GAGUA Oussama

YOVODEVI Zaide

Date : 13/12/2021

TP3 – Représentation visuelle d'objets

I-Présentation du TP :

II-Préparation à faire avant le TP :

II.*1-Utillisation de Pygame :*

❖ Question 1. Explication des lignes de codes :

```
1  import pygame
2  pygame.init()
3  ecran = pygame.display.set_mode((300, 200))
4  pygame.quit()
```

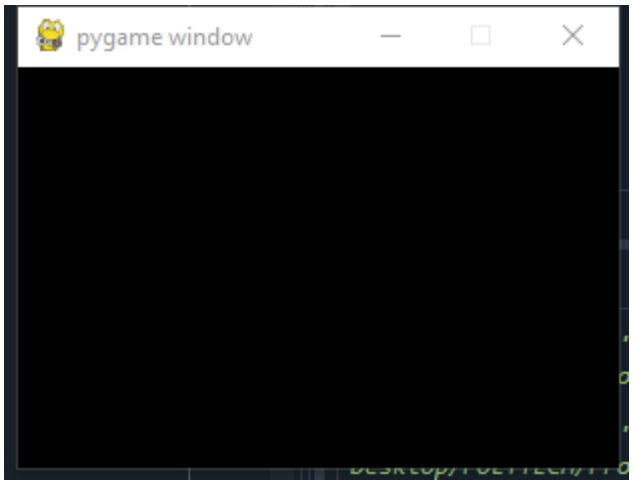
La **1ère ligne** permet d'importer le package pygame

La **2ème ligne** permet d'initialiser pygame pour son utilisation

La **3ème ligne** permet de régler le mode d'affichage de largeur 300 et de hauteur 200

La **4ème ligne** est envoyée lorsque l'utilisateur clique sur le bouton « X » de la fenêtre ou lorsque le système « demande » que le processus s'arrête. S'il est ignoré, il peut toujours être tué par le système. Il permet quand même d'économiser, avant de quitter.

Exécution :



La fenêtre ci-dessus apparaît et se referme en quelques secondes

❖ 2.Test de code et explication :

```
1   import pygame
2
3
4   pygame.init()
5   ecran = pygame.display.set_mode((300, 200))
6
7   continuer = True
8   while continuer:
9       for event in pygame.event.get():
10          if event.type == pygame.KEYDOWN:
11              continuer = False
12
13  pygame.quit()
```

La **1ère ligne** permet d'importer le package pygame.

La **4ème ligne** permet d'initialiser pygame pour son utilisation.

La **5ème ligne** permet de régler le mode d'affichage de largeur 300 et de hauteur 200.

La **7ème ligne** crée la variable continuer qui est initialisée à True(Vrai).

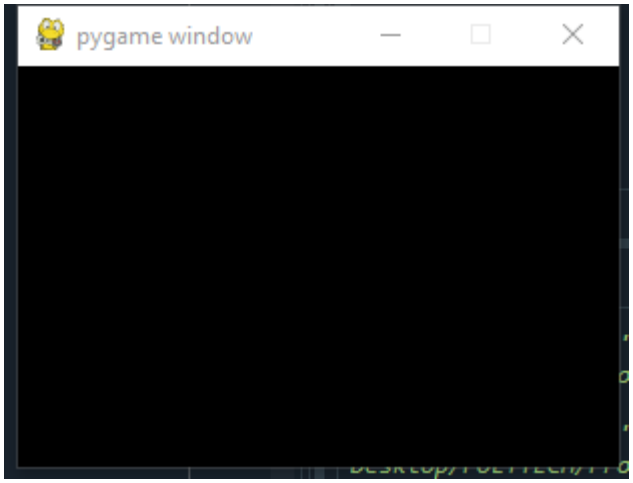
La **8ème ligne** crée une boucle while qui s'exécute tant que continuer est True.

La **9ème ligne** crée une boucle for qui parcourt l'envoie de la liste des évènements par la fonction pygame.event.get ().

La **10ème ligne** vérifie si l'utilisateur appuie sur une touche du clavier dans la liste des évènements

La **11ème ligne** renvoie la valeur False pour la variable continuer suivant la condition de la 10e ligne bouclée dans un for each.

La **13ème ligne** est envoyée lorsque l'utilisateur clique sur le bouton « X » de la fenêtre ou lorsque le système « demande » que le processus s'arrête. S'il est ignoré, il peut toujours être tué par le système. Il permet quand même d'économiser, avant de quitter.



La fenêtre apparaît sans interruption, nous sommes désormais capables de réduire la fenêtre et de la fermer au moment voulu.

II.2-Utillisation de Pyopengl pour représenter des objets 3D :

❖ Question 1. Initialisation de la matrice perspective :

Après avoir copier/coller le code, nous avons initialisé la matrice perspective au niveau du commentaire l'indiquant :

```
# Placer ici l'utilisation de gluPerspective.
glu.gluPerspective(45,1,0.1,50)
```

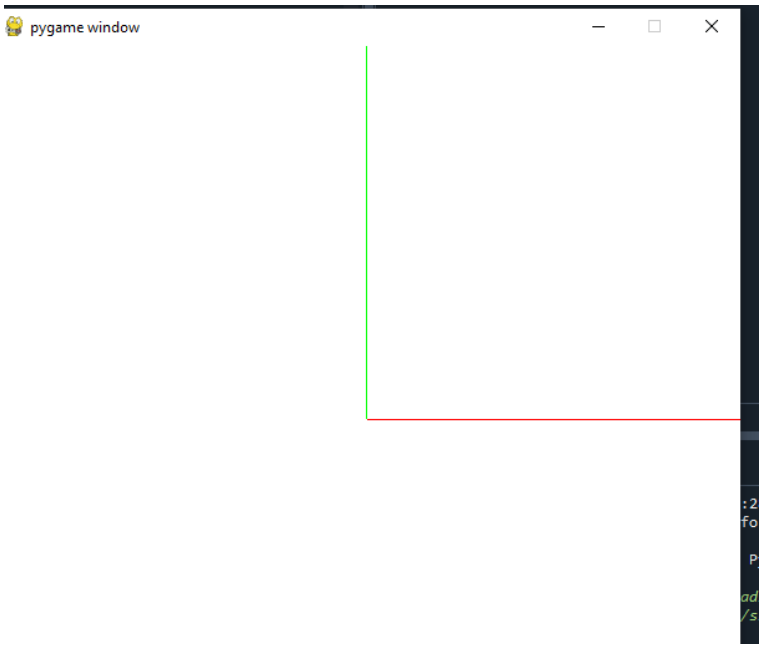
Le code est fonctionnel, nous avons pu obtenir une ligne

❖ Question 2. Traçage des axes :

Pour tracer les axes x(rouge), y(vert) et z(bleu) on procède de la manière suivante :

```python
from sys import exit
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

if __name__ == '__main__':

    pygame.init()
    display=(600,600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
    # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)

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


    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)

    gl.glColor3fv([1, 0, 0]) # Indique la couleur du prochian segment x en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((1, 0, -2)) # DeuxiÃ¨me vertice : fin de la ligne

    gl.glColor3fv([0, 1, 0]) # Indique la couleur du prochian segment y en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((0, 1, -2)) # DeuxiÃ¨me vertice : fin de la ligne

    gl.glColor3fv([0, 0, 1]) # Indique la couleur du prochian segment x en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((0, 0, -1)) # DeuxiÃ¨me vertice : fin de la ligne


    gl.glEnd() # Fin du tracÃ©
    pygame.display.flip() # Met Ã  jour l'affichage de la fenÃªtre graphique

    continuer=True

    while continuer:
        for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    continuer=False
                    pygame.quit()
                    exit()
```

Afin d'être en phase par rapport aux exigences (perspectives, projection et clipping) de la présentation du TP, nous obtenons le résultat ci-dessous :



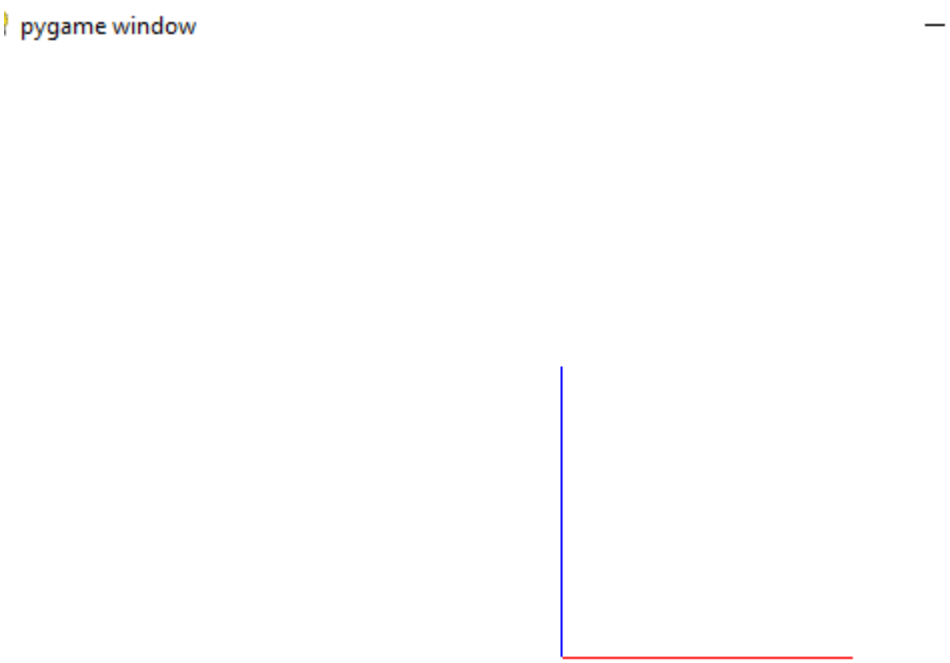
On devait s’attendre à ce résultat car l’axe z est en face de nous

❖ Question 3. Transformations de translation et de rotation :

Afin de déplacer la position de l'écran, on a utilisé les fonctions glTranslatef () et glRotatef (). Nous avons également utilisé la fonction glu Perspective ()

```
from sys import exit
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

if __name__ == '__main__':

    pygame.init()
    display=(600,600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
    # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)

    # Placer ici l'utilisation de gluPerspective.
    glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
    gl.glTranslatef(0.0, 2, -5)
    gl.glRotatef(-90, 1, 0, 0)


    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)

    gl.glColor3fv([1, 0, 0]) # Indique la couleur du prochian segment x en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((1, 0, -2)) # DeuxiÃ¨me vertice : fin de la ligne

    gl.glColor3fv([0, 1, 0]) # Indique la couleur du prochian segment y en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((0, 1, -2)) # DeuxiÃ¨me vertice : fin de la ligne

    gl.glColor3fv([0, 0, 1]) # Indique la couleur du prochian segment x en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : dÃ©part de la ligne
    gl.glVertex3fv((0, 0, -1)) # DeuxiÃ¨me vertice : fin de la ligne


    gl.glEnd() # Fin du tracÃ©
    pygame.display.flip() # Met Ã  jour l'affichage de la fenÃªtre graphique

    continuer=True

    while continuer:
        for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    continuer=False
                    pygame.quit()
                    exit()
```

Après l'exécution du code, nous avons obtenu le résultat suivant :



II.3-Découverte de l'environnement du travail du TP :

❖ Question 1-a : Ajout de la fonction Q1a ()

En ajoutant le code return configuration (), on obtient le graphe ci-dessous :

En utilisant la touche a l'image disparait.

En utilisant la touche z ou Z, on peut effectuer des rotations de l'image dans le sens trigonométrique.

- Analyse du fichier main.py :

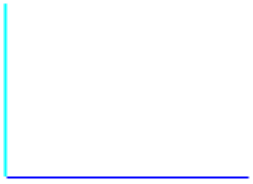
Le fichier main.py est constitué de méthodes se chargeant de l'exécution et de la vérification des questions du TP (Par exemple def Q2a () . Il y'a aussi la méthode def main () qui réunit toutes ses méthodes d'exécution de questions. Elle nous permet d'activer ou de désactiver celles-ci.

- Analyse du fichier configuration.py :

Le fichier Configuration.py contient un constructeur qui initialise la couleur des axes, la position de l'écran et des axes. Elle contient une liste d'objets et d'une option d'initialisation de Pygame.

❖ *Question 1-b : paramètres transmis au constructeur d'un objet de la classe Configuration*

```
def Q1b_f():
    return Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 1]}). \
        setParameter('xAxisColor', [0, 0, 1]). \
        setParameter('yAxisColor', [0,1,1]). \
        display()
```



- Analyse de l'effet de la modification :

La couleur a changé. Cependant, la position de l'écran pouvait être changée pour mieux apprécier le résultat.

- Chainage de méthodes entre SetParameter () et display () :

Le chaînage de méthodes est la pratique des méthodes objet renvoyant l'objet lui-même afin que le résultat soit appelé pour une autre méthode. Le chainage entre setParameter et display est donc possible car setParameter renvoie l'objet par la ligne « retrurn self ».

❖ *Question 1-c : instruction à la méthode initializeTransformationMatrix ()*

→On a pu modifier les axes en faisant une rotation de 90 degrés autour de l'axe x

```
# Initializes the transformation matrix
def initializeTransformationMatrix(self):
    gl.glMatrixMode(gl.GL_PROJECTION)
    gl.glLoadIdentity()
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)

    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glLoadIdentity()
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
    gl.glRotatef(90,-2,0,0)
```

III-Mise en place des interactions avec l'utilisateur avec pygame :

❖ *Question 1-d : Zoom avant et arrière en modifiant la méthode processKeyDownEvent() :*

On a cherché les codes des touches page up et page down sur la documentation de pygame afin de gérer la structure conditionnelle.

```
# Processes the KEYDOWN event
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)

    #Zoom +
    elif self.event.key == pygame.K_PAGEUP:
        gl.glScaled(1.1,1.1,1.1)
    #Zoom -
    elif self.event.key == pygame.K_PAGEDOWN:
        gl.glScaled(1/1.1,1/1.1,1/1.1)
```

→Le code a marché sans problèmes

❖ *Question 1-d : Zoom avant et arrière avec la molette de la souris :*

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.button==4:
        gl.glScaled(1.1,1.1,1.1)
    elif self.event.button==5:
        gl.glScaled(1/1.1,1/1.1,1/1.1)
```

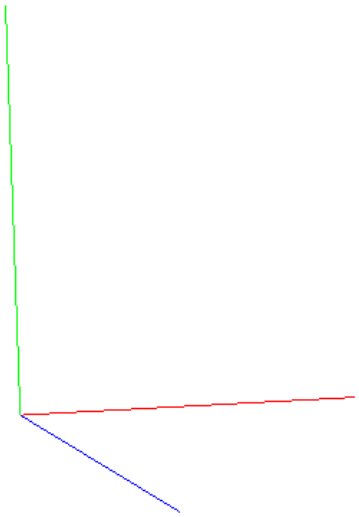
❖ *Question 1-f Déplacement des objets à partir de la souris:*

-On a pu réaliser le changement d'échelle par l'utilisation des méthodes de rotation et de translation définies au début

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0]==1:
        gl.glRotate(self.event.rel[0],1,0,0)
        gl.glRotate(self.event.rel[1],0,0,1)
    elif pygame.mouse.get_pressed()[2]==1:
        gl.glTranslated(self.event.rel[0]/20,0,0)
        gl.glTranslated(0,0,self.event.rel[1]/20)
```

→Afin de diminuer la sensibilité et de mieux voir les résultats, on a divisé les positions de la souris par 20.

→L'exécution du code donne le résultat suivant :



☺☺☺

IV-Création d'une section

❖ *Question 2-a Création d'une section en définissant les faces et les sommets :*

➔ Dans notre travail de fond, on a dessiné un parallélépipède comme celui de la figure 4 puis on a donné à chaque sommet un numéro et, en fonction des dimensions de la figure, on a pu définir tous les sommets.

➔ Sur les faces, on a regroupé les 4 sommets qui forment chaque face.

```python
    # Defines the vertices and faces
    def generate(self):
        self.vertices = [
                # Définir ici les sommets
                [0, 0, 0 ], #Sommet0
                [0, 0, self.parameters['height']],#1
                [self.parameters['width'], 0, self.parameters['height']], #2
                [self.parameters['width'], 0, 0], #3
                [0,self.parameters['thickness'] ,0], #4
                [0,self.parameters['thickness'] ,self.parameters['height']], #5
                [self.parameters['width'],self.parameters['thickness'], 0 ], #6
                [self.parameters['width'],self.parameters['thickness'],self.parameters['height']]#7
                    ]
        self.faces = [
                # définir ici les faces
                [0,3,2,1],
                [6,4,5,7],
                [4,0,1,5],
                [3,6,7,2],
                [0,3,6,4],
                [1,2,7,5]]
```

❖ *Question 2-b Création de la section :*

- **Analyse de Q2b () :**

La méthode Q2b () permet d'ajouter une section à la classe Configuration.

- **Ecriture de la méthode draw () :**

```python
    # Draws the faces
    def draw(self):
        # A compléter en remplaçant pass par votre code

        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
        #1ere face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        #2emeface
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        gl.glVertex3fv( [self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        #3eme face
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        #4eme face
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        #5eme face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        #6eme face
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([0,self.parameters['thickness'] ,self.parameters['height']])

        gl.glEnd()
```

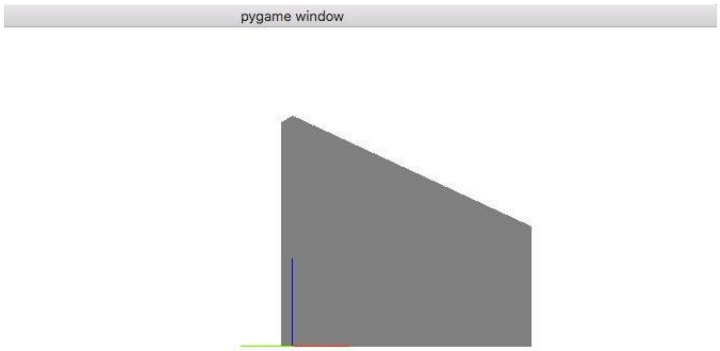
LSP Python: ready conda: intro_programmation (Python 3.9.4)

Remarque :

On a fait un assemblage des faces de façon à construire une section

- **Execution de Q2b ()**

Après l'exécution du code, nous avons obtenu le résultat suivant :



☺☺☺

❖ *Question 2-c Création des arêtes :*

- **Ecriture de la méthode drawEdges () dans la classe Section :**

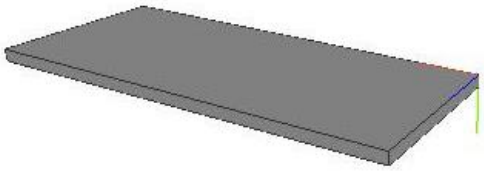
```
    # Draws the edges
    def drawEdges(self):
        # A compléter en remplaçant pass par votre code
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.2, 0.2, 0.2]) # Couleur plus foncée
        #1ere face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        #2emeface
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        gl.glVertex3fv( [self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        #3eme face
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        #4eme face
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        #5eme face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        #6eme face
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([0,self.parameters['thickness'] ,self.parameters['height']])

        gl.glEnd()


LSP Python: ready    conda: intro_programmation (Python 3.9.4)
```

- **Modification de la méthode draw () de la question (2). b)** :

Pour pouvoir afficher les arêtes avant la section, on a juste ajouté la ligne suivante au début de la méthode draw() pour que la fonction drawEdges() s'exécute avant draw() : self.drawedges()

- **Exécution de la fonction Q2c ()** :

Après l'exécution du code, nous avons obtenu le résultat suivant :



☺☺☺

V-Création des murs

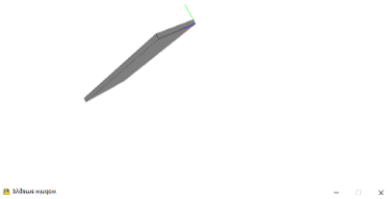
❖ *Question 3-a Création des murs :*

- **Analyse du fichier Wall.py :**

Le fichier Wall.py initialise la position et la couleur du mur et les dimensions du mur. Il y'a également une liste nulles d'objets et une section parentes prédéfinies par le constructeur

- **Écriture dans la méthode draw de la classe Wall :**

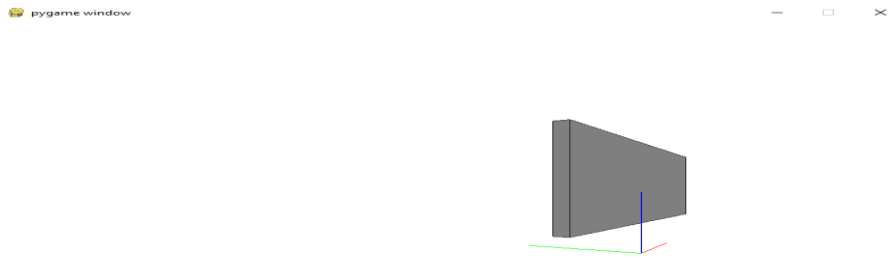
➔ On a parcouru la liste des objets pour tracer le mur



```python
# Adds an object
def add(self, x):
    # A compléter en remplaçant pass par votre code
    self.objects.append(x)
    return self

# Draws the faces
def draw(self):
     for x in self.objects:
         x.draw()
```

➔ Pour pouvoir faire la rotation, on a modifié la fonction draw de la classe Section grâce à l'ajout des fonctions glRotate et glTranslate

```python
# Draws the faces
def draw(self):
        gl.glPushMatrix()

        gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
        gl.glRotate(self.parameters['orientation'],0,0,1)


        # A compléter en remplaçant pass par votre code
        self.drawEdges()

        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
        #1ere face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        #2emeface
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        gl.glVertex3fv( [self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        #3eme face
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
        #4eme face
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        #5eme face
        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
        gl.glVertex3fv([0,self.parameters['thickness'] ,0])
        #6eme face
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
        gl.glVertex3fv([0,self.parameters['thickness'] ,self.parameters['height']])

        gl.glEnd()


        gl.glPopMatrix()
```



☺☺☺

- **Écriture dans la fonction Q3a du fichier Main.py :**

```
def Q3a():
    WALL1=Wall({'position': [1, 1, 0], 'width':7 , 'height':2.6 , 'orientation':90 ,'thickness':0.2 ,'color':[0.7,0.7,0.7]
    return Configuration().add(WALL1)
```

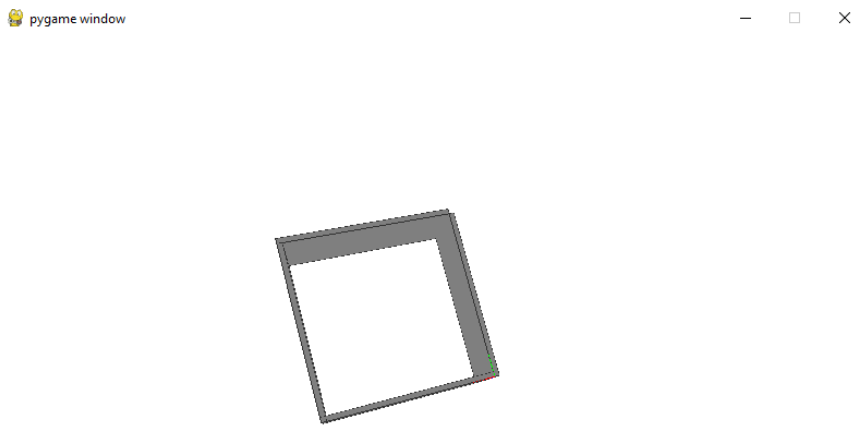
VI-Création d'une maison

- **Ecriture de la méthode draw () de la classe House :**

➔ On a parcouru la liste des objets pour tracer la maison

```
    # Adds an object
    def add(self, x):
        self.objects.append(x)
        return self

    # Draws the house
    def draw(self):
        # A compléter en remplaçant pass par votre code
        # Draws the objects if any
        for x in self.objects:
            x.draw()
```

Voilà notre maison !



☺☺☺

- **Modifier Q4a () dans le fichier Main.py pour créer une maison :**

On a défini les 4 murs qui forment/ constituent la maison :

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':7 , 'height':2.6 , 'orientation':0 ,'thickness':0.2 ,'color':[0.7,0.7,0.7
    wall2 = Wall({'position': [7,0,0], 'width':7 , 'height':2.6 , 'orientation':90 ,'thickness':0.2 ,'color':[0.7,0.7,0.7]
    wall3 = Wall({'position': [0,7,0], 'width':7 , 'height':2.6 , 'orientation':0 ,'thickness':0.2 ,'color':[0.7,0.7,0.7]}
    wall4 = Wall({'position': [0,0,0 ], 'width':7 , 'height':2.6 , 'orientation':90,'thickness':0.2 ,'color':[0.7,0.7,0.7]
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

VII-Création d'ouvertures :

Pour créer l'ouverture, on a dû ajouter un paramètre 'orientation' dans le constructeur

```
class Opening:
    # Constructor
    def __init__(self, parameters = {}) :
        # Parameters
        # position: mandatory
        # width: mandatory
        # height: mandatory
        # thickness: mandatory
        # color: mandatory

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'position' not in self.parameters:
            raise Exception('Parameter "position" required.')
        if 'width' not in self.parameters:
            raise Exception('Parameter "width" required.')
        if 'height' not in self.parameters:
            raise Exception('Parameter "height" required.')
        if 'thickness' not in self.parameters:
            raise Exception('Parameter "thickness" required.')
        if 'color' not in self.parameters:
            raise Exception('Parameter "color" required.')
        if 'orientation' not in self.parameters:
                self.parameters['orientation'] = 0
        # Generates the opening from parameters
        self.generate()
```

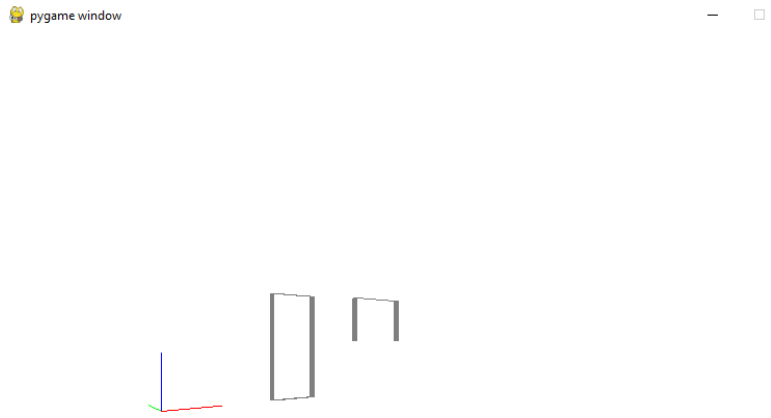
La méthode generate a été modifié de la manière suivante : on a éliminé les 2 faces qui se font faces pour bien définir une ouverture.

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [
            # Définir ici les sommets
                [0, 0, 0 ], #Sommet0
                [0, 0, self.parameters['height']],#1
                [self.parameters['width'], 0, self.parameters['height']], #2
                [self.parameters['width'], 0, 0], #3
                [0,self.parameters['thickness'] ,0], #4
                [0,self.parameters['thickness'] ,self.parameters['height']], #5
                [self.parameters['width'],self.parameters['thickness'], 0 ], #6
                [self.parameters['width'],self.parameters['thickness'],self.parameters['height']]#7
            ]
    self.faces = [
            # définir ici les faces
             #[0,3,2,1],
             #[6,4,5,7],
             [4,0,1,5],
             [3,6,7,2],
             [0,3,6,4],
             [1,2,7,5]
            ]
```

La méthode draw a été écrite de la manière suivante :

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()


    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)


    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    #3eme face
    gl.glVertex3fv([0,self.parameters['thickness'] ,0])
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv( [0,self.parameters['thickness'] ,self.parameters['height']])
    #4eme face
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
    gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    #5eme face
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'], 0 ])
    gl.glVertex3fv([0,self.parameters['thickness'] ,0])
    #6eme face
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'],self.parameters['thickness'],self.parameters['height']])
    gl.glVertex3fv([0,self.parameters['thickness'] ,self.parameters['height']])

    gl.glEnd()

    gl.glPopMatrix()
```

On obtient ainsi le résultat suivant :



☺☺☺

- **Question 5b :**

```
    # Checks if the opening can be created for the object x
    def canCreateOpening(self, x):
        # A compléter en remplaçant pass par votre code
        if x.parameters['thickness'] < self.parameters['thickness'] and \
           x.parameters['height'] < self.parameters['height'] and \
           x.parameters['width'] < self.parameters['width']:
            return True
        return False


    # Creates the new sections for the object x
    def createNewSections(self, x):
        # A compléter en remplaçant pass par votre code
        pass

    # Draws the edges
    def drawEdges(self):
        # gl.glPushMatrix()
```

```
In [2]: runfile('/Volumes/EMTEC C410/tp3-representation-visuelle-d-objets-gagua_yovodevi_tp3-main/src/Main.py',
wdir='/Volumes/EMTEC C410/tp3-representation-visuelle-d-objets-gagua_yovodevi_tp3-main/src')
pygame 2.0.0 (SDL 2.0.12, python 3.9.4)
Hello from the pygame community. https://www.pygame.org/contribute.html
False
False
False
```

On a fait une comparaison entre les dimensions de notre section et celles de l'ouverture qu'on veut construire.

→Après l'exécution du code, le résultat est « False ».

☺☺☺