec la texture AAATRIGGER, et transformez-le en entité-bloc grâce au bouton "To Entity".
Choisissez ensuite l'entité *func_escapezone* dans les propriétés de l'entité.

C'est tout ! Venez pas me dire que le mapping c'est dur quand même !
Grâce à ces informations, vous êtes normalement capable de créer n'importe quelle map pour Counter-Strike, à condition bien entendu de bien lire le reste du cours

Les autres types de maps dont je n'ai pas parlé ne sont pas des types "officiels". Je pense par exemple aux "Speed maps", "Knife maps" etc...
Pour ce type de maps, il n'existe aucune entité particulière et vous n'avez rien de spécial à faire : c'est plus du deathmatch à la Half-Life qu'autre chose... Personnellement, je trouve que ces maps sont sympas 2 minutes mais elles ne font pas le poids par

rapport aux "vraies" maps (enfin c'est mon avis )

Partie 3 : Compilation et optimisation des maps

(Jouez sur votre map, et améliorez votre technique de mapping
C'est la partie la plus difficile : elle vous apprend le mapping professionnel)

La compilation (A : tester sa map)

Au menu de ce chapitre : tout connaître sur le fonctionnement de la compilation. Rien de moins !

Lorsque vous maîtriserez à fond la compilation, vous serez parés pour pouvoir essayer votre map. Vous pourrez vous-même juger ce qu'elle a dans le ventre !

Intêret de la compilation

Pour commencer ce chapitre, nous allons répondre à une question existentielle :

Mais pourquoi dois-je compiler ma map ?

La réponse est à la fois simple et compliquée. Pour aborder le sujet, je vais m'intéresser au format de fichier.

Le format *.bsp

Si vous avez déjà regardé les fichiers de maps se trouvant dans le dossier "maps" de vos mods, vous vous êtes peut-être rendus compte que ces fichiers étaient tous au format *.bsp. Les fichiers BSP sont des maps compilées : si vous placez un fichier BSP dans le dossier "maps" et que vous démarrez Half-Life, vous verrez que vous avez la possibilité de jouer sur cette map.
Il vous suffit pour cela d'aller dans le menu "Multijoueurs / Partie réseau local (LAN) / Créer. La liste des maps disponibles pour le mod apparaissent.

Ces fichiers sont des maps compilées. Ils contiennent des instructions spéciales pour le moteur de Half-Life. N'importe qui est capable de les lire, même s'il n'a pas Worldcraft.

Les formats *.rmf et *.map

Maintenant, regardez dans quel format vous enregistrez vos maps sous Worldcraft. Vous avez le choix entre le format RMF et le MAP :

- Le RMF : ce format est propre à Worldcraft. C'est le seul et unique logiciel capable de lire ces fichiers. Que contiennent ces fichiers ? La position de chaque bloc avec sa texture, de chaque entité, la position de la caméra dans la vue 3D etc... Bref, tout ce qu'il y a dans votre map, plus quelques informations spécifiques à Worldcraft.
- Le MAP : c'est exactement la même chose que pour le RMF. Le fichier contient toutes les informations sur votre map : les blocs, les entités etc...
  Seulement, à la différence du RMF, on ne trouve pas dans ce format d'informations spécifiques à Worldcraft. Le format MAP est universel.

Les formats RMF et MAP correspondent donc à une map à cours de création. On ne peut pas jouer avec eux, car contrairement au RMF, ils ne contiennent pas d'instructions pour le moteur de Half-Life mais pour le logiciel de création de maps.

Il existe plusieurs logiciels pour créer une map Half-Life ou Counter-Strike... Worldcraft est cependant le plus utilisé de tous.
Grâce au format MAP on a pu garder une certaine compatibilité entre ces logiciels pour pouvoir lire une map non compilée.

Schéma

Bon, si on reprenait tout ça ? On se trimballe avec 3 formats de fichiers différents :

- BSP

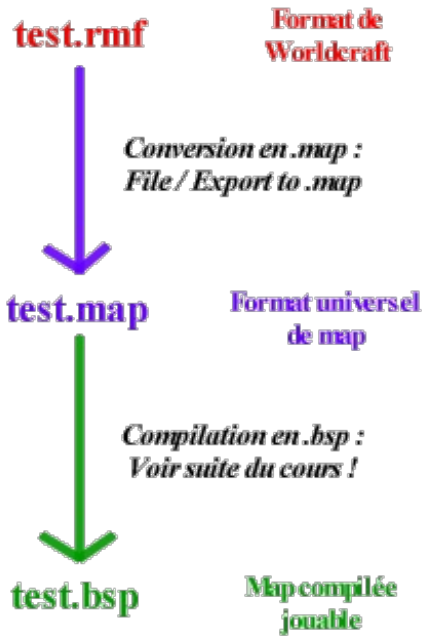
- RMF
- MAP

Dans notre cas, on a une map RMF que l'on veut transformer en BSP. Comment faire ? Vous avez la réponse sous vos yeux : en compilant la map !

Hélas ! Ce serait trop beau si c'était aussi simple... Les ZHLT (compilateurs chargés de transformer votre map en BSP jouable) ne peuvent lire que le format MAP. Ils ne comprennent pas le RMF.

Pour remédier à ce problème, il faudra dans un premier temps convertir le RMF en MAP. Worldcraft sait le faire : menu File / Export to .map. Ensuite, on passera le MAP aux ZHLT qui se chargeront de compiler votre map en BSP. C'est clair ?

Voici le schéma qui représente en gros ce que je viens de vous expliquer :



Vous pouvez déjà faire la conversion en MAP de votre map sous Worldcraft. Ce sera une bonne chose de faite.
Pour ce qui est de la compilation... c'est justement ce dont je vais vous parler

Comment ça marche ?

C'est LA question que bon nombre de mappeurs devraient se poser. Je pense à ceux qui compilent comme des bourrins une map pleine d'erreurs, et qui se demandent après pourquoi la compilation ne marche pas

Alors, à titre préventif, je vais commencer par vous expliquer les rouages de la compilation. Si vous comprenez bien la suite, alors vous pourrez déjà estimer que vous êtes un mappeur de bon niveau, qui comprend à quoi il a affaire. Ce qui est plutôt rare, ceci dit entre nous

Votre fichier .map va passer entre les mains de 4 programmes (fonctionnant en fenêtre MSDOS), qui sont les 4 compilateurs ZHLT.

Très important ! Pour ceux qui n'auraient pas déjà les ZHLT, il faut impérativement que vous les téléchargiez maintenant !!! Ce sont des compilateurs améliorés par rapport aux anciens qbsp, qrad.... qui étaient livrés avec Worldcraft.

Ils sont exécutés dans cet ordre :

- **hlcsg.exe**
- **hlbsp.exe**
- **hlvis.exe**
- **hlrad.exe**

Vous n'êtes pas obligés de lancer tous ces compilateurs. En théorie, les deux premiers suffisent (hlcsg et hlbsp). Mais je

ne vous cache pas que les deux suivants sont très fortement recommandés !

Chacun joue un rôle particulier dans la compilation. Nous allons voir à quoi ils servent plus en profondeur...

- hlcsg : c'est le tout premier compilateur. Son travail, c'est de *placer les polygones* (les blocs) dans l'espace grâce aux informations de votre fichier .map. Hlcsg s'occupe aussi de placer la plupart des entités. Bref, c'est un travail de positionnement.
  Il vérifie par ailleurs que les blocs ont une structure valide, c'est-à-dire qu'ils n'ont pas de forme non conforme aux lois de la géométrie
  Une fois qu'il a fini, il transmet les informations à hlbsp...
- hlbsp : il récupère le boulot qu'a fait hlcsg, et se charge de *créer un fichier .bsp compilé*. En théorie, une fois que c'est fait vous pouvez jouer sur votre map.
  Hlbsp analyse la totalité des blocs et doit déterminer s'ils forment tous une grande pièce hermétiquement close... Cela veut dire qu'il doit vérifier si votre monde a des limites, s'il est bien délimité.

  Si ce n'est pas le cas, alors il y a un LEAK dans votre map : un trou qui laisse entrevoir du vide. C'est une erreur courante qu'il vous faut absolument corriger pour pouvoir jouer.
- hlvis : c'est le premier des compilateurs facultatifs, mais il est très fortement recommandé de le démarrer. Son boulot à lui est très complexe, et peut parfois être très long. Il est à l'origine de plusieurs problèmes de mapping dont on reparlera souvent dans les prochains chapitres de ce cours de mapping.
  Bon, que fait-il concrètement ? C'est assez dur à imaginer : *il calcule la visibilité*. En fonction de chaque position que peut prendre le joueur, il doit déterminer les polygones qui seront calculés par le moteur de Half-Life, et ceux qui ne le seront pas.

  Hlvis permet au bout du compte d'obtenir un plus grand nombre d'images par seconde (aussi appelé fps pour frames per second), que ce soit sur une machine puissante ou une petite config'.

  S'il y a une très grande visibilité, votre map risque de ramer même si vous avez une GeForce 8. On en reparlera, ne vous inquiétez pas
- hlrad : dernier compilateur, et pas le moins important. Il se charge de *calculer la lumière* dans la map. Selon les sources lumineuses, les blocs seront plus ou moins éclairés...
  C'est un gros travail là encore, surtout si votre map se situe à l'extérieur et que vous utilisez une entité light_environement.

  Euh, mais que se passe-t-il si on n'exécute pas hlrad ? Il fait tout noir ?

  Non, au contraire ! Tout est éclairé avec l'intensité maximale, il n'y a aucune zone d'ombre sur la carte, et croyez-moi ça pique les yeux !

J'ai fait un effort pour ne pas parler des erreurs que pouvaient produire les compilateurs, même si c'est tentant

Voilà, maintenant vous avez une meilleure idée du fonctionnement de la compilation. Je ne vous en parlerai pas plus en détail. Après ça devient vraiment trop compliqué...

Je veux surtout que vous compreniez POURQUOI on compile une map. La compilation est très longue, vous le remarquerez vous-même. Mais cela permet par la suite de charger rapidement votre map, en moins d'une minute.

En effet, tous les calculs ayant été faits pendant la compilation, votre ordi n'a plus qu'à lire les résultats des calculs et afficher ce résultat (c'est le travail de nos chères cartes 3D). Aucun PC à l'heure actuelle n'est suffisamment puissant pour faire ces calculs en temps réel, désolé de vous décevoir

Compiler sous Worldcraft

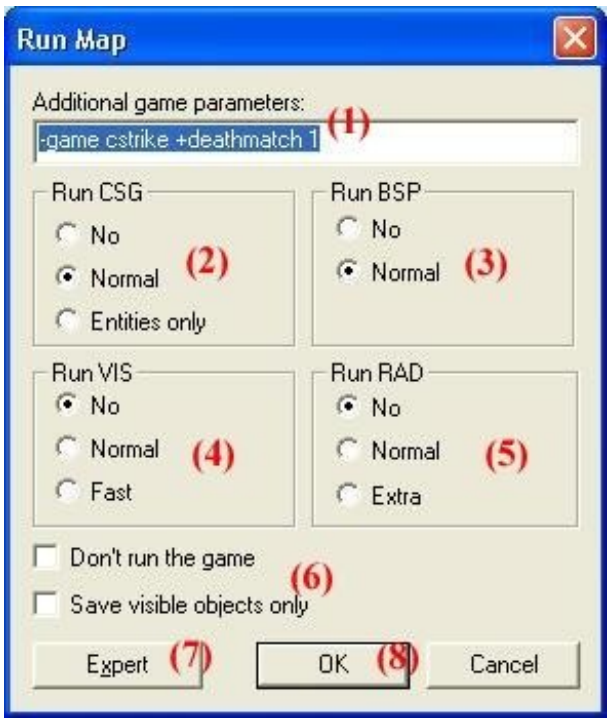
Si vous compilez à l'aide de Worldcraft, votre PC sera bloqué pendant toute la durée de la compilation ! Et ça peut être très long. Je vous explique rapidement à titre indicatif comment ça fonctionne, mais je vous conseille très fortement de sauter à l'étape suivante pour voir comment compiler à l'aide de ZHLT Compile GUI.

Bon, si vous êtes toujours décidés, alors on y va

Commencez par vérifier que vous avez bien configuré Worldcraft, parce que là ça devient très important !

Après avoir ouvert votre map sous Worldcraft, cliquez sur le bouton "Run Map!" , ou appuyez sur la touche F9.

Une nouvelle fenêtre doit apparaître :



1. Additional game parameters : ligne très importante, elle vous permet de donner les arguments à utiliser lorsque Half-Life sera lancé. Cela vous permet de donner une foule d'options, pour la plupart vitales si vous voulez pouvoir tester votre map correctement. Par exemple :

   -console

   Lance Half-Life en mode console (que l'on peut ouvrir avec la touche carré : ²) Très important et très utile.

   -game nomdumod

   Le mod choisi sera chargé. Pour Counter, il faut mettre "cstrike", TeamFortress "tfc" etc...

   +deathmatch 1

   Si c'est un map multijoueurs, comme c'est le cas pour Counter, alors vous devez mettre cet argument.

   -dev

   Mode développeur (vous). Ca vous affiche en haut à gauche de l'écran toutes les opérations effectuées par le moteur de Half-Life, et en temps réel s'il vous plaît !

   +r_speeds 1

   Permet d'afficher les r_speeds. C'est une commande avancée et difficile à exploiter pour les débutants, tout comme -dev. On en reparlera dans un prochain chapitre.
2. Run CSG : vous permet de définir la manière avec laquelle hlcsg sera lancé. Mettez "Normal".
3. Run BSP : pareil pour hlbsp. Vous devez mettre "Normal".
4. Run VIS : idem pour hlvis. Vous n'êtes pas obligés de le lancer mais c'est recommandé.
5. Run RAD : idem pour hlrad. A noter que le mode "Extra" permet d'améliorer la qualité de la lumière, mais c'est plus long bien entendu 
6. Don't run the game : si cette case est cochée, alors la map sera seulement compilée mais Half-Life ne sera pas lancé. Vous devrez le faire manuellement pour pouvoir tester votre map.
   Quant à Save visible objects only, un conseil, ne le cochez pas. On ne compilerait que les objets visibles, et il arrive parfois que l'on ait besoin d'objets en dehors de la map pour les faire fonctionner.
7. Mode Expert : trop compliqué pour nous, et pis de toutes façons ça ne nous intéresse pas !
8. OK : mmh... et si on cliquait dessus pour voir ? 

Après avoir validé, une nouvelle fenêtre s'ouvre. C'est là que vous verrez pourquoi il vaut mieux utiliser ZHLT Compile GUI plutôt que Worldcraft.

En effet, cette fenêtre utilise au maximum les ressources de votre PC, tout puissant qu'il soit. Du coup, tout est bloqué, on ne peut parfois plus bouger la souris !!!

En plus, on est mal mis au courant de l'avancement des opérations, et c'est très frustrant (on ne voit pas la pourcentage en temps réel). Bref, c'est la cata totale.

Mais bon, si vous attendez jusqu'au bout, alors vous pourrez voir le résultat sous Half-Life... à condition que ça n'ait pas planté bien entendu, dans ce cas vous devrez résoudre votre problème avant de pouvoir jouer (le chapitre suivant vous y aidera).

Si la compilation plante, Worldcraft tente quand même de lancer Half-Life sur la map... ce qui est débile puisqu'elle n'a pas été créée correctement.
Donc, si Half-Life vous affiche l'erreur "map not found on server", c'est que la compilation a planté. Votre map est donc buggée  Lisez le chapitre suivant pour y remédier.

Bon, je ne m'attarderai pas plus sur Worldcraft, et je vous montre illico dans le prochain paragraphe ce qu'est une VRAIE compilation.

Compiler avec le GUI

Ah.... ZHLT Compile GUI. Enfin un programme digne de ce nom  Il a été créé par un mappeur qui, comme vous, en avait marre d'utiliser Worldcraft pour compiler. Vous allez voir aussi que bientôt vous ne pourrez plus vous en passer !

Que fait ZHLT Compile GUI ?

Comme Worldcraft, il ne compile pas votre map : c'est le travail des ZHLT. Toute la différence se joue donc dans la manière de lancer les ZHLT : avec ce logiciel, vous aurez droit à un maximum d'options paramétrables ! Et le top du top, c'est que ça va très vite et vous pouvez même faire autre chose pendant que vous compilez. Bref, le paradis :).

Première étape : téléchargez le GUI (GUI est l'abréviation que l'on donne à ZHLT Compile GUI, c'est plus court ;o).

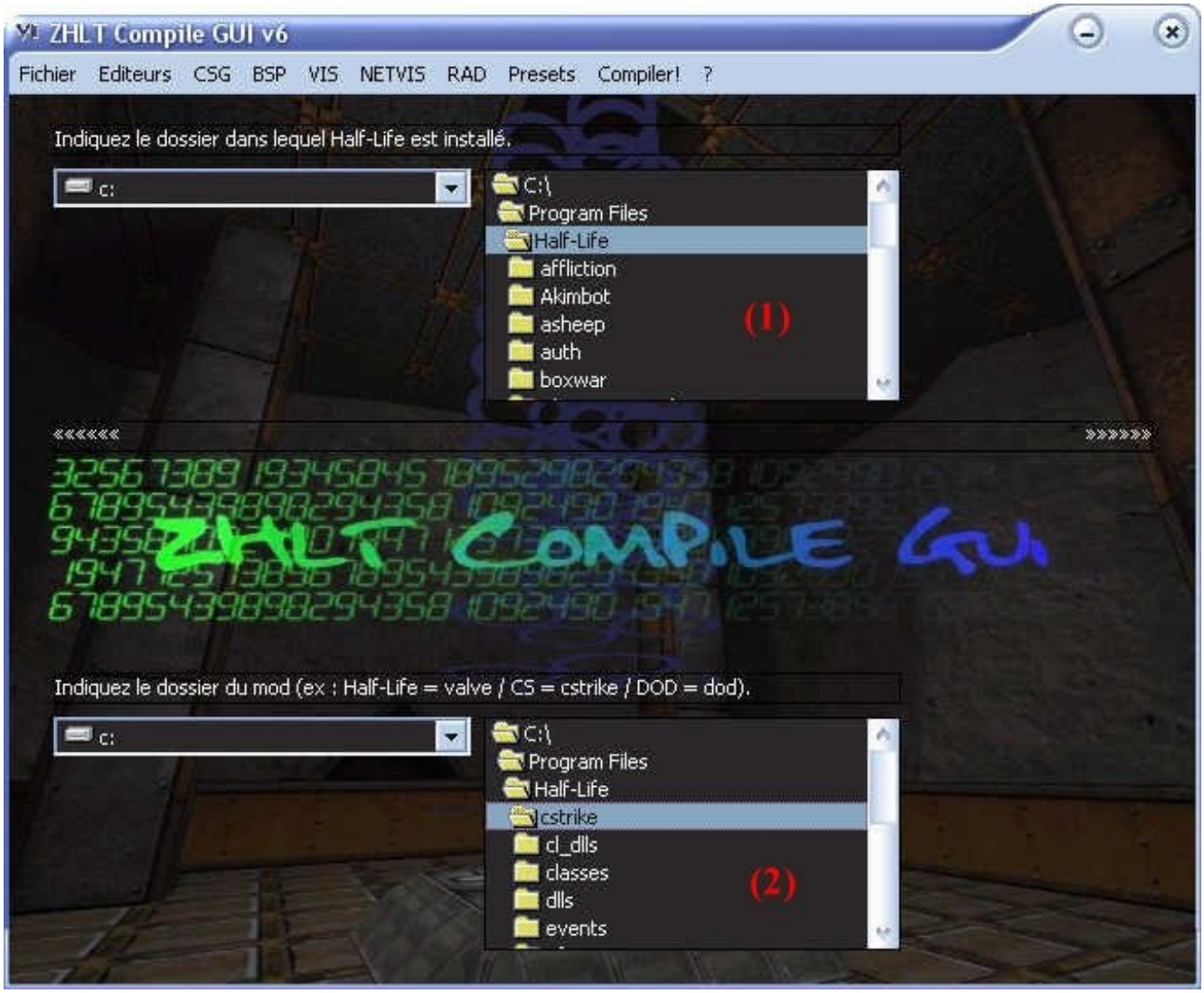
ZHLT Compile GUI v8.zip (2,6 Mo)

Après l'installation, lancez le programme. En cas d'éventuelle erreur au premier lancement, lancez le programme appelé "ZHLT Compile GUI - Error Fix", accessible via le raccourci du menu Démarrer.
Lorsque le GUI est ouvert, voici ce que vous devriez voir :

On vous demande dans un premier temps quelques petites informations...

- En haut, indiquez le répertoire où les ZHLT sont installés. Normalement vous avez dû les télécharger en même temps que le GUI, ils se trouvent donc dans un sous-dossier "ZHLT"
- Ensuite, on vous demandera les compilateurs à utiliser. C'est là que c'est important et qu'il ne faut pas se tromper !
  - Soit vous êtes débutant ou vous n'avez pas envie de vous prendre la tête, et vous cochez **uniquement** la case "Presets".
  - Soit vous êtes expérimenté et vous voulez un maximum de paramétrages, dans ce cas vous cochez les cases que vous voulez sauf "Presets". A noter que NET-Compile sert uniquement pour faire compiler sa map à distance via Internet par quelqu'un d'autre.
- Enfin, vous devrez indiquer en bas où se trouve le fichier à compiler (c'est-à-dire votre map à l'extension .map).

Cliquez ensuite sur les flèches ">>>>>>>" à droite de la fenêtre. De nouvelles options s'affichent :

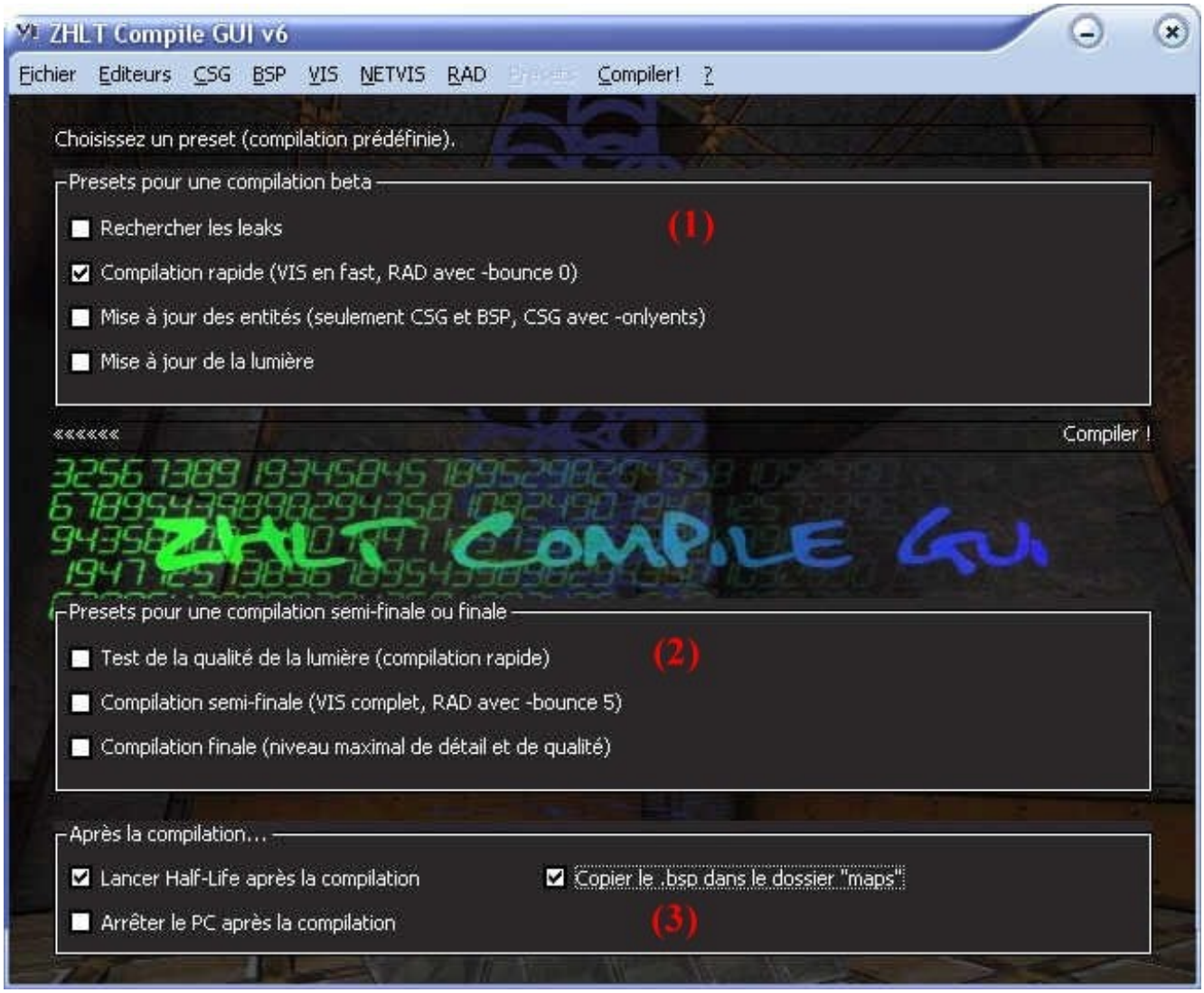


- En haut, indiquez le répertoire où Half-Life est installé. Normalement, vous n'aurez pas à refaire cela la prochaine fois.
- En bas, indiquez le dossier du mod pour lequel vous faites votre map. Si vous faites une map pour Counter-Strike par exemple, vous devrez indiquer le répertoire "cstrike", comme vous le voyez sur le screenshot ci-dessus.

De nouveau, cliquez sur les flèches à droite : ">>>>>>>>>".

La partie "Presets" s'ouvre. Remarquez les menus en haut : "CSG", "BSP", "VIS", "RAD", "Presets", "Compiler!". Ils correspondent aux différentes étapes de la configuration. Vous pouvez basculer de l'un à l'autre en cliquant simplement sur ces menus.

Dans notre cas, nous avons coché la case "Presets", donc nous allons automatiquement dans le menu "Presets" et nous sautons toutes les autres étapes. Vous pouvez basculer d'une étape à l'autre facilement en cliquant sur ces menus.
Pour le moment, on ne se prend pas la tête et on reste dans ce menu "Presets" :



C'est là que vous choisissez votre "Preset", c'est-à-dire le type de compilation que vous désirez.

Surtout, ne cochez qu'une seule case parmi les zones (1) et (2). Il ne doit pas y avoir plus d'une case cochée dans ces 2 zones comprises. Prenez exemple sur mon screenshot : je n'ai coché qu'une seule case dans pour les zones (1) et (2) !

- En haut, vous avez des compilations rapides, utilisées pour le débogage (le test de votre map). Les textes sont expressifs, donc à vous de choisir la compilation qui vous plaît. En temps normal je pense que vous cocherez "Compilation rapide" : la map ne sera pas très jolie, mais ce sera suffisant pour que vous puissiez la tester en quête de bugs.
  Si vous cochez une case dans la zone (1), ne cochez AUCUNE case dans la zone (2).
- Dans cette zone (2), vous avez des compilations plus complètes (et donc plus longues). Utilisez la semi-finale pour distribuer la map à vos amis pour qu'ils la testent (ça sera déjà assez joli pour eux), et n'utilisez la compilation finale que lorsque vous distribuez votre map sur Internet (attention, c'est long !).
- En bas, cochez les cases qui vous conviennent.
  Si vous souhaitez jouer immédiatement après sur votre map, je vous conseille de cocher "Lancer Half-Life après la compilation" et "Copier le .bsp dans le dossier maps".
  Si la compilation est longue (si elle risque de durer toute la nuit), alors cochez uniquement la case "Arrêter le PC après la compilation" : ainsi le GUI éteindra automatiquement le PC lorsque les ZHLT auront fait leur travail. C'est pas beau ça ?

Tout est paramétré ? Alors cliquez sur le bouton "Compiler !" à droite, et ne touchez plus à rien ! Les ZHLT vont s'ouvrir dans des fenêtres MSDOS et faire leur travail. Ne les dérangez pas : je vous conseille de ne rien faire sur votre ordinateur pendant la compilation.

Vous constaterez qu'avec ZHLT Compile GUI, la compilation est très rapide. C'est un de ses nombreux avantages

Une fois la compilation terminée, le GUI fera ce que vous lui avez demandé (arrêter le PC, lancer Half-Life etc...). Il vous indiquera s'il y a eu des erreurs de compilation, et vous donnera le temps qu'a duré la compilation.

S'il y a eu une erreur de compilation, allez regarder dans le fichier votremap.log qui a été créé dans le dossier où se trouve votre map. C'est là que vous pourrez trouver l'erreur s'il y en a eu une (par exemple un Leak).

Décompiler une map

En théorie, c'est totalement impossible. Et pourtant, on l'a fait !!! Quel est le principe ?

D'apparence ça a l'air simple : lire un fichier .bsp et le transformer en .map pour pouvoir l'ouvrir sous Worldcraft. D'apparence seulement, car le .bsp ne contient que les informations pour la carte graphique, et rien qui nous permette de le lire avec un logiciel comme Worldcraft.

La décompilation est très rapide (en comparaison avec la compilation), mais c'est quelque chose de plutôt aléatoire : des fois, il est carrément impossible de décompiler, d'autres fois ça marche avec des erreurs, et (rarement), ça marche plutôt bien...

Pourquoi décompiler une map ?

L'intérêt est évident : vous pouvez voir comment l'auteur a fait pour réaliser tel ou tel effet. Par exemple, vous voudriez savoir comment intégrer les poules de cs_italy dans votre map ? Décompilez cs_italy et regardez par vous-même ! On y apprend une foule de choses intéressantes !

Big WARNING !!! Ne faites pas les idiots et respectez le travail des mappeurs. Ne vous servez pas de ce logiciel pour "copier" les maps des autres.
De toute façon, y'a tellement d'erreurs que la map peut difficilement être recompilée après 

Bon, c'est pas le tout, mais vous devez avoir sacrément envie de télécharger ce logiciel miracle ! Il est tout petit, alors ne vous privez pas ;o)

WinBspc_v1.0.zip (105 Ko)

Exécutez le programme puis allez dans le menu File / Convert. Sélectionnez le fichier .bsp à décompiler. Dans la fenêtre suivante, sélectionnez "Convert to MAP", et choisissez le répertoire où le .map sera déposé.

Après avoir cliqué sur OK, la décompilation commence. Si tout ce passe bien et que vous ne voyez pas trop de Warnings, l'écran devrait afficher : "Map file written in X seconds", où X est le temps qu'a mis WinBspc pour écrire le .map.

Vous pourrez alors ouvrir ce fichier sous Worldcraft et regarder comment il fonctionne !
Eh bien, on peut dire qu'on a fait du bon boulot !

Reste quand même à voir les erreurs de compilation : le plus dur moment du mapping, où on croit que tout est perdu et que sa map ne vaut pas un clou.
Le prochain chapitre va vous donner l'occasion de vous ressaisir 

La compilation (B : débogage)

Ennemis jurés des mappeurs, les erreurs de compilation nous en ont tous fait baver à un moment ou à un autre...

Certains d'entre vous ont même menacé de se suicider si on ne les aidaient pas pour corriger leurs erreurs ! Vous êtes prévenus ! Alors bon, j'ai décidé de leur venir en aide grâce à ce chapitre, parce que j'ai pô envie de me faire accuser d'homicide involontaire

Quand ça plante...

Voici le scénario typique : vous venez de terminer votre première map, et vous êtes impatient de la tester. Qu'à cela ne tienne : vous lisez le chapitre sur la compilation (chapitre précédent), et vous suivez les instructions à la lettre.

Oui mais voilà : à un moment sans crier gare, tout s'arrête brutalement.

- vous compiliez avec le GUI (et si ce n'était pas le cas vous devriez), le programme s'arrête et affiche un message comme celui-ci :

  hlcsg renvoie une erreur !

- Si vous compiliez toujours à l'aide de Worldcraft (allez vite apprendre à vous servir de ZHLT Compile GUI), Half-Life se lance quand même mais ne peut pas charger la carte :

  map nomdelamap.bsp not found on server

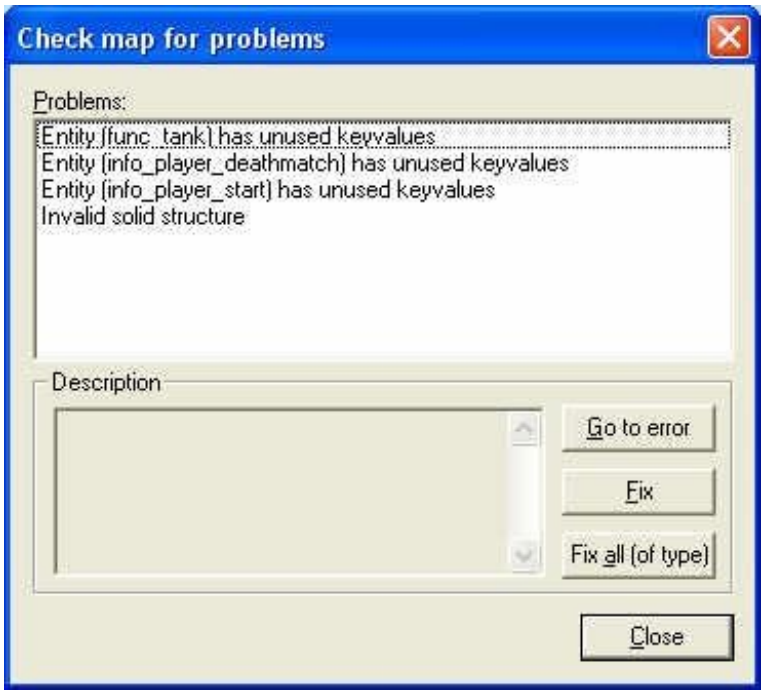
  Ce message est normal puisque le fichier bsp n'a pas été créé à cause de l'erreur de compilation...

A ce moment là vous commencez à croire que votre travail était inutile et que toute votre belle map est perdue. C'est une erreur, car on peut toujours corriger ce genre de problème avec un peu de patience et de méthode...

Ah là là, heureusement que je suis là quand même

Nous n'allons pas dans ce chapitre analyser toutes les erreurs qui existent une par une. Nous allons voir les erreurs les plus courantes, en les classant par groupes.

Tout d'abord, il y a une fenêtre de Worldcraft à connaître absolument : la fenêtre "Check Map for Problems". Vous pouvez y accéder en faisant la combinaison de touches Alt + P ou bien en allant dans le menu *Map / Check for Problems*.



Cette fenêtre est censée afficher les erreurs que Worldcraft a détectées... Si elle ne s'affiche pas, alors c'est qu'il n'y a pas d'erreur détectée par Worldcraft

Worldcraft sait reconnaître les erreurs les plus courantes, mais il y en a beaucoup d'autres qui lui échappent. Pour être sûr que votre map ne comporte pas d'erreur, il faut faire une compilation normale en lançant tous les 4 compilateurs (hlcsg, hlbsp, hlvis et hlrad).

Si un compilateur vous affiche un "Error" ou un "Warning", alors là vous avez un problème.

Nous allons maintenant voir comment repérer et corriger les erreurs de compilation les plus courantes. Ce sont les méthodes de résolution qui vont nous intéresser...

Les blocs invalides

Qu'est-ce que c'est ?

Bon, un bloc vous devriez quand même savoir à quoi ça ressemble lol. Si par hasard ce n'était pas le cas, je vous conseille fortement de lire les premiers chapitres de ce cours.

Mais un bloc invalide ? C'est un bloc qui n'a pas une forme correcte. Sa structure est anormale, irréelle, improbable, elle n'existe pas dans notre dimension. Bref, vous avez créé un bloc que Half-Life ne comprendra pas.

Pour résoudre ce problème, la plupart du temps on ne cherche pas à comprendre : on supprime le bloc qui fait foirer. Cependant, j'aimerais quand même que vous compreniez vos erreurs, donc nous verrons aussi pourquoi ça plante. Le plan que nous allons suivre ressemble donc à ça :

- Les différentes erreurs de blocs invalides
- Comment retrouver et supprimer le bloc qui foire
- Mais pourquoi ça foire ?

1\ Les différentes erreurs de blocs invalides

Lorsqu'un de vos blocs a une structure invalide, les ZHLT ne vous sortent pas à tous les coups la même erreur. Le but ici est donc de voir quelles sont les erreurs qui se rapportent à un bloc invalide, pour que vous sachiez si vous avez ou non une erreur de ce type.

Tout d'abord, il faut savoir que seul hlcsg peut vous donner des erreurs de blocs invalides. C'est lui qui s'occupe de les traquer. Voyons quelques-unes de ses erreurs favorites :

hlcsg: Error: Entity 0, Brush 12: outside world(+/-4096):
(-248,-10016,-276)...

Un bloc sort des limites du monde (et pourtant vous avez de la place pour votre map !). En général, on ne voit rien de spécial sous Worldcraft, mais les coordonnées d'un des points du bloc se trouvent vers l'infini (ce qui est impossible, donc ça plante).

hlcsg: Error: Entity 0, Brush 47, Side 16: plane with no normal

Une des faces du bloc, c'est-à-dire un plan, n'a pas de normale. Une normale est une droite perpendiculaire au plan. Tout plan doit avoir une normale, et apparemment le vôtre n'en a pas... Bref ça plante !

hlcsg: Error: Entity 22, Brush 3, Side 10: has a coplanar plane
at (81, -225, -7), texture LAB1_PANEL1D

Mmh... Là encore c'est un des plans du bloc qui est la cause de l'erreur. Ici, il semble qu'il soit coplanaire à un autre plan, c'est-à-dire que deux plans sont situés au même endroit. Et Half-Life semble ne pas l'apprécier.

Dans le dernier cas, sachez que la texture n'a rien a voir avec l'erreur : c'est juste une indication utile pour retrouver le bloc qui foire...
Et justement, on va maintenant s'intéresser aux moyens qui existent pour réparer ces fichus blocs invalides

2\ Comment retrouver et supprimer le bloc qui foire

En fait, si un bloc est invalide, vous avez deux possibilités : soit vous le supprimez, soit vous "essayez" de corriger l'erreur. Autant vous le dire de suite : la plupart des mappeurs ne cherchent pas à corriger le problème, ils suppriment carrément le bloc qui foire et ils recommencent.

Mais bon, ce que nous devons faire dans un premier temps, c'est retrouver le méchant bloc dans votre map. Imaginez que votre map contient plus de 2000 blocs (et c'est possible), vous risqueriez de passer un moment à retrouver le bloc qui plante.

Heureusement il existe un moyen très simple de retrouver le bloc

Tout bloc (aussi appelé **brush**) de votre map possède un numéro unique. Et les ZHLT vous donnent ce numéro à chaque fois que vous avez une erreur de ce type ! Voyez par vous-même sur cette erreur :

hlcsg: Error: Entity 0, Brush 47, Side 16: plane with no normal

Ici, c'est le bloc numéro 47 qui est la cause du problème. Bien, maintenant que l'on connaît son numéro, on va pouvoir aller l'identifier sous Worldcraft.

Allez dans le menu *Map/Go to brush number*, la fenêtre suivante devrait s'ouvrir :

Comme vous pouvez le voir, c'est très simple : rentrez le numéro 47 en face de "Brush #", et cliquez sur OK. Worldcraft centre alors les 4 vues sur ce %#$##*$ de bloc : ça y est vous l'avez coincé !

Veillez dans ce cas à laisser le champ "Entity #" à sa valeur 0. C'est très important, sinon ça ne marchera pas.
Par ailleurs, si les ZHLT vous donnent un numéro pour "Entity", il faudra alors que vous renseignez aussi ce champ pour que ça marche.

Bon tout ce qu'il vous reste à faire maintenant, c'est d'appuyer sur la touche "Suppr", pour en finir une bonne fois pour toutes avec ce bloc

Je comprends que vous vouliez en savoir plus sur votre erreur, et que vous aimeriez la comprendre... Je vais donc vous faire un petit speech sur la structure des blocs invalides (mais rapide, parce qu'on pourrait y passer des heures).

3\ Mais pourquoi ça foire ?

Vous êtes vraiment sûrs de vouloir savoir ? Si je vous dis que ça nécessite de bonnes connaissances en maths, et plus particulièrement en géométrie dans l'espace, êtes-vous toujours sûrs de vouloir savoir ?

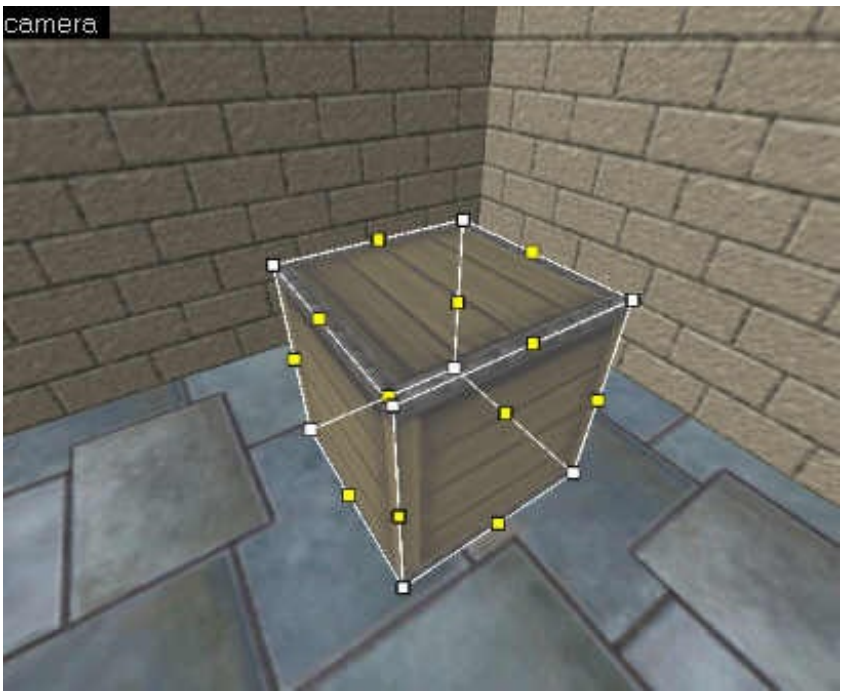
Bon alors voilà quelques trucs à connaître absolument :

- Un Morphing (outil "Vertex") est très souvent la cause d'un problème de forme. En effet, c'est un outil très libre qui risque de vous amener à faire un peu n'importe quoi... et donc à créer une forme impossible qui dépasse les lois de la géométrie. Vous êtes plus forts que votre prof de Maths ?
- Si vous avez intégré un Prefab foireux dans votre map, eh bah ça sera la faute du créateur du Prefab. A cause de lui votre map a buggé !
- Un mauvais Carving aussi peut vous causer des problèmes... Essayez par exemple de carver avec un bloc de forme "Arch". Vous allez immédiatement voir le résultat, je vous le garantis !

Et à quoi ressemblent ces formes invalides ?

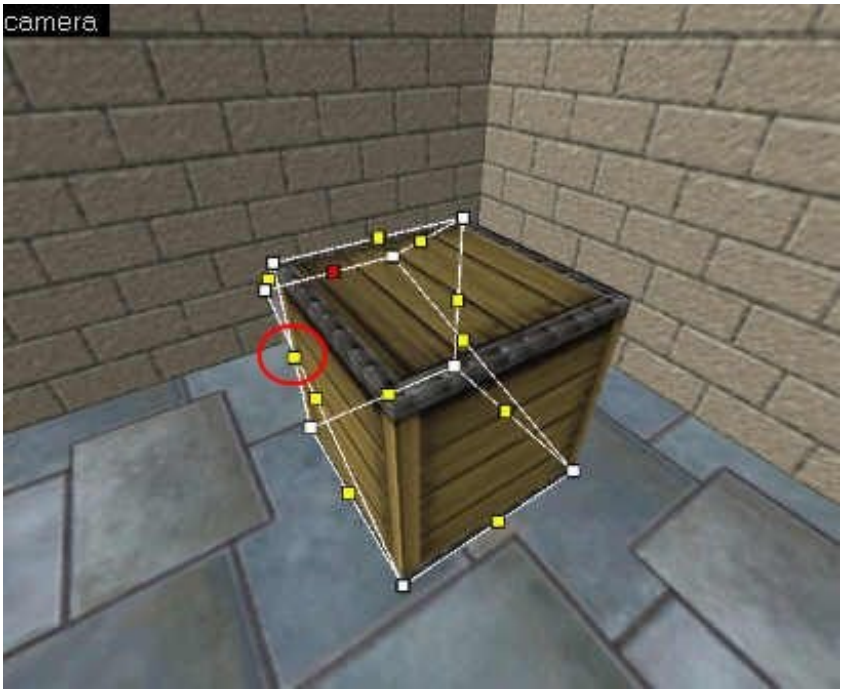
Bon alors là, je ne peux m'adresser qu'aux habitués du "Morphing Tool". Ce n'est qu'avec cet outil que je pourrai mieux vous montrer les causes des problèmes.

Prenons cette caisse qui a une forme très simple :

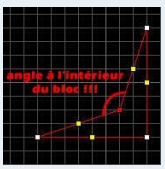


Là, tout est à sa place, tout va bien.

Imaginons que je déplace une des faces et que je la fasse passer à travers une autre, comme ceci :



Vous voyez que les arêtes s'entrecroisent, et ça c'est pas bon du tout. Ca provoquera forcément un plantage.



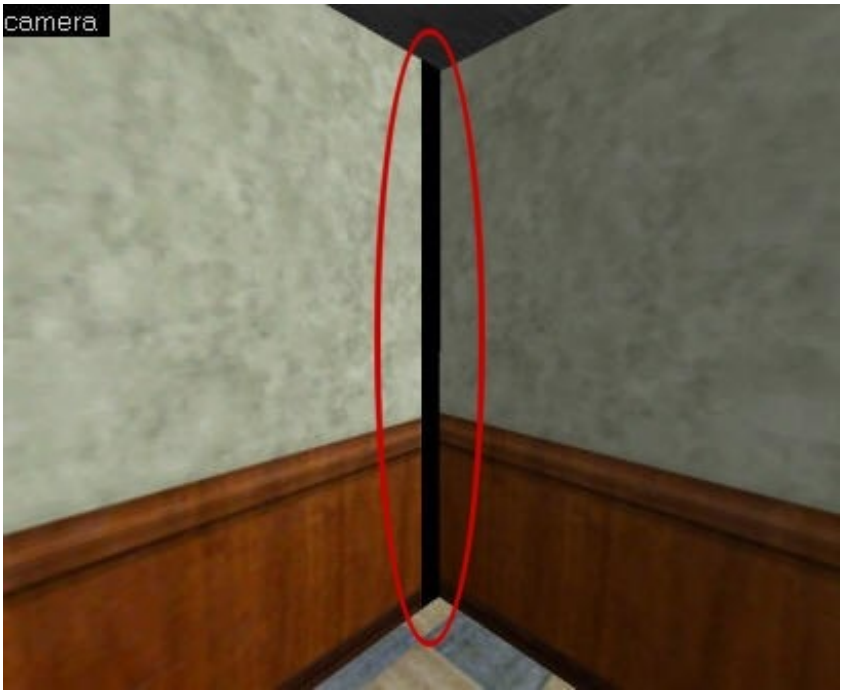
Autre exemple : celui du polygone concave (allez chercher la définition dans un dico si ce mot ne vous rappelle rien). Les polygones de Half-Life sont tous convexes, et si vous voulez en faire un concave, les ZHLT vont planter à tous les coups !
Sur une vue de dessus, ça pourrait donner le résultat de gauche.
Un angle formé par 2 faces se trouve à l'intérieur du bloc, qui devient alors "creux" à l'intérieur : c'est cela un polygone concave.

Les Leaks

Autre méchant problème très courant : les Leaks. Tout le monde en a eu un jour ou l'autre. Vous obtenez une erreur qui ressemble à cela :

hlbsp: Error: Stopped by leak.

A quoi ressemble un Leak ?



Nous avions déjà vu cette image dans le chapitre sur les textures... La zone en rouge est la cause de l'erreur.

Un **Leak** est un trou dans votre map. Vous voyez le vide noir que vous avez lorsque vous commencez une nouvelle map sous Worldcraft ? Eh bien c'est votre pire ennemi : il ne faut jamais montrer ce vide au joueur !

Concrètement, cela veut dire que vous allez devoir en fait créer une sorte de pièce hermétiquement fermée. Dans le cas ci-dessus, pour corriger le problème c'est très simple : il suffit de raccorder les murs pour que le vide ne soit plus visible et le tour est joué !

Oui mais voilà, dans la réalité c'est parfois très difficile de retrouver son Leak : ici le Leak était bien visible, mais l'épaisseur de l'ouverture peut être beaucoup plus petite (1 unité de Worldcraft suffit !). Et alors là c'est le gros bazar pour retrouver le Leak !

Rassurez-vous, il existe des méthodes pour retrouver la position de son Leak : c'est justement ce que nous allons voir...

1\ La bonne méthode

On commence par la meilleure méthode : celle du pointfile. Un pointfile est un fichier à l'extension *.pts qui contient des informations sur des coordonnées de points. Ce type de fichier (avec les *.lin) est automatiquement généré par hlbsp en cas de Leak.

Grâce au pointfile, vous allez pourvoir repérer votre Leak en quelques secondes !

Pour lire les pointfile et les retrouver sous Worldcraft, je vous conseille fortement le logiciel LeakMarker, que vous pouvez télécharger ci dessous :

LeakMarker_v0.1.zip (126 Ko)

Pour l'installer, c'est très simple : copiez le programme dans n'importe quel dossier de votre disque dur (comme celui de Worldcraft) et créez un raccourci dans le menu Démarrer ou carrément sur le bureau si vous aimez l'encombrer d'icônes

Le programme ressemble à ceci :

Le principe est le suivant : vous donnez votre map qui a un Leak à ce programme, et il va créer une entité factice (fausse, inutile, qui n'existe pas, qui ne sert à rien), enfin, une entité de nom inconnu mais placée tout près du Leak... En cherchant cette entité dans Worldcraft, vous vous retrouverez nez à nez avec votre Leak !

- En haut apparaissent les noms des fichiers que vous avez sélectionné en cliquant sur le bouton "Select Files". Ici, il s'agit d'une map de tests appelée "tests2" que j'ai créé spécialement pour avoir un Leak (je suis un peu maso je sais).
  Notez que l'on doit utiliser 2 fichiers : celui de la map (*.map) et celui du pointfile (*.lin) qui contient des informations sur le Leak.
- Ici, vous pouvez choisir le nom de la texture de l'entité qui sera créée. Je vous conseille de laisser RED, comme ça vous pourrez pas la louper
- C'est la classe de l'entité (le type d'entité, au même titre que *func_wall* par exemple). LeakMarker fait exprès de mettre un nom d'entité qui n'existe pas : comme ça Worldcraft dira qu'il y a une erreur et vous montrera directement cette entité (et donc le Leak qui se trouve à côté).
  Le fonctionnement est tordu, je suis bien d'accord, mais lorsque ce programme vous aura aidé à résoudre votre Leak, vous le bénierez croyez-moi
- C'est le nom de l'entité : laissez-le tel quel.
- Ce sont les coordonnées des points données par le fichier *.lin qui déterminent la position du Leak.
- Ce bouton est la première chose que vous devez faire : donnez le nom du fichier pointfile (nomdelamap.lin) et celui du fichier de map (nomdelamap.map) pour que le programme sache sur quoi il doit travailler.
- Enfin, si tout est bon, cliquez sur le bouton "Mark Leak" : le programme va alors modifier votre fichier .map et y ajouter une entité près du Leak.

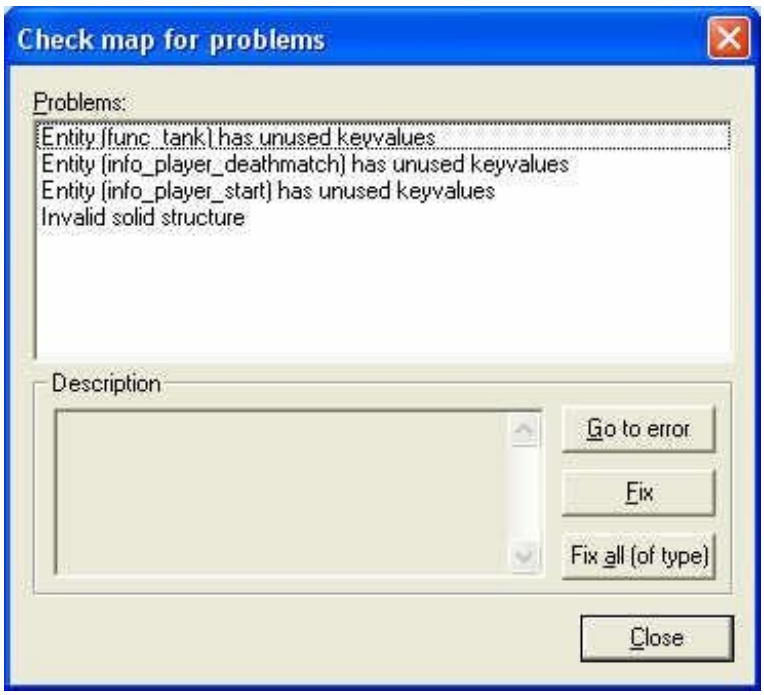
Le fichier *.lin n'a été créé que si les ZHLT ont rencontré un Leak au cours de la compilation. Sinon ce fichier n'existe pas.
Le fichier *.lin se trouve dans le même répertoire que votre map.

A partir de là, vous allez retravailler sous Worldcraft. Ouvrez votre fichier *.map.

N'ouvrez pas le *.rmf car il n'a pas été modifié ! Seul le fichier *.map contient la nouvelle entité, c'est donc lui que vous devez ouvrir.

Bon, si votre map est grosse, il va être assez difficile de retrouver l'entité de LeakMarker à l'aveuglette. Ce serait d'ailleurs complètement débile puisque ça reviendrait à chercher le Leak que vous ne trouvez pas

Heureusement, il existe un moyen pour retrouver l'entité. Allez dans le menu *Map/Entity report*, ce qui aura pour effet d'ouvrir cette fenêtre :



Cette fenêtre recense toutes les entités de votre map. Vous allez ainsi pouvoir repérer en 2 secondes l'entité de LeakMarker.
Et là, cliquez sur le bouton "Go to" : Worldcraft sélectionnera alors l'entité.

Maintenant que vous savez où est votre entité, vous n'avez plus qu'à chercher et à boucher le Leak : il est normalement tout proche de l'entité !

Pensez à supprimer l'entité lorsque vous aurez corrigé votre Leak, parce qu'elle ne vous servira plus à rien. Et n'oubliez pas d'enregistrer votre map en *.rmf et en *.map pour être sûr de ne conserver aucun fichier qui contiendrait toujours le Leak !

2\ La mauvaise méthode

Le (très) mauvais truc que l'on a tendance à faire, c'est "d'entourer" la map de murs géants... L'idée que vous avez est simple : "Puisque je n'arrive pas à trouver mon Leak, je vais créer un très grand cube hollowé autour de la map, comme ça il n'y aura plus de Leak".

Le problème, c'est que cette technique marche... et c'est là tout le danger !

Pourquoi ? A cause de quelque chose dont nous allons beaucoup parler dans les prochains chapitres : les *r_speeds*.

Je ne vais pas vous expliquer ce que c'est maintenant. Vous devez juste savoir qu'avec cette très mauvaise méthode, votre map va ramer même si vous avez la toute dernière GeForce ! Le nombre d'images par seconde (fps) va chuter lamentablement, et votre map deviendra vite injouable !

Bref, je vous recommande fortement d'utiliser LeakMarker comme nous venons de voir : ça c'est une méthode propre

Le sky bug

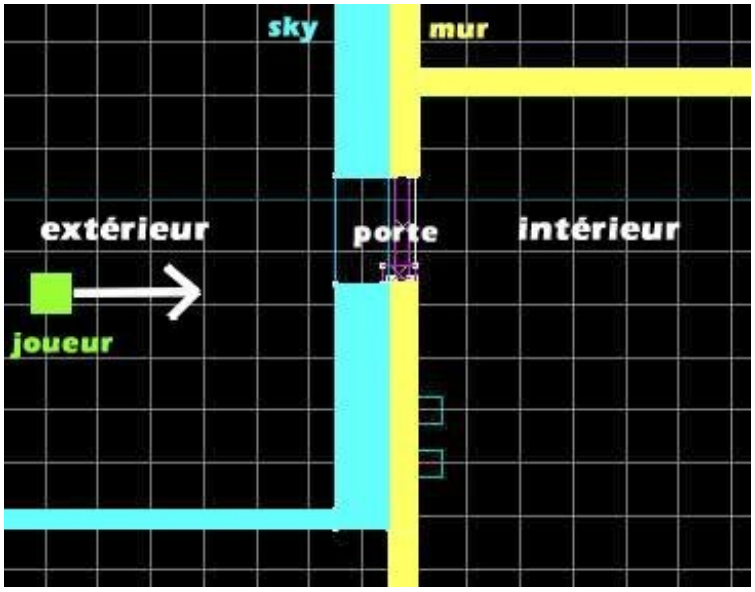
C'est un bug assez méconnu de Half-Life car il est assez rare... Et pourtant, ça m'est arrivé un beau jour, et je dois avouer que j'étais assez désemparé !

A quoi ressemble le "sky bug" ?

Pour vous faire une idée, il vaut mieux avoir un bon screenshot... J'ai donc créé une map dans laquelle j'ai fait en sorte que ce bug apparaisse.
Voilà ce que ça a donné :

Vous voyez le problème maintenant ?

Je me trouve à l'extérieur et il y a donc du sky autour de moi. Je viens de sortir d'un bâtiment par la porte, je me retourne et que vois-je ?
L'intérieur du bâtiment ! Le sky est devenu transparent et du coup maintenant je vois tout ! Pour bien comprendre l'origine de ce problème, il nous faut la vue de dessus ("Top"). Voici ce qu'elle affiche :



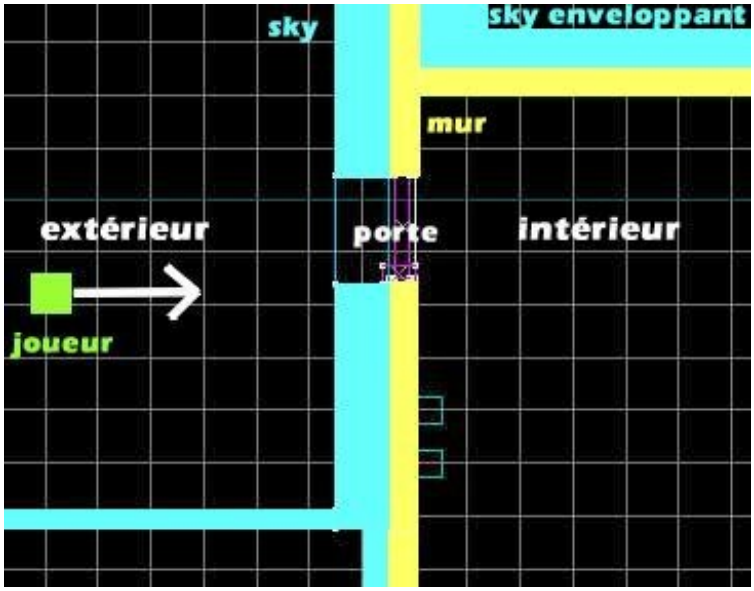
Bon, ok je l'ai un peu modifiée parce que sinon c'était pas très compréhensible...
Enfin, le problème vient de là : le bloc sky est "collé" au mur, et du coup si le joueur est à l'extérieur et qu'il regarde dans la direction indiquée par la flèche, le sky et le mur disparaissent : on voit l'intérieur du bâtiment ! Et ne croyez pas qu'il suffirait d'espacer le bloc sky et le mur ! Non non, ça serait trop facile...

Alors, quelle est la soluce ?

Elle est longue à réaliser... très longue même selon le cas. Il s'agit d'entourer toute les surfaces externes de sa map par des blocs sky. Au-dessus, au-dessous et sur tous les côtés.

Ces blocs sky doivent être impérativement collés aux autres blocs pour que cette technique fonctionne ! Si vous oubliez d'entourer un seul bloc de votre map par un bloc sky, le sky bug recommencera !

Bien, dans notre exemple, voici comment on aurait dû agir et ce dans toute la map :



Des blocs de skys enveloppent tous les murs extérieurs de la map. De cette manière, si vous n'avez oublié aucun mur et si vous avez bien collé vos blocs skys, le sky bug devrait disparaître !
J'espère que vous avez trouvé ce chapitre intéressant, et surtout qu'il vous aura aidé 

Si vous avez une erreur de compilation qui n'est pas donnée ici, deux solutions :

1. Allez voir l'annexe qui recense toutes les erreurs de compilation (prochainement disponible).
2. Allez sur le forum et expliquez votre problème. Pensez à copier votre erreur de compilation !

Optimisation (A : lois de HLvis)

Le chapitre sur l'optimisation est découpé en 2 parties, exactement comme je l'avais fait pour la compilation : j'expliquerai dans ce chapitre comment fonctionne HLvis, puis je vous aiderai à résoudre les nombreux problèmes qu'il vous cause (r_speeds notamment).

Nous n'allons pas vraiment parler de mapping pur et dur. En réalité, on va carrément jouer sur une map, faire des constats puis en déduire le fonctionnement du compilateur HLvis (raisonnement scientifique quoi)

Ce chapitre peut vous paraître simple au premier abord, mais détrompez-vous ! Il s'agit de mapping avancé et de problèmes qui touchent même les mappeurs professionnels de chez Valve !

J'ai voulu traiter le sujet différemment des autres sites afin de vous amener en douceur à comprendre un sujet complexe tel que les r_speeds (c'est un peu ça la méthode du Zér0).

Découpage des polygones

Lorsque vous faites une map, vous la travaillez dans un premier temps sous Worldcraft, puis vous la donnez aux compilateurs afin de pouvoir jouer dessus.

La grosse erreur serait de croire que vous êtes en train de jouer sur la même map que celle que vous avez faite sous Worldcraft...

Pardon ??? Ma map n'est plus la même lorsque j'y joue ???

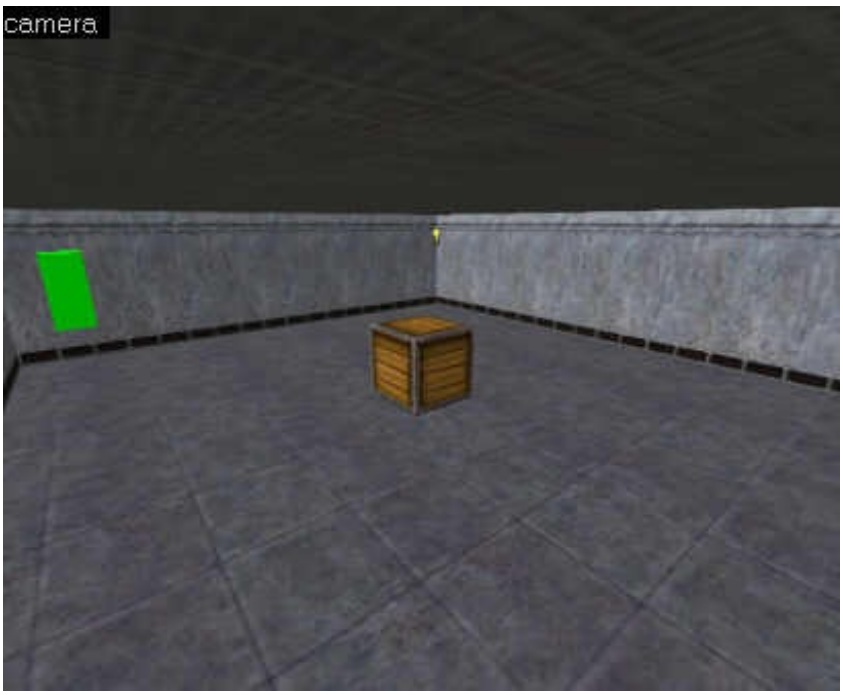
En effet ! Lorsque vous l'avez donnée aux compilateurs, il ne fallait pas croire bêtement que ceux-ci allaient juste la rendre jouable sous Half-Life !
Les compilateurs ont en réalité complètement modifié votre map et elle n'est carrément plus la même...

- Première étape : HLcsg va diviser vos blocs (appelés polygones lorsqu'on joue), pour faire en sorte que tous les blocs aient la forme la plus simple possible. Ainsi le moteur de Half-Life aura moins de mal à les interpréter, mais du coup le nombre de polygones aura été multiplié !!!
- Deuxième étape : HLbsp va se charger de déterminer les limites de votre map (lorsqu'il y a un Leak, il n'y a plus de limites ce qui le fait planter). Tous les blocs situés en dehors de ces limites sont donc totalement supprimés dans le *.bsp !
- Troisième étape : HLvis doit calculer la visibilité, c'est-à-dire déterminer quel polygones doivent être calculés en fonction de la position du joueur. Vous vous en doutez, c'est un très gros travail.

Ceci est juste à titre récapitulatif pour bien mettre les choses au clair.
Je ne vous parle pas de HLrad ici parce qu'il n'intervient pas beaucoup dans les problèmes d'optimisation...
Ce qui nous intéressera dans un premier temps, c'est le travail qu'a effectué HLcsg (première étape). Concrètement, quelles différences peut-on voir entre la map sous Worldcraft et celle sous Half-Life ?

Eh eh, cette différence n'est pas visible en réalité... du moins pour un oeil d'humain ! Je vais vous montrer un exemple typique du découpage des polygones...
Je crée une map toute bête : une salle avec une caisse au milieu (la caisse est un bloc et non pas une entité). Je vous conseille de faire la même chose que moi, vous allez voir c'est assez surprenant !

Voici ce que l'on devrait voir dans la vue "Camera" de Worldcraft :



Si l'on met de côté la laideur de cette salle, combien de blocs pouvez-vous y compter ?
Le sol, les 4 murs, le plafond et la caisse. On peut voir par ailleurs 2 entités : une *light* et un *info_player_start*. Le départ du joueur est tout vert parce que je fais une map pour Half-Life, mais si vous faites pareil pour Counter-Strike ne vous en faites pas, il n'y a aucune différence.
Bon, en tout on a à peu près 7 blocs.

Je compile la map avec le GUI et... zou ! Je lance Half-Life dessus.

Pour que vous puissiez effectuer les manipulations que je vais vous proposer, il faut IMPERATIVEMENT que vous lanciez votre map en mode "Solo" et non pas Multijoueurs.
Le fonctionnement reste le même dans les 2 cas, mais certaines commandes ne fonctionnent pas en mode Multijoueurs...

Voici un screenshot que j'ai pris sous Half-Life :

Apparemment, la map est strictement identique à celle de Worldcraft. Vous pouvez compter les mêmes blocs : le sol, les murs, le plafond et la caisse...

Mais êtes-vous bien sûr qu'il y a 7 polygones dans votre map ?

Pour le vérifier, nous allons activer le mode "fil de fer" de Half-Life. Il ne fonctionne qu'en mode "Solo", c'est pour cela que je vous ai demandé de lancer votre map de cette manière.

Ouvrez la console en tapant sur la touche "²", et tapez la commande suivante :

gl_wireframe 1

Cette commande ne fonctionne que si vous êtes dans une partie Solo ET que vous êtes en mode "OpenGL". Elle ne marche pas avec DirectX ! Il va donc peut-être falloir effectuer le changement dans les options de Half-Life...

Fermez la console en tapant de nouveau sur "²", et observez le résultat :

Surprenant non ?

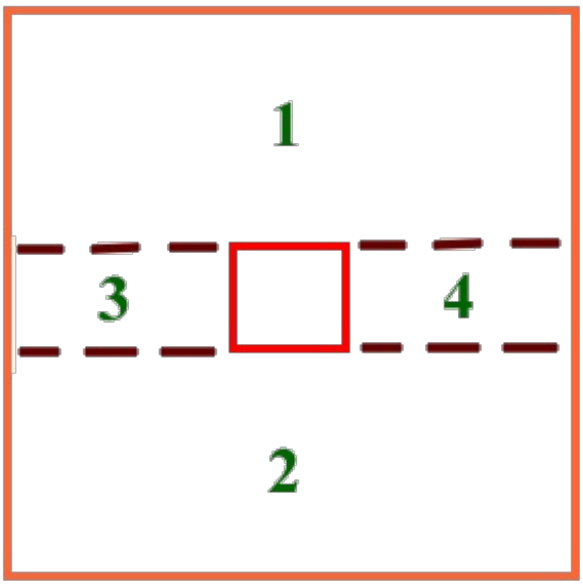
Comme vous pouvez le voir, ce mode affiche les lignes qui séparent les polygones. Il est très facile de les compter ainsi. Amusez-vous à compter les polygones maintenant

Mais pourquoi y a-t-il autant de polygones ?!

Il y a 2 raisons :

- La caisse a découpé le sol sur lequel elle était posée. En effet, HLcsg a martyrisé votre pauvre bloc et l'a découpé en plusieurs parties (4 normalement). C'est ça le découpage des polygones !
  Quel en est l'intérêt ? C'est pour Half-Life que l'on fait ça... On doit lui simplifier au maximum les polygones, au risque d'en augmenter le nombre.

  Schématiquement, ça pourrait donner ça sur une vue de dessus :



La caisse (en rouge) a divisé le sol (en orange) en 4 parties. Et c'est comme ça que ça fonctionne pour tous les blocs de votre map !!!

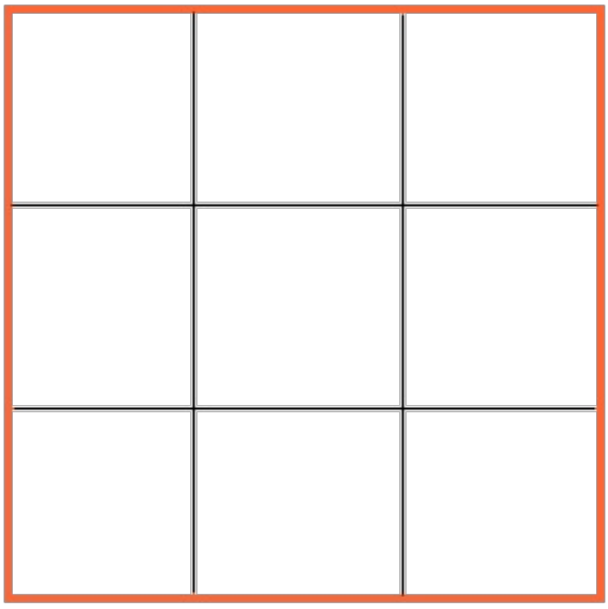
Je vous laisse imaginer si, à la place de la caisse, on avait mis un tonneau, ou n'importe quel objet cylindrique... Le sol aurait été découpé encore plus de fois et bien bizarrement.
Je vous laisse retenter l'expérience avec un objet cylindrique !

- Pour la deuxième raison, on peut se demander pourquoi le plafond a lui aussi été affecté... C'est curieux, on n'avait posé aucun objet dessus et pourtant il a été découpé !?

L'explication tient dans **les flats**. Ce n'est pas évident de définir un flat. Disons à notre niveau que c'est une division d'un bloc à cause de sa texture. Oui, c'est la texture qui est en cause.
Je vous rappelle la définition de texture ? C'est une image qui est répétée en mosaïque... eh bien, à chaque fois que l'on répète la texture, un nouveau flat est créé.

Sur le screenshot que nous avons vu, vous pouvez maintenant discerner la taille de la texture et le nombre de fois qu'on a dû la répéter en mosaïque...

Schématiquement, ça doit donner quelque chose comme ça :



Chaque partie représente un flat, une répétition de la texture.

Cela veut donc dire que si vous créez un très grand bloc, il n'y aura pas qu'un seul polygone, mais plusieurs car la texture aura été répétée de nombreuses fois !

Voilà, c'est à peu près tout ce que vous deviez savoir sur le découpage des polygones

Travail du moteur de Half-Life

On enchaîne maintenant avec un autre aspect de l'optimisation très très important... On va y étudier le fonctionnement du moteur de Half-Life.
Ne négligez surtout pas ce passage car vous y apprendrez des connaissances in-dis-pen-sables à la compréhension des r_speeds.

Rappelons-nous. Que fait HLvis ? Il détermine les blocs qui sont calculés ou pas par le moteur de Half-Life... C'est très important, car si vous vous trouvez à un bout de votre map, vous n'allez quand même pas devoir calculer la position des polygones situés à l'autre bout puisque vous ne les voyez pas !!!

Le travail de HLvis a pour but de dire à Half-Life : "A cet endroit, on va calculer ces polygones et pas ceux-là".

Moins on calcule de polygones, plus le calcul de l'image va vite... et donc, en jouant sur votre map on aura des fps élevées (images par seconde).
Plus les fps sont élevées, moins votre map risque de "ramer". Car c'est bien là le problème : Half-Life est un vieux jeu, mais il peut sans problème faire ramer une carte graphique récente si vous n'y prenez pas garde !

Jusque-là, tout irait bien. Sauf que, la plupart du temps, HLvis va "supposer" que le joueur peut voir certains polygones alors que ce n'est pas le cas. Half-Life va donc faire un travail inutile en calculant des blocs que l'on ne voit pas...

Et ça, c'est très agaçant pour le mappeur, même si ce mappeur est un habitué de Worldcraft. Half-Life calcule trop de polygones pour rien...

Prenons un exemple typique assez impressionnant. Dans une de mes maps, je me trouve à l'intérieur d'une maison derrière la porte d'entrée qui est fermée... Je vois ceci :

Tout simple : des murs, un plafond, des escaliers... bref, peu de blocs sont calculés à priori.

Je vais vous prouver qu'en fait, Half-Life calcule 50 fois plus de blocs que ça (et je n'exagère pas) !

Pour cela, tapez dans la console la commande suivante :

gl_wireframe 2

Je vous rappelle que cette commande ne fonctionne que si vous avez lancé votre map en mode "Solo" et que vous êtes en OpenGL...

Le mode gl_wireframe 2 est encore plus avancé que celui que nous avons étudié tout à l'heure. Il affiche non seulement tous les polygones découpés et les flats, mais aussi TOUS les polygones calculés par le moteur (on les voit par transparence).

Voici donc tous les polygones que Half-Life calculait comme un fou alors que ça ne servait à rien :

Vous avez beau vous frotter les yeux, vous verrez bien la même chose : Half-Life calculait entièrement la pièce qui se trouvait derrière la porte et qu'on ne voyait pas !
Ca fait un sacré paquet de polygones tout ça

Pourquoi, mais pourquoi Half-Life fait-il tous ces calculs inutiles ?

Il anticipe. En effet, la porte que vous voyez là est susceptible de s'ouvrir à n'importe quel moment, par vous ou par un autre joueur... Half-Life se tient prêt et calcule donc déjà les polygones qui se trouvent derrière.
Et si vous avez déjà fait connaissance avec les Leaks, vous savez que Half-Life a horreur de montrer du vide au joueur (c'est le pire cauchemar du mappeur ça, lol)

Contre ce problème, il existe des solutions. Certaines sont préventives (ce sont celles que je préfère), d'autres sont assez bourrines (on force Half-Life à ne pas calculer des polygones, mais alors là gare aux bêtises !).

Les portals

Un portal est une information (et une seule) que HLvis a calculée. Cette information est destinée au moteur de Half-Life et lui indique quels polygones il doit calculer à un endroit précis de la map.

C'est au départ HLbsp qui, après avoir écrit le fichier *.bsp, se charge de transmettre un fichier d'informations sur la map à HLvis pour qu'il puisse faire les calculs nécessaires dessus.

Vous devriez d'ailleurs lire la ligne suivante :

BSP generation successful, writing portal file 'c:\maps\nom_de_la_map.prt'

C'est sur ce fichier *.prt que HLvis va travailler. Il va déterminer les portails (en anglais "portals").

Prenons l'exemple de ma map que vous avez vue tout à l'heure. Il y a un portal qui dit : "SI le joueur se trouve derrière la porte d'entrée, ALORS il faut caluler tous les polygones qui se trouvent dans la pièce derrière".

Il est intéressant de noter que les blocs ne sont pas calculés s'ils sont séparés du joueur par plus d'1 portal. Les entités-blocs tel que le func_wall ne sont pas calculées si elles sont séparées par au moins 2 portals.
Cela veut dire qu'il faut éviter de mettre beaucoup de func_wall parce qu'ils sont beaucoup plus souvent calculés que les blocs normaux. On aura l'occasion de reparler de tout ça...

Les r_speeds

Pour commencer, je tiens à préciser qu'il est **impératif** d'avoir lu le début de ce chapitre pour comprendre les r_speeds. Sinon, vous risquez d'être complètement largués (déjà que c'est pas un chapitre évident, alors si vous sautez les étapes...).

Bien, ceci étant précisé, on va pouvoir se poser la question traditionnelle :

Beuh, c'est quoi les r_speeds ?

Les r_speeds sont des informations que le moteur de Half-Life vous donne en temps réel, c'est-à-dire pour chaque image calculée.
Sachant que notre ordinateur doit afficher au moins 24 images par seconde pour que ça reste fluide à nos yeux d'humains, eh bien il va être difficile de lire 24 r_speeds en une seconde ! Rassurez-vous, ça reste malgré tout très lisible parce que les informations sont quasiment les même d'une r_speed à une autre étant donné que votre ordinateur est très rapide...

Comment afficher les r_speeds ?

Si vous utilisez ZHLT Compile GUI pour compiler, les r_speeds sont déjà sur votre écran. Eh oui, ce sont les lignes qui apparaissent en haut à gauche de l'écran !

Sinon, vous devez taper cette commande dans la console :

r_speeds 1

Puis celle-ci afin que les r_speeds apparaissent en haut de l'écran :

developer 1

Voilà, les r_speeds sont là

Quelles informations donnent les r_speeds pour chaque image ?

Les r_speeds donnent à chaque fois 4 informations différentes par ligne :

Il y a ici 4 lignes, donc ce sont les r_speeds de 4 images calculées par le moteur de Half-Life.
On va s'intéresser à une seule image : la première ligne. Elle contient les informations suivantes :

69fps 5 ms 440 wpoly 256 epoly

Ces 4 informations sont toutes différentes, et certaines sont plus ou moins importantes. Nous allons les étudier dans l'ordre :

- 69fps : comme nous l'avons vu plus haut, les fps (frames per second) représentent le nombre d'image par seconde qui s'affichent sur votre écran. En théorie, 24fps suffisent pour que le jeu soit fluide. On a ici 69fps, c'est donc largement plus qu'il ne nous en faudrait

  Cette information dépend de la puissance de votre ordinateur. En effet, pour une même map, un ordinateur rapide aura des fps élevées alors qu'un ordinateur plus lent (de l'âge de pierre) aura des fps plutôt basses.

Si vous voyez tout le temps 8fps sous Half-Life, alors il serait peut-être temps de changer votre ordinateur, vous ne croyez pas ?

- 5 ms : c'est le temps qu'a mis votre ordinateur pour calculer cette image. Ici, cette image a été calculée en 5 millisecondes (elle est rapide ma bécane hein ?)
  Petit calcul : combien ça fait d'images par seconde si toutes les images sont calculées en 5 ms ? Ca fait 1000 ms / 5 ms = 200fps !!!

Alors pourquoi a-t-on "seulement" 69fps au lieu de 200fps ?

max_fps 90
Eh bien, Half-Life limite volontairement le nombre d'images par seconde pour éviter de fatiguer votre écran, voire de le déteriorer (Valve voulait peut-être éviter des procès de fabriquants d'écran !)
Pour info, vous pouvez augmenter la limite des fps à vos risques et périls. Servez-vous de la commande max_fps. Par exemple, si on veut limiter le nombre de fps à 90, on doit taper :

max_fps 90

Mais ça n'a strictement aucun intérêt ! Enfin, pour la frime peut-être...

- 440 wpoly : cette information est la plus intéressante des 4. Pourquoi ? Parce qu'elle ne dépend pas de la puissance de votre PC ! Cette valeur sera la même pour tout le monde.
  Les wpoly (World Polygons) sont le nombre de polygones que Half-Life a dû calculer pour cette image. Comme vous avez pu le constater, même pour une petite map le nombre de polygones augmente très vite ! Ici, 440 polygones ont été calculés pour cette image (c'est une valeur très correcte)...
  Il faut donc prendre en compte les polygones découpés, les flats, et les polygones que l'on ne voit pas mais que Half-Life calcule quand même !

  Comme nous l'avons vu plus haut, Half-Life sait quels polygones il doit calculer grâce aux portals. Ce sont eux qui lui indiquent s'il doit calculer tel ou tel polygone.

Pourquoi les wpoly sont-ils aussi importants ?

Eh bien, étant donné que cette valeur sera la même pour tous les ordinateurs, vous pouvez ici juger si votre map risque de ramer ou pas.
Plus il y a de wpoly, plus il y a des chances que votre map commence à ramer. On dit à tort que l'on a "des r_speeds élevées" (c'est une erreur parce que ce sont seulement les wpoly qui sont élevés, mais bon que voulez-vous)

Les wpoly prennent en compte non seulement les blocs, mais aussi tout ce qui ressemble à un bloc : les entités func_wall par exemple sont elles aussi comptabilisées et contribuent à augmenter vos wpoly !

Il y a des valeurs à ne pas dépasser. Pour une map multijoueurs (Counter-Strike par exemple), 800 wpoly c'est déjà beaucoup... mais la limite fatidique serait autour de 1000 wpoly. Si vous avez plus que ça, c'est vraiment beaucoup trop et votre map est à revoir !
Les valeurs "idéales" se situent autour de 500 wpoly. Là, on peut se fragger en toute tranquilité

Vous verrez dans le prochain chapitre comment diminuer vos wpoly, car il existe des méthodes pour "arranger" votre map.

- 256 epoly : moins importants dans une partie multijoueurs, les epoly (Entities Polygons) représentent en fait le nombre de polygones liés aux entités (généralement des entités-point).

Les func_wall, tout comme les func_illusionary et autres, ne sont pas comptabilisés dans les epoly, mais bel et bien dans les wpoly !!!

Que reste-t-il alors qui apparaît dans les epoly ?
Il s'agit des "sprites" (explosions, fumigènes...) et des "models" (armes, joueurs, ennemis...).

Pour une map multijoueurs, ces valeurs ne sont généralement pas très élevées. C'est surtout dans les maps "Solo" que ça risque de flamber... En effet, il y a beaucoup plus de models dans ces maps, principalement à cause des ennemis !
Il faut alors prendre en compte les wpoly ET les epoly.

Généralement, les epoly sont beaucoup plus élevées que les wpoly. C'est tout à fait normal d'avoir plusieurs milliers de epoly : l'échelle n'est pas la même que pour les wpoly.

En effet, les epoly sont une toute autre affaire et il y en a généralement beaucoup...

Si vous avez des wpoly élevés, il vaut mieux faire en sorte de réduire un peu les epoly pour que votre map rame moins, et vice versa.

Mais bon, en règle générale on retiendra que les epoly sont "moins importants" pour une map multijoueurs, mais qu'il faut commencer à les prendre en compte si on fait une map "Solo" avec des ennemis.

Voilà. Vous savez tout ce qu'il faut savoir sur les r_speeds, et je reconnais que ce n'est pas une notion évidente. Il ne tient plus qu'à vous d'y faire attention et de vous en méfier...
Dans le prochain chapitre, je vais vous aider à diminuer vos r_speeds... et ce sera assez facile car, maintenant que vous savez comment tout cela fonctionne, vous n'aurez aucun mal à comprendre les méthodes que j'emploie pour réduire les r_speeds !

Optimisation (B : diminuer les r_speeds)

Voilà un chapitre qui devrait en intéresser plus d'un !
Comment diminuer les r_speeds ? C'est bien la question que l'on se pose tous quand on s'aperçoit que notre map explose le compteur de r_speeds (si vous avez plus de 1000 wpoly dites-vous bien qu'il y a un problème !)

Dans le chapitre précédent, je vous ai expliqué le fonctionnement du moteur de Half-Life... ces connaissances vous seront très importantes pour ce chapitre. N'hésitez donc pas à le relire pour être sûr d'avoir bien tout compris !

Eh bien, dans ce chapitre je vais vous apprendre à diminuer vos r_speeds, mais rappelez-vous toujours que dans 90% des cas c'est la structure de votre map qui est en cause !!!

Bien structurer sa map

S'il y a **UNE CHOSE** que vous devrez retenir de ce cours de Worldcraft, c'est bien ce que je vais vous enseigner maintenant sur la structure d'une map.
Il n'y a rien de plus important à savoir quand on veut diminuer ses r_speeds...

Oui, vous l'aurez compris, le meilleur moyen de faire diminuer ses r_speeds c'est de commencer à faire des "vraies" maps !

Qu'est-ce qu'une "vraie" map ?

Comme à peu près tout le monde, j'ai moi aussi commencé à créer des maps gigantesques qui m'affichaient environ 1800 wpoly...

Maintenant je sais comment on doit créer une map, et grâce à cette méthode je dépasse rarement les 800 wpoly ! Ca peut paraître absurde, mais il y a de fortes chances que jusqu'ici vous étiez en train de créer des maps tout à fait foireuses, **même si vous aviez l'impression que tout marchait bien**.

Bon, c'est quoi cette méthode miracle ?

Retenez bien cette phrase :
Que votre map se déroule à l'intérieur ou à l'extérieur, il faut la diviser en plusieurs salles !

Eh oui, c'est aussi bête que ça, et pourtant quand on le fait ça change vraiment tout !

Cas d'une map en intérieur

Si votre map se déroule entièrement à l'intérieur (cas d'une maison, d'une base telle que Black Mesa...), c'est encore assez simple à faire... Vous devez créer plusieurs pièces : des salles plus ou moins grandes, où on pourrait passer de l'une à l'autre grâce à des couloirs ou des portes.

Il faut à tout prix faire en sorte que lorsqu'on est dans une salle on ne puisse pas voir le contenu de l'autre salle. Half-Life ne doit pas calculer les polygones d'une salle que l'on ne voit pas !
Pour cela, il faut créer des zones de transition dans votre map (ascenseur, escalier, couloir...)

Par exemple, voici une zone de transition :

Lorsqu'on descend ces escaliers, on passe d'une pièce (qui se trouve derrière la porte à gauche), à une autre pièce (la cave à droite). Ainsi, grâce à cette zone de transition, si on est dans la cave, seuls les polygones de la cave sont calculés !

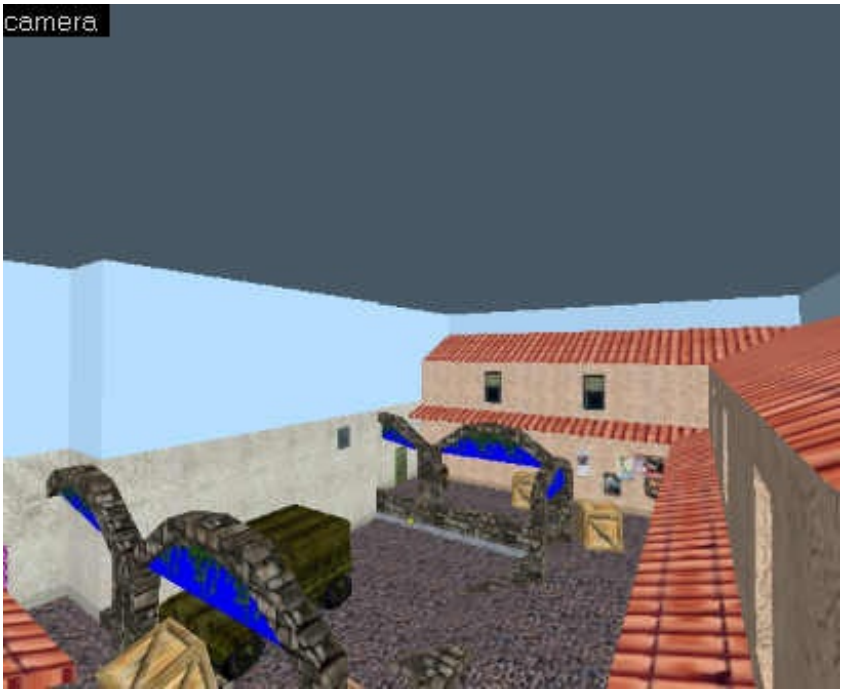
Cas d'une map en extérieur

C'est un cas très fréquent pour les maps Counter-Strike par exemple. Et c'est là que vous risquez de faire le plus d'erreurs...

Pourquoi ? Parce que vous ne comprenez pas ce que veut dire "créer des salles" lorsqu'on se trouve à l'extérieur !

Oui alors, comment peut-on créer des salles à l'extérieur ???

Grâce au sky tout simplement !
Vous savez, la texture sky peut faire bien des choses que l'on ne soupçonne pas. Dans ce cas, elle va servir à créer des murs de skys, afin de créer des salles.
Il faut impérativement que le sky différencie les salles de votre map, et que les murs de sky touchent le sky du haut, un peu comme ceci :



Vous pouvez voir ici une "salle" en extérieur. Observez surtout comment le sky est placé... Il est partout présent ! Les murs de sky partent du haut des toits et s'arrêtent au sky tout en haut.
Et c'est pareil pour toutes les autres "salles" de la map ! Le joueur est constamment entouré de skys, mais lorsqu'il joue sur votre map il ne s'aperçoit pas de cette supercherie !

C'est dans ce sens-là que le moteur de Half-Life est bien fait : vous pouvez mettre du sky où vous voulez, il se chargera d'afficher une image crédible et le joueur n'y verra que du feu : il ne verra même pas qu'il se trouve dans une "salle" entourée de sky !

Efforcez-vous donc de respecter cette méthode pour vos futures maps. Vous verrez que de cette manière vous n'aurez pratiquement plus jamais à vous plaindre des r_speeds !

Si j'ai tellement insisté sur le sky, c'est pour que vous évitiez de commettre cette (grosse) erreur de débutant... Bon, je n'aime pas montrer le mauvais exemple mais là je crois que c'est nécessaire.
Voici ce qu'on appelle le **sky box**, que vous ne devez surtout pas faire :

Le défaut grave ici, c'est que le sky entoure littéralement toute la map (contrairement à tout à l'heure où l'on ne voyait qu'une pièce de la map) ! C'est cela un sky box... et du coup il n'y a qu'une seule pièce ! Où que l'on se trouve dans cette map, tous les polygones sont calculés !

Et comme vous pouvez le voir, il y a beaucoup trop de polygones à la fois... Du coup, je vous laisse imaginer les r_speeds...

Eh oui, dans ce cas c'est toute la map qui serait à revoir : il faudrait la restructurer en plusieurs salles. La map que vous voyez là est donc un mauvais exemple ! Evitez de faire pareil, parce que c'est bel et bien à cause de ça que vous vous retrouvez avec des r_speeds faramineuses !

Eviter le découpage des polygones

Grâce à ce que vous venez d'apprendre, vous ne devriez presque plus jamais avoir de problèmes de r_speeds.
Je dis "presque" car il peut arriver parfois une situation assez gênante... à cause par exemple du découpage des polygones.

Il existe des méthodes plus ou moins délicates qui vous permettront d'éviter ce genre de problème que nous avons vu dans le précedent chapitre.

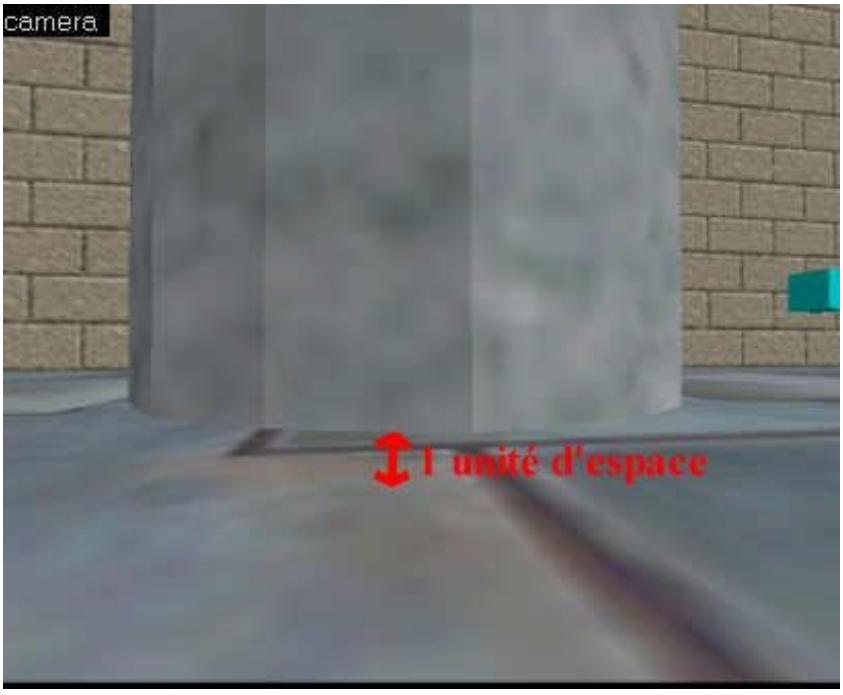
Eviter le contact direct des polygones

Lorsqu'il y a découpage des polygones, c'est que 2 blocs sont en contact l'un avec l'autre, et que le plus petit découpe le plus grand...

Si vous voulez éviter ce genre de désagrément, la première méthode consisterait à éviter que les blocs ne se touchent !

Prenons un exemple : lorsque vous créez une pièce avec des piliers au milieu. Ceux-ci étant des cylindres, ils risquent de méchamment découper le sol !

L'astuce ici consiste à laisser un espace infime entre le pilier et le sol (1 unité suffit). Si cet espace est suffisamment petit, le joueur en passant par là ne devrait pas s'en rendre compte... et comme les blocs ne se touchent pas, il n'y aura pas de découpage du sol !



Si vous laissez juste 1 unité d'espace entre le pilier et le sol, je peux vous assurer que ça ne se verra pas. D'ailleurs, même sous Worldcraft on a beaucoup de mal à repérer cette supercherie.

L'astuce du *func_wall*

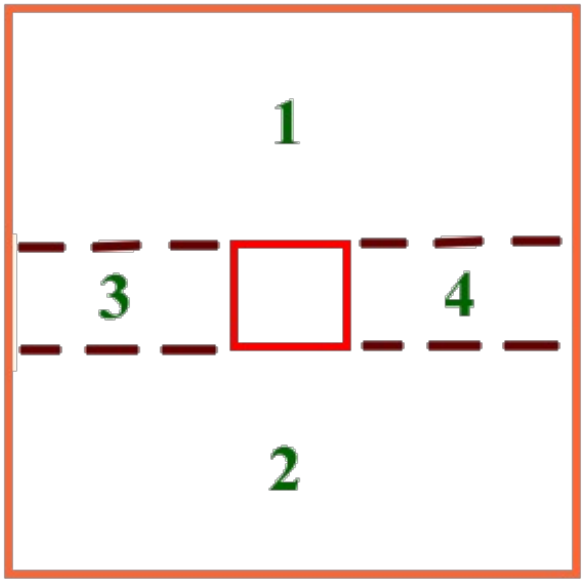
Comme je vous l'ai appris dans le chapitre précédent, les blocs se découpent allègrement entre eux... mais je n'ai jamais parlé des entités-blocs.
Y a-t-il découpage des polygones avec des entités-blocs ? Eh bien non ! Voilà pourquoi entre autres l'entité *func_wall*, de toute apparence identique à un bloc, peut nous être très utile.

Comment peut-on s'en servir ?

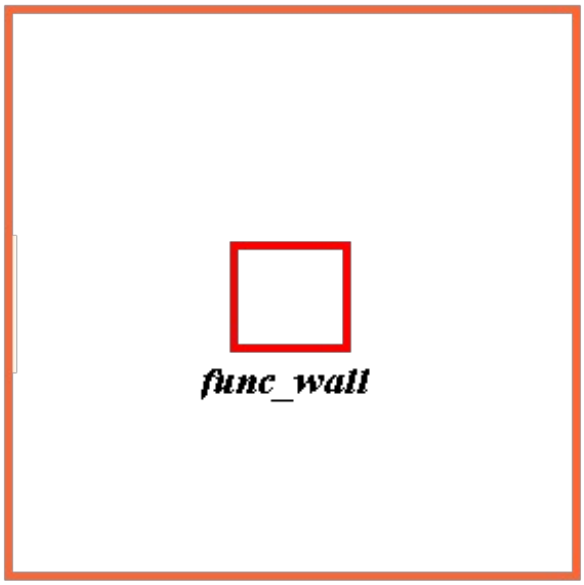
Dans le cas des piliers que nous venons de voir, plutôt que de laisser un espace entre le pilier et le sol, on aurait pu convertir le pilier en *func_wall*. Même s'il le touchait, le pilier n'aurait pas découpé le sol !

Bingo

Rappelez-vous aussi du test que l'on avait fait dans le chapitre précédent avec une caisse au milieu d'une salle. Dans ce cas-là, le sol était découpé comme ceci :



Si la caisse est convertie en *func_wall*, alors le sol ne sera plus découpé !



Attention cependant à ne pas abuser de cette méthode... En effet, je vous ai déjà dit que les entités sont calculées même si elles sont séparées par 2 portals : cela veut donc dire que les func_wall son beaucoup plus souvent calculés que les blocs normaux.. et risqueraient donc de faire augmenter vos r_speeds plutôt que de les baisser si vous en utilisez trop !

Ah là là, c'est dur quand même le mapping lol

Les Hint Brush

Introduits avec les ZHLT, les Hint Brush peuvent être un moyen très pratique de faire baisser vos r_speeds... à condition que votre map soit déjà bien structurée.

Il peut arriver dans certains cas, même si votre map est bien structurée, que Half-Life calcule des polygones alors que ça ne sert à rien. Les Hint Brush sont des informations qui vont en quelque sorte interdire à Half-Life de calculer certains polygones.

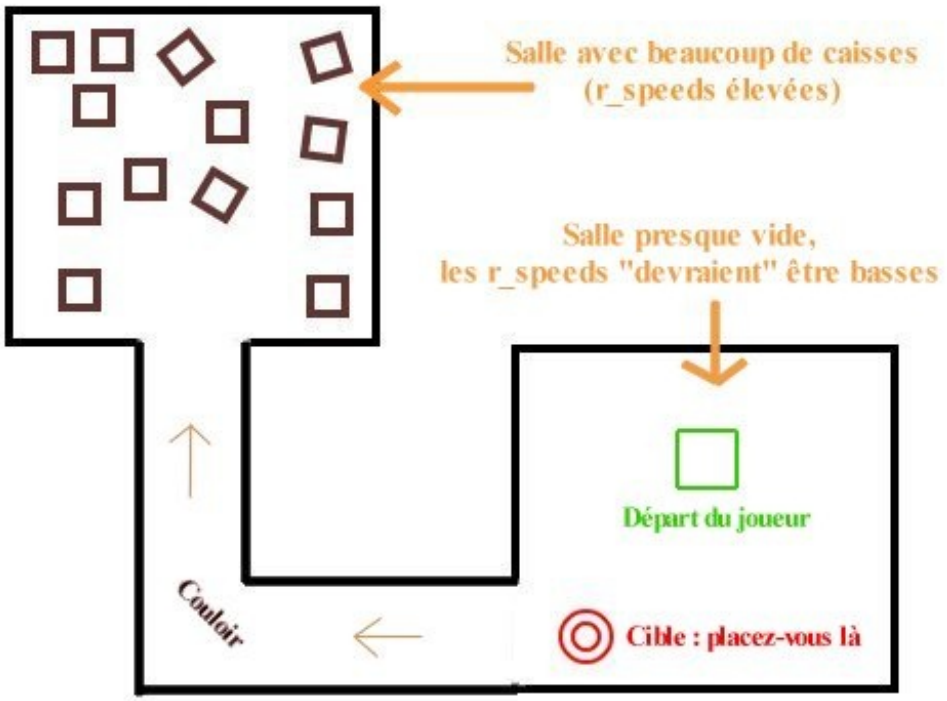
Oui mais voilà, chaque situation est très différente, et se sera à vous de bien réfléchir à chaque fois avant de vous servir des Hint

Brush... En plus, il ne faut pas oublier que le monde est en 3D : il faut aussi prendre en compte les éléments sur différents étages !

Moi, je vais vous montrer un exemple typique où les Hint Brush peuvent nous servir. A vous d'adapter cet exemple à votre map !

Exemple d'une map

Voici le plan schématique de la map "Hint" que j'ai créée pour l'occasion :



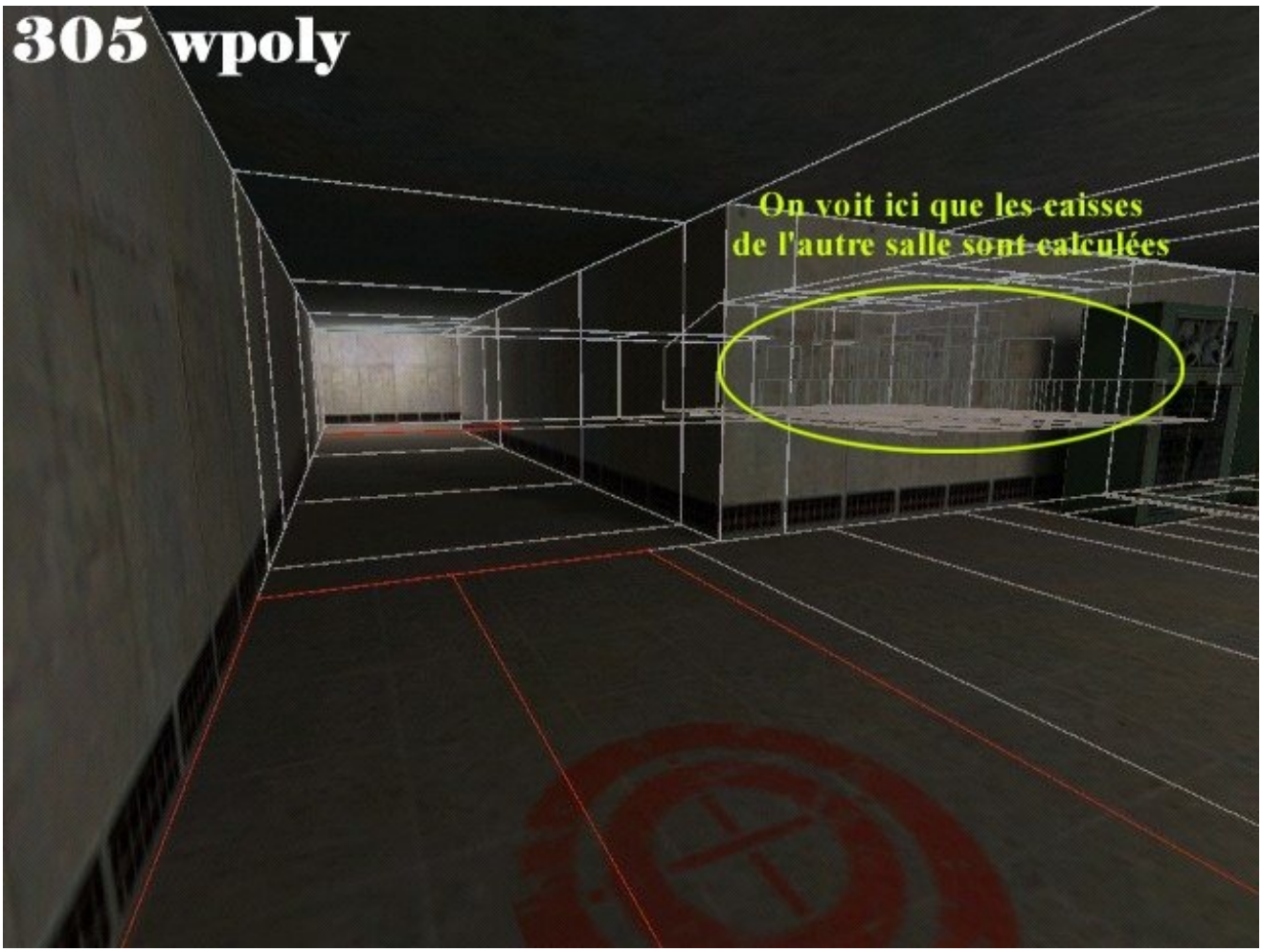
L'histoire est assez simple : le joueur part de la salle en bas à droite, puis traverse un couloir pour se rendre dans la salle en haut à gauche.

La salle en haut à gauche contient beaucoup de caisses que j'ai placé un peu n'importe comment histoire de faire monter les r_speeds. Il est normal que, lorsqu'on rentre dans cette pièce, les r_speeds se mettent à monter un peu...
Mais ce qui n'est pas normal, c'est que même si le joueur se trouve dans la pièce en bas à droite, il ait des r_speeds trop élevées (alors qu'il n'y a rien dans cette pièce) !

Ben ça alors ! Half-Life calcule les polygones des caisses de l'autre salle ?

Oui ! Si on place le joueur sur la cible et qu'il regarde en direction du couloir, il pourra voir ses r_speeds augmenter terriblement... à cause des caisses de l'autre salle !
Dans ce cas, c'est plutôt agaçant : le joueur ne voit pas ces caisses, et pourtant Half-Life les calcule. Cela fait que **les r_speeds sont élevées partout dans cette map !**

Vous êtes sceptiques ? Voici un screenshot que j'ai pris en jouant sur cette map (je me suis placé sur la cible et j'ai activé le "gl_wireframe 2") :



Résultat : 305 wpoly !!! C'est aberrant, c'est stupide parce qu'on ne voit pas les caisses, mais c'est comme ça.

Explications

Pourquoi Half-Life calcule-t-il ces polygones ???

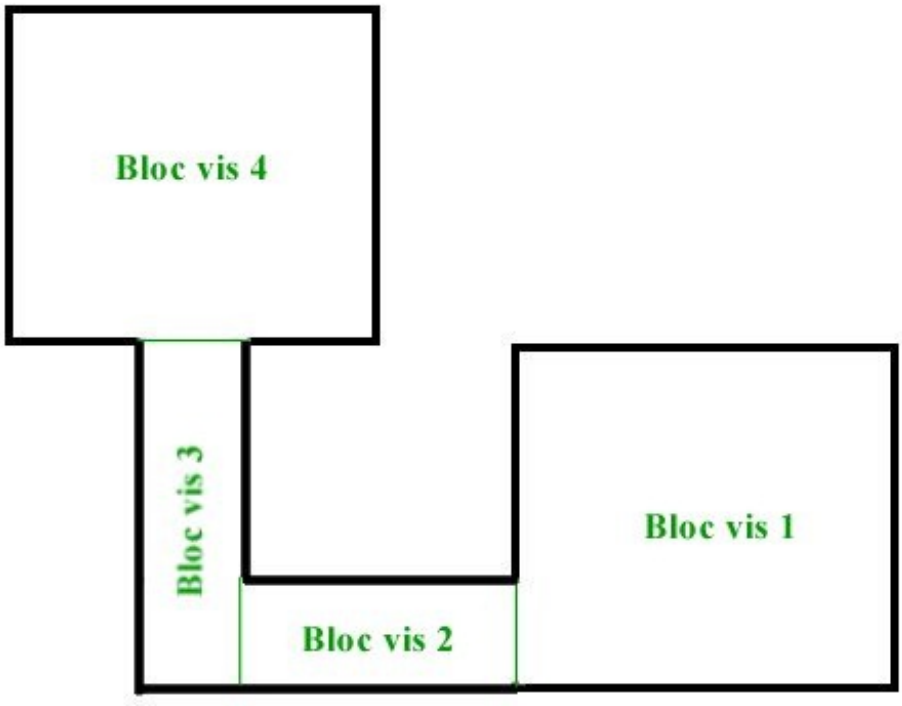
C'est à cause de HLvis (encore lui !), qui, en compilant la map, a "cru" que ces polygones étaient visibles (alors que ce n'est visiblement pas le cas).

Il faudrait ici revenir à la source : les portals. Ici, il y a un portal qui dit "SI le joueur se trouve sur la cible ET qu'il regarde en direction du couloir, ALORS il faut calculer les polygones de l'autre salle".

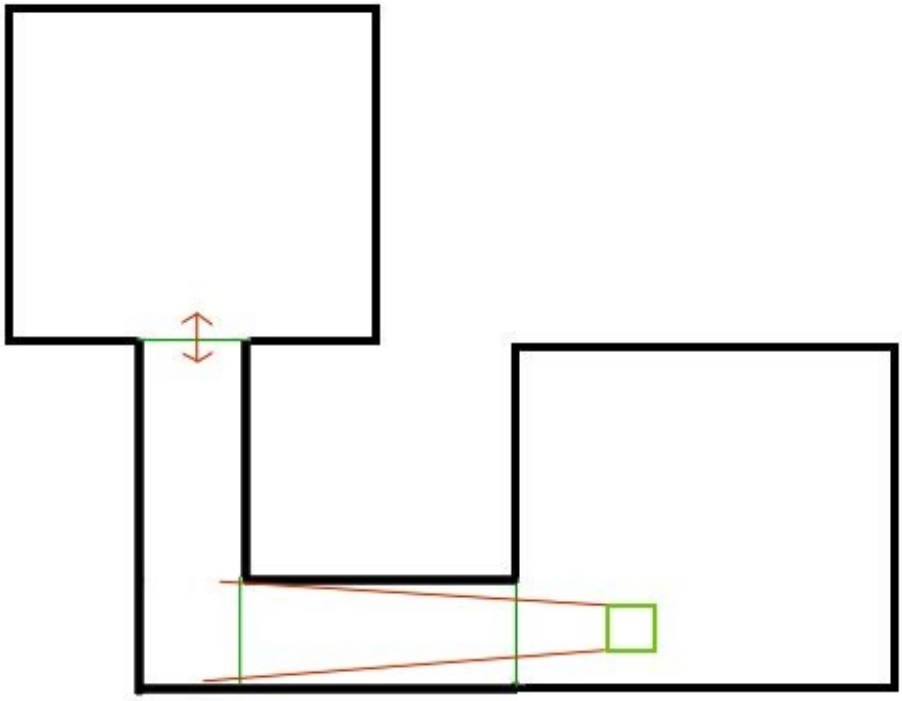
HLvis divise à sa manière les blocs, en ce qu'on appelle des "blocs vis".

Grosse erreur à ne pas faire : il ne faut surtout pas confondre les blocs et le découpage des polygones avec les blocs vis. Ca n'a RIEN à voir, le découpage est différent !

Voici la division qu'il a dû faire dans ce cas (schéma simplfié) :



Supposons maintenant que le joueur se place sur la cible et qu'il regarde vers le couloir :



Il "voit" un bout du bloc vis 3. Dès lors qu'un bloc vis est vu, HLvis pense qu'il faut calculer tous les autres blocs vis avec lesquels il est en contact... c'est-à-dire le bloc vis 4 avec toutes les caisses qui vont avec !

Moralité : 305 wpoly pour rien

Heureusement, les Hint Brush vont nous tirer d'affaire !

Apparence des Hint Brush

A quoi ressemble un Hint Brush ?

Pour pouvoir introduire des Hint Brush dans votre map, il faut tout d'abord charger dans Worldcraft le fichier de textures "ZHLT.wad"

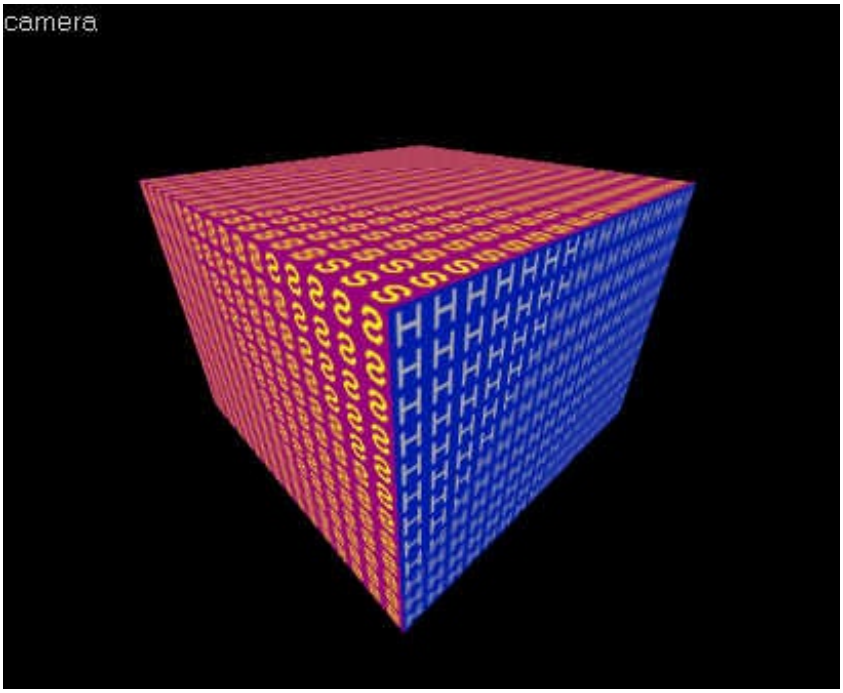
Vous ne trouverez pas ce fichier dans le répertoire de Half-Life mais dans celui où vous avez installé les ZHLT !

Ce wad contient 3 textures dans les versions récentes des ZHLT, et seulement 2 textures dans les anciennes versions.
La texture rajoutée est NULL : tout bloc possédant cette texture ne sera pas compilé par les ZHLT dans le *.bsp. Cela n'a aucun rapport avec les Hint Brush, donc vous pouvez oublier cette texture

Pour ce qui est des 2 autres textures, voici à quoi elles ressemblent :

HINT	La texture HINT, c'est celle qui va donner une information à HLvis lorsqu'il calculera les portals. Elle permettra en quelque sorte d'éviter que de trop nombreux blocs soient calculés pour rien.
SKIP	Quant à la texture SKIP, elle est toujours utilisée en même temps que HINT. En fait, cette texture ne sert strictement à rien ("Skip" en anglais signifie "Passer"). Seul HINT va donner une information : SKIP n'a aucun effet mais on en aura besoin, vous allez voir...

Un Hint Brush sous Worldcraft, c'est en réalité un bloc normal à 6 faces, dont toutes les faces sont recouvertes de la texture SKIP, sauf une (et pas n'importe laquelle) qui possède la texture HINT :



La position de ce bloc va donner des informations précises à HLvis pour la compilation (nous verrons cela plus bas). Une fois que HLvis s'en est servi, bien entendu il supprime le bloc pour qu'il ne soit plus visible lorsque vous jouez sur votre map

Fonction des Hint Brush

Comme nous l'avons vu tout à l'heure, HLvis s'est très mal débrouillé en découpant les blocs vis... ce qui nous a amené à avoir des r_speeds élevées pour rien !

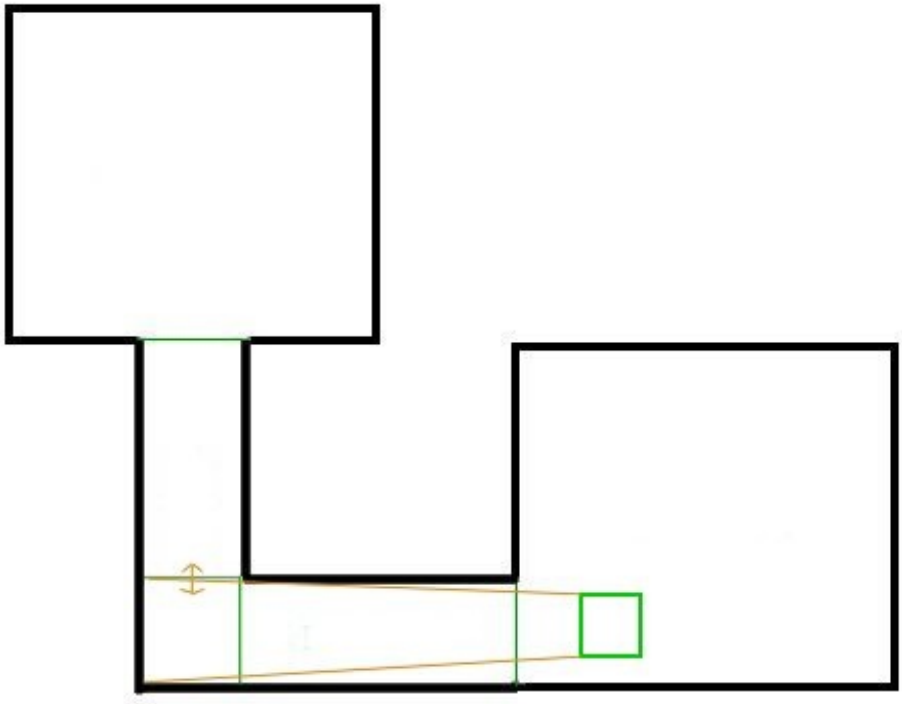
Concrètement, les Hint Brush vont nous permettre d'ordonner à HLvis de découper les blocs vis d'une autre manière. Si vous

prenez le temps de réfléchir 5 minutes sur votre problème, vous trouverez tous seuls où vous devez placer vos Hint Brush !

Comme je vous l'ai dit, seule la face recouverte de la texture HINT va avoir un effet. Toutes les autres (avec la texture SKIP) seront oubliées.

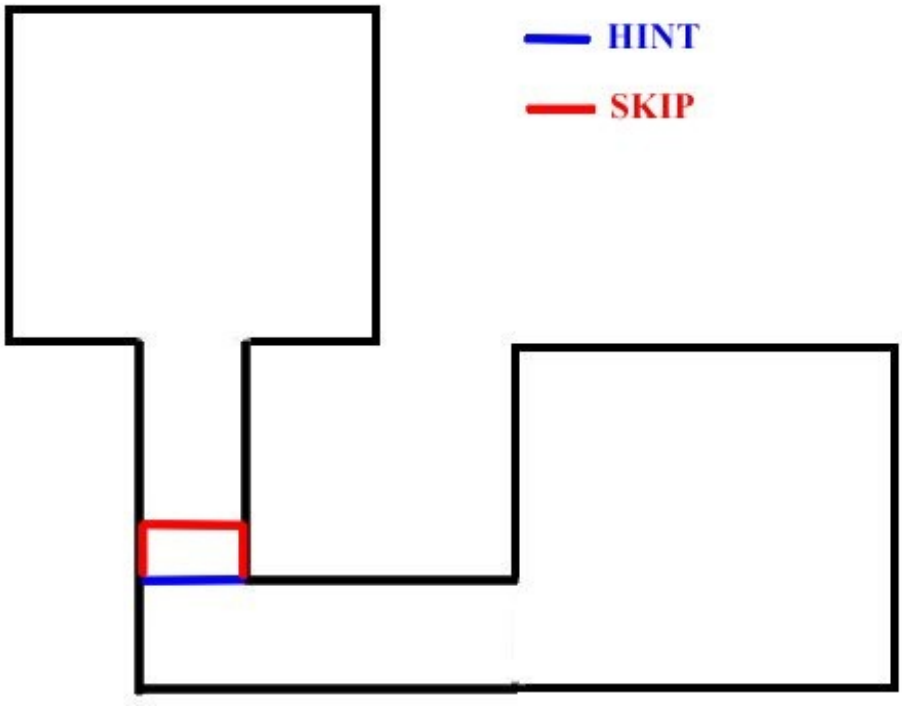
La face (et je dis bien la face) recouverte de la texture HINT va découper les blocs vis avec lesquels elle est en contact.
Voilà tout simplement à quoi sert un Hint Brush, et si vous avez bien suivi mes explications jusqu'ici, alors vous venez de tout comprendre !

Sur le schéma des blocs vis, voici le découpage idéal que l'on aimerait obtenir :



Vous voyez, ici on a rajouté un découpage des blocs vis au niveau de l'intersection du couloir... Cela fait que seule l'autre partie du couloir sera calculée (puisqu'elle est en contact direct avec le bloc vis de l'intersection), et du coup l'autre pièce avec ses nombreuses caisses n'est pas calculée !

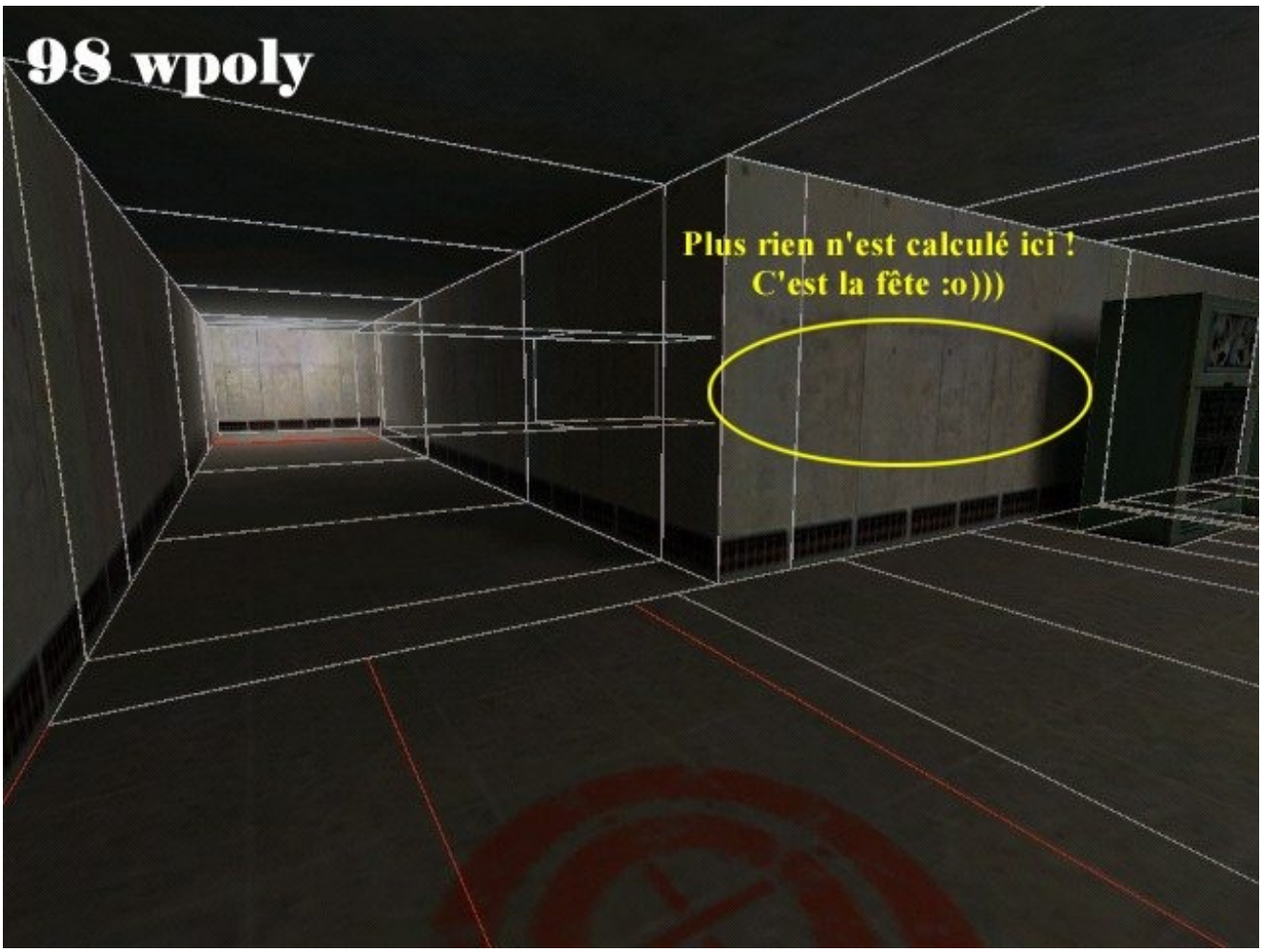
Pour faire ce découpage, j'ai donc placé mon Hint Brush comme ceci dans Worldcraft :



La face Hint a découpé les blocs vis et le tour était joué !

Conclusion

Ah, il ne me reste plus qu'à vous faire admirer le résultat :



La différence se fait ressentir : ça nous fait 200 wpoly de moins d'un coup !!! Quel soulagement

A noter que si j'avais mis encore plus de caisses dans l'autre salle, les Hint m'auraient fait descendre encore plus les wpoly (la différence aurait été plus grande). Bref, vive les Hint Brush !
Le tout est de savoir les placer judicieusement, et surtout de ne pas en abuser...

Ne retenez pas simplement que les Hint Brush servent à faire baisser vos r_speeds ! Parfois, c'est la structure de votre map qui est à remettre en cause, et les Hint Brush ne peuvent plus rien faire d'autre pour vous !
De plus, il faut éviter de mettre trop de Hint Brush car ils augmenteraient beaucoup le nombre de blocs vis... Plus il y a de blocs vis, plus il y a de portals à calculer, et plus la compilation est longue.

D'ailleurs, j'ajouterai à ce sujet que si vous doublez le nombre de blocs vis dans votre map, alors le temps de compilation quadruple !!! Mais c'est très mathématique alors ne cherchez pas trop à comprendre lol
Ca veut juste dire qu'il ne faut pas trop en abuser. Enfin, dernière chose : je vous offre cette map de test appelée "Hint" dont je me suis servi tout au long des explications. Vous pouvez la télécharger ci-dessous :

Hint.zip (73Ko)

Elle comprend le *.rmf et le *.bsp compilé en "Full compil". Je vous conseille de la lancer en Half-Life Solo pour pouvoir profiter du "gl_wireframe"

Les overviews

Beuh, c'est quoi un overview ?

Introduits avec la HLTV, et maintenant disponibles en tant que spectateur sous Counter-Strike, les overviews sont des vues de dessus de votre map.

Elles permettent donc d'avoir une idée globale de la forme de la map, et de savoir où se trouvent les joueurs sur la map.
Lorsque votre map est terminée, vous devriez créer un overview ! A noter que ce que je vais vous apprendre vous sera aussi très utile pour avoir un plan d'une map et élaborer une stratégie dessus avec sa team !

Récupérer l'overview

Première étape : on va récupérer le fichier d'overview sous Half-Life ainsi que d'autres informations importantes.
Suivez bien mes instructions dans l'ordre et vous verrez que c'est très facile !

- Lancez Half-Life ou Counter-Strike en mode "Console" (paramètre -console). Normalement, la plupart des joueurs ont la console activée.
- Vérifiez que la résolution sous Half-Life est bien définie à 1024 x 768 (j'espère que vous êtes en OpenGL ou DirectX).

  Il est très important de mettre la résolution à 1024 x 768 ! C'est la taille obligatoire pour l'overview !
- Cliquez sur le bouton "Console" du menu principal :

- La console s'affiche. Pour l'instant elle est vide. Je vais vous demander de rentrer la commande suivante :

  sv_cheats 1

  Pensez à taper Entrée pour valider.
- Vous devez ensuite lancer votre map avec la commande "map". Par exemple, pour la map "cs_thunder" :

map cs_thunder

Tapez Entrée et votre map sera chargée.

- Lorsque votre map est chargée, vous ne devriez rien voir à l'écran à part la map (pas de HUD : c'est-à-dire pas d'indications de munitions, de vie etc...). Cela est tout à fait normal.

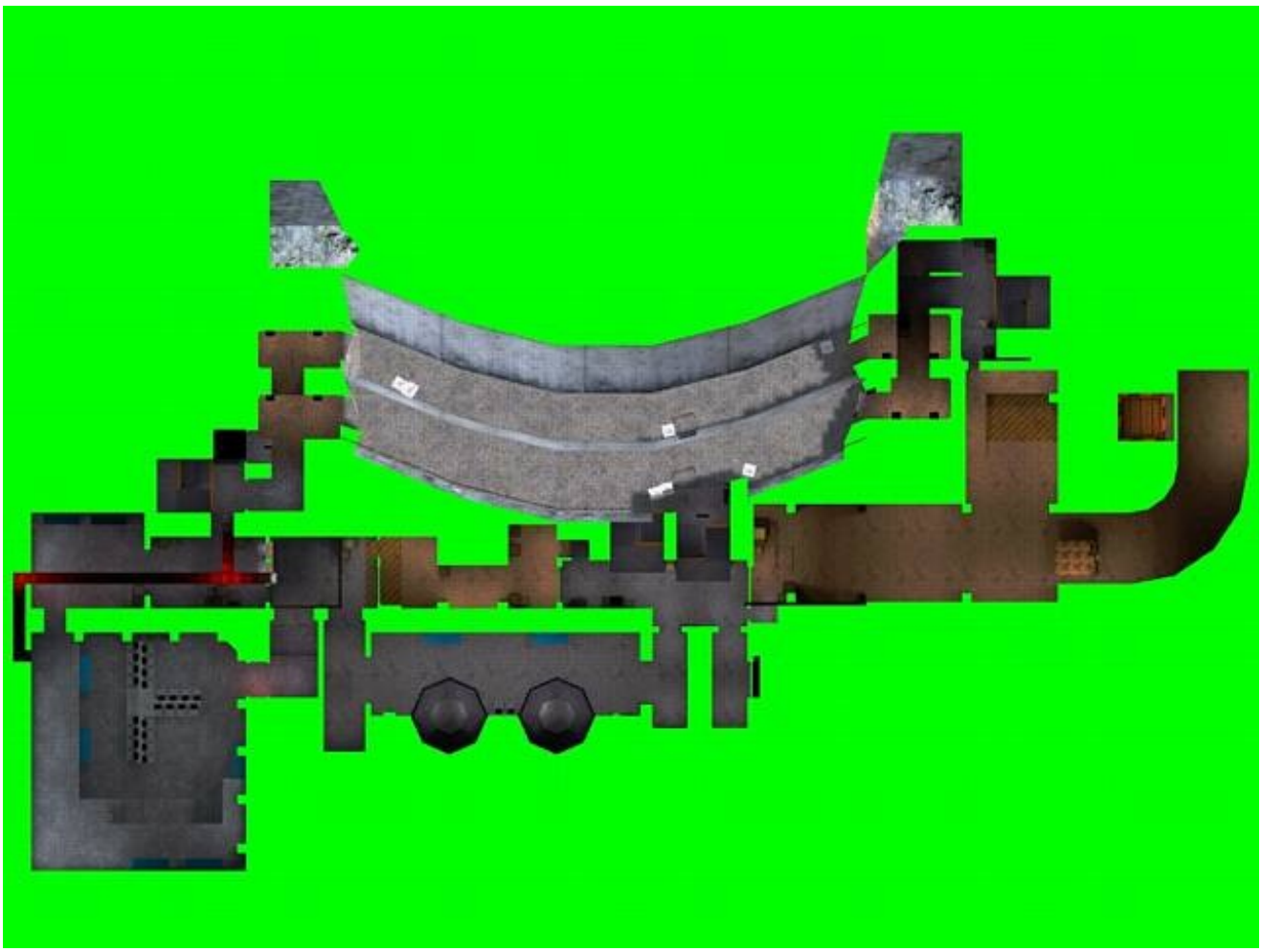
  S'il reste encore le HUD, tapez la commande "hud_draw 0". Si vous avez mis le net_graph (indication des fps en temps réel), tapez "net_graph 0"

  Voilà maintenant votre écran doit être parfaitement vierge, comme ceci :

- Ouvrez la console avec la touche "²", et tapez la commande suivante :

  dev_overview 1

  Retirez la console puis admirez l'overview que Half-Life a généré :



Le vert fluo sera rendu transparent à la fin, ne vous en faites pas pour ça

- Il va maintenant falloir modifier cet overview jusqu'à ce qu'il nous convienne. Vous avez la possibilité de zoomer :
  - Zoom avant : maintenir le bouton gauche de la souris enfoncé.
  - Zoom arrière : maintenir le bouton droit de la souris enfoncé.

  Vous avez aussi la possibilité de centrer votre overview : servez-vous pour cela des flèches du clavier. Essayez si possible de placer le joueur au centre de la map pour que l'overview soit centré.
- Une fois que l'overview vous convient, tapez F5 pour prendre un screenshot.
- Activez ensuite la console. Vous devriez voir de nombreuses lignes contenant des informations qui vont nous être précieuses. Relevez seulement de la dernière ligne sur un bout de papier.

Je vous recopie cette ligne pour que vous puissiez la lire plus facilement :

Overview: Zoom 1.22, Map Origin (-578.00, -144.00, -128.00),
Z Min 513.00, Z Max 763.00, Rotated 0

- Voilà, vous pouvez maintenant sortir de Counter-Strike, la manoeuvre est terminée.

Convertir l'overview

Cette deuxième étape sera normalement plus rapide. Le seul hic, c'est que vous devriez avoir besoin d'un logiciel de dessin un peu plus évolué que Paint (moi j'utilise Corel Photo-Paint par exemple).

Mais en quoi va-t-on convertir le fichier d'overview ?

Un overview doit être un bitmap (ça c'est bon), mais en 256 couleurs (8 bits). Or, le screenshot que nous avons pris est en 24 bits ! (voir le cours sur les Images numériques pour plus d'infos là-dessus).

L'objectif est donc : transformer l'image en 8 bits.

Paint peut le faire, mais la palette de couleurs n'est pas adaptée : du coup votre overview risque d'être sacrément moche ! Voici 2 moyens pour y parvenir :

- Sous Corel (ou un autre bon logiciel de dessin) : ouvrir le bitmap. Aller dans le menu "Image / Convertir en / 256 couleurs (8 bits)". Une fenêtre s'ouvre : il vous faut sélectionner le type de palette. Choisissez "Adaptée" et validez.
  Voilà, il ne vous reste plus qu'à enregistrer le bitmap dans le dossier "Half-Life\cstrike\overviews" avec le même nom que la map. Ici : "cs_thunder.bmp".
- Sous Paint : c'est vraiment plus délicat et je vous recommande de vous servir d'un autre logiciel. Si toutefois vous n'avez pas le choix, essayez l'astuce suivante :
  - Ouvrez un overview déjà existant dans une première fenêtre de Paint (ex : "de_dust.bmp").
  - Ouvrez votre overview dans une deuxième fenêtre.
  - Dans votre overview, faites CTRL + A (Sélectionner tout) et CTRL + C (Copier).
  - Dans la première fenêtre (celle qui contient dust par exemple), faites CTRL + V (Coller).
  - Toujours dans la première fenêtre, allez dans le menu Fichier / Enregistrer Sous, et enregistrez l'overview dans le dossier Half-Life\cstrike\overviews, sous le nom de la map ("cs_thunder.bmp" dans mon cas).

Voilà qui est fait, votre overview est presque prêt !

Il nous reste maintenant à éditer ses propriétés...

Edition des propriétés

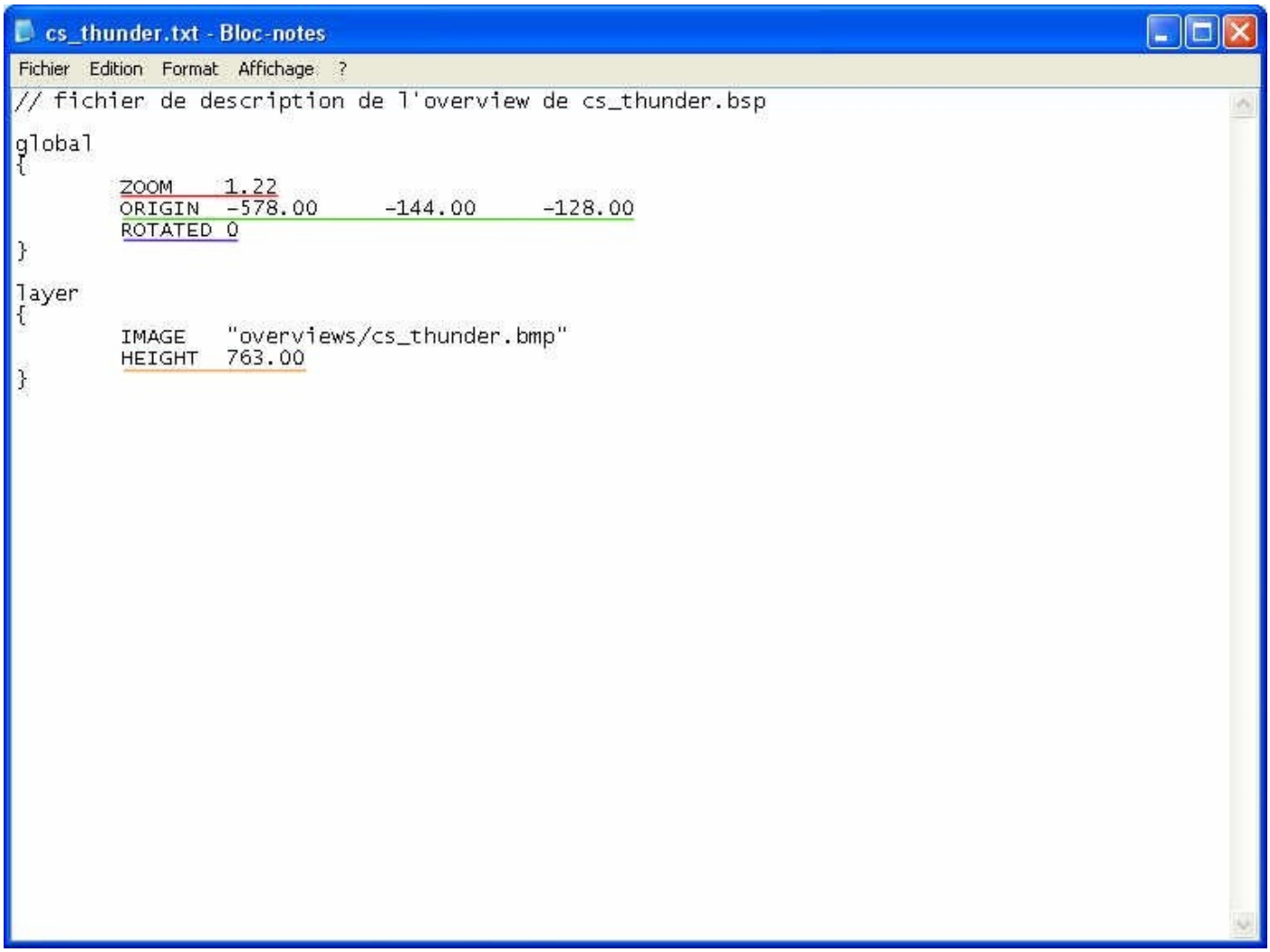
Un overview est en fait composé de 2 fichiers : le bitmap et un fichier texte (contenant les propriétés de l'overview).

Nous allons devoir créer ce fichier texte. Je vous recommande fortement d'en ouvrir un déjà existant, puis de faire "Enregistrer Sous". Enregistrez le fichier dans le dossier "Half-Life\cstrike\overviews", avec le même nom que la map ("cs_thunder.txt" par exemple).

Vous allez maintenant devoir modifier les valeurs, en vous servant pour cela des informations que vous avez noté sur un bout de papier...
Je vous recopie ici celles que j'avais noté pour cs_thunder :

Overview: Zoom 1.22, Map Origin (-578.00, -144.00, -128.00),
Z Min 513.00, Z Max 763.00, Rotated 0

Voici ce que ça doit donner dans le fichier texte :



Aidez-vous des codes de couleur que je vous ai donné pour remplir à votre tour le fichier texte. N'oubliez pas de renseigner le champ "Image" par le nom du bitmap de l'overview.

Voilà, si vous avez correctement recopié les valeurs et enregistré les fichiers avec les bons noms, votre overview est prêt !

Pour le tester, il faudrait que vous soyez plusieurs à jouer sur la map (servez-vous des bots par exemple).
Lancez une partie LAN sur votre map, et au moment de choisir les équipes, cliquez sur "Spectator". Appuyez sur "Espace" autant de fois que nécessaire afin d'avoir la vue de l'overview.

Si vous pouvez voir l'overview, c'est que vous avez tout fait comme je vous ai dit, bravo !

Notez que le bitmap est certainement perfectible, il faudrait un peu retoucher les bords qui ne sont pas complètement verts, et qui du coup se voient encore sur l'overview (ça arrive rarement rassurez-vous).

Vous pouvez maintenant distribuer l'overview avec votre map lorsqu'elle est terminée. Pensez-y, surtout si vous participez au concours de mapping !

Partie 4 : Annexes

(En complément du cours, voici les annexes.
Vous y trouverez des informations complémentaires.
Cela peut se révéler très utile...)

Les waypoints

Waypoints ? Késako ?
Beaucoup de mappeurs ne savent pas du tout ce qu'on appelle "waypoints", d'autre savent ce que c'est mais ne savent pas les utiliser...

Grâce à cette annexe, vous pourrez jouer seul sur votre map, contre des bots. Dès que vous avez terminé une map, il est conseillé de livrer les waypoints. C'est le meilleur moyen pour que le joueur reste longtemps sur votre map, et ça voudra dire qu'il s'amuse comme un petit fou

Bien, dans un premier temps on va voir ce que sont ces fameux waypoints.

Qu'est-ce que c'est ?

Primo, les waypoints dont je vais vous parler ne concernent que le mod Counter-Strike.
Il s'agit en fait d'une fonctionnalité particulière des PODBots.

Mmh, c'est quoi les PODBots ?

Counter-Strike est un mod exclusivement multijoueur. Oui, mais voilà, si vous avez un petit forfait Internet limité et que vous ne pouvez pas rester connecté des heures, ça va être difficile de jouer longtemps !
Heureusement, on a inventé les Bots. Ce sont des joueurs gérés par l'ordinateur, qui possèdent une IA (intelligence artificielle).
Du coup, ça vous permet de jouer déconnecté à Counter-Strike contre votre ordinateur !

Il existe plusieurs types de Bots (c'est un peu comme des "marques"), mais les plus connus et les plus utilisés de loins sont les PODBots. Pour les télécharger (si vous ne les avez pas déjà), rendez-vous sur vossey.com par exemple.

Une fois que vous les avez installé, vous pourrez jouer sur les maps officielles contre les bots... Mais bizarrement, si vous jouez sur une map non officielle (par exemple la map que vous venez de créer), les bots n'apparaîtront pas...

Pourquoi il n'y a pas de bots qui jouent sur ma map ?

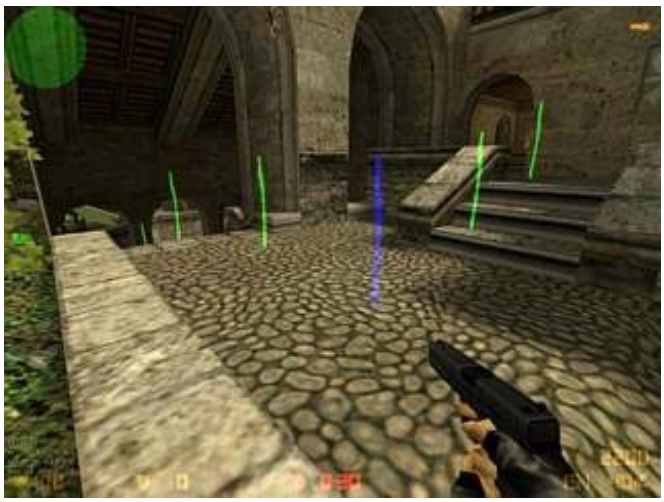
Parce qu'ils ne la connaissent pas. Les Bots doivent connaître une map pour jouer dessus avec vous. Sinon, vous auriez affaire à des aveugles qui ne savent pas du tout où il vont.

Pour "apprendre" aux bots à jouer sur votre map, vous allez créer des **waypoints**.

Les waypoints ressemblent à des balises, qui servent à "guider" les bots. Par exemple, il existe une balise pour dire "Tu peux marcher là, vas-y il n'y a pas de mur qui va te gêner", ou bien "Il y a une échelle, tu peux monter si tu veux", ou encore "Les otages sont ici, et le point de secours des otages est là".
Vous comprenez le principe ? Ca peut paraître compliqué à faire, mais vous verrez que c'est en fait très simple. Et après avoir créé vos waypoints, vous pourrez enfin jouer sur votre map contre des bots

A quoi ressemblent ces waypoints ?

Vous pouvez les voir lorsque vous jouez sur une map en mode "Waypoints activés". Les waypoints ressemblent à ceci :



Comme vous pouvez le voir, chaque waypoint a la forme d'un gros bâton posé au sol. Ils indiquent où peuvent passer les bots. Ici par exemple, on voit bien que les bots ont le droit de descendre (ou monter) les escaliers.
Il existe plusieurs types de waypoints, comme nous le verrons plus tard. Les waypoints verts par exemple, signifient simplement que les bots peuvent passer par là.

Commandes de base

Comment fait-on alors pour passer en mode "Waypoints activés" et pour poser des waypoints ? Eh bien, il va falloir taper des commandes dans la console.
Lancez votre map en LAN (menu Multijoueurs / LAN). Une fois que vous êtes dans votre map, vous devez afficher la console en appuyant sur la touche "²" (carré), située en haut à gauche de votre clavier.

C'est maintenant que vous devez taper des commandes (des mots-clés) pour activer les wayoints.

Attention ! Je vous rappelle que votre clavier sous Counter-Strike est en QWERTY ! Il va falloir vous habituer aux changements des touches pour pouvoir taper les commandes correctes (ex : A = Q, Z = W etc...).

Voici les commandes de base dont nous aurons besoin :

Commande	Signification
waypoint on	Active le mode Waypoints. C'est la première commande à taper.
waypoint add	Permet d'ajouter un nouveau waypoint. Nous aurons très souvent besoin de cette commande !
waypoint delete	Supprime le waypoint le plus proche de vous.
waypoint off	Désactive le mode Waypoints. C'est donc la dernière commande à taper

Il existe d'autres commandes un peu plus compliquées à utiliser. Je vous les montrerai petit à petit dans ce chapitre, dès que nous en aurons besoin.

Commencez donc par taper

waypoint on

Voilà, le mod Waypoints est activé.

Les binds

Nous allons bientôt voir comment placer un waypoint à l'aide de la commande "waypoint add". Oui, mais comme il y a en général des centaines de waypoints par map, ça risque d'être long de taper à chaque fois "waypoint add / waypoint add / waypoint add / waypoint add...". Heureusement qu'il existe les binds !

Un bind consiste à attribuer une commande à une touche. Par exemple, en appuyant sur la touche N la commande "waypoint add" est automatiquement exécutée. C'est quand même plus rapide !

Pour faire un bind, utilisez la commande suivante par exemple :

bind n "waypoint add"

Rentrez d'abord bind, puis la touche à binder, et enfin la commande correspondante entre guillemets.
Voilà, maintenant vous pourrez, sans ouvrir la console, ajouter un waypoint simplement en appuyant sur la touche que vous voulez ! (ici il s'agit de la touche N)

Placer et lier des waypoints

Nous allons placer maintenant notre premier waypoint à l'aide de la commande "waypoint add" que je viens de vous montrer (n'hésitez pas à vous servir de votre bind !).

Lorsque vous tapez cette commande, un menu s'affiche à gauche de l'écran :

On vous demande quel type de waypoint vous voulez poser (je vous rappelle qu'il en existe plusieurs).
Pour commencer, on ne va pas se préoccuper des différents types de waypoints, on va donc poser le waypoint normal, celui que l'on rencontre le plus souvent.

Tapez "1" pour poser un waypoint normal.

Vous devriez entendre un petit "tac", qui veut dire que le waypoint a été posé. Retournez-vous, il est juste derrière vous. Si tout va bien vous devriez voir un waypoint vert comme ceci :

Félicitations ! Vous venez de poser votre premier waypoint !

Pour info, la map que je waypointe s'appelle de_piranesi, une map officielle qui ne disposait malheureusement pas de waypoints (car elle est nouvelle pour les PODbots).
Ici je me situe au départ des terros (au niveau du bateau), et j'ai posé un waypoint à cet endroit pour indiquer aux bots qu'ils doivent passer par là pour monter.

Vous devrez mettre de nouveaux waypoints verts au fur et à mesure de la montée, pour que les bots sachent où avancer. Voici ce que vous devriez voir lorsque c'est fait :

Voilà, c'est tout simple comme vous le voyez ;o)
Les bots comprendront qu'ils doivent monter.

Ensuite, il va vous falloir mettre des waypoints dans la zone en haut.

Laissez un espace suffisant entre les waypoints, et n'en mettez pas une concentration trop importante ou sinon les bots ne sauront plus où donner de la tête.

Voici ce que ça donne une fois que la partie en hauteur a été correctement waypointée :

C'est l'espace qu'il faut entre les waypoints. N'en mettez pas plus à la fois, ça ne sert à rien !

Autowaypoint

Si la map est grande, ça risque d'être un peu gonflant de la parcourir en entier pour mettre des waypoints. Même si vous avez fait un bind, il est vrai que c'est peu pratique.

Heureusement qu'il existe l'autowaypoint !

Une fois activée, cette commande magique se charge de placer des waypoints verts sur votre passage. Elle respecte une distance entre les waypoints (elle n'en met pas trop, juste ce qu'il faut).
Vous, tout ce que vous aurez à faire, c'est de parcourir votre map à pied, marcher partout dans tous les recoins, marcher sur les places en long en large en travers pour qu'il y ait des waypoints partout !

Tapez cette commande dans la console pour activer l'autowaypoint :

autowaypoint on

Voilà, vous pouvez maintenant vous balader tranquillement sur votre map et laisser l'autowaypoint rajouter automatiquement les waypoints :o)

Pour arrêter l'autowaypoint, tapez simplement :

autowaypoint off

Et voilà le travail !

Liaisons entre les waypoints

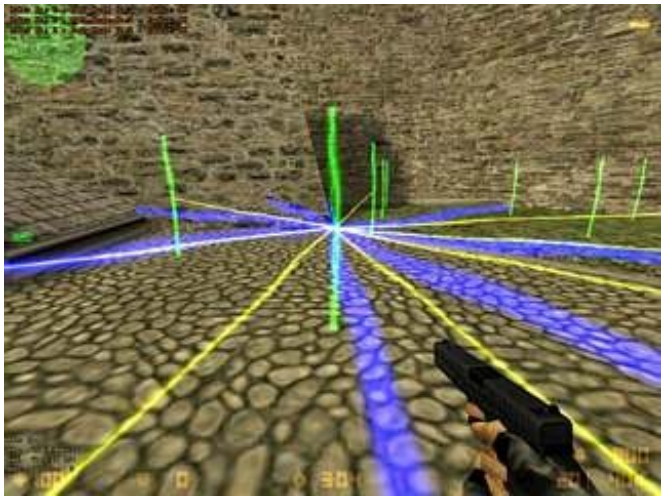
Voilà une notion importante que vous devez connaître : les waypoints sont liées entre eux, formant une sorte de gigantesque toile d'araignée.

En temps normal, vous ne pouvez pas voir ces liaisons. Mais il existe une commande pour demander de les afficher :

pathwaypoint on

A noter que "pathwaypoint off" désactive ce mode, comme vous vous en doutez
Allez-y, activez le pathwaypoint, vous allez apprendre quelque chose d'intéressant (et d'important).

C'est fait ? Alors voilà ce que vous devriez avoir :



Distinguons trois choses :

- **Les lignes bleues** : le bot considère qu'il peut aller où il veut dans la zone bleue. Ca n'a pas vraiment beaucoup d'intérêt ici.
- **Les lignes jaunes** : beaucoup plus importantes, ce sont elles qui montrent les liens entre les waypoints. Le waypoint que nous avons en face de nous est donc lié aux waypoints autour. Cela indique un chemin qu'il peut emprunter.
  En d'autres termes, la ligne jaune signifie "Tu peux te rendre au waypoint suivant sans problème, il n'y a pas de mur qui viendra te gêner".
- **Les lignes blanches** : on n'en voit pas sur mon screenshot car elles sont assez rares, mais il faut les connaître. Elles fonctionnent comme les lignes jaunes : elles indiquent un lien entre deux waypoints. Sauf que là, le lien ne fonctionne que dans un sens : le bot peut aller du waypoint A vers le waypoint B, mais il ne fera pas l'inverse !

Méfiez-vous ! Si jamais deux waypoints séparés par un mur se lient, le bot va tenter désespérement de traverser le mur !

La commande "pathwaypoint delete X" permet de supprimer un lien entre deux waypoints. Placez-vous sur le premier waypoint, et tapez cette commande en remplaçant X par le numéro du deuxième waypoint (indiqué dans la console).

De même, il est possible d'ajouter un lien à l'aide de la commande "pathwaypoint add X". A noter que, si on veut que le lien soit réciproque (ligne jaune), il faudra le faire 2 fois : la première fois à partir du waypoint A, et la deuxième fois à partir du waypoint B.

Voilà en gros ce qu'il faut savoir sur les liens entre les waypoints.
Si vous regardez la console, vous verrez que le programme crée automatiquement des liens entre les waypoints.

Les différents types de waypoints

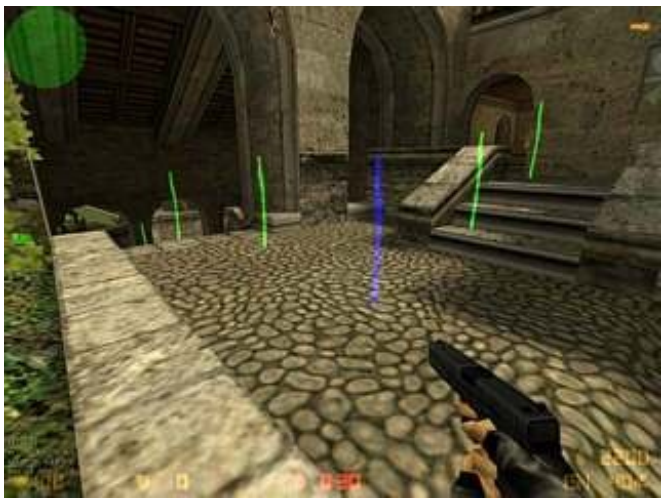
Je ne vous ai montré qu'un seul type de waypoint jusqu'ici : les waypoints verts. Mais il faut savoir qu'il en existe bien d'autres, qu'il faut absolument connaître.
Je vais vous lister ci-dessous tous les types de waypoints qui existent, chacun possèdant sa propre couleur et chacun a une

fonction différente.

- Normal waypoint : c'est le waypoint vert que nous avons vu jusqu'ici. C'est de loin celui que l'on pose le plus souvent dans une map. Sa fonction est toute simple : il indique que le bot peut passer par là. Rien de plus.
  A vous d'en mettre partout dans votre map, en vous aidant de la fonction "autowaypoint".

- Important terrorist waypoint : vous devez mettre environ 4 ou 5 waypoints de ce type dans une map. Ce waypoint ne concerne que les terros, et indique un point "chaud" de la map vers lequel ils doivent se rendre.
  Pour une DE, mettez des waypoints de ce type à plusieurs endroits de la map, de sorte à faire avancer les bots progressivement jusqu'au point de pose de la bombe. Je vous rappelle qu'il ne faut surtout pas en mettre trop, 5 ou 6 maximum !
  Pour une CS, mettez 2 ou 3 waypoints de ce type près des otages pour qu'ils les protègent. Mettez-en d'autres à des points où les 2 teams se rencontrent le plus souvent pour se fragger 

- Important counter-terrorist waypoint : inutile de me répéter, c'est le même waypoint, mais il concerne cette fois les counters.
  Dans une DE, mettez plusieurs waypoints de ce type aux points de pose de la bombe.
  Dans une CS, mettez-en au fur et à mesure du chemin qu'ils doivent emprunter pour se rendre aux otages. Je vous rappelle qu'il ne faut les mettre qu'à des points stratégiques de la map !



Pour créer une zone de méga-baston, il vous suffit de mettre un "counter-terrorist waypoint" à côté d'un "terrorist waypoint". Vu que les 2 teams vont se rendre à cet endroit, c'est le carnage garanti ! 
N'abusez pas de cette méthode, parce que sinon les bots feront tout le temps la même chose !

- Goal map waypoint : il représente l'objectif. Ce waypoint violet doit être posé là où se trouve l'objectif de la map. Il y en a donc très peu dans chaque map.
  Pour une CS, il doit y en avoir un près des otages.
  Pour une DE, il doit y en avoir un à chaque point de pose de la bombe.

- Rescue waypoint : placez un waypoint de ce type à chaque point de secours des otages (rescue point). C'est vers cet endroit que les counters se dirigeront une fois qu'ils ont les otages.

- Le ladder waypoint : c'est le waypoint pour les échelles. Il indique aux bots qu'ils peuvent monter à une échelle.

Vous devez en mettre 2 par échelle : un en bas et un en haut. Placez celui du bas dès que vous êtes "accroché" à l'échelle, et celui du haut dès que vous allez quitter l'échelle.
Si l'échelle est grande, pensez à activer l'autowaypoint pour poser des waypoints verts (normaux) au milieu afin que les bots sachent que l'échelle est grande.

Sur le screen ci-dessous, il n'a pas été nécessaire de mettre des waypoints verts au mileu car, comme vous pouvez le voir, les waypoints sont liés entre eux pas un trait jaune. S'il n'y avait pas eu de lien, il aurait fallu mettre un waypoint vert au mileu.

- Camp waypoint : ah ah, les bots savent tout faire... même les campouzes ! La preuve avec ce maaagnifique waypoint qui indique au bot de camper, si possible avec un snipe, de manière à fragger dans le dos les pauvres gars qui passeraient par là.
  Bon, par contre ce waypoint est assez difficile à utiliser au début. Tout d'abord, accroupissez-vous si nécessaire, ou restez debout si vous préférez. Regardez dans une direction (à gauche par exemple), et activez le "camp waypoint" (campstart waypoint). Ensuite, regardez à droite, et terminez le camp waypoint (avec un campend waypoint).

  Sur le screen ci-dessous, vous voyez deux lignes rouges qui indiquent le champ de vision du bot lorsqu'il campe. C'est dans cette direction qu'il va regarder, en attendant que des joueurs passent.

  Si vous restez accroupi pendant que vous posez le waypoint, le bâton posé au sol sera plus court : cela indiquera au bot qu'il doit rester accroupi lui aussi à cet endroit.

A noter qu'il existe un mode "jump waypoint", situé dans "other waypoints", qui surveille vos sauts et dira aux bots de sauter comme vous.
Même si un bot sait sauter une barrière tout seul, il n'en sera pas de même pour un précipice. C'est là que sert le "jump waypoint", qui ordonnera au bot de sauter à cet endroit.

Enregistrer et distribuer vos waypoints

Enregistrer les waypoints

Dernière étape : il va falloir enregistrer vos waypoints. Il serait dommage en effet d'avoir fait tout ce travail pour rien

La commande à taper est toute simple :

waypoint save

Si aucune erreur n'est détectée, les waypoints sont enregistrés et vous avez gagné

Attention : lorsque vous enregistrez vos waypoints, vérifiez que votre map possède son nom final. Si jamais votre map change de nom, vos waypoints ne seront plus valides et vous devrez les recréer !

Malheureusement il peut y avoir des erreurs. Si vous essayez d'enregistrer vos waypoints et qu'un message d'erreur s'affiche, c'est qu'il y a un problème (parfois, c'est un waypoint précis qui foire, d'autres fois c'est parce que vous n'avez pas mis de goal waypoint etc...).
A vous de corriger cette erreur par vous-même, ce qui peut parfois prendre un certain temps (mais ne vous découragez pas ;).

Autres commandes à connaître : "waypoint check" sert à chercher les erreurs, "waypoint load" charge les derniers waypoints enregistrés, "waypoint save nocheck" enregistre les waypoints même s'il y a des erreurs.

N'utilisez la commande "waypoint save nocheck" que pour le débogage (pour essayer), et ne livrez SURTOUT pas des waypoints invalides à d'autres joueurs ou sinon vous risquez d'avoir des centaines de joueurs sur le dos

Livrez vos waypoints

Comment donner les waypoints que j'ai créé à d'autres joueurs ?

Il suffit en fait de donner un seul fichier, au format PWF. Vous le trouverez dans le dossier : "Half-Life\cstrike\PODBot\WPTDefault".
Ce fichier a le même nom que votre map, par exemple : "cs_assault.pwf". Même si vous trouvez d'autres fichiers dans ce répertoire, c'est le seul que vous aurez à livrer.

Les longueurs

Le problème des longueurs sous Worldcraft : voilà un sujet qui inquiète beaucoup de mappeurs débutants... Du coup, ils ont tendance à faire des maps GIGANTESQUES !!!

Pourtant, quand on prend l'habitude de mapper, on ne se pose plus ce type de questions. Ca devient quasiment inné Du coup, on pense à réduire sérieusement la taille de ses maps

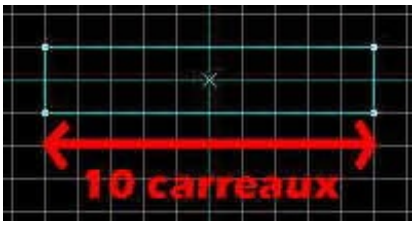
Donc en clair, je rédige cette annexe uniquement pour rassurer les débutants. Les habitués de Worldcraft, eux, devraient quand même apprendre pas mal de choses, comme les distances officielles de pénétration des balles dans les murs.

Calculs de longueurs

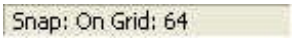
Avant toute chose, il va d'abord falloir apprendre à calculer les longueurs sous Worldcraft, car ce n'est pas aussi simple qu'il n'y paraît !

En fait, vous allez devoir faire une multiplication entre le nombres de carreaux dans la vue 2D et la précision de la grille. Je m'explique avec un exemple :

Voici un bloc dans la vue "Top". J'aimerais savoir quelle est la longueur de ce bloc :



Ce bloc fait 10 carreaux de longueur, ce qui ne veut pas dire qu'il fasse 10 unités de longueur. Il faut aussi prendre en compte la précision de la grille, indiquée en bas de l'écran :



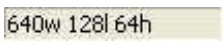
Ici, la précision est de 64 unités par carreau.

Pour calculer la longueur de se bloc, on multiplie le nombre de carreaux par la précision, ce qui nous fait :

10 x 64 = 640 unités !

Euh oui, je reconnais mon bloc était un peu gros, mais bon c'est pour l'exemple

Si vous augmentez la précision de la grille, faites attention ! Les carreaux deviendront de plus en plus petits, jusqu'à un stade où l'écran ne sera plus assez précis pour tout afficher !
Dans ce cas, faites très attention dans vos calculs, et servez-vous plutôt de l'indication au bas de l'écran :



Cela signifie que le bloc fait 640 unités de largeur, 128 unités de longueur et 64 unités de hauteur. Comme vous pouvez le voir, je ne me suis pas trompé dans mes calculs

Conversion pixels/mètres

Voici maintenant les longueurs officielles de conversion entre les mètres et les unités de Worldcraft (que nous venons d'étudier plus haut) :

Unités de Worldcraft	Unités réelles
1 unité	2,54 cm
4 unités	10,1 cm
8 unités	20,3 cm
16 unités	40,6 cm
32 unités	81,3 cm

64 unités	1,62 m
128 unités	3,25 m
256 unités	6,5 m

Ce tableau devrait aider les débutants un peu perdus

Quant aux autres, lisez la suite. Vous allez apprendre pas mal de choses...

Longueurs connues

Maintenant que vous vous êtes familiarisés avec les longueurs de Worldcraft, nous allons voir quelques exemples de valeurs officielles pour certaines longueurs (taille des murs, taille du joueur...)

Les murs

Un mur normal fait dans la réalité environ 20 cm, soit 8 unités.
Bien entendu, il existe des murs bien plus épais : vous pouvez par exemple doubler la taille d'un mur normal. Cela nous fait un mur de 40 cm, soit 16 unités.

Par contre, évitez de faire des murs plus petits que 8 unités, parce qu'après on a l'impression de jouer dans une map où les murs sont en carton

Les portes

Une porte moyenne fait 64 unités de largeur et 128 unités de hauteur. Avec une telle porte, vous êtes sûrs que le joueur passera sans avoir à baisser la tête ou à serrer les coudes

Le joueur

Taille debout : 72 unités = 180 cm
Taille accroupi : 38 unités = 69,5 cm
Largeur : 34 unités = 86,3 cm

Ainsi, notre ami Gordon Freeman (toujours porté disparu) faisait 1m80 ! Une taille plutôt grande, mais tout à fait potable

Si par exemple vous faites un conduit d'aération dans lequel le joueur est obligé de s'accroupir, je vous conseille de prévoir un peu plus de place que la taille exacte du joueur.
Plus le passage est étroit, plus on a du mal à progresser. Si vous laissez 24 unités d'espace libre autour du joueur, c'est parfait !

Les sauts

Si vous souhaitez faire des plates-formes éloignées mais quand même accessibles, voici les hauteurs et longueurs maximales qu'un joueur peut sauter :

Hauteur de saut normale : 44 unités
Hauteur de saut aggripé (accroupi en l'air) : 60 unités

Longueur de saut normale : 236 unités
Longueur de saut avec le LongJump de Half-Life : 400 unités

Voilà, ainsi si vous voulez mettre la dextérité du joueur à rude épreuve, vous n'avez qu'à vous servir de ces données
Je vous rappelle que ce sont les valeurs maximales, par conséquent évitez de trop forcer la dose ou votre map risque d'être plutôt hard

Pénétration des balles dans les murs (CS)

Enfin, pour terminer cette annexe, voici quelques valeurs qui risquent d'intéresser les mappeurs pour Counter-Strike...
Elles concernent la profondeur que peuvent traverser les balles dans les murs. Eh oui, on peut tuer son ennemi à travers un mur (lol désolé si je vous prends pour des newbies, mais ce site s'appelle le "Site du Zér0", alors je suis bien obligé de faire comme si)

Type de munitions	Distance traversée
.45 ACP	6 unités
9 mm	8 unités
50 AE	12 unités
762 mm	14 unités
556 mm	12 unités
338 Mag	16 unités

Ces valeurs sont les valeurs officielles données par Gooseman, un des développeurs de CS... Il n'y a donc aucun doute donc sur la fiabilité de ces informations.

Bien entendu, ces valeurs dépendent des munitions et non pas de l'arme (ce n'est pas l'arme que vous voulez lancer à travers le mur quand même ?)

Les plus perspicaces pourront noter donc que les munitions des Glock, USP et autres SMG (.45 ACP et 9 mm) peuvent traverser les murs... Je n'ai pas testé, mais cela me semble possible étant donné qu'on ne voit jamais des murs de 6 unités dans une map (trop fin !)