IUT de Provence – Site d'Arles
Département d'Informatique
2ème année

2006 – 2007

IN Connaissances complémentaires

TP N°2 – Max Script

Programmer en MAXScript

MAXScript fournit des commandes qui permettent de créer des conditions et des répétitions. En voici brièvement les syntaxes.

Condition

Voici deux exemples de syntaxes :

```
if mybox.height == 10 then mybox.width = 20

if mybox.height == 10 then mybox.width = 20 else mybox.width = 10
```

Ici, on regarde si la hauteur de la boîte est égale à 10 (2 signes =)

Les autres types de conditions possibles :

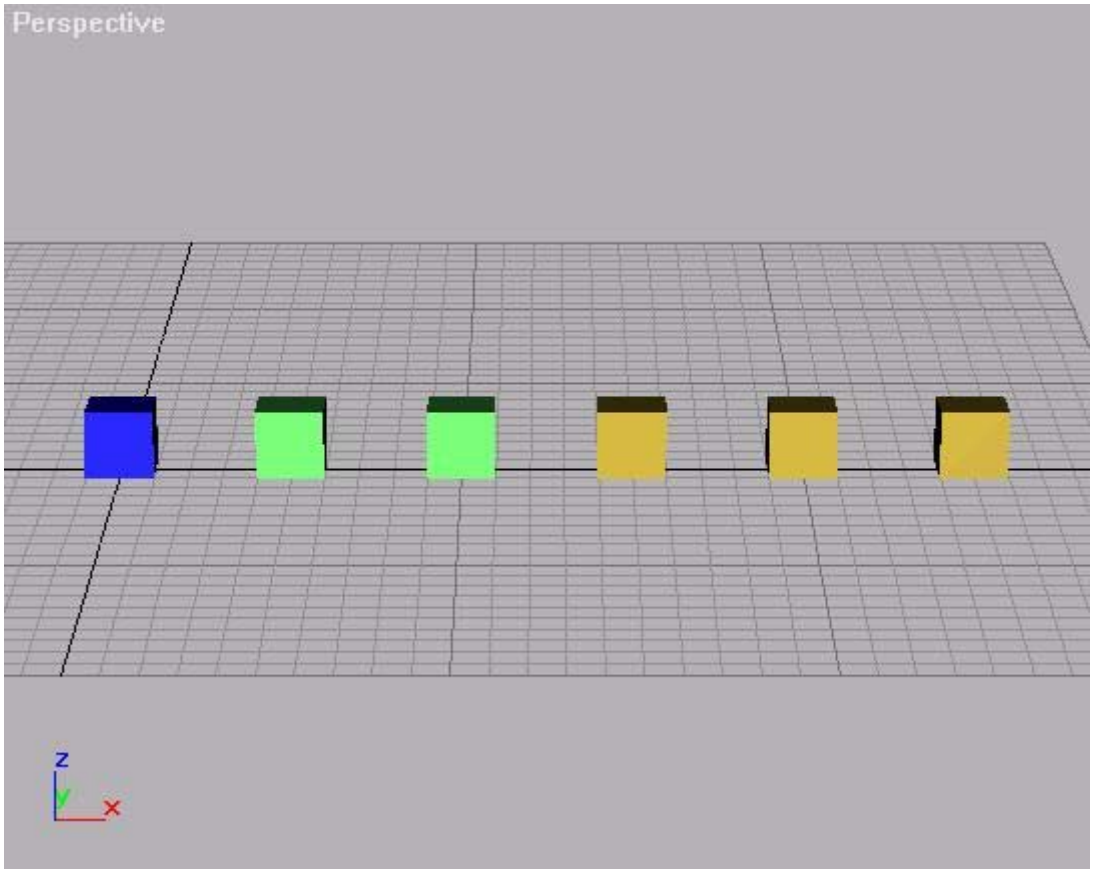
==	égal à
!=	non égal
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Répétitions

La boucle pour

Exemple 1 :

```
for i = 1 to 5 do
(
	box_copy = copy mybox
	box_copy.pos = [i * 50, 0, 0]
)
```



Exemple 2 :

```
for i = 1 to 5 by 2 do
(
	box_copy = copy mybox
	box_copy.pos = [i * 50, 0, 0]
)
```

Répétition avec While et Do

Syntaxe:

do <expr> **while** <expr>

while <expr> **do** <expr>

Exemple :

```
x=10
while x>0 do
(
	x-=1
	print x
)
```

Retourne: 9
8
7
6
5
4
3
2
1
0
0

Variables Locales et Globales

En MAXScript, vous pouvez créer deux types de variables : les variables « locales » et les variables « globales ». Une variable globale peut être utilisée n'importe où dans le programme. La seule chose que vous ne pouvez pas faire est de bien sûr utiliser une variable globale avant de la déclarer !

Voici un exemple de déclaration :

```
global rad = 10
sphere radius:rad
```

Il est aussi possible de déclarer des variables locales. Elles ne sont utilisables que dans le bloc dans lequel elles ont été déclarées. Voici un exemple :

```
for i = 1 to 3 do
(
	local rad = 10
	s = sphere()
	s.pos.x = i * 10
	s.radius = rad
)
```

Dans l'exemple précédent, bien que la variable *rad* ait été initialisée à une valeur spécifique, il n'est pas obligatoire d'assigner une valeur à la création de la variable. Le nom de la variable sera alors reconnu mais on ne connaîtra pas le type de la variable. MAXScript assigne alors automatiquement une valeur spéciale « indéfinie ». La première fois qu'une valeur sera assignée à la variable, le type correspond sera choisi pour la variable. En pratique il est préférable d'assigner une valeur initiale à la variable.

L'utilisation de la notion de variable globale et de variable locale n'est pas obligatoire. MAXScript le fera de façon implicite en fonction de l'endroit où vous déclarez votre variable.

Création de fonctions

Les fonctions sont définies en MAXScript en utilisant la syntaxe suivante :

(**function** | **fn**) <name> { <parameter> } **=** <expr>

<name> est le nom de la fonction

L'expression après le **'='** (signe égal) est le corps de la fonction.

Exemples :

Fonction calculant la somme de deux nombres **a** et **b** passés en paramètre. La valeur retournée provient implicitement de la dernière opération effectuée :

```
function add a b =
(
	a + b
)
```

Fonction calculant récursivement la factorielle d'un nombre **n** passé en paramètre :

```
fn factorielle n =
(
	if n <= 0 then
		1
	else
		n * factorielle(n - 1)
)
```

La fonction suivante teste si la variable **val** passée en paramètre à la fonction est négative, nulle ou positive, et affiche une boîte d'alerte en conséquence :

```
function signe val =
(
	if val == 0 then
		messagebox ("Egal à 0")
	else if val > 0 then
		messagebox ("Supérieur à 0")
	else
		messagebox ("Inférieur à 0")
)
```

Afin de tester cette fonction, vous devez l'appeler en tapant par exemple le programme suivant :

```
Valeur = -2
signe (Valeur)
```

Utilisation de tableaux de valeurs

Un tableau est une séquence de valeurs de taille variable. Les valeurs dans un tableau peuvent être de tous types

Syntaxe :

#(<value>, <value>, ...)

#() -- un tableau vide

Propriétés :

<tableau >.**count** : Integer

Permet de lire / Stocker le nombre d'éléments contenus dans un tableau.

Opérations :

<Tableau>**[**<integer>**]**

Retourne l'élément d'un tableau. On commence à 1.

<Tableau>**[**<integer>**]** = <value>

Stocke la valeur value dans le tableau. La taille du tableau augmente si nécessaire

Exemple:

a = #(1,2,3,4)

Méthode Mesh Vertex

La méthode suivante fournit un accès au maillage représentant l'objet et les normales aux sommets. Le maillage éditable peut par exemple être créé grâce à un tableau de sommets et un tableau de faces. Chaque valeur Point3 dans le tableau de sommet spécifie la position du sommet dans le système de coordonnées courant. Chaque valeur Point3 dans le tableau de face spécifie les 3 indices des sommes qui forment la face. Le tableau materialIDs spécifie l'identifiant du matériel qui doit être assigné à chaque face. Chaque Point3 du tableau tverts spécifie les coordonnées UVW de la texture (voir sujet de tp suivant).

Syntaxe :

Mesh vertices : <tableau_de_points3> **faces :** <tableau de points3> **[materialIDs :** <tableau d'entiers>**] [tverts :** <tableau_de_point3>**]**

Un exemple de construction d'un mesh utilisant cette forme de constructeur est donné ci-dessous :

```
mesh vertices:#([0,0,0],[10,0,0],[0,10,0],[10,10,0]) \

faces:#([1,2,3],[2,4,3]) materialIDS:#(1,2)
```

Une fois le mesh créé, il est possible d'utiliser les méthodes suivantes :

Les méthodes liées

getNumVerts <mesh>
Retourne le nombre de sommets comme un entier. On peut aussi utiliser **<mesh>.numverts**

getNumFaces <mesh>
Retourne le nombre de faces comme un point3. On peut aussi utiliser **<mesh>.numfaces**. Les indices.

setNumVerts <mesh> <vert_index_integer>
Stocke le nombre de sommet spécifié

getVert <mesh> <vert_index_integer>
Retourne la position (dans le système de coordonnées courant) du sommet comme un point3

setVert <mesh> <vert_index_integer> (<point3> | <float> <float> <float>)

Stocke la position des coordonnées du sommet comme un point3, ou 3 floats X, Y, Z

Exemple :

```
obj = sphere()                      -- créé une sphère
convertToMesh obj                   -- Conversion en mesh editable
for v = 1 to getNumVerts obj do     -- boucle sur tous les sommets
(
      vert = getVert obj v          -- obtient le v-ième sommet
      vert.z = 0.0                  -- met la coordonnée Z à 0.0
      setVert obj v vert            -- replace la valeur dans le
                                    -- v-ième sommet
)
update obj                          -- Mise à jour du mesh
                                    -- → La sphere est applatie
```

deleteVert <mesh> <vert_index_integer>

Efface le sommet indiqué du maillage, ainsi que toutes les faces qui lui sont liées. Renumérote les sommets et faces pour tenir compte de la suppression

Exemple :

```
p = plane()                 -- crée un plan
convertToMesh p             -- transforme le plan en mesh éditable
deleteVert p 13             -– supprime le vertex 13 (le vertex du
                            -- milieu)
update p                    -- Mise à jour du mesh
```

getNormal <mesh> <vert_index_integer>

Retourne la normale au sommet indiqué.

Exemple :

```
obj = geosphere()       -- crée une GeoSphere
convertToMesh obj       -- transforme la GeoSphere en mesh éditable

// Affiche les normales de chaque sommet de la GeoSphere
for v = 1 to obj.numVerts do
(
      print (getNormal obj v)
)
```

setNormal <mesh> <vert_index_integer> <point3>

Permet de stocker la valeur du vecteur normal à un sommet

Vous pouvez aussi utiliser les méthodes suivantes :

getVertSelection <node> [<modifier_or_index>] [**name**:<name>]

getVertSelection <mesh>

setVertSelection <node> [<modifier_or_index>] (<sel_bitarray> | <sel_array>)

[**name**:<name>] [**keep**:<boolean>]

setVertSelection <mesh> (<sel_bitarray> | <sel_array>) [**keep**:<boolean>]

averageSelVertCenter <node>

averageSelVertNormal <node>

Selection d'un objet

Il est possible de sélectionner à la souris un objet dans la scène au moyen de la fonction **`pickobject`**.

Le programme suivant affiche un utilitaire (panneau **Utilitaires** → bouton **MAXScript** → choisir **Infos objet** dans la liste déroulante **Utilitaires**) permettant, lorsque l'on clique sur un bouton, de sélectionner ensuite un objet de la scène. Si cet objet est un objet géométrique (de classe de base **`Geometryclass`**), alors on affiche son nombre de sommets et de faces.

```
utility creation_objet "Infos objet"
(

    button bouton_infos "Choisir objet" width:60 height:20

    fn GetGeometry o =
    (
        Superclassof o == Geometryclass and classof o != TargetObject
    )

    on bouton_infos pressed do
    (
        obj = pickobject filter:GetGeometry
        if isValidNode obj then
        (
            tmesh = snapshotAsMesh obj

            num_verts = tmesh.numverts
            num_faces = tmesh.numfaces

            print num_verts
            print num_faces
        )
    )
)
```

Parcours des éléments de la scène

Les objets de la scène (objets 3D, lumières, caméras, etc.) sont stockés dans le tableau **`objects`**.

On peut connaître le nombre d'objets grâce à **`objects.count`**

Pour parcourir tous les objets de la scène :

```
for o=1 to objects.count do
(
      obj = objects[o]
      if GetGeometry obj then
      (
         -- il s'agit d'un objet 3D
      )
)
```

Questions

1 Essayez les différents scripts donnés en exemples.

2 Ecrivez un script de type utilitaire permettant de compter le nombre de polygones des différents objets de la scène.

2.1 L'interface comportera un contrôle de type spinner qui permettra de régler un nombre de faces. On voudra détecter dans la scène les objets dont le nombre de faces est supérieur à ce nombre.

2.2 L'interface comportera un bouton « **Compter** ». Lorsque l'utilisateur cliquera sur ce bouton, vous parcourrez tous les objets de la scène. Si l'objet courant est de type objet 3D, alors vous regarderez son nombre de faces. Si ce nombre est supérieur au nombre de face du spinner, alors vous afficherez le nom de l'objet et son nombre de faces.

3 Ecrivez un script de type utilitaire permettant de créer un cylindre de type mesh.

3.1 Ecrire une fonction **`cylindre`** permettant de créer le corps d'un cylindre. Elle recevra en paramètre le rayon et la hauteur du cylindre, ainsi que le nombre de points sur la circonférence du cylindre. Vous y créerez un objet de type mesh.

4 Ajoutez dans l'interface un bouton « **Créer** ». Lorsque l'utilisateur cliquera sur ce bouton, vous appellerez la fonction **`cylindre`**.

4.1 Ajoutez à la fonction **`cylindre`** le calcul des deux « couvercles » du cylindre.

4.2 Ajoutez des contrôles de type **`spinner`** permettant de définir dans l'interface la hauteur, le rayon et le nombre de points sur la circonférence du cylindre lors de la création.