IEVA - Interface Adaptative

Eric Maisel

Automne 2024

L'objectif de ce travail est d'explorer une base de données rassemblant des oeu- vres impressionnistes en utilisant une métaphore 3d : celle du musée virtuel. Cette exploration correspond dans cette métaphore à une visite adaptée à un visiteur particulier : l'avatar de l'utilisateur.

Le musée virtuel est constitué de N*N salles de 10m*10m organisées selon un schéma matriciel. Chaque salle est repérée par ses indices dans cette matrice. Initialement le visiteur se trouve dans la salle d'indice (0,0). Cette salle contient des oeuvres choisies de façon aléatoire (ou non).

Le visiteur peut se déplacer librement dans l'environnement 3d. A chaque fois qu'il entre dans une salle vide (sans oeuvre d'art) un calcul sera effectué sur un serveur de façon à générer les oeuvres qui seront présentées dans cette salle. Cette génération devra satisfaire les contraintes suivantes :

- Une fois généré, le contenu d'une salle ne peut pas être modifié
- Le niveau 1 contient les sommets qui représentent les étiquettes (ou tags) qui caractérisent les oeuvres
- Les autres niveaux contiennent les concepts les plus abstraits qui structurent l'ensemble des étiquettes

1 Appropriation

1.1 Exécuter le programme

Lancer le serveur :

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

Pour exécuter le client (à partir d'un autre onglet ou d'une autre fenêtre) :

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

1.2 Ajouter des objets à la scène

Actuellement le serveur peut répondre aux types de messages suivants :

- `assets` : renvoie les matériaux disponibles
- `init` : renvoie un décor initial

- `click` : décleché quand on clique sur un objet
- `tictac` : déclenché toutes les secondes (synchrone)
- `salle` : déclenché quand on entre dans une nouvelle salle

```
@app.route('/init')
def init():
  laScene = scene.Scene()
  ...
  return jsonify(laScene.jsonify())

@app.route('/tictac')
def tictac():
  t = request.args.get("Time",default=0,type=float)
  ...
  return jsonify(resultat)

@app.route('/salle/')
def onSalle():
  i = request.args.get("I",default=0,type=int)
  j = request.args.get("J",default=0,type=int)
  print("CHANGEMENTDE SALLE : i=",i," - j=",j)
  ...
  return jsonify(resultat)

@app.route('/click/')
def onClick():
  x = request.args.get('X', default=0,type=float)
  y = request.args.get('Y', default=0,type=float)
  z = request.args.get('Z', default=0,type=float)
  nomObjet = request.args.get('Nom')

  if nomObjet != None :
    print("Objet sélectionné : ",nomObjet)
    print("Point d'intersection : ",x," - ", y ," - ",z)
  ...
  return jsonify(resultat)
```

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 `onSalle` invoquée quand on passe d'une salle à une autre).

Vous travaillerez dans un premier temps dans la fonction `init`. Il faut d'abord créer la scène :

```
import scene

@app.route('/init')
def init():
```

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

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

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

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

On peut associer des composants à cet acteur, à la fois pour l'incarner et le placer. Ici une sphère verte de diamètre 20cm :

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

Programmer le monde revient donc :

- A créer/supprimer des entités
- 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 = laScene.actor("lulu","actor").add(scene.box("lulu",3,1,1,"rouge"))
a.add(scene.position(5,2,3)).add(scene.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.

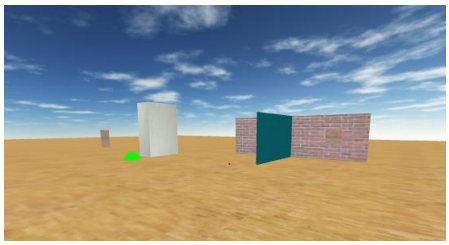
```
laScene.actor("boite01","actor").add(scene.box("boite01",1,1,1,"rouge"))
laScene.getActor("boite01").add(scene.position(2,0.5,3)).add(scene.rotation(0,math.pi/4,0)

# Ajoût de la sphère

laScene.actor("sphere01","actor").add(scene.sphere("sphere01",1,"vert"))
laScene.getActor("sphere01").add(scene.position(0,0.5,0))
laScene.getActor("sphere01").add(scene.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 `scene.poster`) à un mur (composant créé en appelant la fonction `scene.wall`).

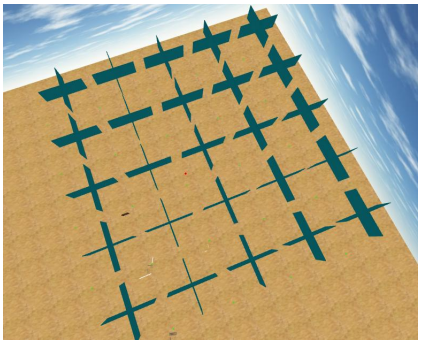
Exercice proposez les instructions qui permettent d'obtenir une scène correspondant à l'image ci-dessous :



2 Génération aléatoire du contenu d'une salle

2.1 Représentation 3d du musée

Le musée est une matrice de 4*4 salles de 10*10m chacune. Chaque salle communique avec les autres salles par des ouvertures situées sur ses cotés nord, sud, est ouest. Comme les cloisons sont partagées entre deux salles il sera



plus simple, pour placer des tableaux au murs de les placer non pas par rapport aux murs mais par rapport au centre de la salle dans laquelle ils se trouvent. Centre matérialisé par une sphère. On pourra alors placer les tableaux en (a,h,b),(b,h,a),(b,h,-a),(a,h,-b),(-a,h,-b),(-b,h,-a),(-b,h,a) et (-a,h,b).

Exercice : modifiez lea fonction `init` du module `serveur` de façon à obtenir un tel musée (n'oubliez pas le toit !!).

```
import scene

...


@app.route('/init')
def init():
  laScene = scene.Scene()

  dx = 10
  dz = 10

  laScene.actor("toit","actor").add(scene.box("toit",50,0.1,50,"blanc"))\
                               .add(scene.position(25,3,25))

  for i in range(0,5) :
    for j in range(0,5):
      x = i*dx
      z = j*dz
      suffixe = str(i)+"-"+str(j)
      nomSalle = "salle-" + suffixe
      laScene.actor(nomSalle,"actor").add(scene.sphere(nomSalle,0.2,"vert"))\
                                      .add(scene.position(?,?,?))
      nomMurH = "H-"+suffixe
      laScene.actor(nomMurH,"actor").add(scene.wall(nomMurH,8,3,0.1,"murBleu"))\
                                     .add(scene.position(?,?,?))
      nomMurV = "V-"+suffixe
      laScene.actor(nomMurV,"actor").add(scene.wall(nomMurV,8,3,0.1,"murBleu"))\
                                     .add(scene.position(?,?,?)\
                                     .add(scene.rotation(?,?,?))
  return jsonify(scene.jsonify())
```

2.2 Représentation du contenu du musée par des données

Le musée expose des tableaux. Les classes nécessaires à leur représentation se trouvent dans le module `musee.js`. Il s'agit des classes `Tableau` et `Musee`.

Un tableau est représenté par son titre (nom), le peintre qui l'a créé, son année de création ainsi qu'une liste de mots-clé.

Un musée est caractérisé par une liste de peintres et un dictionnaire qui associe à une clé une peinture correspondante.

2.3 Calcul du contenu du musée

Les oeuvres d'art utilisées dans la génération d'une salle sont sélectionnées de façon aléatoire.

Exercice modifiez la fonction `init` du module `serveur` de façon à affecter de façon aléatoire des tableaux à des salles et de les y placer.

3 Génération du contenu d'une salle à partir de mots-clés

Proposez une méthode `calculerObjetsDInteret` de la classe `Graphe` qui renvoie la liste des N objets ayant les degré d'intérêt les plus forts connaissant l'intérêt porté par l'utilisateur aux tags qui le décrivent. Pour cela on utilise la relation suivante :

$$I(o) = \sum_{w \in V^+(o)} I(w)$$

Avec :

- o : un objet
- $I(s)$: le degré d'intérêt associé au noeud s
- $V^+(o)$: l'ensemble des mots qui caractérisent l'objet o

Question 1 Pour tester cette procédure créez un graphe sémantique. Pour l'instant un tel graphe a comme arcs (o, w) qui indique que w est un mot-clé qui caractérise l'objet o.

lors de l'entrée dans une nouvelle salle sélectionnez de façon aléatoire un mot-clé. Utilisez le résultat obtenu pour générer le contenu de la salle vide dans laquelle le visiteur vient de rentrer. Vérifiez que les oeuvres retenues correspondent bien au mot-clé sélectionné.

Question 2 Traitez le cas où le nombre d'oeuvres à placer est supérieur à la capacité de la salle courante : elles seront traitées dans la salle suivante.

Question 3 (optionnel) proposez une procédure de sélection des oeuvres probabilistes : les degrés d'intérêt associés à l'ensemble des oeuvres sert à définir leur proba- bilité d'être sélectionnée.

4 Co-construction

Dans les exemples précédents le contenu du musée était défini avant que l'utilisateur ne commence sa visite. On cherche à adapter ce contenu à l'utilisateur. Pour cela on applique un principe de co-construction du musée entre le programme et l'utilisateur. Le contenu n'est défini a priori mais pendant la visite de l'utilisateur en tenant compte des réactions de l'utilisateur aux objets jusque là ajoutés au contenu.

Une interaction avec un objet o dénote l'intérêt du visiteur pour les mots-clé liés à l'objet o (ensemble $V^+(o)$). Cela est pris en compte par la modification des valeurs des intérêts associés aux différents sommets. La valeur du degré d'intérêt des mots-clé associés à o augmente. Comme on cherche à garder constante la quantité totale d'intérêt il faut réduire l'intérêt des autres mots-clé. On appelle C la quantité d'intérêt ajouté aux élements de $V^+(o)$.

$$C = \sum_{w \in W} \tau I(w)$$

et

$$V = \frac{C}{||V^+(o)||}$$

Finalement on peut définir la variation de l'intérêt des mots-clés de la façon suivante :

$$\Delta I(w) = \begin{cases} R - \tau I(w) & \text{si} \quad w \in V^+(o); \\ -\tau I(w) & \text{sinon} \quad . \end{cases}$$

Le code mettant en oeuvre ce calcul est à placer dans la méthode `asynchrone(o)` de la classe `Graphe` avec `o` un sommet du graphe correspondant à une oeuvre. Cette méthode est appelée à chaque fois que le visiteur interagit avec une oeuvre.

Exercice Mettez en oeuvre cette méthode et discutez de l'influence de la valeur du paramètre τ.

L'appel à cette méthode se fait à partir de la fonction `click` du module `serveur`.

5 Amortissement des degrés d'intérêt

L'intérêt que porte un utilisateur à un thème, donc à un tag, finit par s'émousser. Cela se traduit par le fait que les valeurs des degrés d'intérêt tend vers une même valeur : la valeur moyenne des intérêts des tags, notée $\bar{I}$. Une quantité $\tau(I(w) - \bar{I})$ d'intérêt, avec $o \in [0, 1]$, est collectée sur tous les tags dont l'intérêt est supérieur à $\bar{I}$, l'intérêt moyen. Il est redistribué de façon uni- forme à tous les tags dont l'intérêt est inférieur à $\bar{I}$.

$$\begin{cases} \bar{I} & = & \frac{1}{||W||} \sum_{w \in W} I(w) \\ C & = & \sum_{w \in W / I(w) >= \bar{I}} I(w) \\ R & = & \frac{C}{||w / I(w) < \bar{I}||} \\ \Delta I(w) & = & \begin{cases} -\sigma(I(w) - \bar{I}) & \text{si} \quad I(w) >= \bar{I} \\ R & \text{si} \quad I(w) < \bar{I} \end{cases} \end{cases}$$

Ce calcul est mis en oeuvre dans la méthode synchrone de la classe Graphe. Cette méthode est synchrone : elle est activée régulièrement avec une période constante. Concrètement à patir de la fonction `tictac` du module `serveur`.

Exercice Mettez en oeuvre cette méthode et discutez de l'influence de la valeur du paramètre σ.

6 Génération du contenu d'une salle par des taxonomies

Généralisez ces méthodes aux taxonomies ayant plus de 2 niveaux (cf annexe). La génération du contenu se fait en deux temps :

- **Propagation** les degrés d'intérêt sont propagés du niveau des oeuvres au niveau des concepts les plus abstraits (propagation bottom-up) puis de ceux-ci vers le niveau des oeuvres (propagation top-down)

- **Génération** une sélection de T oeuvres (T la capacité de chaque salle) parmi toutes les oeuvres est effectuée comme contenu de la salle vide dans laquelle le visiteur vient de rentrer.

Inspirez vous de l'article de Bonis.

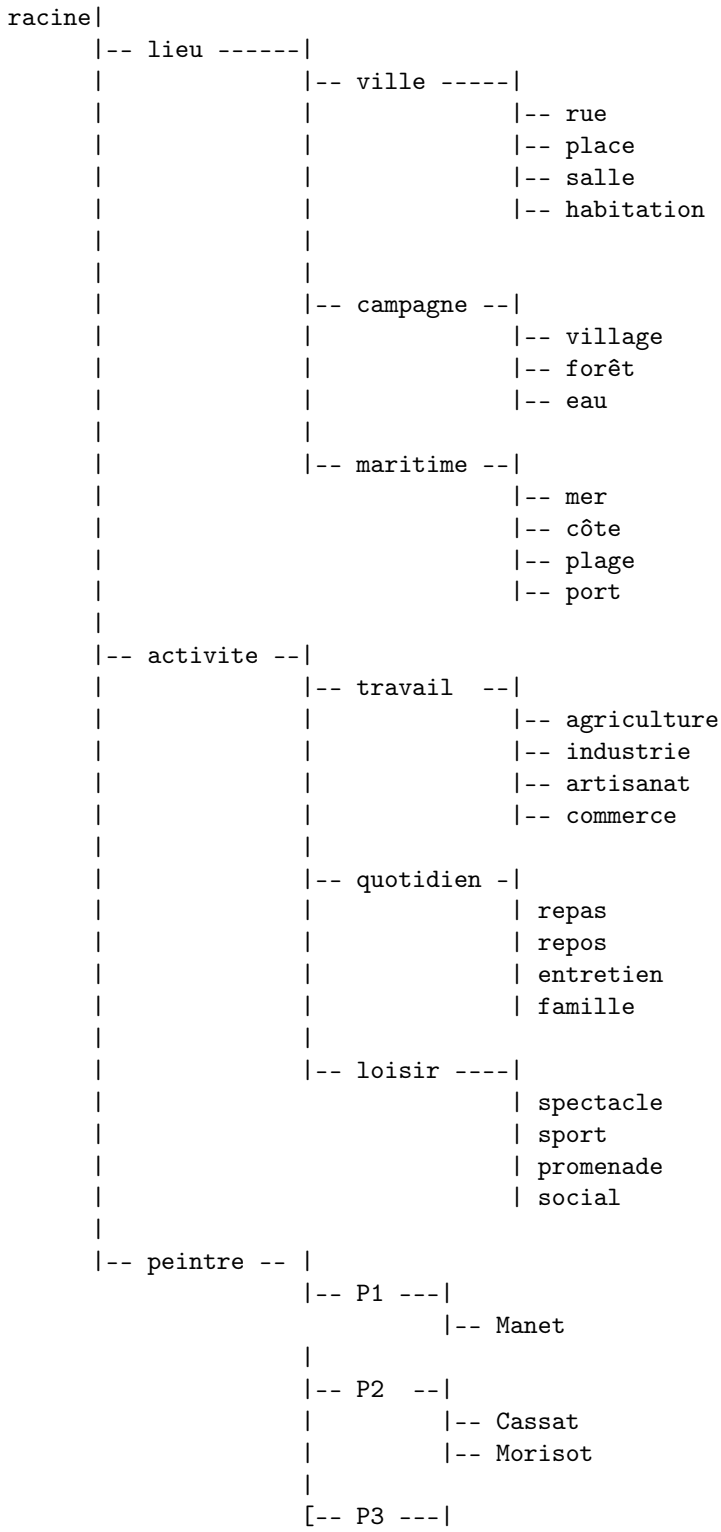
Annexe A : fonctions de création d'objets 3d (module scene.py)

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

7 Annexe B : matériaux disponibles

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

8 Annexe C : taxonomie



```
|          |-- Monet
|          |-- Pissaro
|          |-- Renoir
|          |-- Sisley
|
|-- P4 ---|
|          |-- Bazille
|          |-- Caillebotte
|          |-- Degas
|
|-- P5 ---|
           |-- Cézanne
           |-- Seurat
```