# IEVA - Interface Adaptative

Eric Maisel - Pierre Deloor

Octobre 2023

Au cours des différentes scéances de laboratoire nous chercherons à mettre en oeuvre un moyen d'explorer une base de données de façon pertinente pour un utilisateur au moyen d'une métaphore 3d. Il s'agit ici de prendre en main quelques concepts liés à cette représentation 3d.

## 1 Appropriation

### 1.1 Lancer le programme

Le système est basé sur l'architecture client-serveur.

Lancer le serveur :

```
> python serveur/serveur.py &
```

Pour exécuter le client :

```
> firefox client/index.html
```

**ATTENTION** pour restituer correctement les objets 3d il faut modifier l'option `security.fileuri.strict_origin_policy` et lui affecter la valeur `false` (en accédant à la page Web `about:config`).

### 1.2 Ajouter des objets à la scène

Le client et le serveur interagissent à travers des envois de messages (les requêtes) envoyées par le client au serveur. Celui-ci renvoie des instructions agissant sur les objets de la scène, instructions contenues dans des trames au format JSON.

Actuellement le serveur peut répondre à deux types de messages. Ces réactions sont implémentées dans le fichier `serveur/serveur.py` dans les fonctions `init` et `click`.

Certains des objets sont créés dans le monde lors de l'initialisation du client (dans la fonction `init`) d'autres le sont en réaction à l'occurrence d'événements(par exemple dans la fonction `click`).

Vous travaillerez dans un premier temps dans la fonction `init`.

Il faut d'abord créer la scène :

```
scene = Scene()
...
return jsonify(scene.jsonify())
```

L'instruction suivante permet de créer un acteur connu par le serveur sous le nom "toto" et de type actor".

```
scene.actor("toto","actor")
```

Cet acteur est complètement abstrait (pas d'incarnation et non situé dans l'espace) qu'il est impossible de restituer.

On peut associer un objet à cet acteur. Ici une sphère verte de diamètre 20cm :

```
scene.actor("toto","actor").add(sphere("toto",0.2,"vert"))
```

Cette instruction illustre l'architecture de l'application 3d de type entités-composants : elle est composée d'entités (les acteurs) auxquelles on ajoute des composants qui permettent de décrire la forme et l'aspect des objets mais également leur comportement.
On verra ultérieurement que des sections de code des composants et des entités sont régulièrement exécutés pour modifier l'état des entités.

Programmer le monde revient donc :

1. A créer/supprimer des entités
2. A ajouter/retirer à des composants à ces entités (à l'initialisation ou en cours d'exécution)

Voir en annexe la liste des composants qui correspondent à des primitives 3d qu'il est possible d'utiliser dans ce travail.

Pour l'instant les objets 3d utilisés sont placés à l'origine du repère de la scène. Des composants permettent de les placer et de les orienter dans l'espace :

```
a =scene.actor("lulu","actor").add(box("lulu",3,1,1,"rouge"))
a.add(position(5,2,3)).add(rotation(0,math.pi/4,0))
```

Ici un acteur incarné par une boite rouge est crée. On lui applique

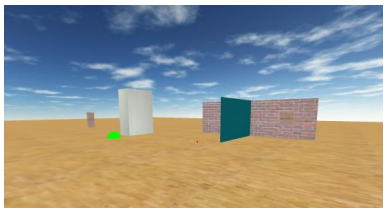
- une rotation paramétrée par les angles d'Euler suivants : 0 autours de l'axe $x$, $\frac{\pi}{4}$ autours de l'axe $y$ et 0 autour de l'axe $z$
- une translation de vecteur $(5, 2, 3)^t$

Il peut être pratique de placer les objets non pas par rapport au repère de la scène mais par rapport à un objet de la scène. Par exemple on veut poser une sphère sur une boite placée à un endroit quelconque de la scène.

```
scene.actor("boite01","actor").add(box("boite01",1,1,1,"rouge"))
scene.getActor("boite01").add(position(2,0.5,3)).add(rotation(0,math.pi/4,0))

# Ajoût  de la sphère

scene.actor("sphere01","actor").add(sphere("sphere01",1,"vert"))
scene.getActor("sphere01").add(position(0,0.5,0))
scene.getActor("sphere01").add(anchoredTo("boite01"))
```



**Exercice** essayer d'exécuter le code donné ci-dessus sans la dernière instruction. Puis avec la dernière instruction.

- Qu'observez vous ?
- Que pouvez vous en déduire ?

**Exercice** en utilisant ce principe accrochez deux tableaux (composant créé en appelant la fonction `poster`) à un mur (composant créé en appelant la fonction `wall`).

**Exercice** proposez les instructions qui permettent d'obtenir une scène correspondant à l'image ci-dessus :

# 2 Une galerie de tableaux

On veut représenter dans un monde 3d des tableaux contenus dans une base de données. Ces tableaux seront placés comme s'ils étaient dans une galerie, c'est à dire dans le cadre de ce travail un "couloir".

## 2.1 Modélisation

La galerie est représentée de façon "simplifiée" par deux murs parallèles de même longeur $L$ et de largeur $l$. Sur chaque mur sont placés $n$ tableaux différents les uns des autres.

**Exercice** proposez une suite d'instructions qui va permettre de créer la galerie de tableaux.

## 2.2 Peuplement automatique d'une galerie

La classe `Musee` définie dans le fichier `serveur/serveur02.py` permet de charger une base qui contient un ensemble de tableaux décrits par les éléments suivants :

- le peintre qui a créé ce tableau
- le nom de ce tableau
- l'année de son achèvement
- sa hauteur (en cm)
- sa largeur (en cm)

Exécutez le serveur implémenté dans le fichier `serveur/serveur02.py`. Il a pour rôle de placer de façon aléatoire des représentations de tableaux spécifiés dans le fichier `base.json`.

**Exercice** Proposez une sous-classe de la classe `Musee` qui permet de visualiser une galerie dédiée à un peintre particulier. Dans cette galerie on ne trouvera que des tableaux de ce peintre ordonnés selon l'ordre chronologique.

# 3 Un musée

En utilisant les primitives 3d disponibles proposez les instructions qui permettent de modéliser un environnement architectural composé de salles carrées identiques (10m*10m*3.5m) placées selon une disposition matricielle (matrice $n * n$). Chaque salle communique avec ses 4 voisines. Placez des tableaux dans ces salles.

**Exercice** proposez une suite d'instructions permettant de créer une telle représentation 3d appliquée à la base `base.json`.

**Exercice** des étiquettes sont associées à chaque oeuvre. Elles sont accessible à travers l'attribut `tags` de la classe `Tableau`. Proposez une procédure permettant de classer les tableaux dans les différentes salles selon les différents attributs (peintre, année d'achèvement, tags, ...). Il faudra que le passage d'une salle à une de ses voisines suive une logique "compréhensible".?

# 4 Annexes

## 4.1 Annexe A : description de primitives 3d

- `sphere(nom,diametre, materiau)`
- `box(nom,dx,dy,dz, materiau)`
- `wall(nom,dx,dy,dz,materiau)`
- `poster(nom,l,h,acces-image)`

## 4.2 Annexe B : les matériaux

- "rouge"
- "vert"
- "bleu"
- "blanc"
- "murBriques"
- "murBleu"
- "parquet"