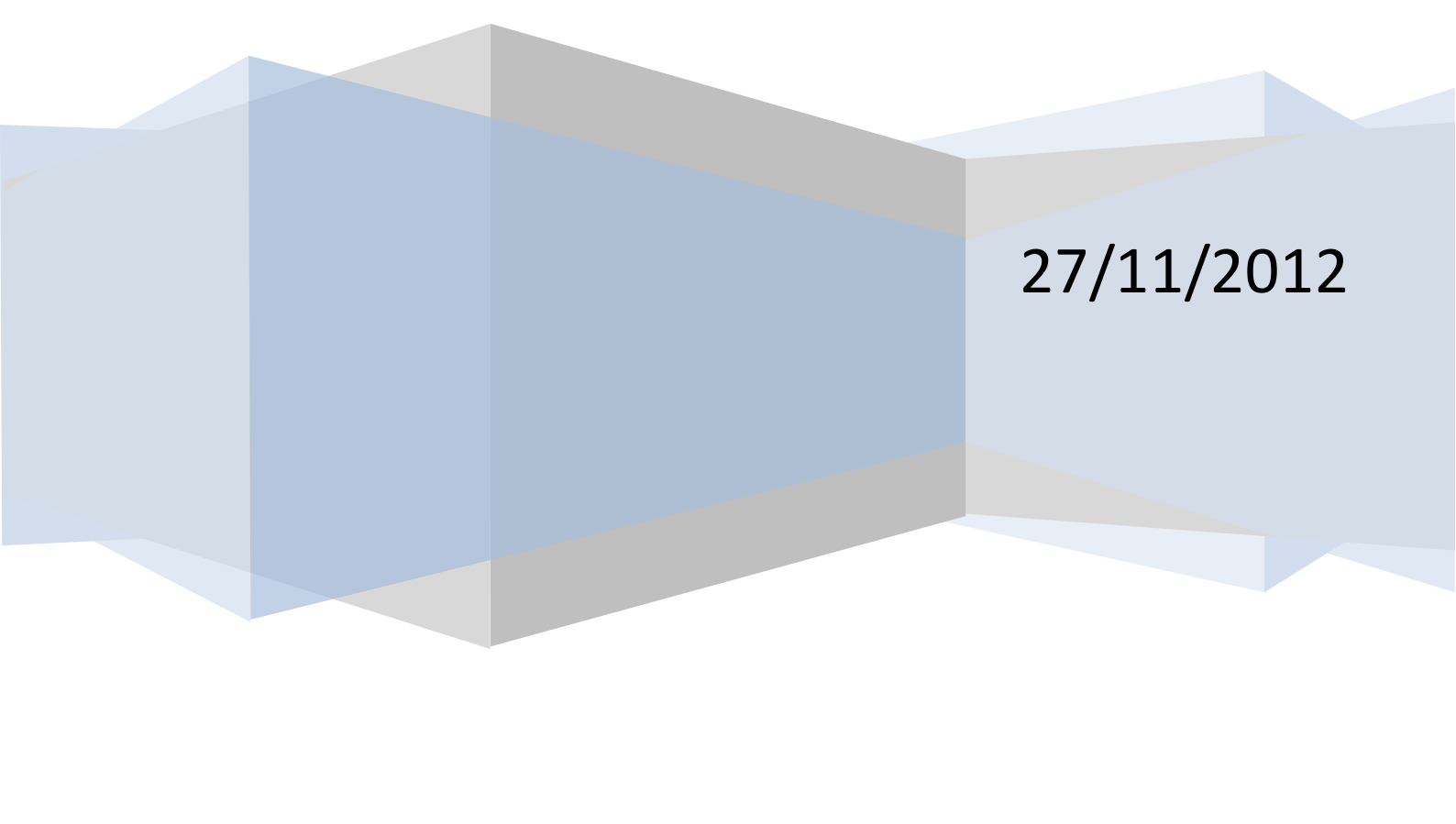


NeoAxis engine

Points

MONIER Vincent



27/11/2012

Sommaire

NeoAxis: Affichage de points.....	2
Description.....	2
Créer un petit objet 3D qui représentera le point	2
Création d'une sphère wireframe	3
Affichage par intersection d'axes locaux	10
Attacher le point à un objet.....	14
Pour aller plus loin....	16

NeoAxis: Affichage de points

Description

Dans un environnement virtuel 3D, il est très utile de pouvoir afficher un point de l'espace dont on connaît les coordonnées XYZ. Généralement, un tel affichage est intéressant pour :

- Un débogage, car l'ordinateur manipule des vecteurs 3D, donc des coordonnées de points
- L'affichage d'un point d'origine (si on veut afficher une ligne ou si on souhaite définir un trajet comme le trajet des piétons par exemple)
- L'affichage de coordonnées GPS, qui se réfèrent à un point

Dans le dernier cas, il faudra toutefois être capable de faire la conversion entre les coordonnées GPS et les coordonnées XYZ de l'environnement 3D. Cette conversion ne fait pas l'objet de ce document.

On considérera donc que l'utilisateur souhaite afficher, avec le moins de code possible, un point de l'espace XYZ, fixe ou mobile (un point mobile peut, par exemple, être le centre d'une balle qui se déplace). Trois méthodes sont envisageables, mais seules les deux dernières seront détaillées, la première présentant trop de défauts.

Créer un petit objet 3D qui représentera le point

La première solution, assez évidente, est de créer un objet 3D suffisamment petit pour que l'on ait le sentiment qu'il s'agisse d'un point. Par exemple, on pourrait créer une sphère (petite) que l'on positionne aux coordonnées XYZ demandées :



La sphère est placée de sorte que son centre soit sur les coordonnées XYZ du point.

Cette méthode requiert un objet 3D, il vous faudra donc en créer un qui sera réutilisé pour chacun des points. Pour créer cet objet, utilisez l'éditeur de ressources. Créez un fichier *.type de classe « MapObject », et attachez-lui un mesh sphérique (il en existe déjà un dans Types/Dynamic/Ball). Vous pouvez modifier le highmaterial (shader définissant la texture) de ce

mesh, et/ou créer un nouveau highmaterial à appliquer à ce mesh (il est recommandé de créer un highmaterial définissant une texture translucide, pour voir le centre de la sphère).

Une fois l'objet créé, vous pouvez :

- Soit l'ajouter manuellement via l'éditeur de map (cliquez sur le fichier *.type dans la section « Entités », et cliquez dans la fenêtre 3D là où vous souhaitez l'ajouter ; si vous sélectionnez une entité dans la fenêtre 3D, vous pouvez la positionner via ses coordonnées XYZ dans la section « Properties », champ « Position »)
- Soit l'ajouter via un script (qui ne sera pas détaillé ici, mais utilisez le tutoriel de la documentation, http://www.neoaxis.com/wiki/Documentation/Programming_Articles/Entity_System#Creating_objects)

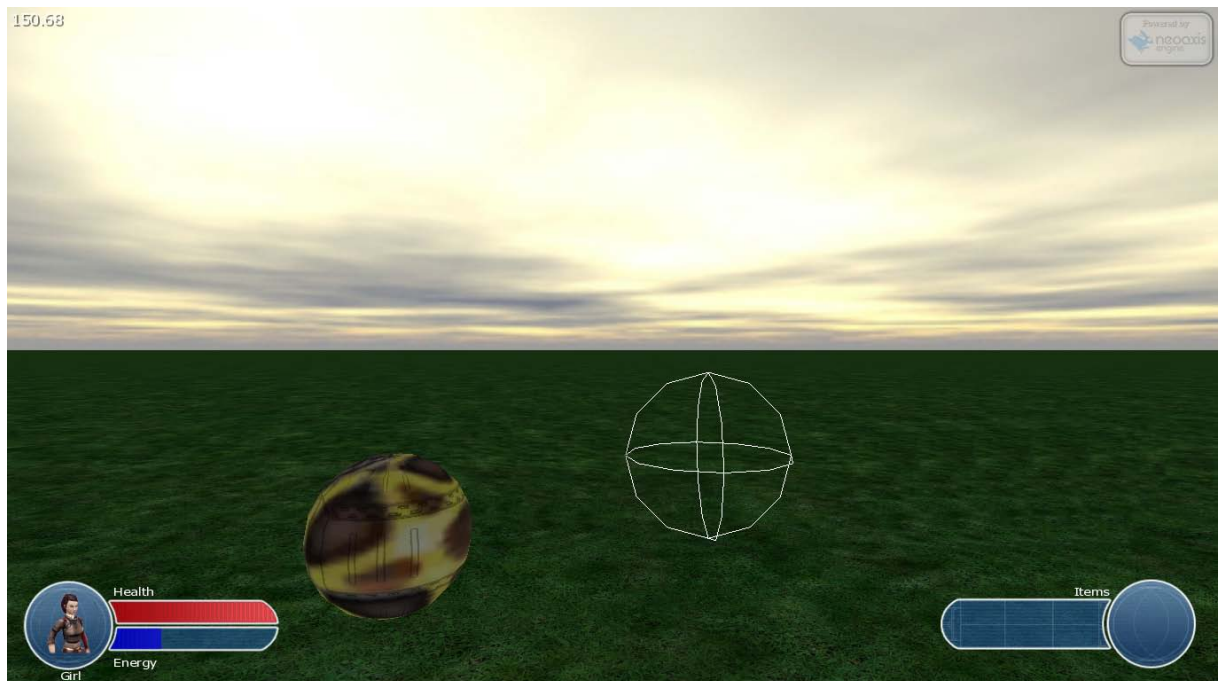
Cette méthode présente l'avantage d'être assez simple, mais elle est très lourde, et symboliser un point par un objet 3D n'est pas toujours heureux car l'objet 3D possède un volume, on n'est donc pas absolument certain de la position du point représenté. Avec ce système, il est également difficile de savoir si deux points sont superposés, surtout si vous ajoutez un peu de code pour faire varier la taille de la sphère avec la distance à la caméra (ainsi, plus la caméra est loin, plus la sphère sera grosse, de sorte qu'elle garde toujours la même taille à l'écran).

Création d'une sphère wireframe

La seconde méthode consiste à créer une sphère filiforme (« wireframe ») et à l'afficher de sorte que son centre coïncide avec le point de coordonnées XYZ que l'on souhaite représenter.

Cette méthode, comme la précédente, est donc peu précise quant à la position exacte du point à afficher, mais elle est bien plus légère que la précédente.

Elle va nécessiter de créer une nouvelle classe pour gérer cet affichage, mais une fois la classe créée, il suffira d'une seule ligne pour ajouter un point à l'écran. De plus, ce point pourra être ajouté sur un objet, et il y sera donc attaché.



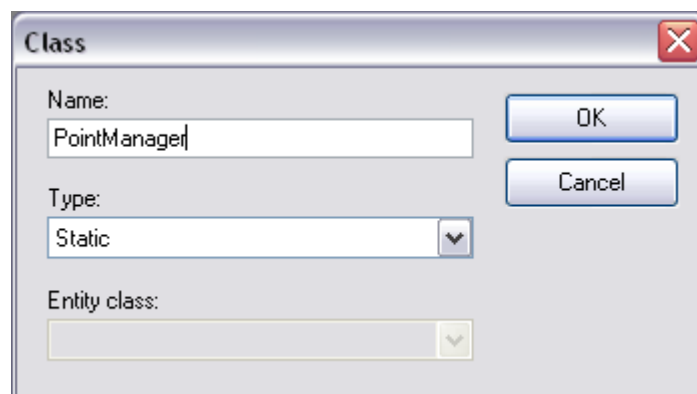
Le point à afficher est le centre de la sphère wireframe, en fil de fer blanc.

Pour réaliser ceci, créez une nouvelle carte (ajoutez-y une heightmap, une skybox et un spawn point, vous pouvez y ajouter une lumière ou modifier l'ambient light en sélectionnant « Map » dans « MapEntities » puis, dans la section « Properties », modifiez « AmbientLight ») ou ouvrez-en une existante. Cliquez sur « World » dans la barre des menus, puis « Logic Editor ».



L'éditeur des scripts associés à la carte actuelle.

Une fois l'éditeur de script ouvert, faites un clic droit sur « classes » puis « New class », et sélectionnez « Static ». Vous pouvez nommer la classe comme vous le souhaitez :

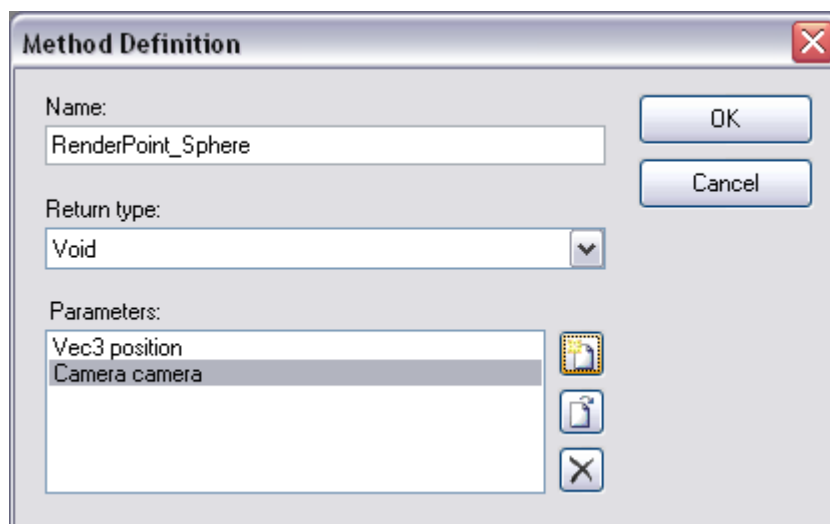


La classe doit être de type « Static ».



La nouvelle classe apparaît dans la section de gauche, « ClassList ».

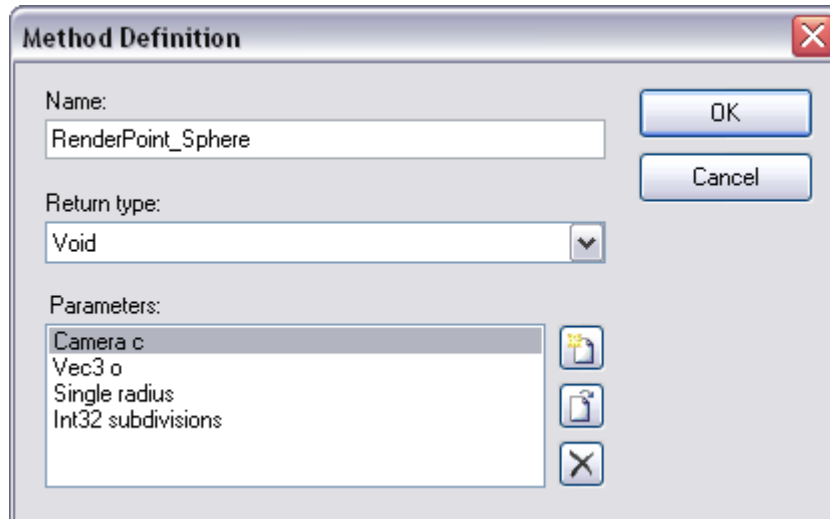
Faites alors un clic droit sur la classe ajoutée, puis « Create Method », « Script type » pour créer une nouvelle méthode pour cette classe. Nommez la méthode comme vous le souhaitez, et définissez ses paramètres via les boutons de la zone « Parameters » :



Créez une nouvelle méthode avec ses paramètres.

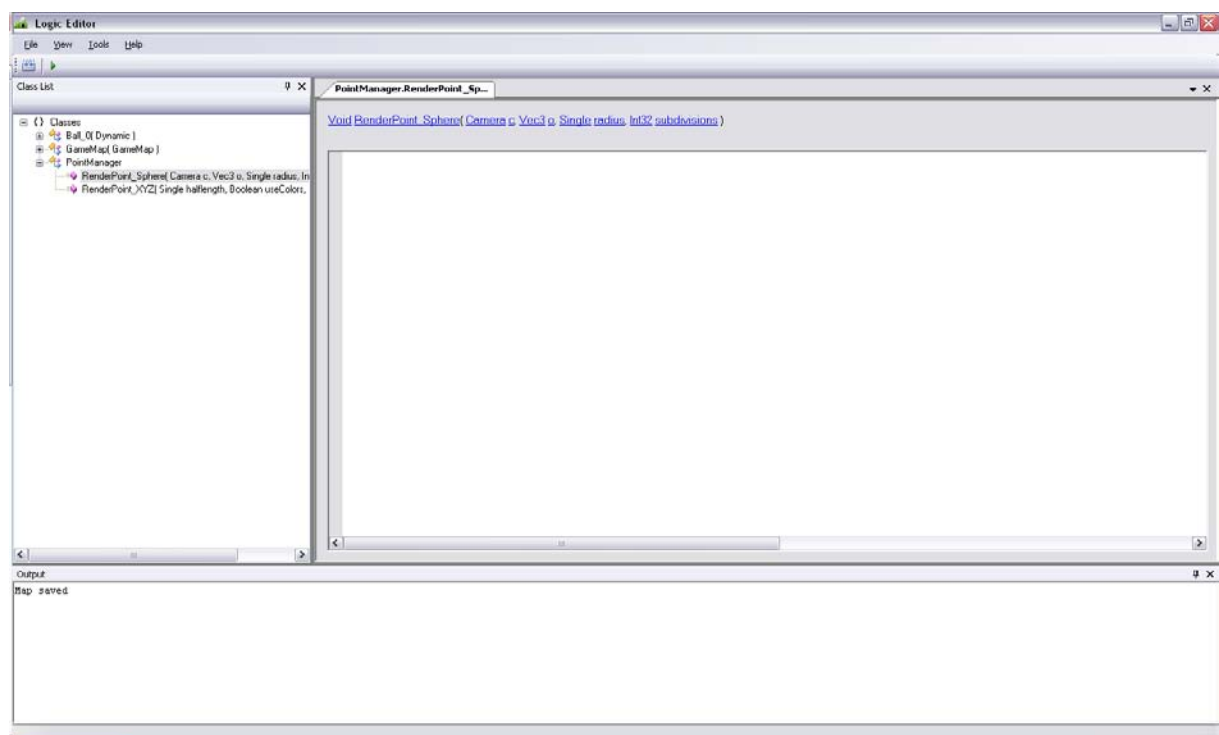
La méthode que vous allez créer va servir à afficher une sphère, utilisez donc un nom explicite (exemple : « RenderPoint_Sphere »). Elle prend en paramètres :

- La caméra, de type « Camera »
- La position du point à afficher, de type « Vec3 »
- Le rayon de la sphère à créer, de type « Single » (équivalent de « Float »)
- Le nombre de subdivisions dans l’affichage wireframe, de type « Int32 »



Créez la méthode avec ses paramètres ; elle retourne « void » mais vous pouvez lui faire retourner autre chose si besoin.

La méthode est alors créée :

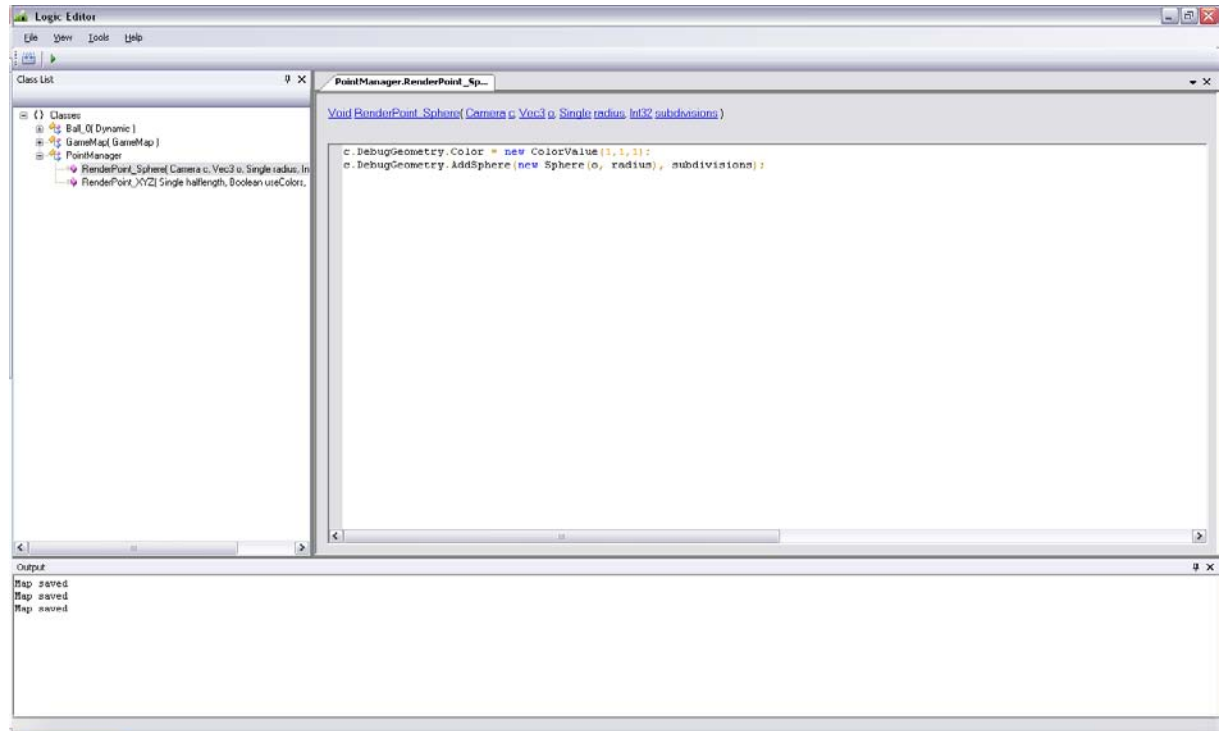


L'éditeur de script a créé la méthode.

Copiez-y le code suivant :

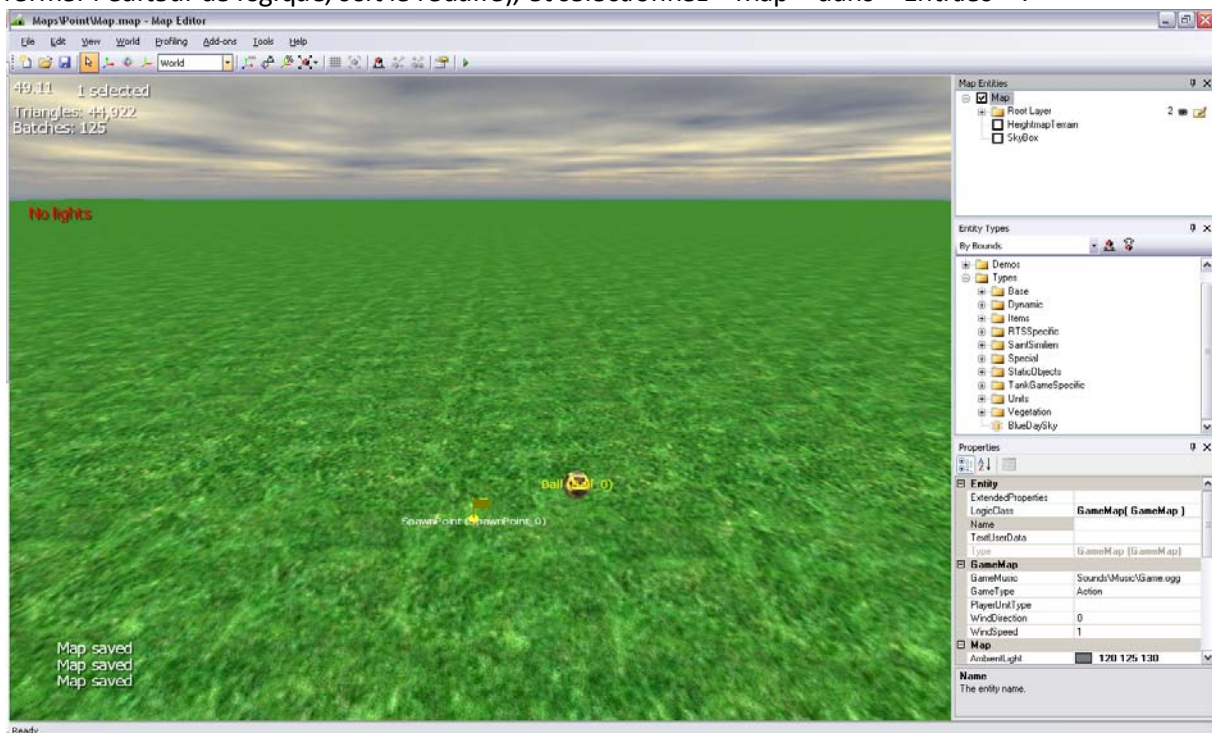
```
c.DebugGeometry.Color = new ColorValue(1,1,1);  
c.DebugGeometry.AddSphere(new Sphere(o, radius), subdivisions);
```

où « c » est la caméra (« Camera »), o la coordonnée du point qui sera le centre de la sphère (« Vec3 »), radius le rayon de la sphère (« Single »), et subdivisions la finesse d'affichage de la sphère (« Int32 », le minimum est 3, évitez de dépasser 32 ; 4 génèrera une bi-pyramide à base carrée).



Collez le code précédent dans la méthode, et sauvez la carte (Ctrl+S)

Maintenant, il vous suffira d'appeler la méthode que vous venez de créer chaque fois qu'il faudra afficher la sphère représentant le point. Pour ce faire, retourner à l'éditeur de carte (vous pouvez soit fermer l'éditeur de logique, soit le réduire), et sélectionnez « Map » dans « Entités » :



Dans la section « Entités », sélectionnez « Map ».

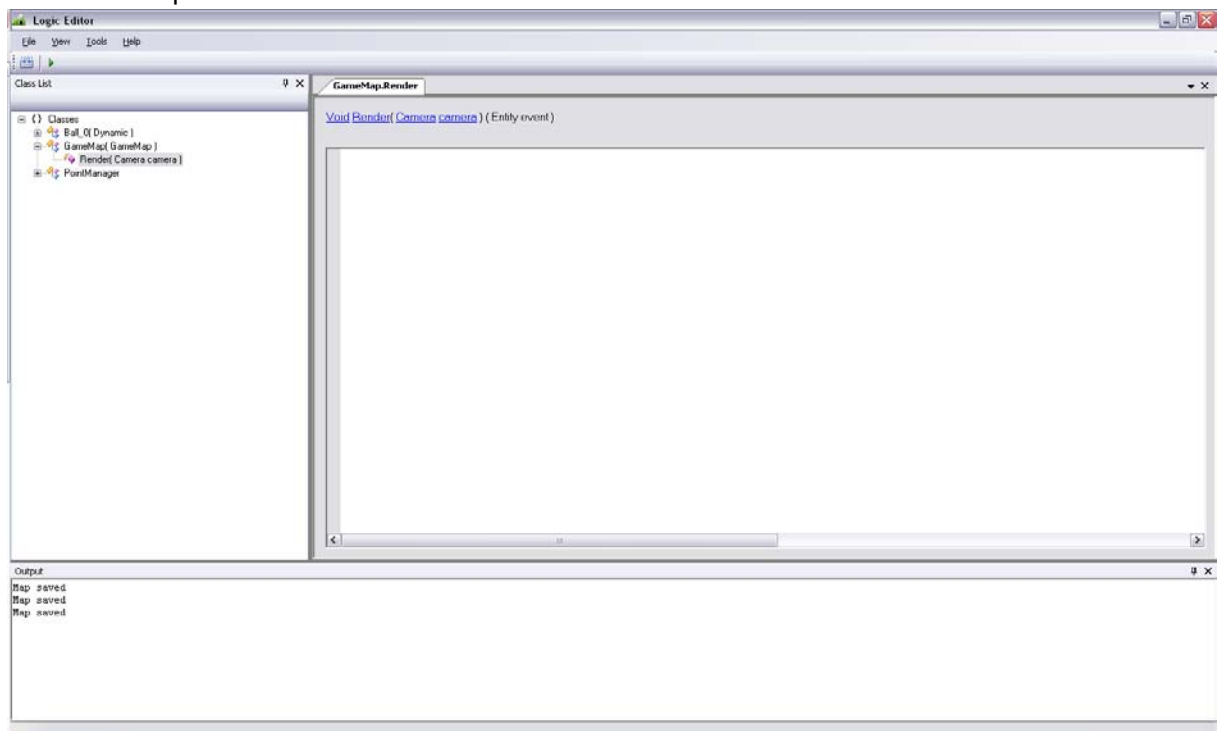
Ensuite, dans la section « Properties », double cliquez sur « LogicClass » pour associer la map à une nouvelle classe dans le script (c'est-à-dire dans l'éditeur de script). Il se peut qu'il faille double-cliquer deux fois sur le nom « LogicClass » (la première fois pour sélectionner la propriété

« LogicClass », la seconde fois pour que l'éditeur de carte associe automatiquement l'objet « Map » à une nouvelle classe dans le script) :



Dans l'éditeur de script, la classe « GameMap (GameMap) » est apparue.

Maintenant, cliquez droit sur cette nouvelle classe, puis « Create Method », « Render », « Script Type ». La méthode « Render » est créée pour cette classe et apparaît sous le nom de la classe. La méthode est vide pour le moment :



La méthode « Render » de l'objet « GameMap » est vide. Cette méthode est appelée à chaque fois qu'une caméra veut afficher la carte.

Dans cette méthode, collez le code suivant :

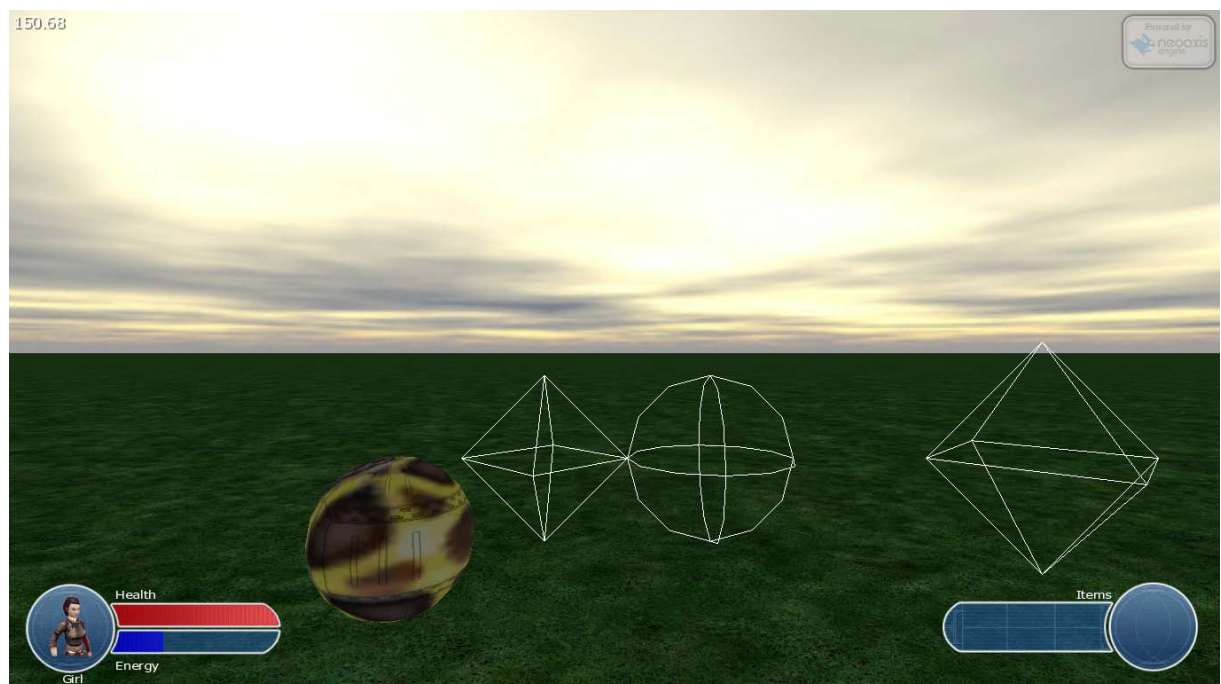
```
Camera c = camera;  
if(c != RendererWorld.Instance.DefaultCamera)  
    return;  
  
PointManager.RenderPoint_Sphere(c, new Vec3(13f, 7f, 1f), 0.7f, 4);
```

La première ligne est un alias (cette ligne est inutile si vous remplacez « c » par « camera » dans toute la méthode).

La seconde ligne teste si la caméra qui essaye d'afficher la carte est bien la caméra par défaut, c'est-à-dire celle qui affiche les informations à l'écran. En effet, imaginez que l'environnement 3D possède, quelque part, un écran virtuel qui affiche une vue aérienne de la carte : il peut alors être intéressant de ne pas afficher les points pour cette caméra. Cette ligne vous donne donc le contrôle de quelle caméra peut afficher les points XYZ.

La dernière ligne affiche la sphère qui représente le point. Le premier paramètre, c, est la caméra d'affichage. Le second, la position du point (« new Vec3(X, Y, Z) »), le troisième est le rayon de la sphère (notez le « f » à la fin du nombre), et le dernier paramètre est le nombre de subdivisions (4 est une bipyramide à base carrée).

Voilà, le point est affiché ! Vous pouvez varier les paramètres de position, taille et finesse :



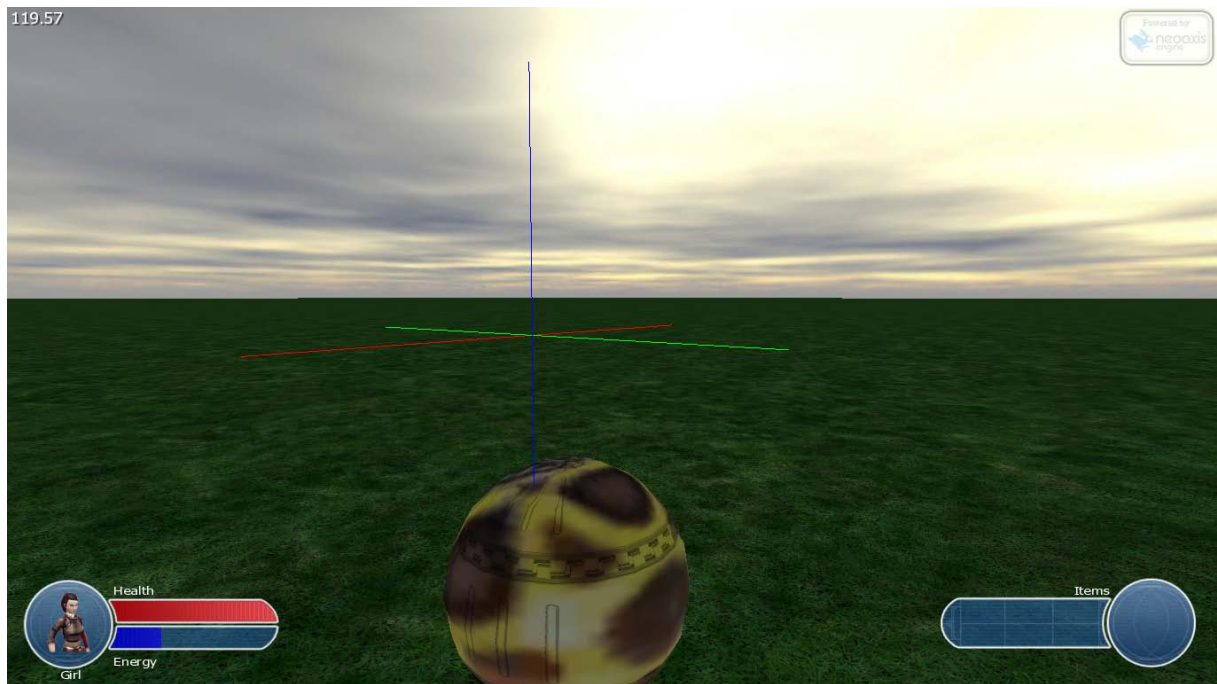
3 « sphères » sont affichées en format fil de fer, pour 3 points différents.

Vous pouvez modifier les méthodes des scripts pour changer, par exemple, la couleur des sphères. Vous pouvez utiliser, dans la méthode de la classe « PointManager », la distance entre le point XYZ et la position de la caméra pour ajuster le rayon de la sphère, et grossir la sphère si la caméra s'éloigne.

Cette méthode est peu précise, car le point est « le centre » de la sphère, ce qui n'est pas forcément facile à voir. La dernière méthode, ci-dessous, est bien plus précise.

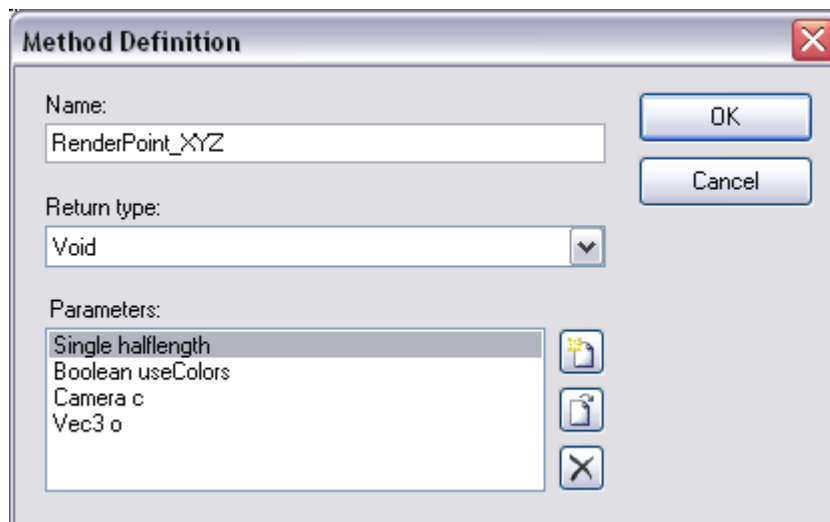
Affichage par intersection d'axes locaux

Une dernière méthode, la plus recommandée, consiste à afficher les axes X Y et Z du repère 3D de sorte que ces axes se croisent en un point, le point à afficher :



Le point est l'intersection des axes X (rouge), Y (vert) et Z (bleu) de l'espace virtuel 3D.

Cette méthode est assez légère, et très précise. Elle reprend les mêmes principes que l'affichage d'une sphère en wireframe. D'abord, créez une classe « PointManager » ou réutilisez la classe précédente, puis créez une méthode pour afficher un point :



La méthode d'affichage d'un point.

Les paramètres sont :

- La demi-longueur des axes à afficher (« Single »)
- L'affichage des couleurs (« Boolean », si true, les axes seront rouge-vert-bleu, si false, ils seront tous blancs)
- La caméra (« Camera »)
- La position du point à afficher (« Vec3 »)

Une fois la méthode créée, collez-y le code suivant :

```
Vec3 x = new Vec3(halflength, 0, 0);
Vec3 y = new Vec3(0, halflength, 0);
Vec3 z = new Vec3(0, 0, halflength);
ColorValue r = new ColorValue(1,0,0);
ColorValue g = new ColorValue(0,1,0);
ColorValue b = new ColorValue(0,0,1);

if (useColors)
    c.DebugGeometry.Color = r;
else
    c.DebugGeometry.Color = new ColorValue(1,1,1);
c.DebugGeometry.AddLine(o-x, o+x);

if (useColors)
    c.DebugGeometry.Color = g;
c.DebugGeometry.AddLine(o-y, o+y);

if (useColors)
    c.DebugGeometry.Color = b;
c.DebugGeometry.AddLine(o-z, o+z);
```

Les trois premières lignes définissent les vecteurs +X, +Y et +Z. Vous pouvez utiliser une matrice de rotation pour tourner ces trois axes si besoin.

Les trois lignes suivantes définissent les couleurs des axes (0 → noir, 1 → blanc).

Les tests « if » changent la couleur des axes si l'option « useColors » est à « true » ; sinon, la couleur est définie une seule fois sur « blanc ».

Les autres lignes, avec la méthode « AddLine », ajoutent une ligne à afficher pour la caméra.

Pour utiliser ce code, revenez sur « GameMap.Render », et utilisez le code suivant pour afficher un point :

```
PointManager.RenderPoint_XYZ(0.3f, false, c, new Vec3(10.0f, 11f, 1.5f));
```

Le Vec3(X, Y, Z) définit la position du point. Le paramètre « useColors » est ici à « false », donc les axes seront en blanc. La valeur « 0.3f » indique que les axes auront une demi-longueur de 0.3, donc leur longueur totale sera 0.6.

Vous pouvez, dans la méthode « PointManager.RenderPoint », ajouter un code qui agrandira les axes en fonction de la distance à la caméra. Le résultat est le suivant :



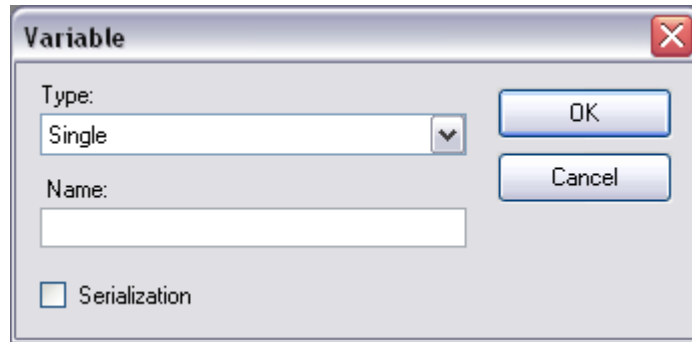
Affichage d'un point

La taille peut être modifiée en temps réel : « GameMap.Render » est appelé avant chaque rendu d'image à l'écran :



Vous pouvez augmenter ou diminuer dynamiquement la taille des axes.

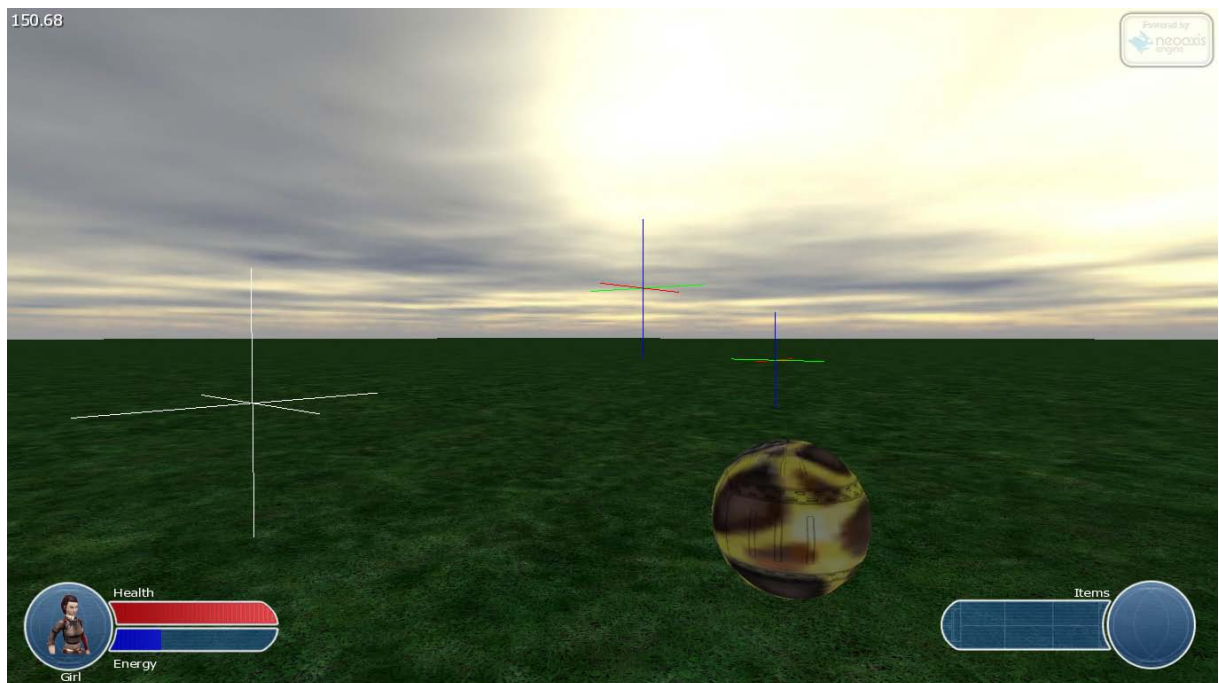
Pour ajouter une variable à une classe (un « attribut »), faites un clic droit sur la classe, puis « Create Variable » :



Vous pouvez créer un attribut pour une classe.

Cet attribut pourrait vous servir à stocker la taille des axes pour l'augmenter ou la diminuer avec le temps par exemple.

Vous pouvez afficher plusieurs points si besoin :

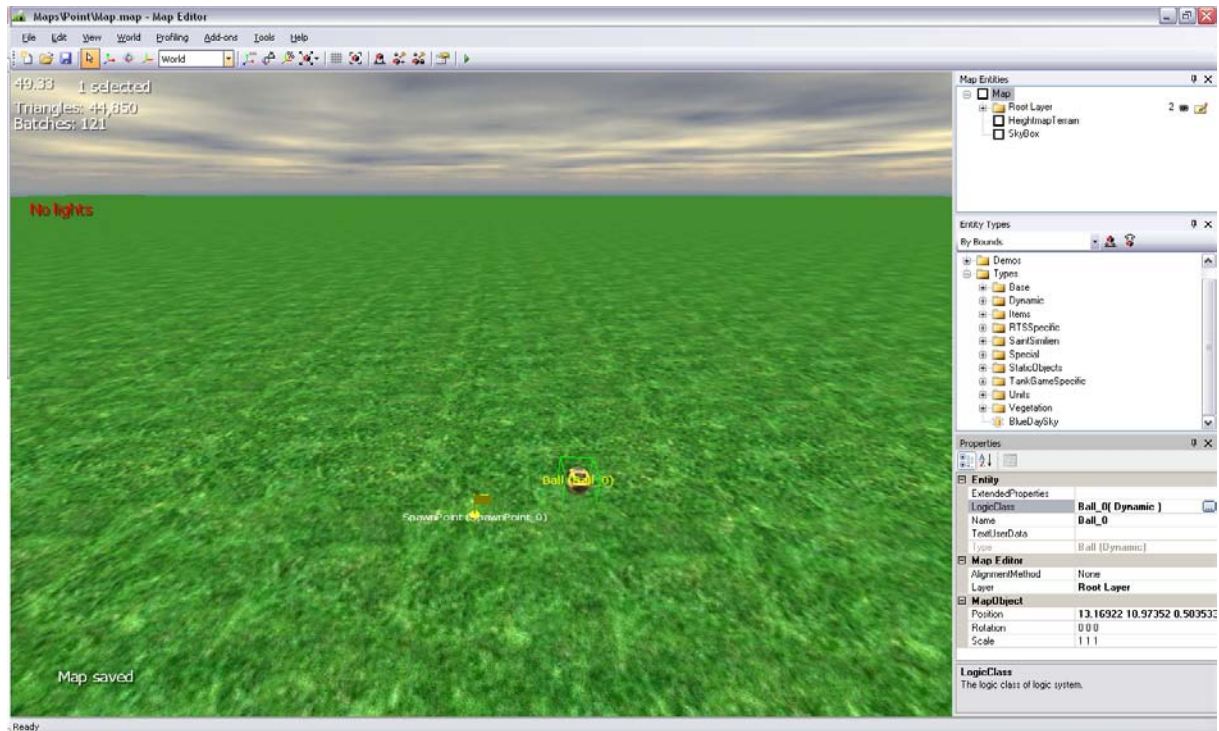


Plusieurs points peuvent être affichés en même temps.

Mais les points sont, pour l'instant, statiques. Il peut être utile d'attacher le point à un objet qui peut se déplacer ou être supprimé. Référez-vous au paragraphe suivant pour attacher le point à un objet.

Attacher le point à un objet

Pour ce faire, ajouter un objet à la carte (soit via l'éditeur de carte, soit via le script), et associez-le à une classe logique (double-double-clic sur « LogicClass » dans l'éditeur de carte). Créez la méthode « Render » pour cette classe, et ouvrez-la :



L'objet « Ball_0 » est associé à la classe logique « Ball_0 (Dynamic) ». Si votre objet n'a pas de nom (« Name » est vide dans la section « Properties »), ajoutez-lui un nom unique.

Dans l'éditeur de logique, modifiez la méthode « Render » de cette nouvelle classe et collez-y le code suivant :

```
MapObject obj = (MapObject)Entities.Instance.GetByName("Ball_0");;
```

```
Vec3 o = obj.Position;
```

```
Camera c = camera;
```

```
if(c != RendererWorld.Instance.DefaultCamera)
    return;
```

```
PointManager.RenderPoint_Sphere(c, o, 1.1f, 32);
```

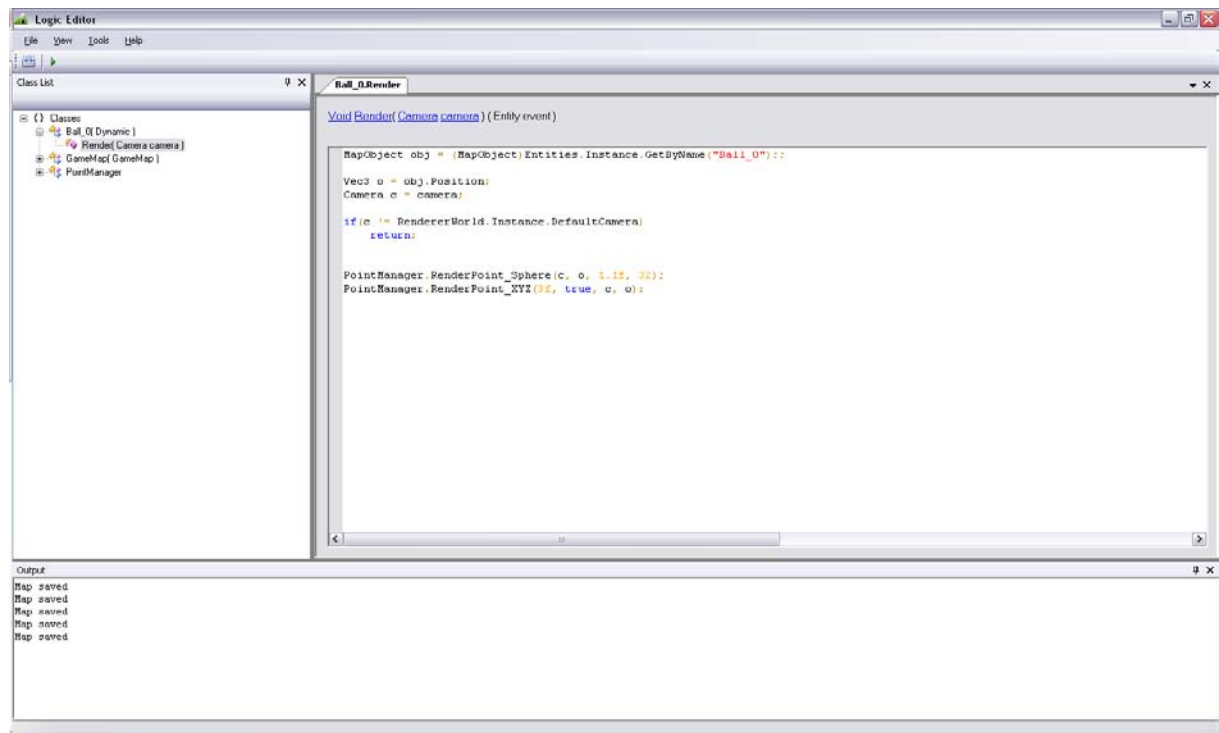
```
PointManager.RenderPoint_XYZ(3f, true, c, o);
```

La première ligne récupère l'objet dynamique (remplacez « Ball_0 » par le nom de votre objet). La suivante récupère la position de l'objet (vous pouvez ajouter un delta à cette position, par exemple, « Vec3 o = obj.Position + new Vec3(0, 0, 10) » pour décaler la position du point de 10 unités sur l'axe Z par rapport au centre de l'objet).

La suivante récupère la caméra.

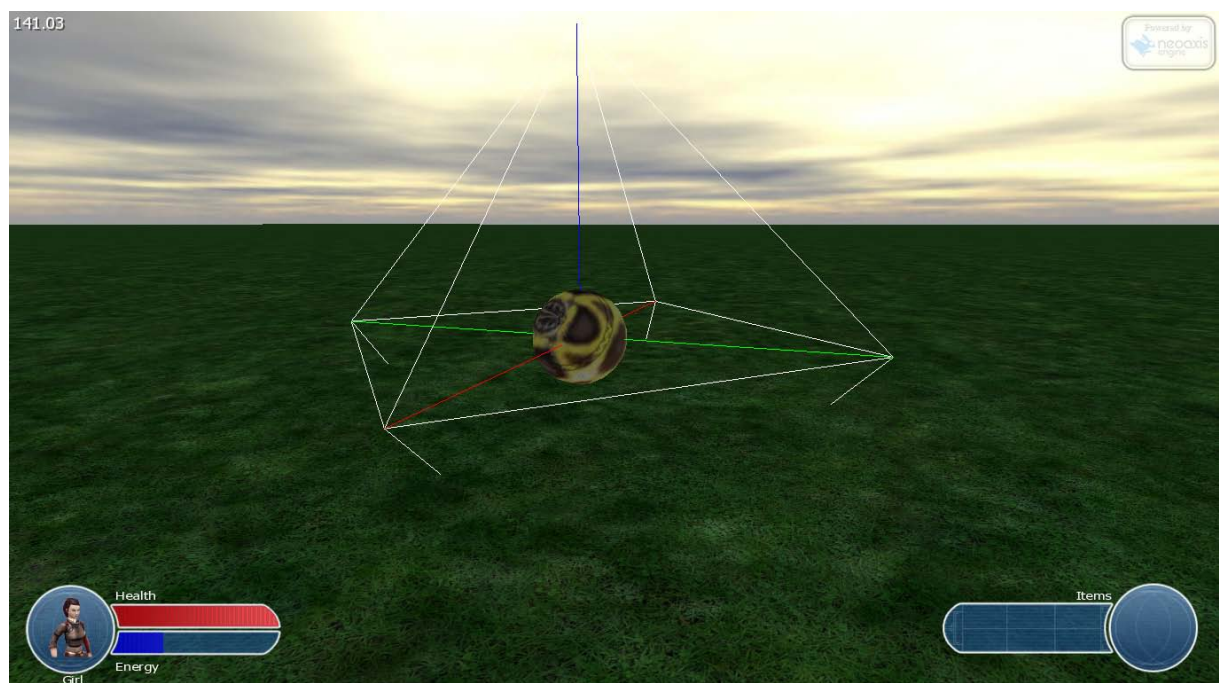
Le texte « if » permet de s'assurer qu'il s'agit bien de la caméra par défaut (celle qui s'affiche à l'écran).

Les deux dernières utilisent les méthodes « RenderPoint_Sphere » et « RenderPoint_XYZ » définies précédemment. Vous pouvez n'utiliser qu'une des deux méthodes.

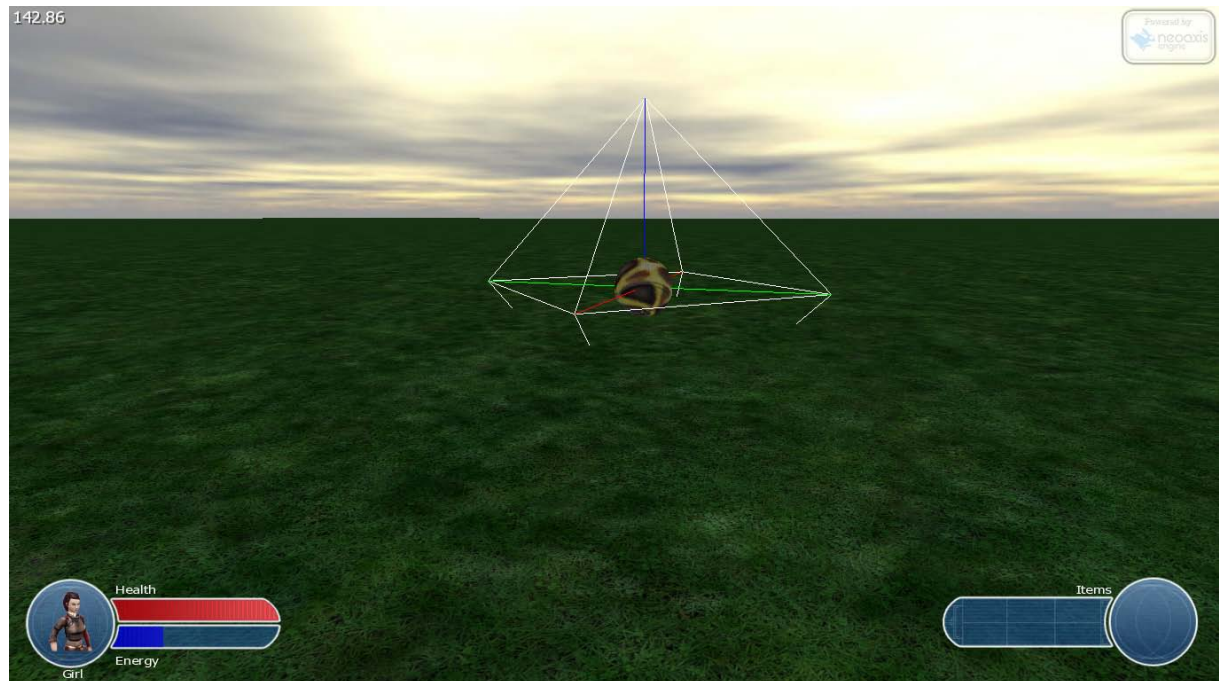


La méthode « Ball_0.Render » une fois le code collé.

Le point affiché est alors le centre de l'objet. Même si l'objet bouge, le point se déplace avec lui :



Les deux affichages sont utilisés (axes RGB et bipyramide). Le point est le centre de la balle ...



...si la balle se déplace, le point se déplace aussi.

Pour aller plus loin...

La méthode « Addline » peut être utilisée pour d'autres choses si besoin. Par exemple, pour afficher le trajet d'un piéton, ou pour dessiner des lignes de champ. N'hésitez pas à réemployer la méthode suivant vos besoins.