COIN Alix
LEGRAND Léo-Paul
07/12/2021

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Le but de ce TP est d'afficher une fenêtre graphique interactive avec python contenant des objets 3D.

II. Travail Préparatoire

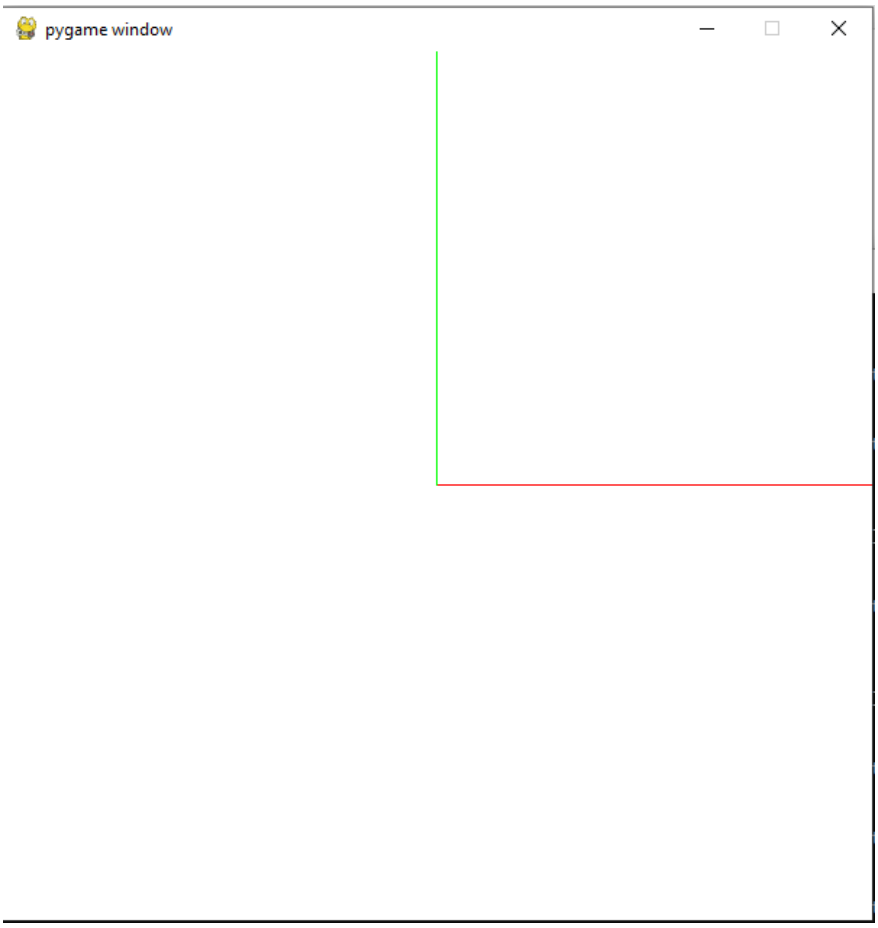
1. Question (1).

On importe la librairie pygame, on initialise le module, on crée une fenêtre de taille (200, 300) et on ferme cette fenêtre et quitte.

2. Question (2).

On fait la même chose que précédemment, mais au lieu de quitter juste après avoir créé la fenêtre, on attend que l'utilisateur appuie sur une touche du clavier pour quitter le programme.

3. Question (3)

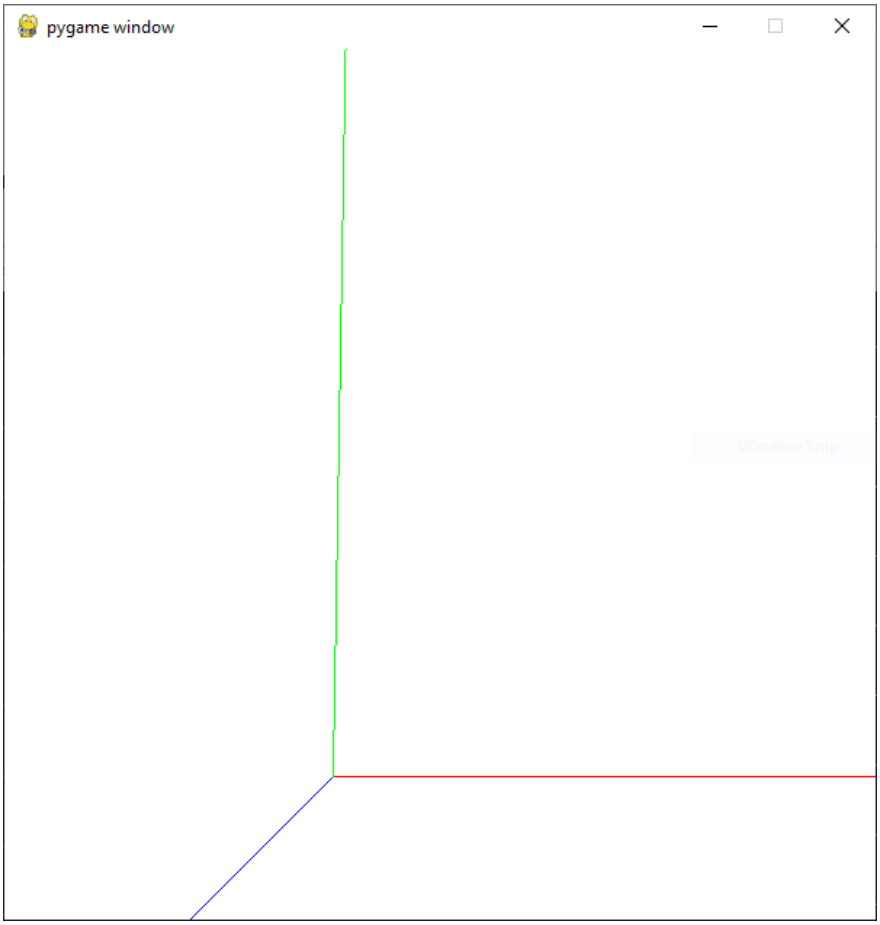


4. Question (4)

En utilisant :

gl.glTranslatef(-0.2, -0.2, 0)

gl.glRotatef(-10, 1, 0, 0)



5- Question (1)

Le fichier main.py contient une multitude de fonctions (*Qx…*) qui génèrent des configurations différentes d'affichage avec plus ou moins d'objets et les affichent. Une *configuration* est un objet qui initialise Pygame et OpenGL selon les paramètres passés dans son constructeur, et auquel on peut ajouter des objets. On obtient la représentation des axes x y z au centre de l'écran.

Le chaînage de *setParameter* et *display* est possible car *setParameter* retourne un objet *configuration*.

Le traitement particulier de *screenPosition* est effectué car sa modification implique de modifier la matrice de projection.

On ajoute *gl.glRotatef(-90, 1, 0, 0)* à la fin de la méthode.

III. Mise en place des interactions avec l'utilisateur avec Pygame

1. Question (1)

Si la touche « Page Up » ou « Page Down » est pressée, on choisit de modifier la matrice ModelView qu'on multiplie par 1,1 ou 1/1,1 grace à la fonction gl.scalef

Pour que le code fonctionne, il est nécessaire d'utiliser les codes des touches « Page Up » et « Page Down » (1073741899 et 1073741902), obtenus grâce à la fonction print(self.event.key)

2. Question (2)

De la même manière, si (IF) la molette est actionnée vers le haut (mousebutton == 4) ou (ELIF) vers le bas (mousebutton == 5)

3. Question (3)

La methode processmouseMotionEvent gère le clic gauche pour la rotation et le droit pour la translation.
La rotation pouvant s'effectuer autour de deux axes, il est nécessaire d'implémenter deux lignes, une pour x et une pour z. On utilise la fonction gl.glRotatef.

Gl.glRotatef(self.event.rel[1]*0.5, 1, 0, 0) -> si la souris est déplacée le long de l'axe y (1), la fonction effectue une rotation de la moitié de la valeur du déplacement autour de l'axe x.

Gl.glRotatef(self.event.rel[0]*0.5, 0, 0, 1) -> si la souris est si la souris est déplacée le long de l'axe x (0), la fonction effectue une rotation de la moitié de la valeur du déplacement autour de l'axe z.

Pour la translation, gl.Translatef permet de translater sur trois axes en même temps : une seule ligne est donc nécesaire :

gl.glTranslatef(self.event.rel[0]*0.1, 0, self.event.rel[1]*0,1) -> si la souris est déplacée le long de l'axe x, la fonction effectue une translation sur x. Si elle est déplacée sur y, la fonction est déplacée sur z.

IV. Création d'une section

1. Question (1)

La méthode generate (self) contient deux fonctions :

- La première (self.vertices) permet de créer une liste de points dont les coordonnées sont définies grâces aux paramètres width, height, thickness. Huit points sont nécessaires à la définition d'un parallélépipède rectangle.
- La seconde (self.faces) créé les faces en associant quatre points dans les sens anti horaire.

2. Question (2)

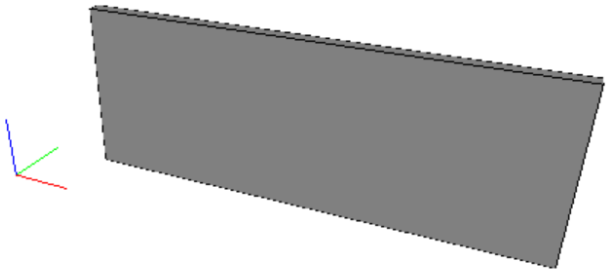
La fonction **Q2b()** permet, grâce à Configuration().add(section), d'initialiser OpenGL et PyGames, avant de créer une instance de la classe section (donc un mur)

L'instruction **Configuration().add(section).display()** permet d'afficher un nouvel objet section grâce à la méthode display, après avoir ouvert et configuré une fenêtre OpenGL.

Pour chaque face, la methode **draw()** trace et colorise la surface d'un quadrilatère puis associe des coordonnées x y z à chaque sommet selon l'index du point.

3. Question (3)

La méthode **drawEdges()** reprends exactement la syntaxe de **draw()** (après la partie modifiant la matrice de position) mais remplace GL_FILL par GL_LINE.



V. Création des murs

1. Question (1)

Le code Wall.py contient la classe Wall, dont le constructeur **__init__** donne des valeurs par défaut aux paramètres non définis et crée une section avec les paramètres définis ou par défauts, section stockée dans une liste d'objets. Ce code contient également une méthode permettant de vérifier si un objet peut être inséré dans le mur (porte, fenêtre…)

La méthode **draw** de la classe Wall emploie la méthode **draw** de l'objet spécifié.

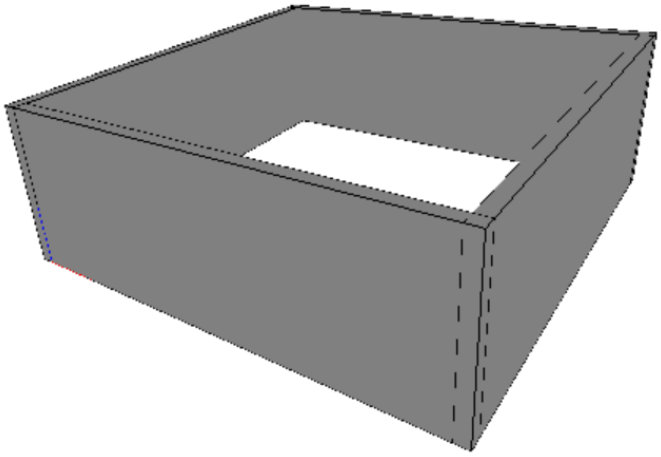
La fonction Q3a() crée un objet Wall selon des paramètres définis et l'ajoute à la configuration.

VI. Création d'une maison

1. Question (1)

La methode **draw()** de la classe House emploie la methode **draw()** de l'objet spécifié.

La fonction Q4a() crée quatre murs et une maison, indique que les murs composent la maison, et ajoute la maison à la configuration.



VII. Création d'ouvertures

1. Question (1)

La classe opening contient la méthode **generate(self)** qui crée une liste de vertex contenant les coordonnées des sommets des volumes entourant les ouvertures, et attribuant ces sommets à chaque face de ces volumes. La méthode **draw(self)** dessine des quadrilatères selon cette liste de sommets.

2. Question (2)

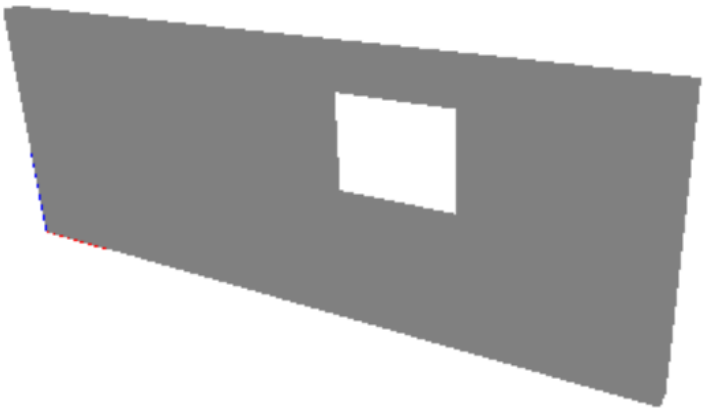
Le dernier affichage vaut false car le haut de l'ouverture est positionné en z=2,7 ce qui dépasse du mur, haut de 2,6. Sa valeur vaut donc *False*.

```
Hello from the pygame community. https://www.pygame.org/contribute.html
True
True
False
```

3. Question (3)

On considère que de manière générale, une ouverture crée quatre sections. La méthode **createNewSection(self, x)** vérifie l'ouverture n'est pas coïncidente avec un coté (ex : porte) et crée une section selon des paramètre dépendant de l'ouverture voulue ou copiés de la section initiale. Les nouvelles sections se superposent, ce qui ne pose pas de problème dans notre cas, mais pourraient en créer si leurs propriétés étaient distinctes (plusieurs couleurs) on ne devaient pas être doublées (exemple simulation de masse avec des propriétés de masse volumique, le volume final ne serait pas bon donc le résultat serait faux).

Q5c1 crée une porte, et Q5c2 une fenêtre.

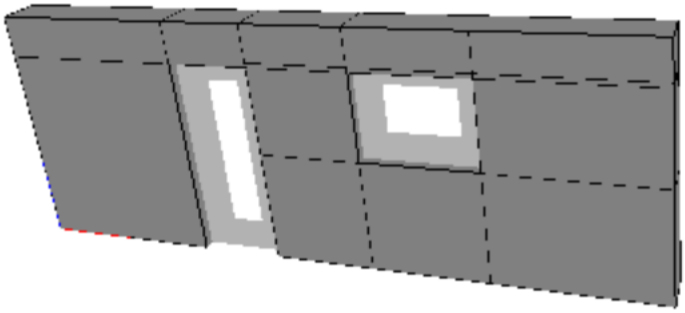


4. Question (4)

Enumerate renvoie une paire contenant l'index dans la liste objects et l'objet associé. L'utiliser permet de simplifier le code en sortant l'index et l'objet de la liste, plutôt que de d'abord utiliser la longueur de la liste pour la boucle for, puis d'aller chercher l'objet dans la liste à la position i.

Ce code renvoie « 0 » et une instance de la classe section.

La methode **add(self, x)** commence par rechercher une section dans laquelle il est possible de créer l'ouverture grâce à **self.findSection(x)**. Si il est possible de créer l'ouverture, la section est supprimée de la liste et remplacée par les nouvelles sections grâce à **createNewSections(x)**. Un nouvel objet (ex : une fenêtre) est ajouté à la liste pour combler l'ouverture.



VIII. Pour finir…

1. Question (1)

Les classes **Door** et **window** ont presque le même fonctionnement que la classe wall par exemple, elles créent des objets de type sections. Par conséquent, il est possible de créer une fenêtre dans une porte (ou une porte dans une fenêtre, ou une porte dans une porte…)