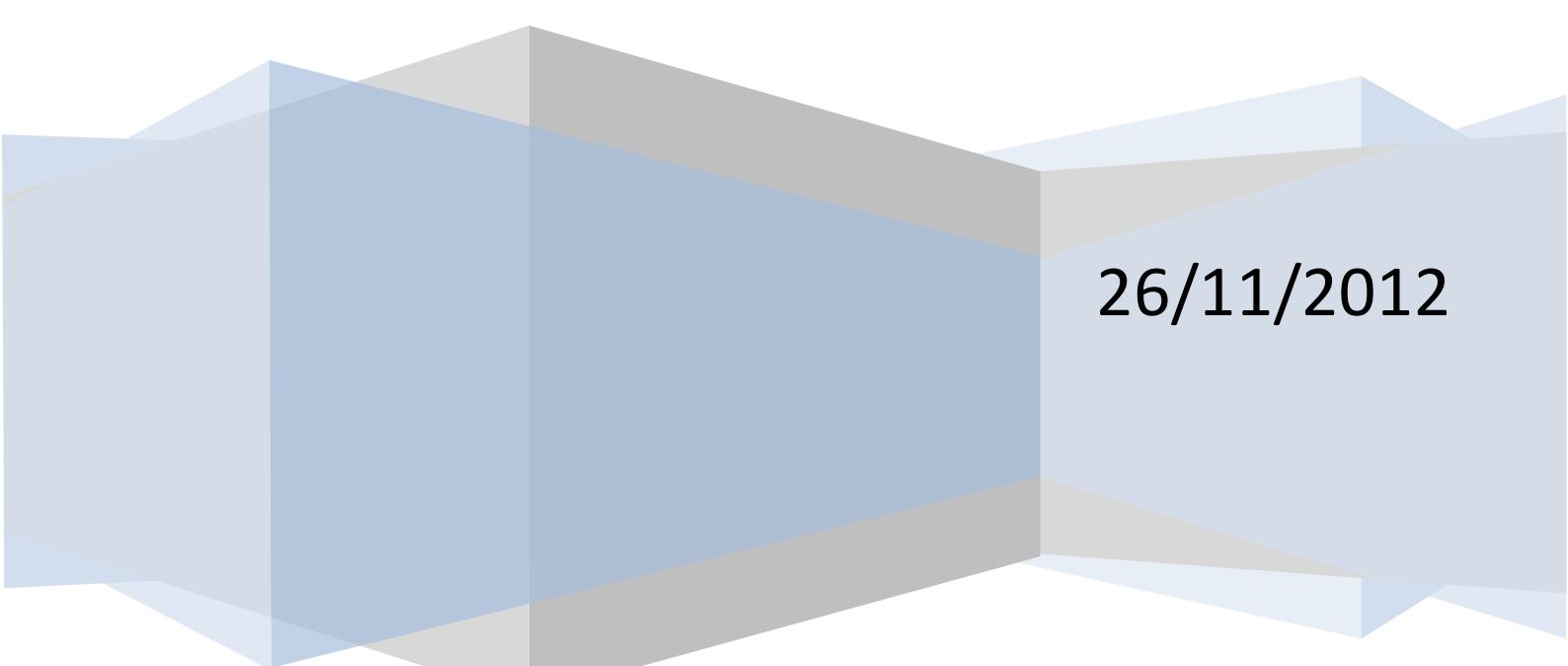


# NeoAxis engine

Labels

MONIER Vincent



26/11/2012

**Sommaire**

NeoAxis: Créer des labels.....	2
Définition du label.....	2
Exemples .....	2
Création de labels dans NeoAxis.....	3
Billboard et texture.....	3
HUD dynamique.....	19
Ingame GUI .....	20
Croisement des techniques.....	27

## NeoAxis: Créer des labels

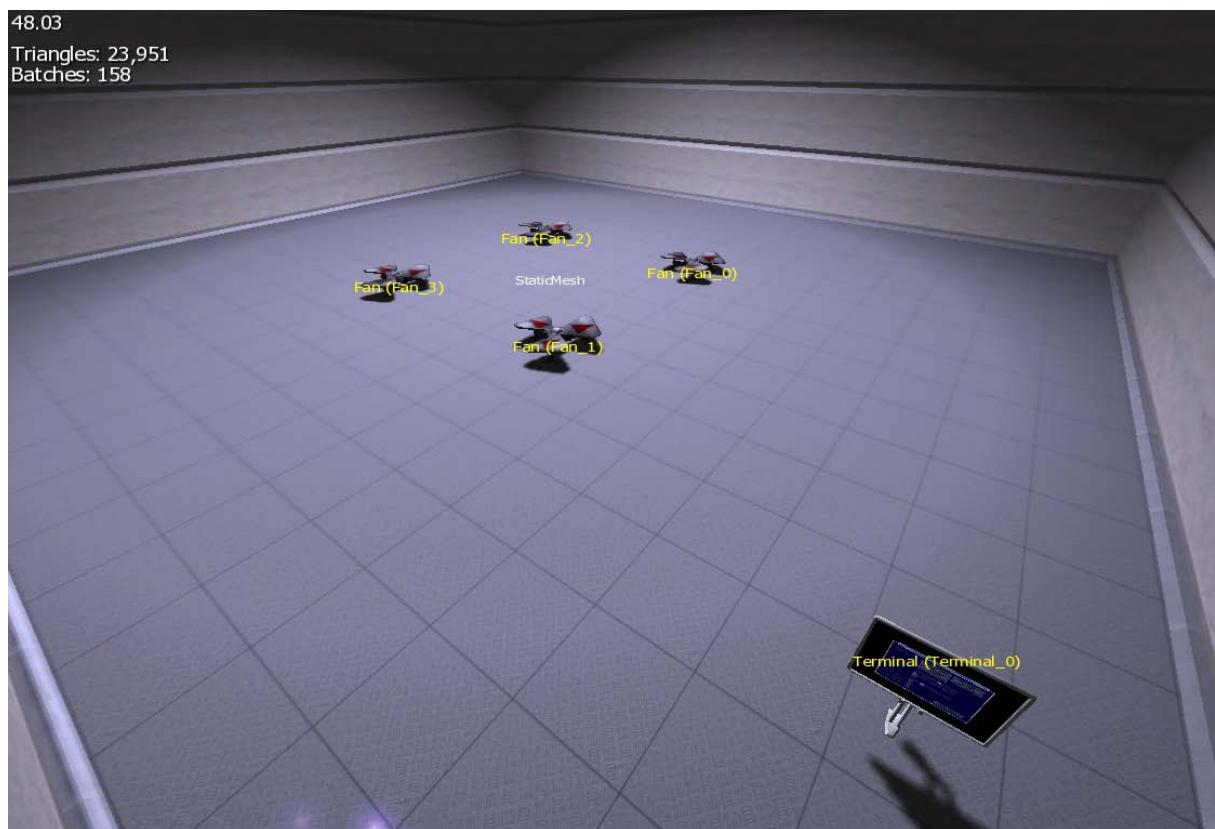
### Définition du label

Les labels sont des textes 2D affichés dans un environnement 3D donné. Ils sont liés à un point de cet environnement, point auquel le texte du label réfère. Le label doit se positionner et se tourner automatiquement de façon à se rendre le plus visible possible sur l'écran. Il doit, entre autres, essayer de faire face à l'écran (le label peut être vu comme un plan 2D contenant un texte, ce plan devant donc être tourné en direction de la caméra) et éviter de se trouver masqué derrière un objet (si le label est un bâtiment, le texte de ce label aurait intérêt à être devant le bâtiment si il veut se rendre visible sur l'écran de l'utilisateur et ce quelque soit la position de l'utilisateur).

Un positionnement classique serait donc inapproprié, car un positionnement classique consiste à définir les coordonnées d'un objet (ici, le texte du label), l'angle de l'objet (rotation selon les axes X Y Z) et sa taille (aussi appelé « PRS », « Position Rotation Scale ») et tout ceci de façon statique. Un label étant dynamique, son positionnement PRS dans l'espace est également dynamique.

### Exemples

Il n'existe pas d'exemple de label dans la démo fournie avec NeoAxis, mais les logiciels d'édition en fournissent de bons exemples (les labels sont principalement utilisés par les développeurs et les créateurs pour faciliter la gestion d'un environnement 3D ; ils sont rarement utilisés dans les jeux ou les produits finaux) :



*Dans l'éditeur de carte, les labels servent à afficher le nom des objets auxquels ils sont rattachés.*

## Création de labels dans NeoAxis

Plusieurs méthodes peuvent être envisagées. Elles sont décrites ci-dessous.

### Notes :

- « Type » désigne un typage d'entité (fichier \*.type), qui peut s'assimiler à une classe d'objets que l'on peut instancier dans le moteur 3D
- « HighMaterial » est le nom désignant les fichiers « Shader », qui définissent les textures. Un mesh peut utiliser ces shaders pour être colorisé. Le shader peut aussi être utilisé par les GUI (HUDs), ou les particides (effets spéciaux 2D) ou par les billboard (qui peuvent se ranger dans la catégorie « particides »).

### Billboard et texture

Il est possible d'utiliser le type d'objet « Billboard » pour créer un label. En ce cas, le label est une image, qui fera toujours face à la caméra. Si l'image contient un texte, et seulement un texte, alors on aura l'impression qu'il s'agit d'un label classique.

#### Avantages:

- Facilité de mise en œuvre
- Possibilité d'ajouter des effets sur la texture du billboard
- Possibilité d'incruster les ombrages sur le billboard

#### Inconvénients:

- Impossible d'éditer le texte du billboard (car il s'agit en fait d'une texture)
- Obligation de modifier la taille du billboard en fonction de la distance à la caméra

#### Méthode :

D'abord, il faut créer l'image. Deux méthodes peuvent s'envisager :

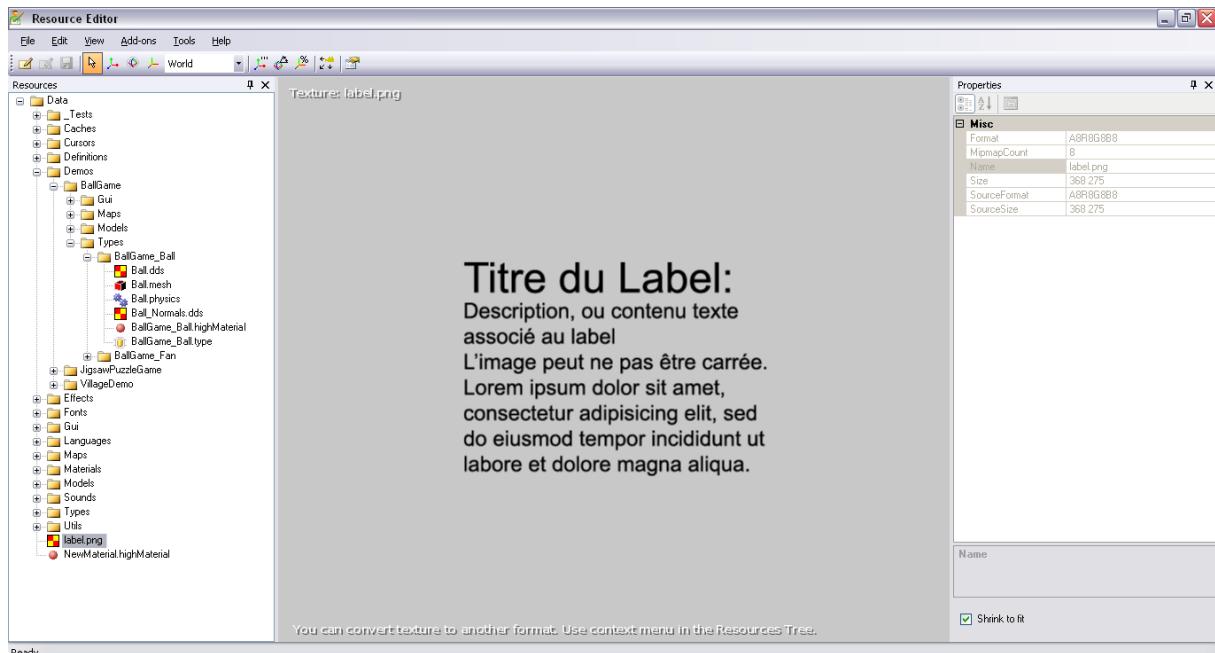
- Créer manuellement l'image via un logiciel comme photoshop
- Créer un logiciel qui génère automatiquement l'image à partir d'une police et d'un texte

La première méthode est la plus aisée pour le prototypage, puisque tout le monde peut utiliser paint (ou autre logiciel d'édition d'image) pour créer une image avec un fond transparent, et y écrire le texte qu'on veut.

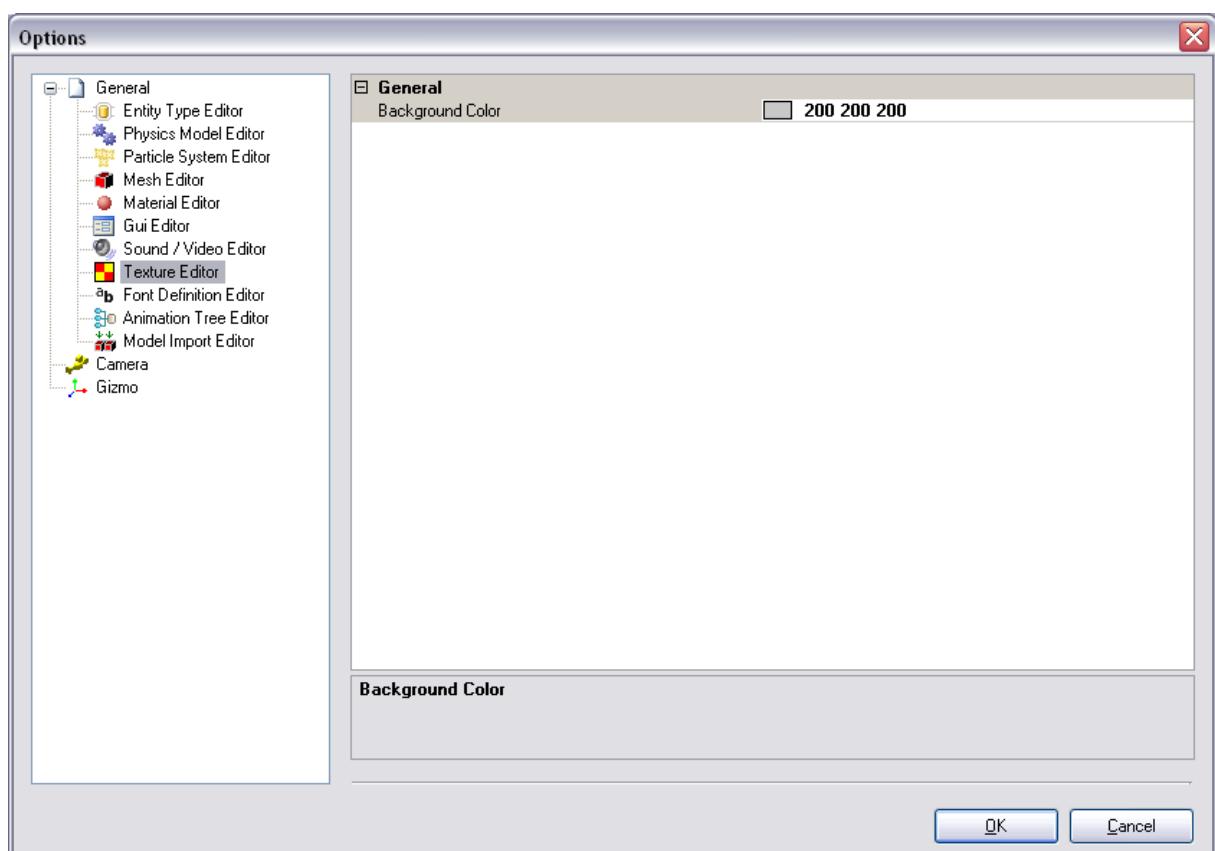
La seconde présente l'avantage d'être automatisée, et donc rapide, et serait à utiliser pour un déploiement et non un prototypage.

L'image du billboard doit, dans tous les cas, respecter les conditions suivantes :

- Un fond transparent (on peut utiliser un fond opaque, mais ce fond sera alors affiché derrière le texte sur le label)
- 32 bits par pixel (RGBA)
- Format PNG (autres formats acceptés : TGA, JPG, BMP, DDS)
- Taille maximale de 4096x4096 pixels (suivant l'implémentation, la taille de l'image peut monter à ~6000x6000, mais elle sera rétrécie à 4096x4096 dans tous les cas ; attention, au-delà de 6000x6000, le moteur peut crasher !)

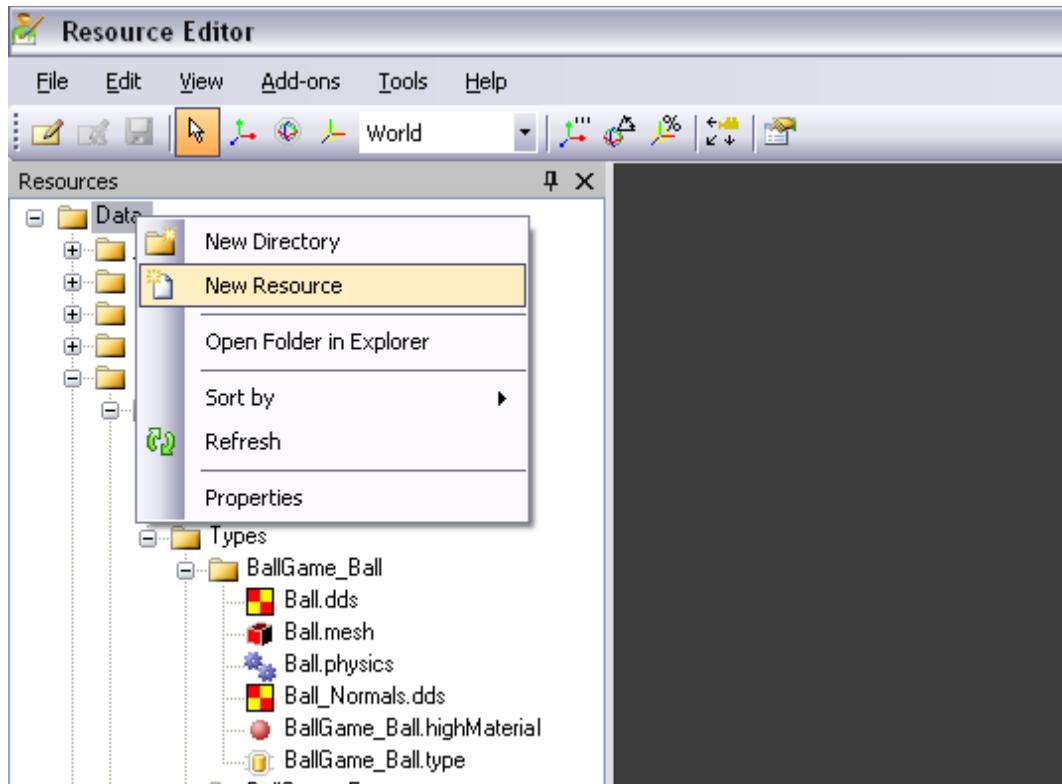


*Exemple d'image pouvant être utilisée comme label complet d'un bâtiment.*

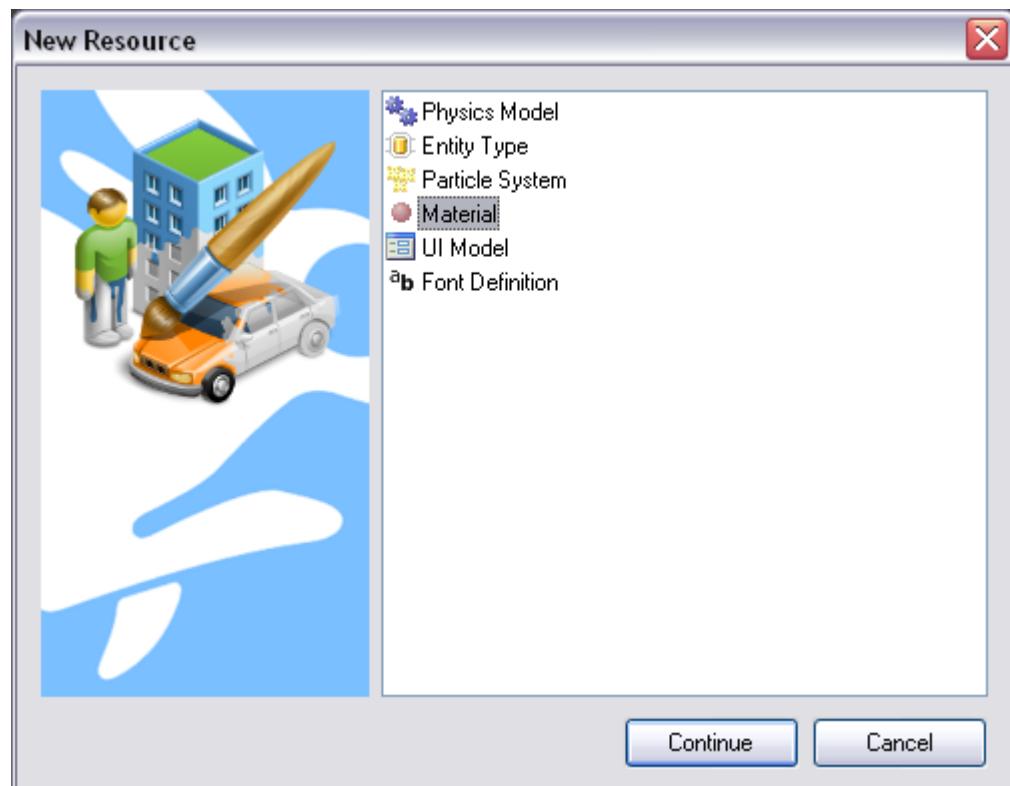


*Vous pouvez modifier la couleur de fond de l'éditeur NeoAxis via « Tools » / « Options » / « Texture Editor » / « Background Color »*

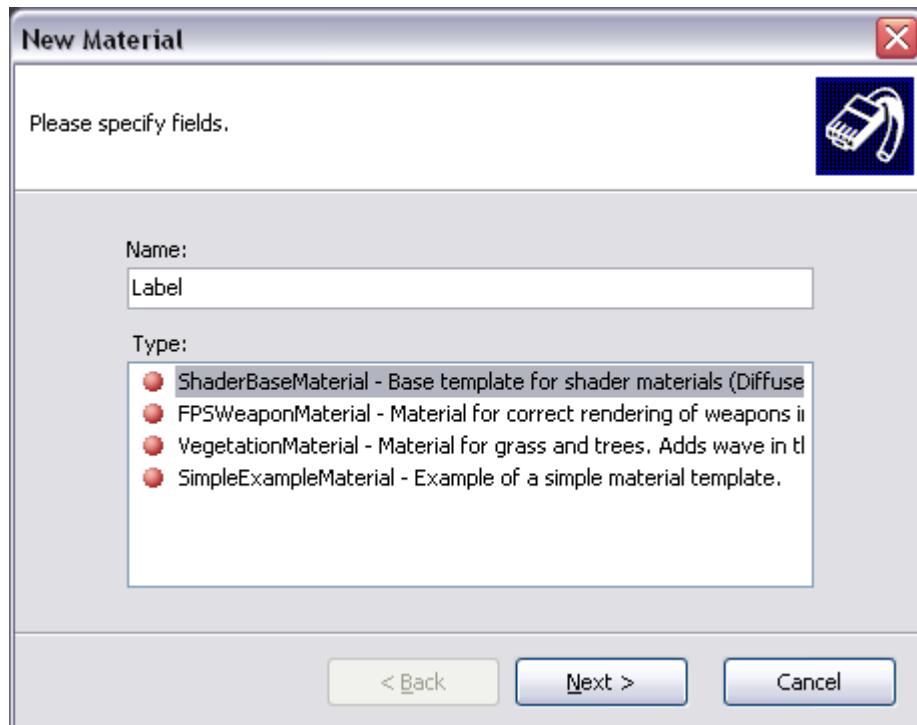
L'image doit ensuite être utilisée comme « diffusemap » dans un shader HighMaterial. Créez un nouveau HighMaterial et ouvrez-le en édition :



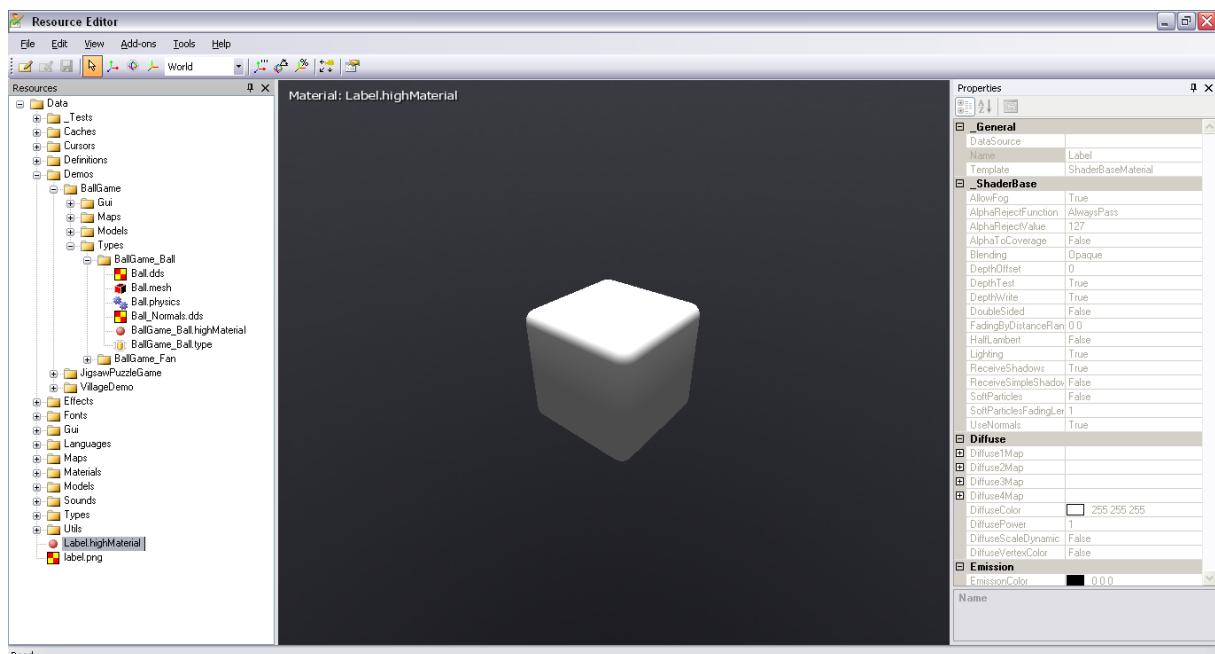
Faites un clic droit sur le dossier où vous souhaitez créer le « HighMaterial », puis sélectionnez « New Resource »



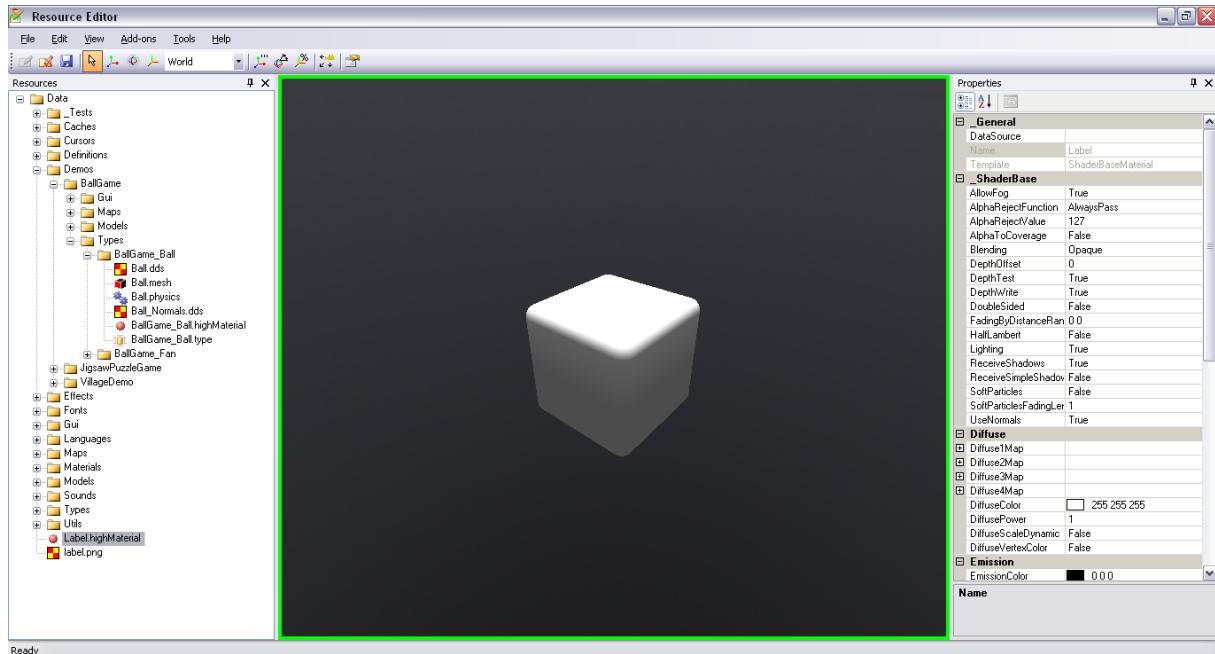
Choisissez « Material »



*Entrez un nom et sélectionnez « ShaderBaseMaterial »*

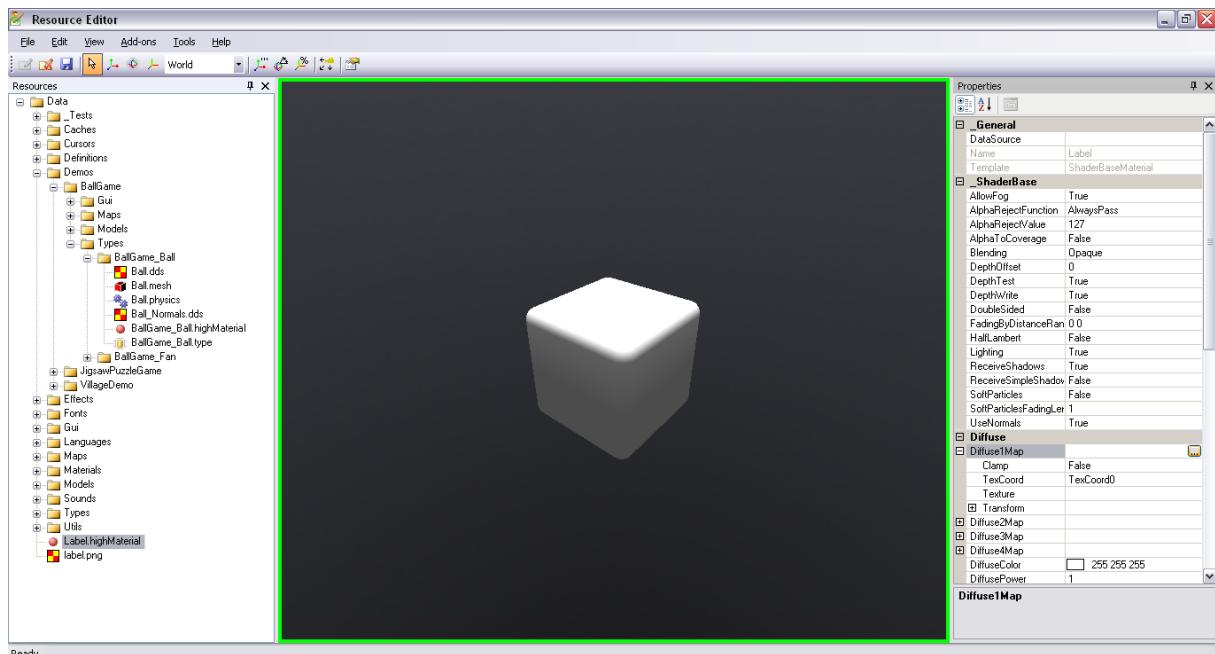


*Le HighMaterial est alors créé*



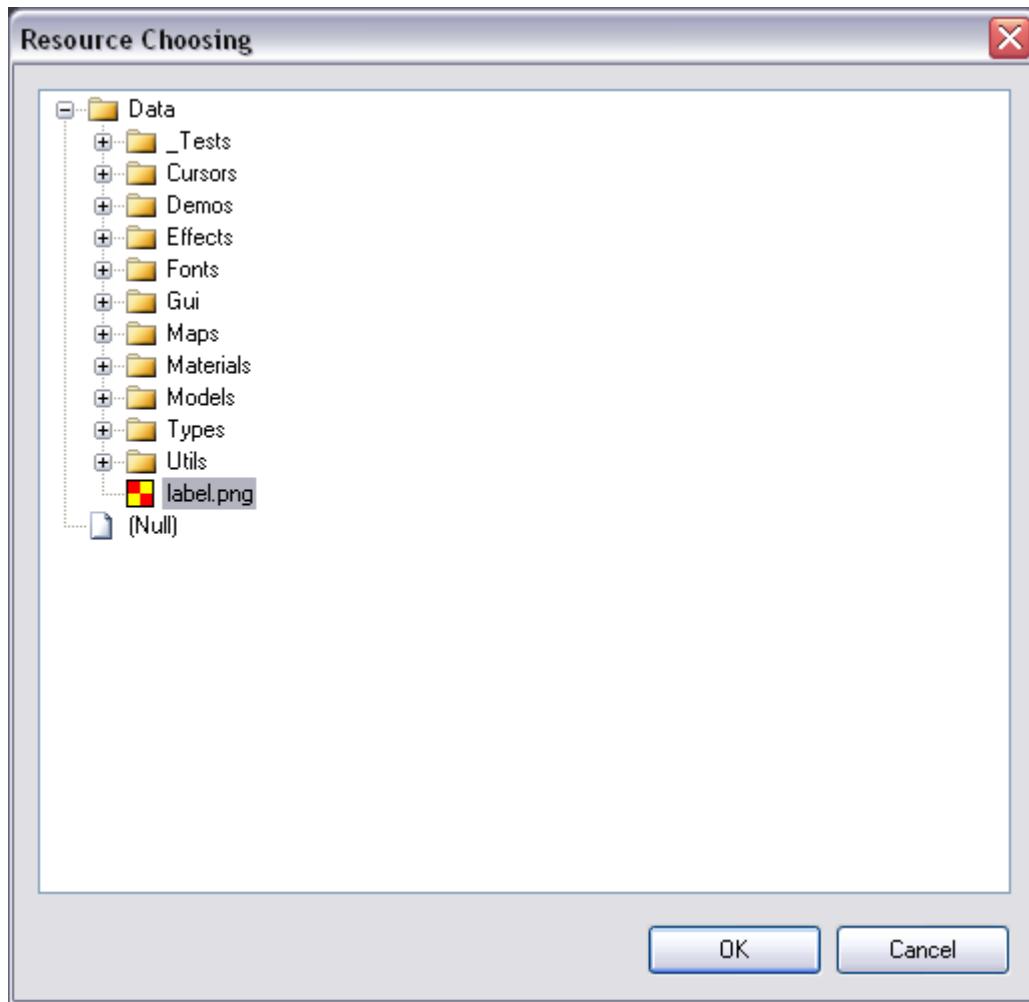
Double-cliquez dessus (sur « Label.highmaterial » dans la zone de gauche) pour l'ouvrir en édition

Une fois le HighMaterial ouvert en édition, vous pouvez le manipuler comme n'importe quel HighMaterial. Pour ajouter la texture (« DiffuseMap ») précédemment créée, cliquez sur « Diffuse1Map » (zone de droite, « Properties ») puis sur le bouton avec les points de suspensions (à droite de la valeur, actuellement vide, du champ « Diffuse1Map ») :

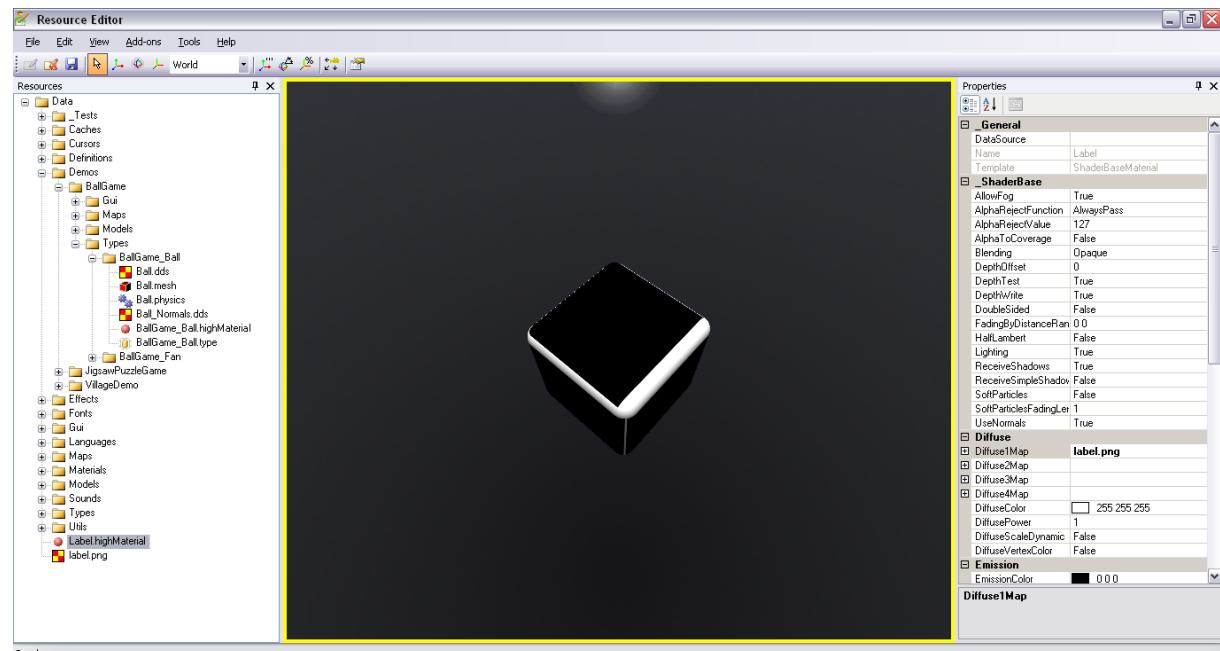


Cliquez sur « Diffuse1Map » puis cliquez sur le bouton « ... » pour choisir la texture à appliquer

Choisissez la texture précédemment réalisée :

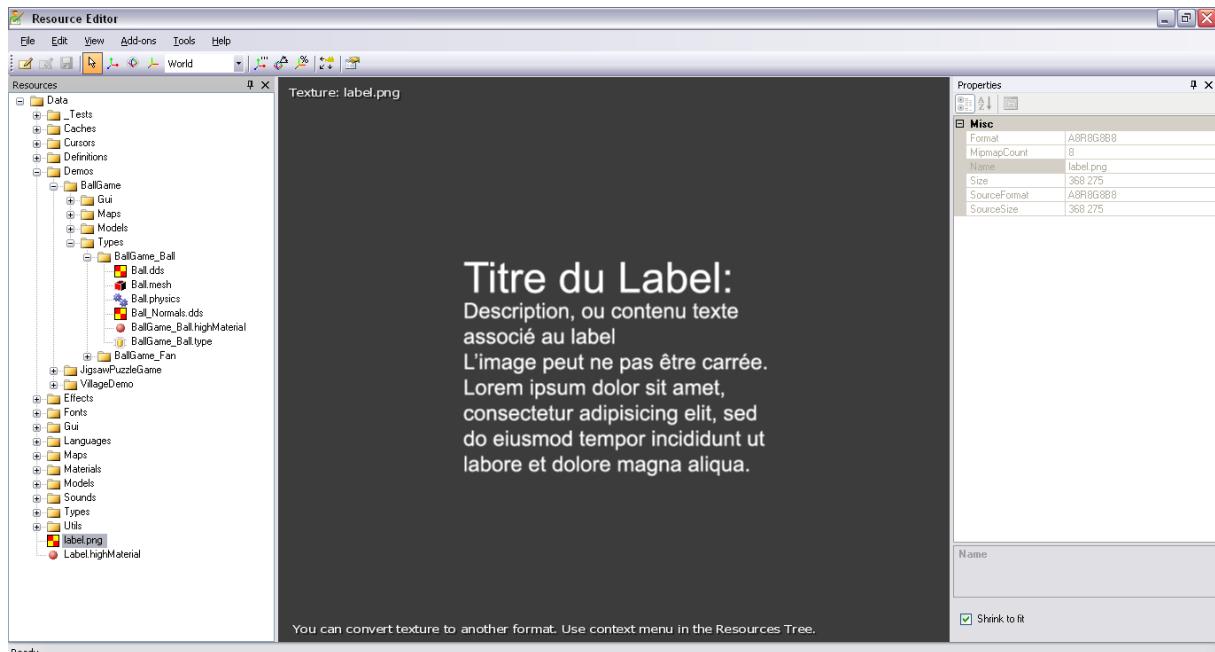


Choisissez la texture à appliquer (« Label.png » dans cet exemple) et cliquez sur « Ok ».

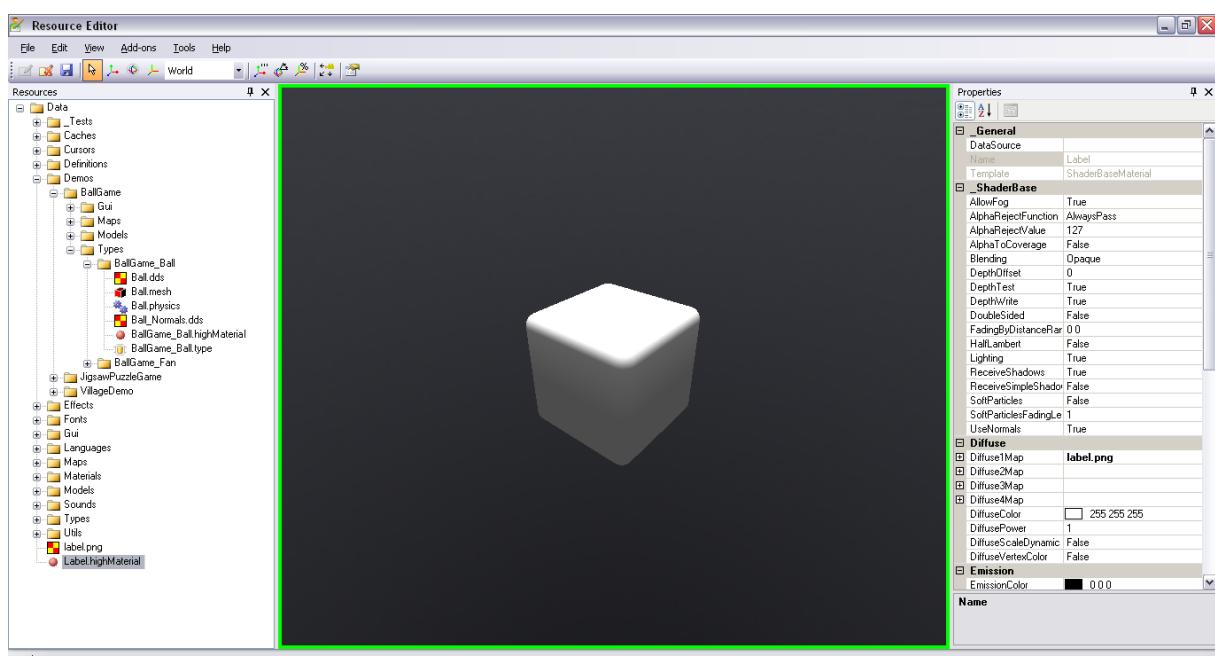


La texture est appliquée... Le résultat est assez inattendu !

Si vous avez utilisé un fond transparent, le résultat peut ne pas être celui que vous espérez. Pourquoi ? Parce que les pixels transparents sont rendus opaques par NeoAxis . Or, tout pixel est de la forme « RGBA », et possède donc une couleur et une transparence. Ainsi, même si le pixel est totalement transparent, il possède une couleur (qui n'est généralement pas utile, car le pixel est censé être transparent). Cette couleur, définie par le logiciel d'édition d'image, n'est généralement pas déterminée ni connue par l'utilisateur.



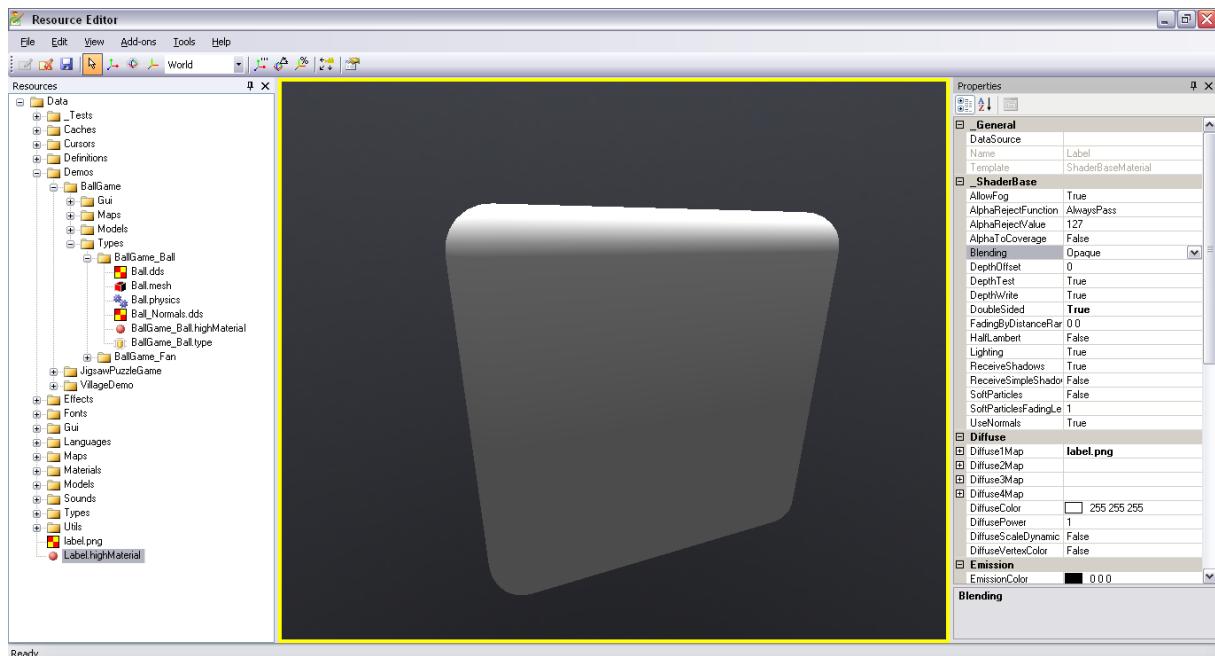
*Si vous avez écrit le texte en blanc sur un fond transparent...*



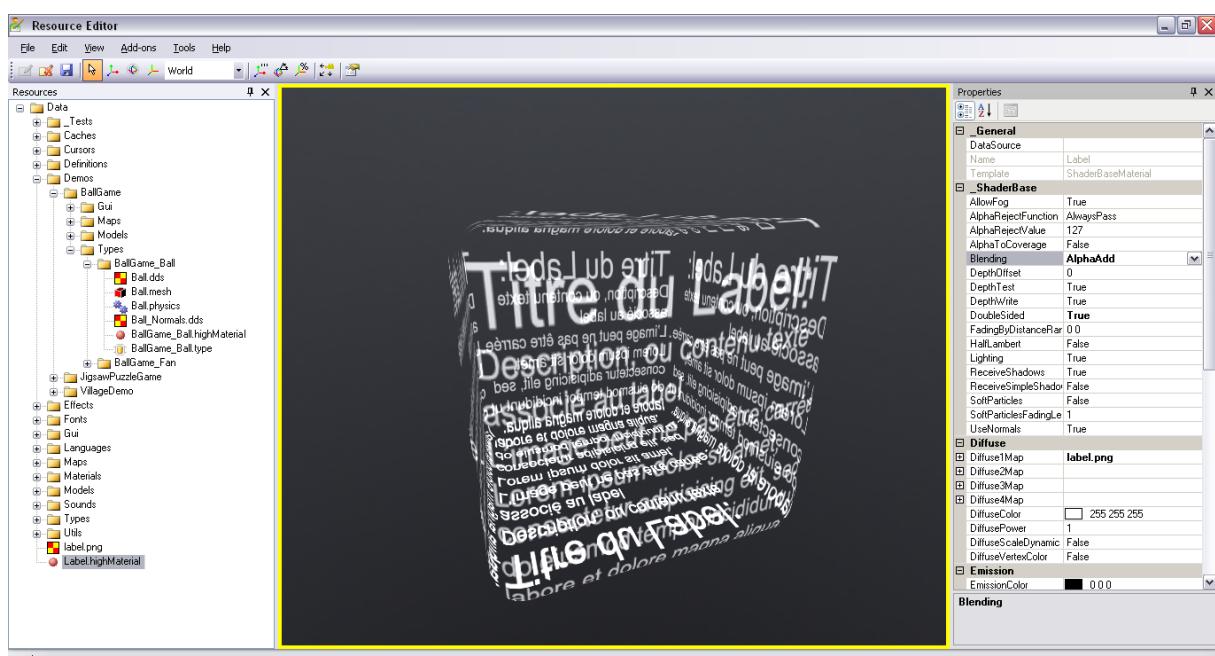
*Alors le cube sera blanc car le logiciel d'édition (ici Photoshop) a défini la couleur des pixels transparents sur « blanc » .*

Pour gérer correctement la transparence, changer l'option « BlendFunc ». Dans le rendu 3D, le moteur calcule l'image de ce qui se trouve derrière l'objet à rendre. Cette image est appelée

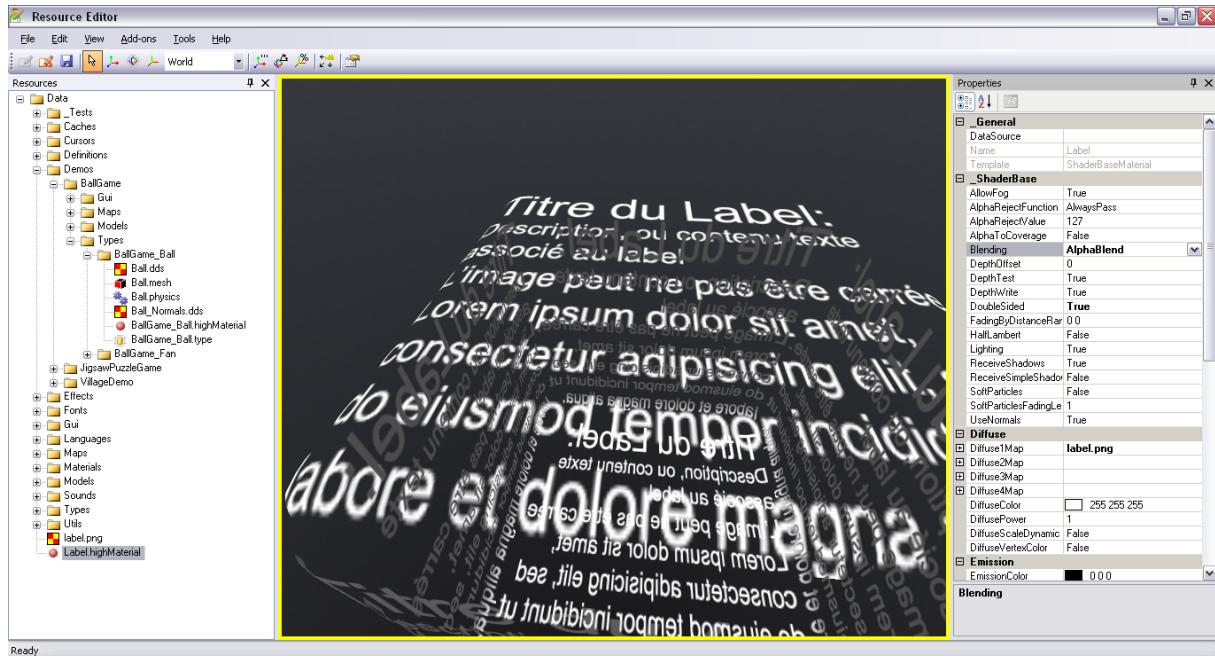
« buffer », et n'est jamais affichée à l'écran (c'est une image temporaire). L'objet est alors rendu par-dessus cette image (car il est « devant » dans la 3D). Si « AlphaAdd » est sélectionné, la couleur des pixels de l'objet est ajoutée à la couleur de l'image du buffer. Cette méthode est utile si l'objet laisse passer la lumière ET qu'il en émet en plus (exemple : halos de lumière autour des sources lumineuses). Si « Opaque » est sélectionné, les pixels de l'objet écrasent les pixels du buffer (c'est le cas de TOUS les objets totalement opaques, c'est-à-dire que ne laissent pas du tout passer la lumière, qu'ils en émettent eux-même ou non). Enfin, « AlphaBlend » est utilisé pour des objets qui possèdent des zones transparentes, et qui laissent donc passer la lumière, sans forcément en émettre eux-même. C'est notre cas ici.



*En mode « Opaque », la transparence est ignorée, et le cube est donc intégralement blanc.*

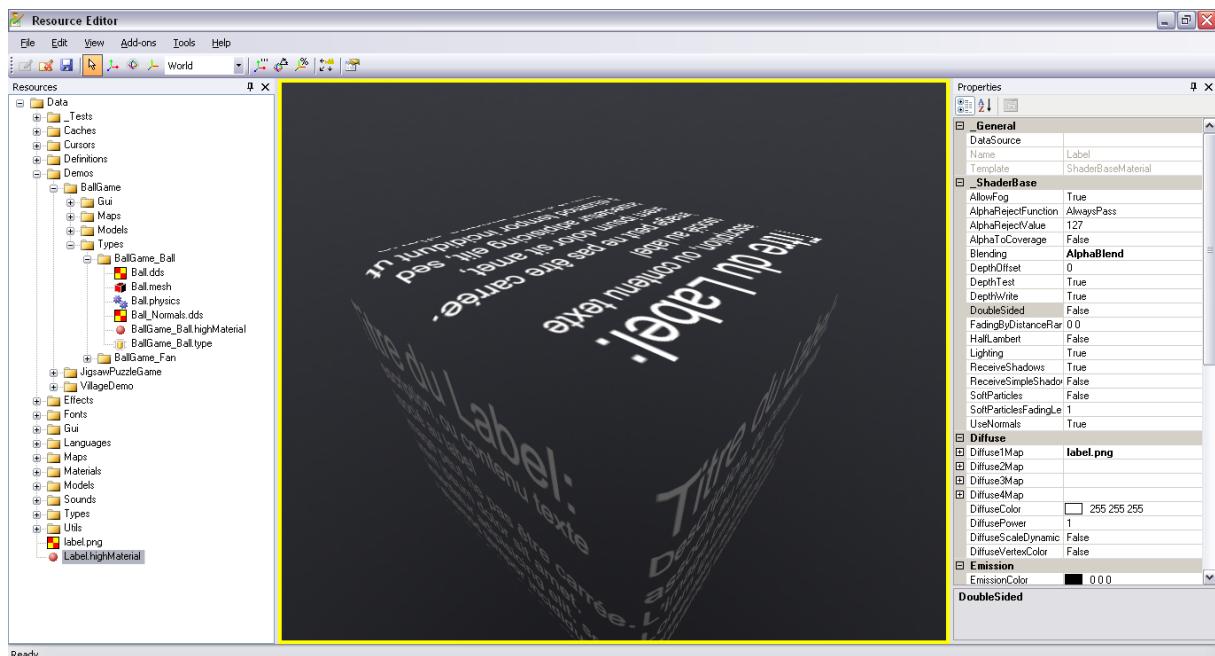


*En mode « AlphaAdd », on s'aperçoit que certaines zones sont transparentes, d'autres sont en blanc simple, et d'autre enfin, un blanc double car le texte est rendu « 2 fois ».*



En mode « Alpha Blend », le rendu est celui attendu : un objet classique avec transparence

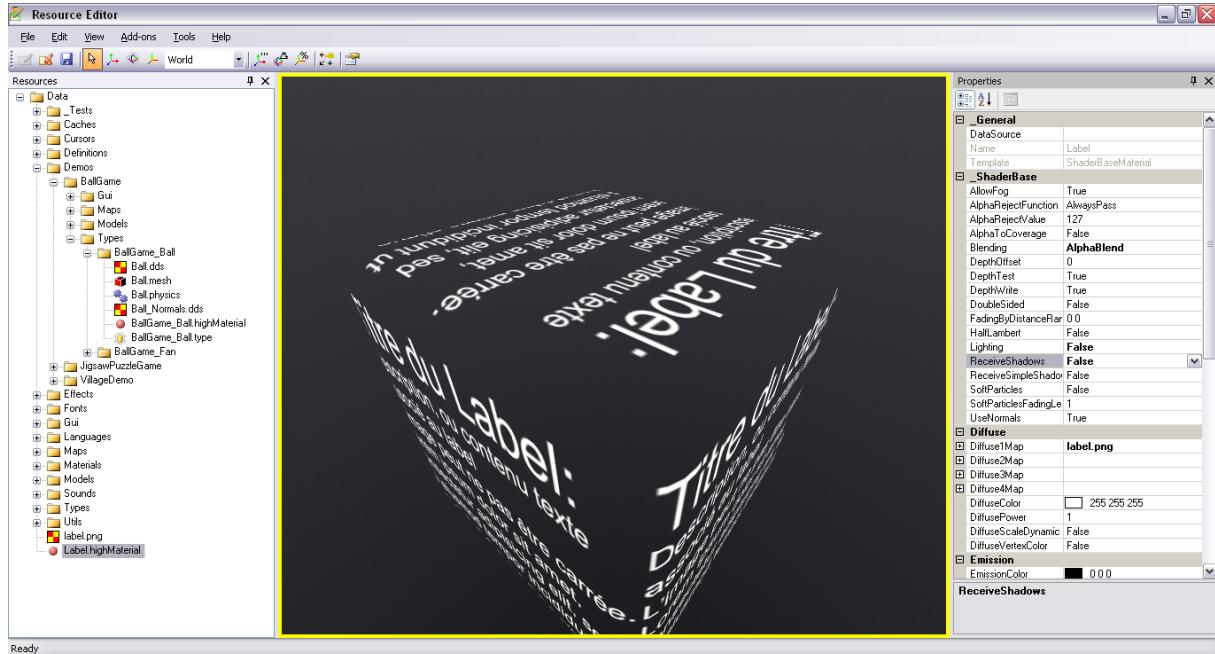
Sur la screen précédente, vous aurez peut-être remarqué qu'à certains endroits, le texte gris des bords du cube est rendu « par-dessus » le texte blanc, alors que (normalement), le texte gris est « derrière » le texte blanc. Cela vient de la désactivation du « DepthWrite » dans le cas des objets non-opaques. Ce défaut apparaît si l'option « Double Sided » est active. Cette option indique que l'objet sera visible de face et de dos. Dans notre cas, le Label est toujours tourné vers la caméra, il est donc toujours vu de face. Vous pouvez donc mettre l'option « Double sided » sur « False ».



Double-sided étant à « false », les pixels de la texture ne seront rendu que s'ils sont vus « de face » : les cotés cachés du cube ne sont plus visible et le problème précédent (depthwrite) est résolu

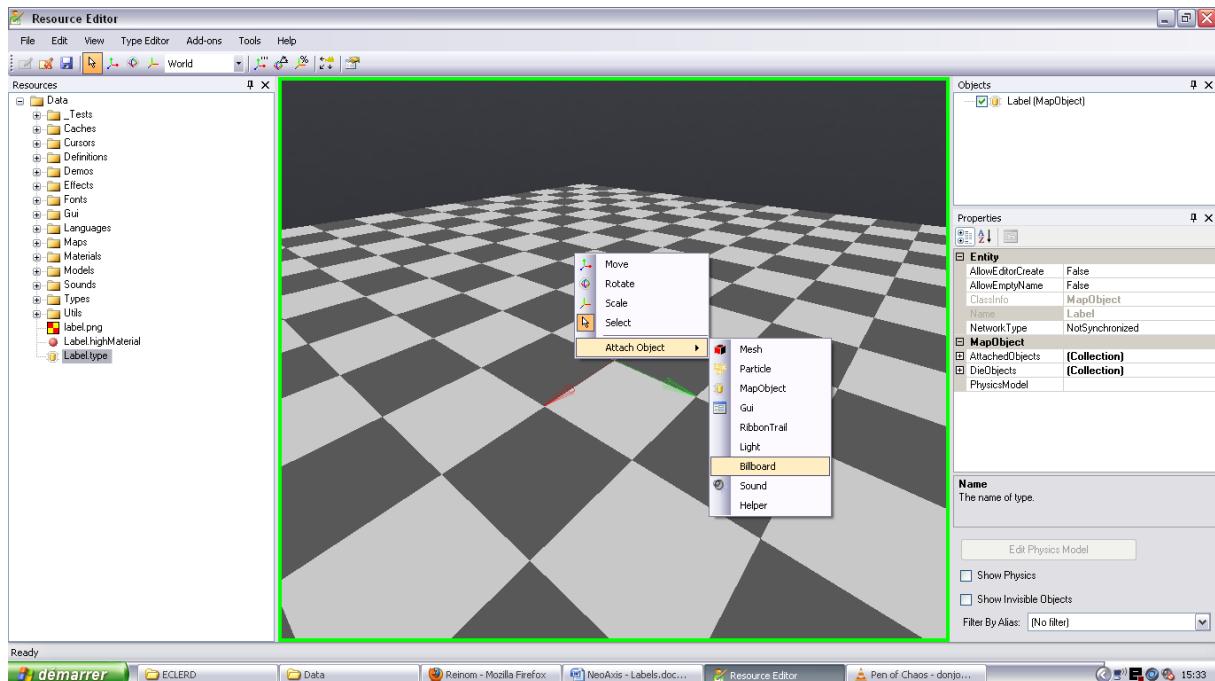
Vous pouvez éditer d'autres options du highmaterial pour avoir le rendu que vous souhaitez. Il est quand même conseillé de mettre l'option « Lighting » sur « False » et « ReceiveShadow » sur

« False ». Ainsi, le label sera indépendant du lighting (même dans un coin sombre à l'ombre d'un bâtiment, le label sera en blanc pur) et il ne générera pas d'ombrage, ce qui serait inutile.



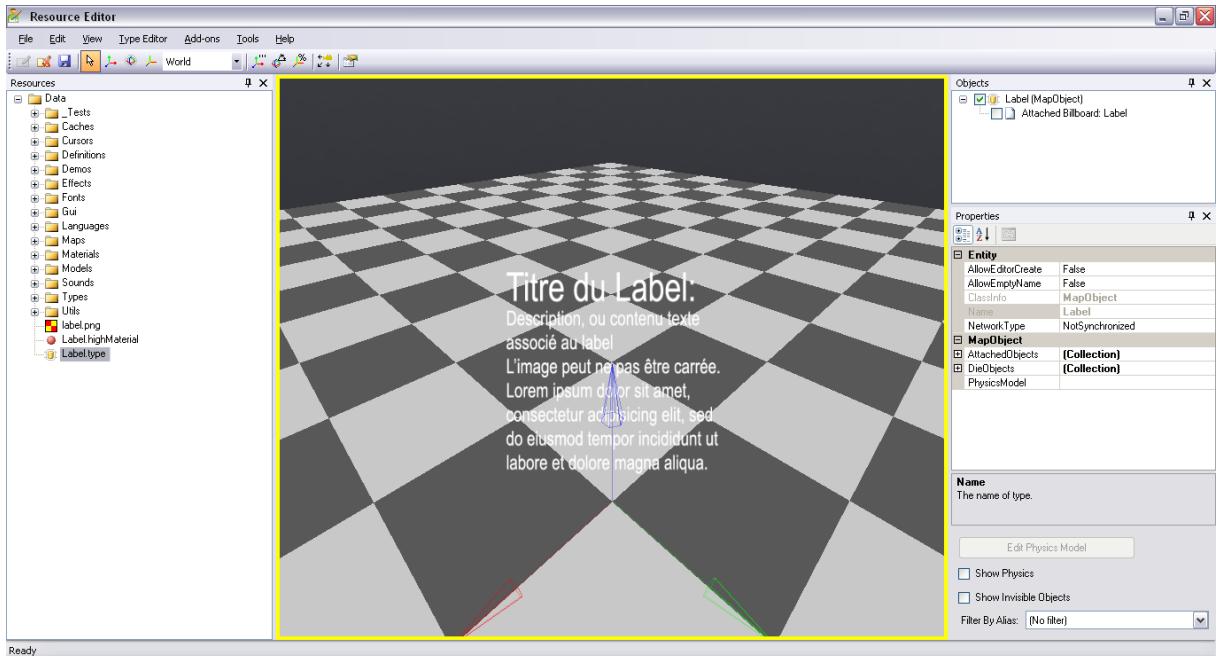
En règle générale, le label aura les options « Lighting », « Double Sided » et « Receive Shadow » à « false », et son « Blending » sera sur « Alpha Blend »

Maintenant, il est possible d'ajouter un bill board à une entité (ouvrez l'entité en édition par un double-clic sur le fichier « \*.type », puis clic droit dans une zone vide, « Attach Object », « Billboard » et définissez l'option « Material » de ce billboard sur le material que vous avez créé, le \*.HighMaterial) :



Ajoutez le « Billboard » dans un « \*.type » (que vous avez peut-être vous-même créé avant via « New Resource », « Entity Type »), vous pouvez aussi l'ajouter par un clic droit sur « Label »

*(MapObject) » dans la zone « Objects », ou en cliquant sur l'option « AttachedObjects » dans la zone « Properties ».*

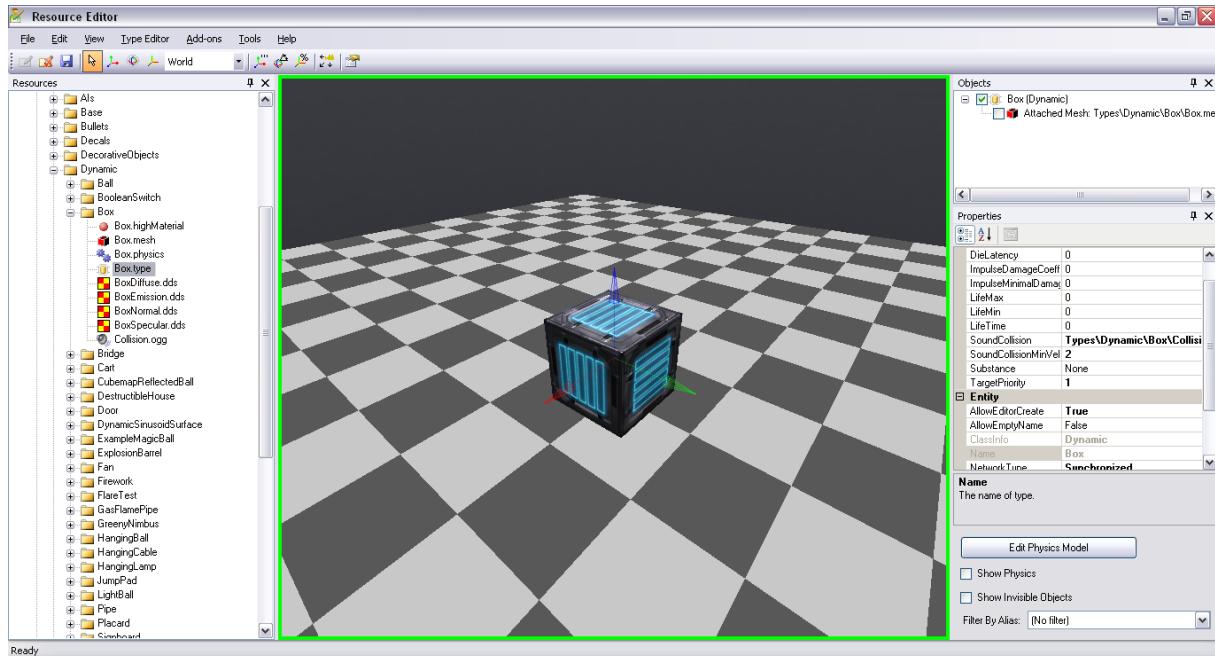


*Cliquez sur l'option « Material » dans la zone « Properties » (après avoir sélectionné le Billboard dans « Objects ») et sélectionnez le HighMaterial que vous avez créé pour qu'il s'applique à ce billboard*

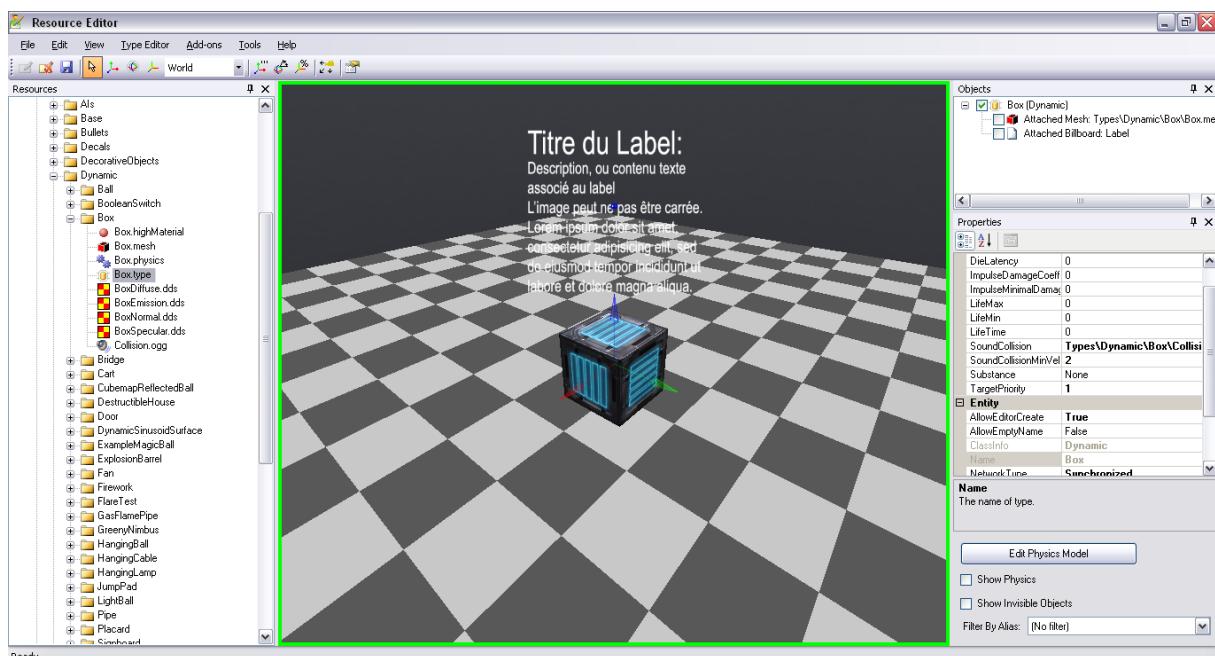
Vous pouvez bien sûr déplacer le billboard. Il est inutile de le tourner en revanche, car il fera toujours face à la camera. Vous pouvez éditer sa taille. Il est conseillé de ne pas déplacer le billboard pour que le centre de celui-ci soit confondu avec le centre de l'entité (objet « \*.type »), sachant que l'entité est directement manipulable dans le moteur 3D. Donc, si vous déplacez le billboard dans l'entité, il faudra enregistrer la valeur de ce décalage pour positionner le billboard dans le monde 3D.

Il est également conseillé de créer un nouvel objet \*.type pour le billboard, si vous souhaitez le manipuler de façon indépendante dans le moteur de jeu. Toutefois, si le billboard (ou label) doit être rattaché à un bâtiment, il est parfaitement possible d'ajouter ce billboard au fichier \*.type de ce bâtiment, et ainsi gérer l'ensemble des deux d'un seul coup. Il est normalement possible de déplacer un objet d'une entité de façon indépendante des autres objets (ainsi, il sera possible de déplacer le billboard par rapport au modèle 3D dans le fichier \*.type).

Il est aussi possible d'ajouter plusieurs billboard à un même objet (par exemple, des billboards plus ou moins détaillés) et, une fois dans le moteur 3D, d'afficher et de masquer certains billboard suivant la distance à l'objet.



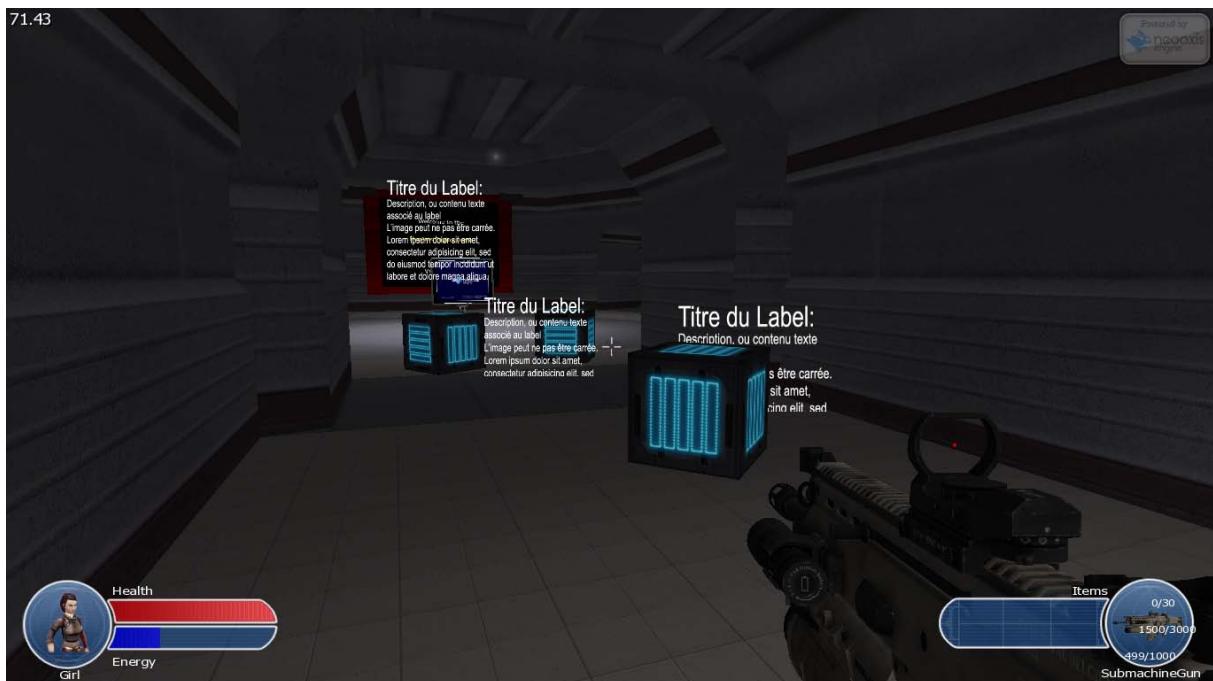
Exemple avec un type d'origine, fourni avec le SDK : un cube métallique bleu



On ajoute le billboard



*Le billboard est maintenant associé à tout objet utilisant ce type d'entité*



*Si l'objet bouge (ici, le cube est dynamique), le billboard sera positionné en (0 0 2) dans le repère local du cube (qui n'est pas forcément le repère global : ici, l'axe Z de deux des cubes est à l'horizontal et les billboards ne sont plus affichés « au-dessus » des cubes).*

*Vous pouvez faire des jeux d'essais en manipulant certains paramètres des highmaterial, pour visualiser leur effet. Ci-dessous, quelques paramètres utiles :*



Résultat pour « Lighting » à true et « UseNormals » à « false » (« UseNormal » indique au jeu d'utiliser la normale en chaque pixel d'un objet pour calculer le rendu lumineux ; comme un billboard n'a pas de normale puisqu'il fait toujours face à l'utilisateur, « Use Normals » doit être à « false »).



Avec un « Lighting » à false, « Receive Shadow n'est pas pris en charge, car le texte, même dans l'ombre de ce panneau, est toujours en blanc.

L'option « FadeByDistanceRange » permet de définir deux valeurs, A et B, de sorte que l'objet (le label) :

- soit complètement visible si la caméra est à une distance inférieur à A de l'objet
- soit complètement invisible si la caméra est à une distance supérieure à B
- entre les deux, l'objet disparait progressivement



*A une distance faible, le label est totalement visible*



*Il s'efface ensuite progressivement...*



... jusqu'à disparaître complètement. Ce système permet de n'afficher le label qu'entre deux valeurs de distances (par exemple, un label court est affiché pour des distances importantes, et s'efface à des distances proches, alors qu'un label plus complet ne doit s'afficher que si on est proche de l'objet).

Le système du billboard avec une texture peut donc être utilisé sans problème, mais il ne permet pas d'afficher des vidéos. Il est, en revanche, possible d'induire une image dans le billboard, puisqu'un billboard est, en pratique, une image.

Néanmoins :

- Le billboard est une image raster, et non vectorielle, ce qui peut ne pas être esthétique
- Le billboard a besoin que l'image soit faite « à l'avance » : on ne peut pas changer le texte à la volée, une fois dans le moteur 3D
- Le billboard peut être créé plus facilement avec un outil générant directement l'image à partir du texte et de la police (par exemple, un formulaire HTML peut générer, via PHP, une image avec un texte mis en forme via les balises HTML)

## HUD dynamique

Une autre solution consiste à créer un HUD sur l'écran dont le contenu sera l'ensemble des labels à afficher. Pour chaque label, à chaque frame, il faut initialiser le label :

- Créer un élément « text » dans le HUD des labels
- Définir le texte de cet élément, ainsi que sa couleur, police, taille etc

Ensuite, tant que le label existe, il faut le calculer :

- Calculer la projection du point de l'espace sur l'écran
- Positionner le texte du HUD en ce point
- Définir la taille du texte sur le HUD (cette taille est normalement fixe et non, cette étape peut être omise)

### Avantages :

- Possibilité de changer le contenu texte instantanément
- Possibilité de changer la police, la couleur, la taille du texte
- Possibilité de désactiver l'affichage de tous les labels d'un coup (en masquant tout le HUD)
- Le label est rendu en 2D après la scène (donc il n'est pas masqué par la scène 3D, cf l'exemple du label sur un bâtiment)

### Inconvénients :

- Calcul de la projection potentiellement difficile
- Calcul de projection à refaire à chaque frame (perte de performances)
- HUD 2D qui peut donner un effet étrange si l'écran est un écran 3D

### Méthode :

Pour le HUD dynamique, la méthode ne sera pas intégralement détaillée car elle est assez longue à implémenter.

Pour créer un HUD, référez-vous au document sur les HUDs. Le positionnement du texte sur le HUD est à la charge du développeur.

Un Label sera alors composé d'un point de l'espace (ou d'un objet du monde 3D) et d'informations, de différents types (textes, images, vidéos) qui seront, en pratique, inscrites dans un GUI dédié, qui sera affiché dans le HUD dynamique. Les commandes pour réaliser ce système ne sont pas directement données sur le site du moteur 3D, et il faudra disséquer le HUD « HUD example » fournit avec le SDK (ce HUD possède des zones « dynamiques », c'est-à-dire dont l'affichage varie en fonction d'un script défini pour l'environnement 3D).

### Note :

Cette méthode requiert des connaissances en C# puisqu'elle passe par la création d'un script. Toutefois, un développeur pourra se charger de ce code C# et faire en sorte que l'utilisateur n'ai plus qu'à lier ses données aux bâtiments auxquels il souhaite les rattacher (liaison pouvant se faire par un simple fichier texte par exemple).

## Ingame GUI

Une troisième solution consiste à utiliser un système similaire aux moniteurs contenant des GUIs (cf le document concernant les HUDs dans NeoAxis). Le label serait alors une surface plane, pouvant faire parti d'un objet, dans laquelle est rendu un GUI classique, contenant le texte du label.

### Avantages :

- Facilité de mise en place
- Possibilité de faire des labels complexes (avec plusieurs textes, un titre, des images ou des vidéos)
- Possibilité d'ajouter des boutons sur les labels (pour les masquer par exemple)
- Possibilité de créer un système de LOD (si on s'éloigne du label, seul le titre est visible, en gros, et si on se rapproche, le contenu des paragraphes s'affichent, si on se rapproche encore, les images sont affichées et les vidéos aussi)

### Inconvénients :

- Redimensionnement manuel de la taille du label en fonction de la distance (ceci n'étant pas « obligatoire », cf le LOD ci-dessus, cet inconvénient peut être écarté)
- Lourdeur pour le moteur 3D (autrement dit, un grand nombre de labels risque de ralentir le moteur 3D)

### Méthode :

La méthode est bien plus simple que le HUD dynamique, et peut être appliquée par n'importe qui, y compris un non-développeur, de façon similaire à un label par billboard-texture. La seule difficulté consistera à orienter le Label dans la direction de la caméra, cette orientation devant se faire via un script (ce script pourra être écrit une fois et utilisé ensuite pour chaque Label dans le jeu).

D'abord, il faut créer le ou les Labels, sous la forme de GUI (dans cet exemple, le GUI est créé à partir du template de base « Control »). Référez-vous au document sur les HUDs. On ne va considérer ici qu'un seul GUI pour cet exemple, mais un objet peut parfaitement être rattaché à plusieurs labels. Le GUI est donc créé manuellement (note : avec un script C# custom, il devrait être possible de créer le HUD « en temps réel » ; en effet, il est possible de créer une instance de GUI via un C#, et d'y ajouter, déplacer, et éditer des composants comme un texte, une image ou une vidéo) :



Le label est créé. Vous pouvez y insérer des textes, images ou vidéos.

A la base, un GUI est un menu. Donc, la souris sera affichée si l'utilisateur regarde en direction du GUI :



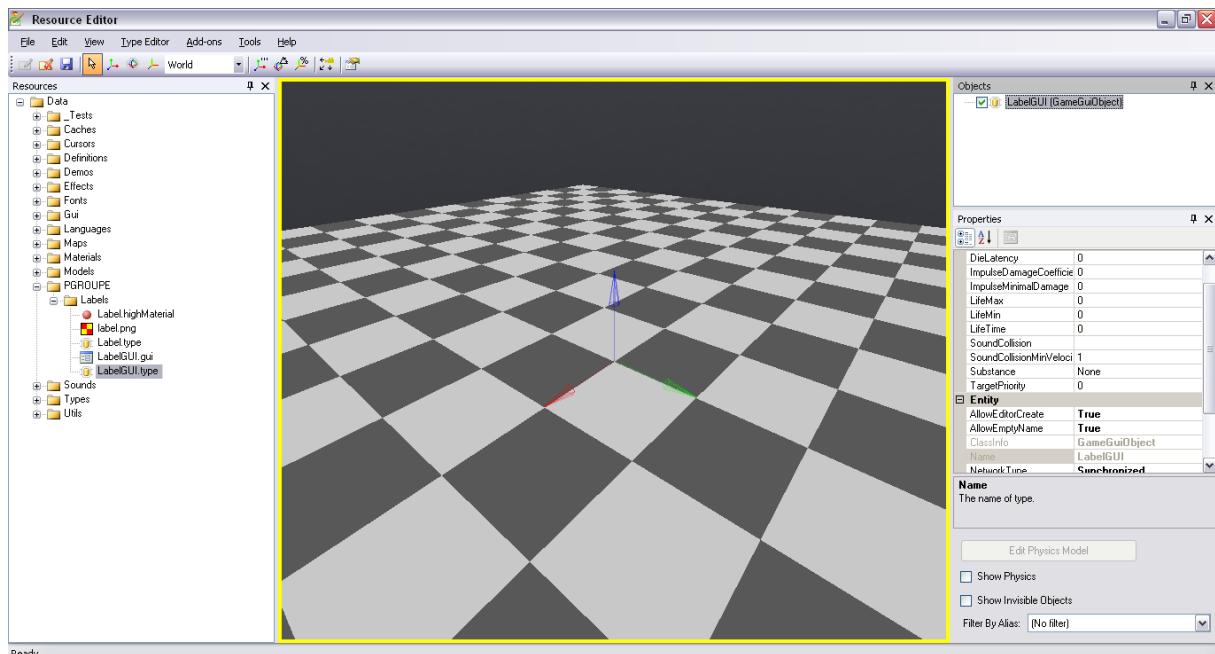
La souris (crosshair au centre de l'écran) est affichée lorsque l'utilisateur regarde le label.

Pour éviter ce comportement, et masquer la souris quand l'utilisateur regarde le label, il suffit de définir la valeur de la propriété « General/Enable » du GUI sur « False » :



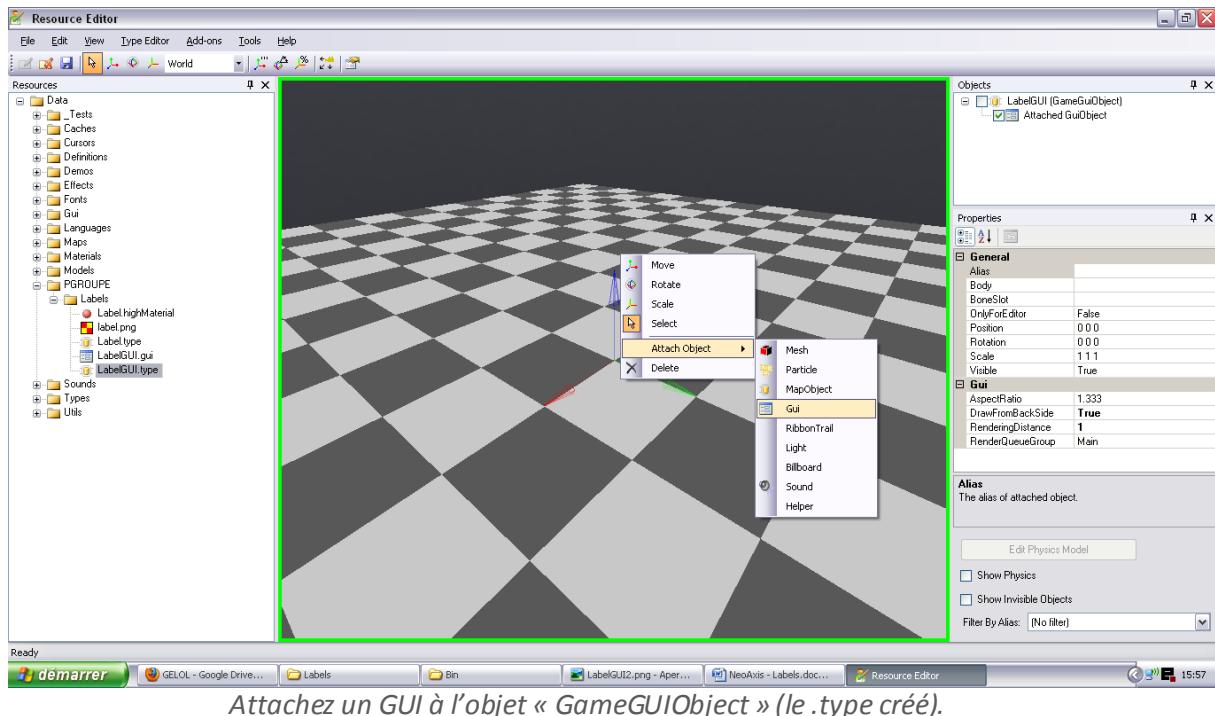
*Enable* (à droite) doit être à « false » pour que la souris ne soit pas affichée et que le GUI soit inerte. Ce paramètre « Enable » pourra, in-game, être mis à « True » via un script C# pour que le label devienne dynamique. Et cliquable (avec, par exemple, un bouton « plus de détails »).

Ensuite, il faut créer le \*.type associé au label (autrement dit, la « classe » du label). Ceci pourra aussi se faire en temps réel, via un C# script. Pour le prototypage, il sera plus rapide de le faire par l'éditeur de ressources. Ce \*.type est créé à partir de la classe « GameGUIObject » :



*Le .type est basé sur « GameGUIObject ».*

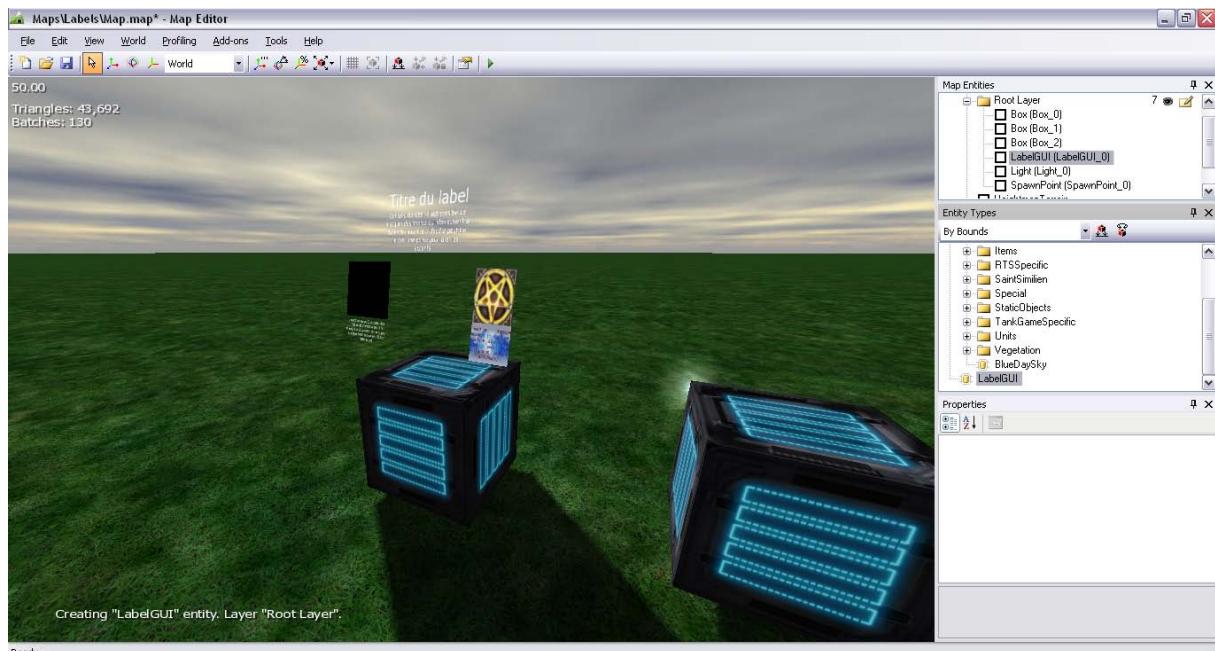
Ensuite, attachez un GUI à ce .type (dès droit, « AttachObject », « GUI ») :



Attachez un GUI à l'objet « GameGUIObject » (le .type créé).

La propriété « RenderingDistance » permet de masquer le GUI lorsque la caméra s'éloigne de celui-ci (c'est-à-dire du label). La distance est en unités (arbitraires) du moteur 3D. Le GUI sera forcément affiché en dessous de cette distance. Il peut être intéressant de créer un script C# personnalisé pour gérer plus en profondeur cette option et, par exemple, masquer le GUI en dessous d'une certaine distance.

Maintenant, dans l'éditeur de carte, il est possible de créer une instance de ce label :



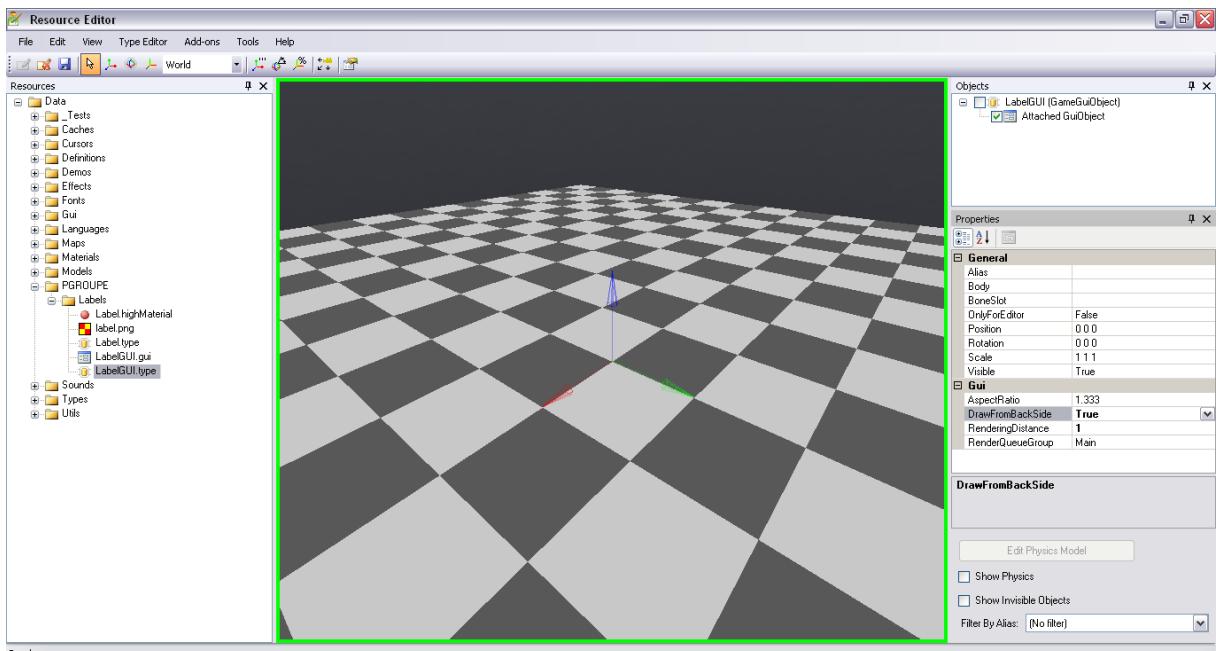
Dans le mapeditor, section « Entity Type », cliquez sur le \*.type que vous avez créé ; puis cliquez dans la fenêtre 3D pour ajouter le Label à l'endroit du clic. Le label peut alors être déplacé, tourné, et redimensionné.

Le label apparaît alors dans le monde 3D (un script C# pourra permettre de l'orienter face à la caméra) :



*Le label est affiché dans le monde 3D.*

L'option « DrawFromBackSide » de l'éditeur de ressource pour le fichier \*.type permet d'afficher le label même s'il est vu de dos :



*L'option « DrawFromBackSide », dans « GUI » (Section « Properties ») permet d'afficher le Label s'il est vu de dos ; par défaut, le label n'est affiché que s'il est vu de face (valeur « false »).*

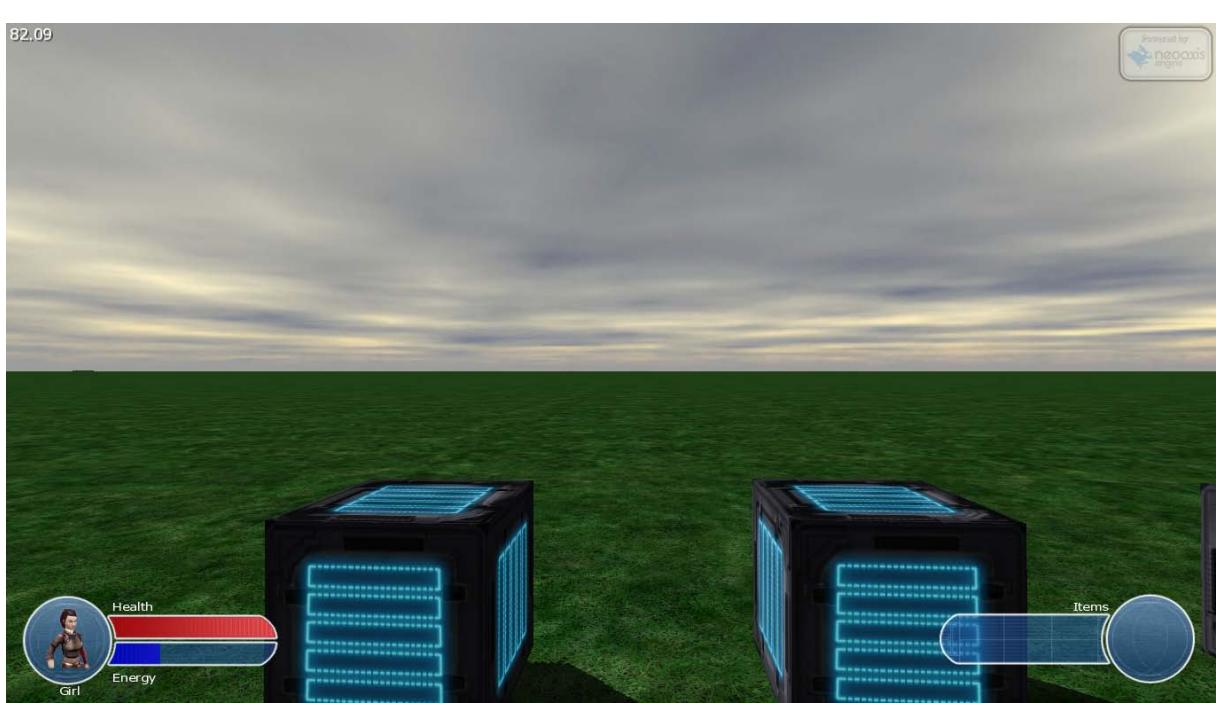


*Le label est visible de dos (le texte est donc à l'envers).*



*Par défaut, le GUI ne fait pas forcément face à la caméra (d'où l'option « DrawFromBackSide »).*

Une autre option du \*.type, section « Properties », permet de masquer le label au-delà d'une distance donnée (« RenderingDistance ») :



Cette technique permet donc de créer des Labels riches, orientables à volonté (puisque il faut créer le script de réorientation soi-même, on a donc un contrôle total sur cette orientation), mais ce système est un peu plus lourd que le premier.

## Croisement des techniques

La meilleure solution consiste à croiser les méthodes énoncées ci-dessus, principalement le billboard et l'in-game GUI, pour utiliser la bonne méthode en fonction :

- Du type de label
- Du nombre de labels
- De la distance au label

Ainsi, les labels les plus importants (fiche d'information sur un bâtiment par exemple) peuvent être affichés via un GUI in-game, sous condition que la caméra doit assez proche de ce bâtiment pour que l'utilisateur ait un réel intérêt à voir ce label. Si l'utilisateur est loin (donc, il y a potentiellement beaucoup de labels), on pourra utiliser la méthode du billboard (si le rendu à l'écran est en 3D).

La méthode du HUD dynamique est un peu trop complexe pour justifier sa mise en place.