BUSSEUIL Lucas
BLATRIX Alexis
SNI 3 - TD E

TP 3 : Représentation visuelle d'objets

Utilisation de Pygame

1) Ceci permet d'ouvrir une fenêtre noire, et la referme instantanément car aucune instruction n'est donnée ensuite.
2) Ici, la fenêtre reste ouverte. Il suffit de cliquer sur n'importe quelle touche du clavier pour fermer la fenêtre. Cependant on ne peut pas fermer la fenêtre en cliquant sur le bouton en haut à droite.

Utilisation de Pyopengl

Ici, la condition pour fermer la fenêtre est de cliquer sur le bouton croix

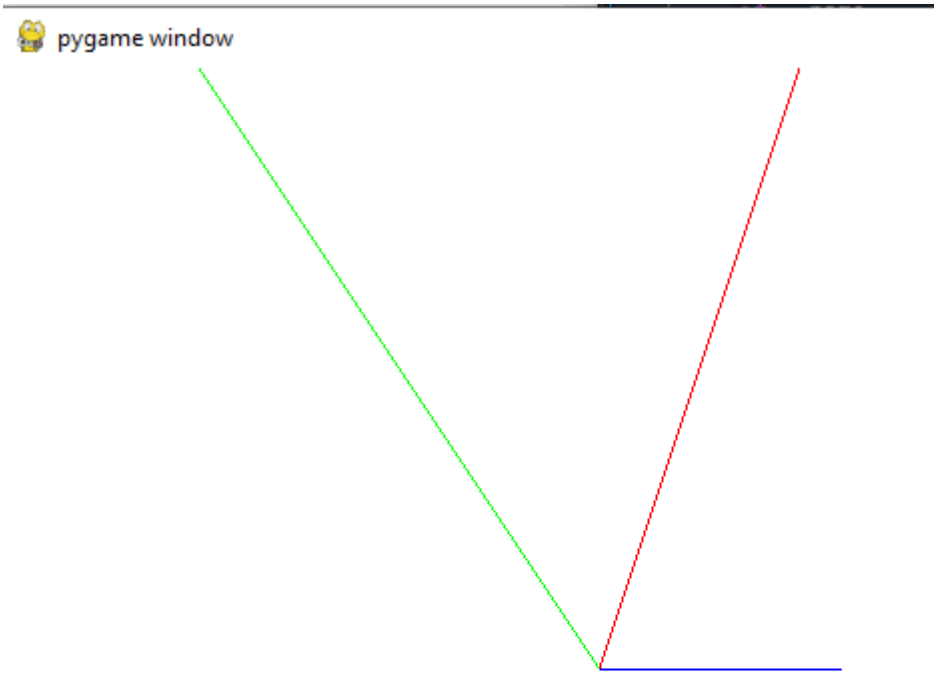
1) Pas d'erreur

```
# Placer ici l'utilisation de gluPerspective.
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
```

2) Fenêtre s'affiche désormais en blanc avec les 3 axes

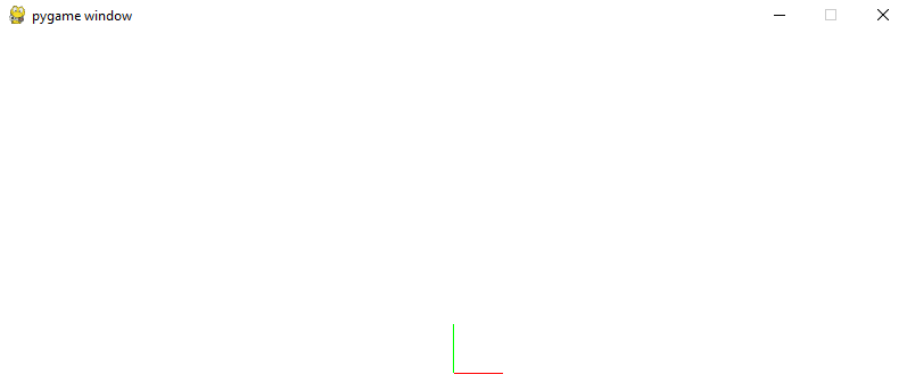
```
    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
    gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((1, 1, -2)) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((-2, 1, 1)) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((1, -2, 1)) # Deuxième vertice : fin de la ligne
    gl.glEnd() # Find du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

3)



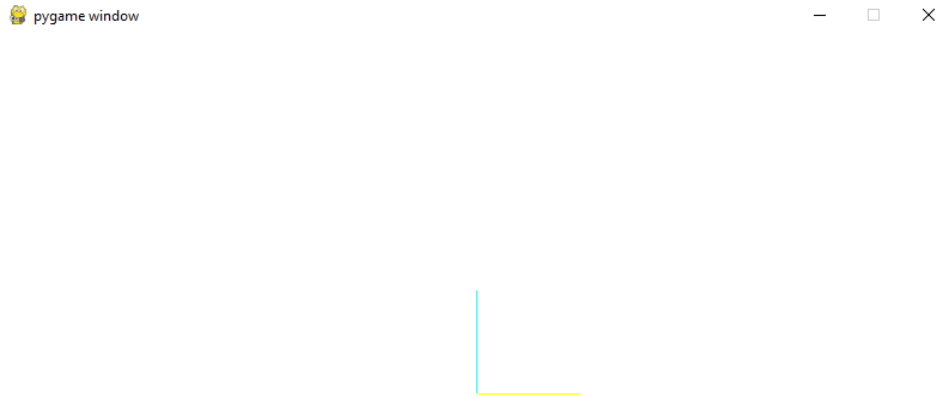
Découverte de l’environnement du travail du TP

1) On remarque après essai que la touche “a” fait disparaître les axes, “z” crée une rotation dans le sens inverse des aiguilles d’une montre, et “maj+z” une rotation dans l’autre sens.
   Configuration.py correspond aux fonctions de création de la fenêtre et de ce qu’elle contient (axes, fonctions de mouvements comme celles utilisées lors de cette question, etc). Main.py va utiliser configuration pour créer la maison objectif du tp.



---

2) Changer les valeurs de Screenposition ou xaxiscolor change la taille de l’axe, ou plutôt sa position, et sa couleur.



L'utilisation des .\ permet d'enchainer les méthodes sans avoir besoin de créer plusieurs fonctions. ScreenPosition n'étant pas une matrice nous n'avons pas la même manière de le modifier.

3) On ajoute dans Initialize Transformation la ligne suivante :

```
gl.glRotate(-90, 1, 0, 0)
```

qui nous permet d'avoir l'axe Z vertical et l'axe Y en profondeur dans notre direction. (L'axe Z est bleu foncé).

Mise en place des interactions avec l'utilisateur Pygame

1) Pour zoomer et dézoomer on utilise les lignes de code suivantes :

```
elif self.event.dict['unicode'] == 'Page Up' or self.event.key == pygame.K_PAGEUP:
    gl.glScale(1.1,1.1,1.1)

elif self.event.dict['unicode'] == 'Page Down' or self.event.key == pygame.K_PAGEDOWN:
    gl.glScale(0.9,0.9,0.9)
```

2) (modifié ensuite)

```
def processMouseButtonDownEvent(self):
    if self.event.button == 4:
        gl.glScale(1.1, 1.1, 1.1)

    elif self.event.button == 5:
        gl.glScale(0.9, 0.9, 0.9)
```

On a constaté qu'après avoir effectué la question suivante, cette méthode ne fonctionnait plus, or, elle fonctionnait de base.On a vu avec le professeur mais nous n'avons pas trouvé le problème.

3)

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0] and not pygame.mouse.get_pressed()[1]:
        gl.glRotate(self.event.rel[0],-1*self.event.rel[1],0,1)


    elif pygame.mouse.get_pressed()[2] and self.event.rel[0]:
        gl.glTranslatef(0.01*self.event.rel[0],0,-0.01*self.event.rel[1])
```

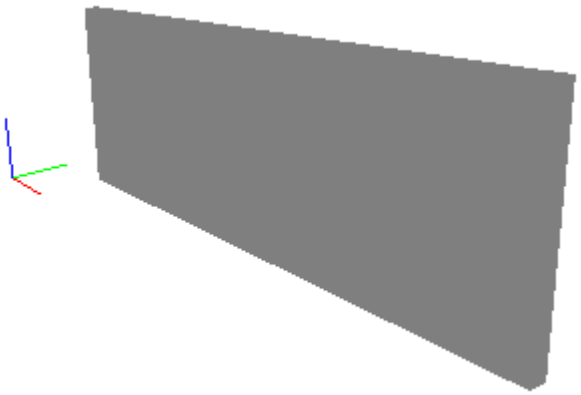
Création d'une section

2)a)

```
def generate(self):
    self.vertices = [
            [0, 0, 0 ],
            [0, 0, self.parameters['height']],
            [self.parameters['width'], 0, self.parameters['height']],
            [self.parameters['width'], 0, 0],
            [0,self.parameters['thickness'] , 0 ],
            [0, self.parameters['thickness'], self.parameters['height']],
            [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
            [self.parameters['width'], self.parameters['thickness'], 0],
            ]
    self.faces = [
            [0, 3, 2, 1],
            [4,7,6,5],
            [2,3,5,6],
            [0,1,4,5],
            [0,3,4,7],
            [1,2,6,7],
            ]
```

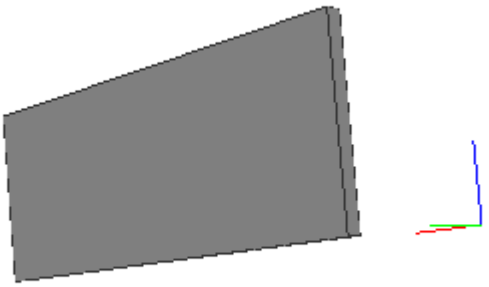
2)b)

L'instruction de la question 2b crée une section d'origine [0,0,0], de longueur 7 et de hauteur 2,6. Ci dessous une représentation de nos 6 sections.



2c)

Ci dessous, une représentation de nos 6 sections avec une couleur différente pour les arêtes.



Création des murs

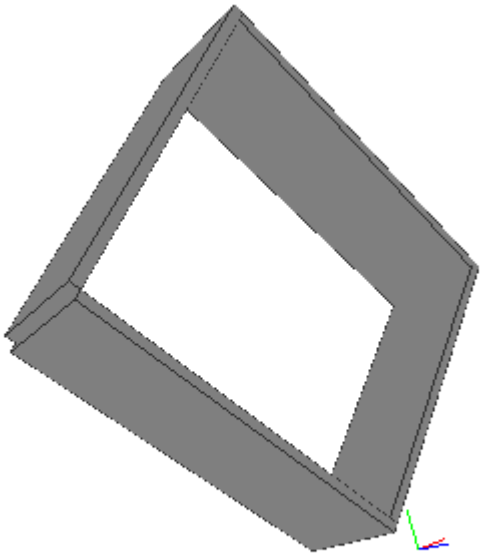
3a) Voici l'instruction qui permet d'afficher un mur ci-dessous. Dans la fonction Wall, on a constaté qu'un mur peut être constitué de plusieurs objets donc c'est une liste. On a fait une boucle qui permet d'afficher chaque objet de la liste.

```
def Q3a():
    Mur = Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'edges': True})
    return Configuration().add(Mur)
```

Création d'une maison

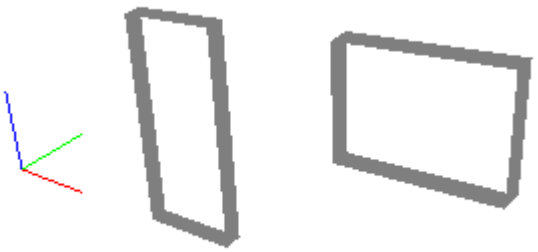
4)a) Ci-dessous le code permettant de tracer les 4 murs de la maison. Ainsi que la représentation actuelle de nos 4 murs.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':7, 'height':2.6, 'edges': True, 'orientation':0})
    wall2 = Wall({'position': [7, 0, 0], 'width':7, 'height':2.6, 'edges': True,'orientation':90})
    wall3 = Wall({'position': [0, 0, 0], 'width':7, 'height':2.6, 'edges': True,'orientation':90})
    wall4 = Wall({'position': [0, 7, 0], 'width':7, 'height':2.6, 'edges': True,'orientation':0})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```



Création d'ouvertures

5a) Nous avons repris la même formule que pour dessiner les sections, sauf qu'ici nous n'avons que 4 sections et nous ne prenons pas en compte l'orientation. Voici ci dessous une représentation de 2 sections (1 fenêtre et 1 porte)



5b)
Pour cette question, nous devons créer une fonction permettant de savoir si l'ouverture entrée en paramètre rentre dans la section. Nous avons donc vérifié si la position de l'ouverture + la longueur/largeur et hauteur étaient comprises dans notre section. On remarque bien que la troisième ouverture dépasse comme annoncé dans l'énoncé

```
    def canCreateOpening(self, x):
        if self.parameters['width'] >= x.parameters['position'][0]+x.parameters['width'] :
            if self.parameters['height'] >= x.parameters['position'][2]+x.parameters['height'] :
                if self.parameters['thickness'] >= x.parameters['position'][1]+x.parameters['thickness']  :
                    return True
        return False
```

```
In [18]: runfile('C:/Users/ASUS/Desktop/TP3/tp3-representation-visuelle-d-objets-
tp3_blatrix_busseuil-main/src/Main.py', wdir='C:/Users/ASUS/Desktop/TP3/tp3-representation-
visuelle-d-objets-tp3_blatrix_busseuil-main/src')
True
True
False
```

Pour conclure, ce TP était compliqué mais intéressant, le fait de coder quelque chose de physique est plus parlant pour progresser et comprendre nos erreurs. Par manque de temps nous n'avons pu finir entièrement ce TP, mais je suppose que nous avons compris les notions principales.
Ce que nous avons pu faire a dans l'ensemble bien marché.