HERVE Grégoire
CASTEUR Axel
14/01/2022

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Lors de la réalisation de ce TP nous nous intéressons à la représentation d'objets 3D à l'écran dans une fenêtre graphique. On aura pour objectif la représentation de plusieurs maisons à partir d'objets simples que l'on va construire progressivement comme les murs, les portes, les fenêtres. Tout cela sera possible en utilisant le module Pygame et le module PyOpenGL.

II. Préparation avant TP

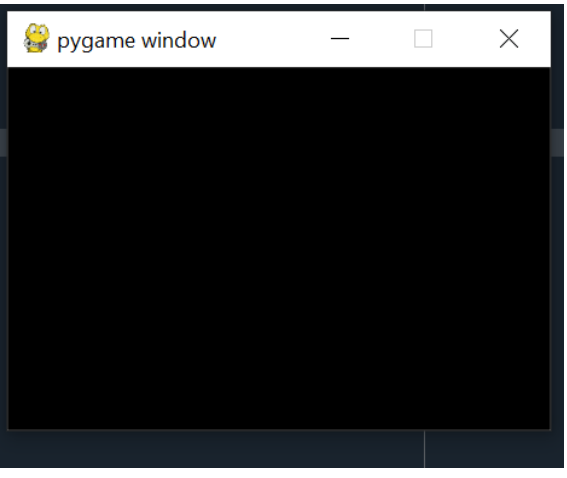
1. Question (1).

```
import pygame
pygame.init()
ecran = pygame.display.set_mode((300,200))
pygame.quit()
```

Dans un premier temps, nous importons la libraire pygame. Ensuite nous initialisons pygame avant de paramétrer l'affichage à 300 de large et 200 de hauteur. Pour finir on quitte l'affichage. On obtient donc une fenêtre qui s'ouvre et se ferme directement.

2. Question (2).

```
import pygame

pygame.init()
ecran = pygame.display.set_mode((300,200))

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



Ici on réalise la même chose que précédemment mais avant de refermer la fenêtre nous codons le fait qu'il faut un appui sur le clavier. Autrement dit, la fenêtre ne se ferme pas tant que nous n'appuyons pas sur le clavier.

3. Question (3).

Voici, le code :

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

On a pas d'erreur.

4. Question (4).

On obtient le code suivant :

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

gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -1)) # Premier vertice : départ de la ligne
gl.glVertex3fv((-1, -1, 0)) # Deuxième vertice : fin de la ligne

gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -1)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 0, 0)) # Deuxième vertice : fin de la ligne

gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -1)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 1, 0)) # Deuxième vertice : fin de la ligne
```

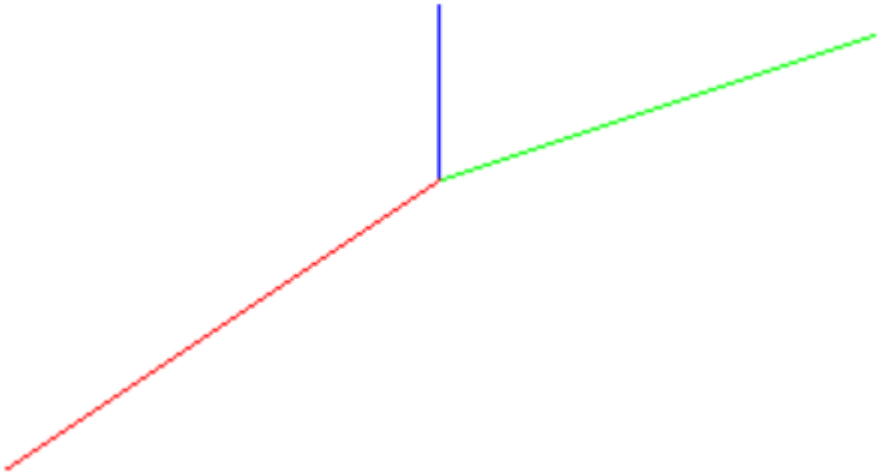
Et l'affichage suivant :

5. Question (5).

On écrit le code suivant :

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
gl.glTranslatef(0.0, 2, -5)
```

On obtient l'affichage suivant :



Ensuite on effectue une rotation à l'aide de la fonction **glRotatef**, pour faire une rotation de $90°$ Autour de l'axe ***X***. Afin de pouvoir observer le tracé de l'axe ***Z*** effectué plus tôt.

On produit le code suivant :

```
# 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)
```

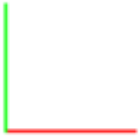
On obtient l'affiche ci-dessous :

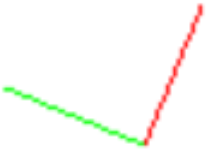
6. Question (6).

a)

La touche a fait disparaître l'objet 3D. Quant à La touche z, elle permet de faire une rotation de droite à gauche dans le sens trigonométrique. Exemple :

Puis avec appuie sur z :





Maintenant, quand on appuie sur les touches z+maj cela fait une rotation dans le sens horaire :

Voici le code qui permet de réaliser cela dans configuration.py :

```
# 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_z):
        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)

    # Draws or suppresses the reference frame
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)
```

Concernant les touches z+maj : on a défini que lorsqu'on appuie sur la touche z+maj cela est relié à la fonction glRotate qui fait donc une rotation autour de l'axe des x de 2.5° par rapport au sens horaire.

Maintenant concernant la touche z : on a défini que lorsqu'on appuie sur la touche z cela est relié à la fonction glRotate qui fait une rotation autour de l'axe des x de 2.5° par rapport au sens trigonométrique. Soit -2.5° par rapport au sens horaire pour pouvoir comparé aux touches z+maj.

Pour la touche a : on a défini que lorsqu'on appuie sur la touche a cela renvoie l'inverse du self.parameters. Cela signifie que lorsqu'on appuie et qu'il n'y a rien cela fait apparaitre les axes et inversement, s'il y a les axes et qu'on appuie sur a cela les fait disparaitre.

b) On écrit le code suivant :

```
def Q1a():
    return Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

On remarque bien que le trait rouge de l'axe des x change de couleur et passe en jaune :

On écrit le code suivant :

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

On obtient bien des changements de couleur des axes x et y :

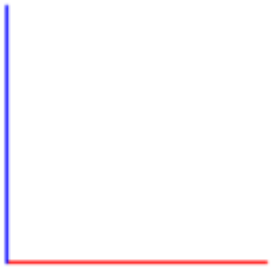
- Le chaînage de l'appel des méthodes setParameter() et display() est possible car elles retournent self.

- Après test sur machine de z = -5 et z = -2, nous remarquons que plus z est petit dans les négatifs, plus la représentation est grande (zoomée). Ce qui explique pourquoi srcreenPosition est fixé.

c) On ajoute l'instruction ce qui donne :

```
# Initializes the tranformation 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(-90,1,0,0)
```

On obtient en sachant qu'on a repris le code couleur de base:
donc z en bleu et x en rouge :



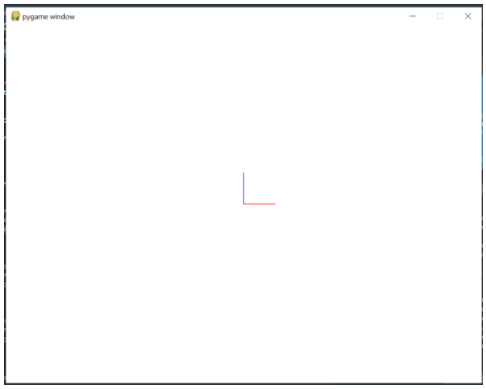
III. Mise en place des interactions avec l'utilisateur avec Pygame

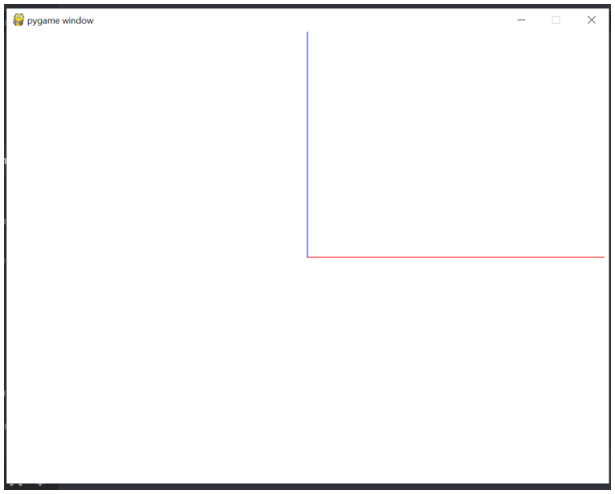
7. Question (7).

On obtient :

```
# Zoom in/out
elif self.event.dict['unicode'] == 'PAGEUP' or self.event.key == pygame.K_PAGEUP:
    gl.glScalef(1.1, 1.1, 1.1)
elif self.event.dict['unicode'] == 'PAGEDOWN' or self.event.key == pygame.K_PAGEDOWN:
    gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Ce qui donne : Dézoomé : zoomé :





8. Question (8).

On remplace l'instruction pass dans la méthode **processMouseButtonDownEvent()** pour gérer l'effet de zoom avec la souris ce qui donne :

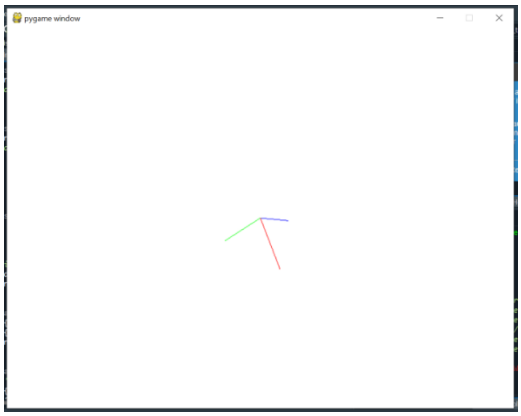
```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.button == 4 :
        gl.glScalef(1.1, 1.1, 1.1)
    elif self.event.button == 5 :
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

9. Question (9).

Voici notre code :

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[2]==1:
        gl.glTranslatef(self.event.rel[0]/10, 0, - self.event.rel[1]/10)

    elif pygame.mouse.get_pressed()[0]==1:
        gl.glRotatef(self.event.rel[1], 0, 1, 0)
        gl.glRotatef(self.event.rel[0], 1, 0, 0)
```

Pour la rotation :



Pour la translation :

Après appuie sur le bouton gauche on obtient bien une rotation de même si l'on appuie sur le bouton droit on obtient bien une translation.

IV. Création d'une section

11. Question (11).

En utilisant le fichier Configuration.py, on écrit la méthode generate(self) de la classe Section qui crée les sommets et les faces d'une section orientée selon l'axe x et dont le coin bas gauche de la face externe est en (0, 0, 0). Nous avons défini les coordonnées des sommets en utilisant les paramètres width, height et thickness.

On obtient le code suivant :

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [                   ##Définition des sommets
            [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 = [                      ##Définition des faces
            [0, 3, 2, 1],
            [0, 1, 5, 4],
            [0, 4, 7, 3],
            [3, 2, 6, 7],
            [4, 7, 6, 5],
            [1, 5, 6, 2]
            ]
    return self
```

12. Question (12).

Analyse fonction Q2b() : elle ajoute un objet de type Section à la configuration :

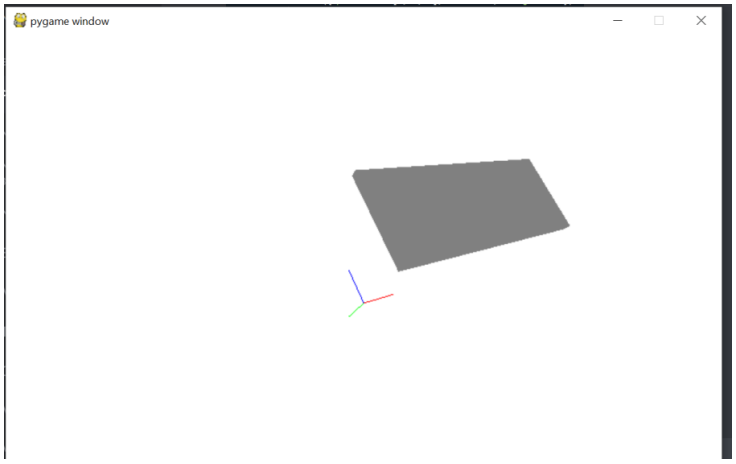
```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
            Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
            )
```

Configuration().add(section).display() permet de renvoyer le repère ainsi que la section ajouté (la fonction add permettant l'addition donc ici l'ajout d'une deuxième fonction). De plus que dans le cas où le code de la section ne marche pas ou code n'est pas encore crée, le repère sera quand même est situé en ce point.

Voici le code le code :

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

    gl.glPushMatrix()
    gl.glTranslatef(0, -2, 0)
    for face in self.faces:
        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
        gl.glVertex3fv(self.vertices[face[0]])
        gl.glVertex3fv(self.vertices[face[1]])
        gl.glVertex3fv(self.vertices[face[2]])
        gl.glVertex3fv(self.vertices[face[3]])
        gl.glEnd()
    gl.glPopMatrix()
```

On obtient visuellement :



13. Question (13).

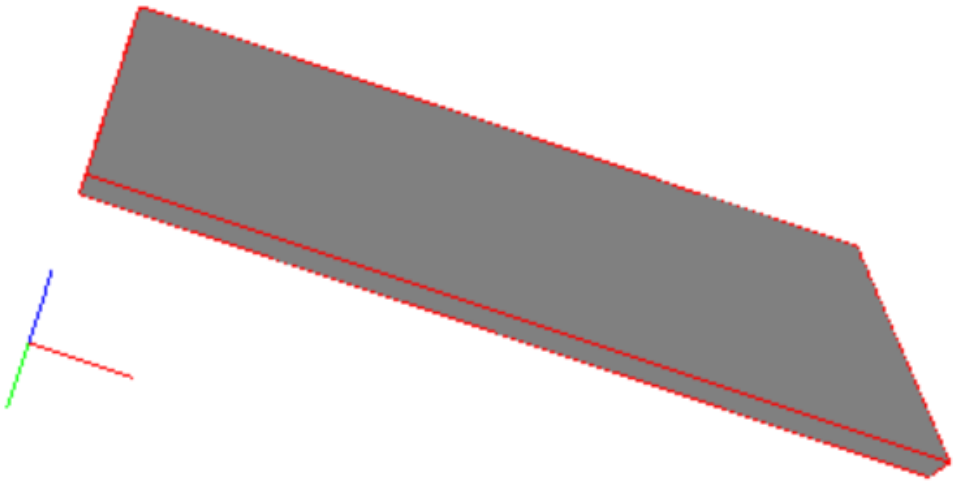
Nous avons construit notre programme de la même manière que pour draw :

```
# Draws the edges
def drawEdges(self):
    gl.glPushMatrix()
    gl.glTranslatef(0, -2, 0)
    for face in self.faces:
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les arrêtes : GL_LINE
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([4, 4*0.0, 4*0.0]) # Couleur rouge
        gl.glVertex3fv(self.vertices[face[0]])
        gl.glVertex3fv(self.vertices[face[1]])
        gl.glVertex3fv(self.vertices[face[2]])
        gl.glVertex3fv(self.vertices[face[3]])
        gl.glEnd()
    gl.glPopMatrix()
```

Modification de Draw :

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    if self.parameters['edges']==True:
        self.drawEdges()
    gl.glPushMatrix()
    gl.glTranslatef(0, -2, 0)
    for face in self.faces:
        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
        gl.glVertex3fv(self.vertices[face[0]])
        gl.glVertex3fv(self.vertices[face[1]])
        gl.glVertex3fv(self.vertices[face[2]])
        gl.glVertex3fv(self.vertices[face[3]])
        gl.glEnd()
    gl.glPopMatrix()
```

On obtient visuellement avec les arrêtes en rouges pour les mettre en évidence. :



Et au niveau des tests d'erreurs :

```
----------------------------------------------------------
---------------
Ran 9 tests in 0.017s

OK
```

V. Création des murs

14. Question (14).

On analyse le code suivant :

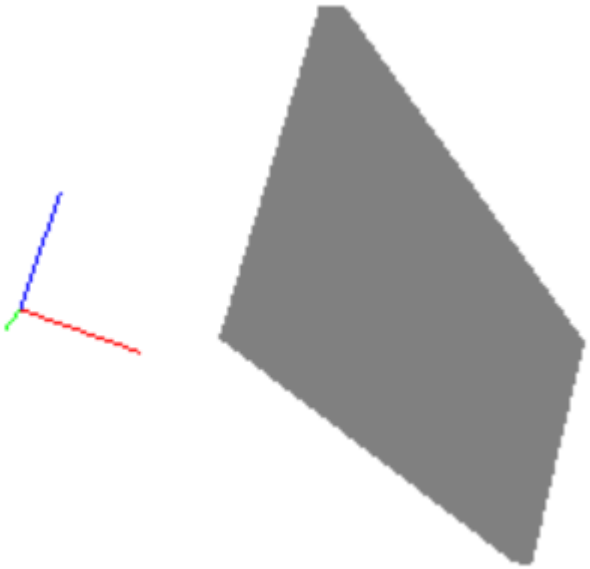
```
# Constructor
def __init__(self, parameters = {}) :
    # Parameters
    # position: position of the wall
    # width: width of the wall - mandatory
    # height: height of the wall - mandatory
    # thickness: thickness of the wall
    # color: color of the wall

    # Sets the parameters
    self.parameters = parameters

    # Sets the default parameters
    if 'position' not in self.parameters:
        self.parameters['position'] = [0, 0, 0]
    if 'width' not in self.parameters:
        raise Exception('Parameter "width" required.')
    if 'height' not in self.parameters:
        raise Exception('Parameter "height" required.')
    if 'orientation' not in self.parameters:
        self.parameters['orientation'] = 0
    if 'thickness' not in self.parameters:
        self.parameters['thickness'] = 0.2
    if 'color' not in self.parameters:
        self.parameters['color'] = [0.5, 0.5, 0.5]

    # Objects list
    self.objects = []

    # Adds a Section for this object
    self.parentSection = Section({'width': self.parameters['width'], \
                                  'height': self.parameters['height'], \
                                  'thickness': self.parameters['thickness'], \
                                  'color': self.parameters['color'],
                                  'position': self.parameters['position']})
    self.objects.append(self.parentSection)
```

Analyse de Wall.py : le codage est fait de telle sorte que des exceptions sont prérentrées pour faire bugger le programme si des paramètres ne sont pas rentrés. De plus on remarque que Wall.py crée une section automatiquement, la section parente.

On écrit ensuite dans la méthode draw de la classe Wall :

```
# Draws the faces
def draw(self):
    gl.glPushMatrix()
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for i in self.objects:
        i.draw()
    gl.glPopMatrix()
    print(Wall,self.parameters)
```

On obtient :



Nous avons obtenu :

```
----------------------------------------------------------------------
---------------
Ran 3 tests in 0.005s

OK
```

VI. Création d'une maison

15. Question (15).

Après avoir géré la création de murs, nous allons construire une maison en créant 4 instances de murs. Celle-si sera gérée par la classe House du fichier House.py. Pour cela nous allons nous aider de la méthode Draw de la classe Configuration qui permet l'affichage de plusieurs objets. Donc après écriture de la méthode draw de la classe House nous allons modifier Q4a dans le fichier Main.py (nous allons devoir transformer notre section initiale en plusieurs sections). La modification de Q4a a pour but la création de 4 murs que l'on vérifie bien positionner entre eux. Puis on créait un objet house qui s'appuie sur la classe House qui va permettre de regrouper les 4 murs en un élément house. Voici le code :

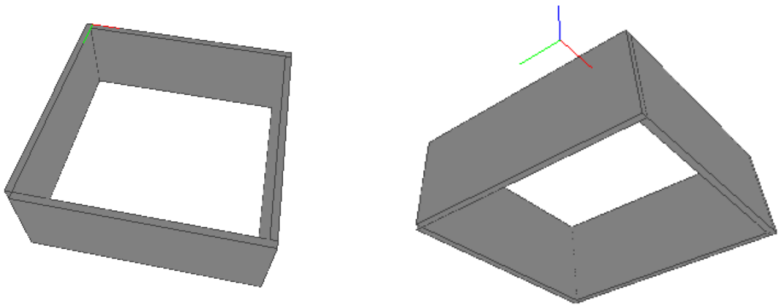
```python
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position':[0,0,0], 'width':7, 'height':2.6, 'edges':True})
    wall2 = Wall({'position':[0,0,0], 'width':7, 'height':2.6, 'edges':True, 'orientation':90})
    wall3 = Wall({'position':[0,7,0], 'width':7, 'height':2.6,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

    return Configuration().add(house)
```

Voici un exemple d'une maison carrée que nous avons décalé par rapport au repère. En effet pour décaler la maison nous avons produit le code suivant :

```python
# Draws the house
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)
    for i in self.objects:
        i.draw()
    gl.glPopMatrix()
```

```python
house = House({'position':[3,0,0]})
```

La maison carrée :



Après lancement de test_house, on a :

```
---------------
Ran 3 tests in 0.006s

OK
```

VII. Création d'ouvertures

Notre objectif est maintenant de créer des ouvertures dans un mur pour pouvoir mettre des portes et des fenêtres. Pour créer une ouverture dans un mur, nous allons transformer la section initiale en plusieurs sections.

17. Question (17).

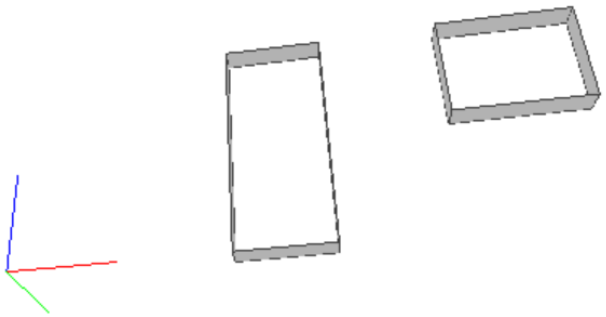
Nous allons nous intéresser à la représentation graphique du contour de l'ouverture, constitué de 4 faces. On sait que la position de l'ouverture est définie par la coordonnée du coin inférieur gauche par rapport au repère de la section. Nous avons complété la classe Opening, qui est définie dans le fichier Opening.py, et exécuté le fichier Main.py avec la fonction Q5a() afin de produire un affichage. Voici notre code :

```
# Defines the vertices and faces
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],
        [self.parameters['width'], self.parameters['thickness'],0],
        [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
        [0, self.parameters['thickness'], self.parameters['height']],
        [0, self.parameters['thickness'], 0]
        ]

    self.faces = [
        [1,2,5,6],
        [7,4,3,0],
        [0,7,6,1],
        [2,3,4,5]
        ]
```

```python
def drawEdges(self):
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    for face in self.faces:
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les arrêtes : GL_LINE
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.5*0.5, 0.5*0.5, 0.5*0.5]) # Couleur gris moyen
        gl.glVertex3fv(self.vertices[face[0]])
        gl.glVertex3fv(self.vertices[face[1]])
        gl.glVertex3fv(self.vertices[face[2]])
        gl.glVertex3fv(self.vertices[face[3]])
        gl.glEnd()
    gl.glPopMatrix()

# Draws the faces
def draw(self):
    self.drawEdges()
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],
    self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) #on trace les faces : GL:FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) #Tracé d'un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for j in i :
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

```python
def Q5a():
    # Ecriture avec mélange de variable et de chaînage
    opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    return Configuration().add(opening1).add(opening2)
```

Ce qui donne :



18. Question (18).

Pour cette question nous avons modifié la classe Section en commençant par ajouter la méthode canCreateOpening(self, x) où x est un objet. Cette méthode retourne *True* si une ouverture, représentée par **x**, peut être ajoutée dans la section, représentée par self, et *False* sinon. Ce test s'appuie sur la comparaison des informations fournies par les paramètres de position, de largeur, de hauteur de l'ouverture avec celles fournies par les paramètres de la section.

Codage de canCreateOpening :

```python
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    if x.parameters['width'] + x.parameters['position'][0] <= self.parameters['position'][0]+self.parameters['width']:
        if x.parameters['height'] + x.parameters['position'][2] <= self.parameters['position'][2] + self.parameters['height']:
            return True
    return False
```

On vérifie le bon fonctionnement :

```
In [29]: runfile('C:/Users/Gregoire/Downloads/tp3-representation-visuelle-d-objets-eit_tp3_herve_casteur-main/tp3-representation-visuelle-d-objets-eit_tp3_herve_casteur-main/src/Main.py', wdir='C:/Users/Gregoire/Downloads/tp3-representation-visuelle-d-objets-eit_tp3_herve_casteur-main/tp3-representation-visuelle-d-objets-eit_tp3_herve_casteur-main/src')
Reloaded modules: Configuration, Section, Wall, Window, Opening, Door, House, Main
True
True
False
```

On obtient False ce qui est normal pour le troisième test car l'ouverture donnée dépasse la section le long de l'axe z.

19. Question (19).

On a écrit la méthode **createNewSections(self, x)** de la classe **Section** où **x** est un objet, porte ou une fenêtre, qui retourne la liste des sections engendrées par la création de l'ouverture. Voici le code :

```
    # Creates the new sections for the object x
    def createNewSections(self, x):
        # A compléter en remplaçant pass par votre code
        if self.canCreateOpening(x) == True:
            section1 = Section({'position':self.parameters['position'],
                    'width':x.parameters['position'][0],
                    'height':self.parameters['height'],
                    'thickness': self.parameters['thickness']})

            section2 = Section({'position':[x.parameters['position'][0],x.parameters['position'][1],x.parameters['position'][2]+x.parameters['height']],
                    'width':x.parameters['width'],
                    'height':self.parameters['height']-x.parameters['position'][2]-x.parameters['height'],
                    'thickness': self.parameters['thickness']})

            section3 = Section({'position':[x.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2]],
                    'width':x.parameters['width'],
                    'height':x.parameters['position'][2],
                    'thickness': x.parameters['thickness']})

            section4 = Section({'position':[x.parameters['position'][0]+x.parameters['width'],x.parameters['position'][1],self.parameters['position'][2]],
                    'width':self.parameters['width']-x.parameters['position'][0]-x.parameters['width'],
                    'height':self.parameters['height'],
                    'thickness': x.parameters['thickness']})

            return [section1,section2,section3,section4]
```

On vérifie le bon fonctionnement en lançant Q5c1() et Q5c2() :

```
def Q5c1():
    section = Section({'width':7, 'height':2.6})
    opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createNewSections(opening1)
    configuration = Configuration()
    for x in sections:
        configuration.add(x)
    return configuration

def Q5c2():
    section = Section({'width':7, 'height':2.6})
    opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createNewSections(opening2)
    configuration = Configuration()
    for section in sections:
        configuration.add(section)
    return configuration
```

On obtient donc côté droit :

On obtient donc côté gauche :

20. Question (20).

La fonction enumerate() associe un nombre à chaque objet dans enumerate. S'il n'est pas spécifié, le compteur par défaut est 0. Cela créera un tuple contenant le compteur et l'objet.

La section [0] vaut 0 et section [1] vaut 1.

On écrit la méthode add(self, x) :

```
# Adds an object
def add(self, x):
    # A compléter en remplaçant pass par votre code
    findsection = self.findSection(x)
    self.objects.pop(findsection[0])
    self.objects.extend(findsection[1].createNewSections(x))
    return self
```

On exécute :

Dans un premier temps nous avions fait :

```
def Q5d():
    wall1 = Wall({'position': [0, 0, 0],'width': 6.9,'height': 2.6,'orientation': 0,'edges': True})
    opening1 = Opening({'position': [2, 0, 0],'width': 0.9,'height': 2.15,'thickness': 0.2,'color': [0.7, 0.7, 0.7]})
    opening2 = Opening({'position': [4, 0, 1.2],'width': 1.25,'height': 1,'thickness': 0.2,'color': [0.7, 0.7, 0.7]})
    wall1.add(opening1)
    wall1.add(opening2)
    configuration=Configuration()
    configuration.add(wall1)
    return configuration
```

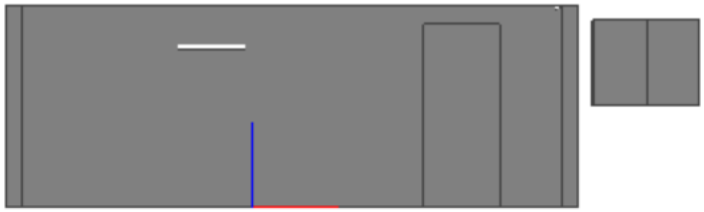
On n'obtenait pas la bonne chose, il nous manquait la fenêtre. On s'est rendu compte que le programme faisait soit la porte soit la fenêtre si les wall.add étaient écrits sur deux lignes. Donc on les a écrits sur une ligne et on a obtenu :

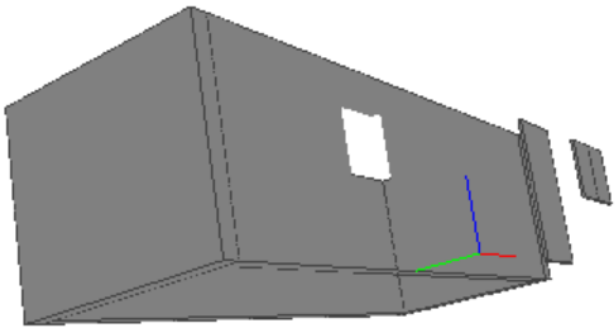
```
def Q5d():
    wall1 = Wall({'position': [0, 0, 0],'width': 6.9,'height': 2.6,'orientation': 0,'edges': True})
    opening1 = Opening({'position': [2, 0, 0],'width': 0.9,'height': 2.15,'thickness': 0.2,'color': [0.7, 0.7, 0.7]})
    opening2 = Opening({'position': [4, 0, 1.2],'width': 1.25,'height': 1,'thickness': 0.2,'color': [0.7, 0.7, 0.7]})
    wall1.add(opening2).add(opening1)
    configuration=Configuration()
    configuration.add(wall1)
    return configuration
```

VIII. Pour finir

On a essayé d'écrire dans la fonction Q6() du fichier main.py des instructions permettant d'obtenir la maison suivante mais nous avons un problème d'affichage et par manque de temps nous n'avons pas pu finir.

Nous obtenions :





Nous avons réussi à réaliser la porte et la fenêtre mais celle-ci n'est pas parfaite :

IX. Conclusion

Lors de ce TP nous avons abordé de nouvelles notions tel que le codage sur Pygame et PyOpenG…. Il nous a aussi permis d'en approfondir d'autres comme le codage python déjà utilisé précédemment. A titre personnel, nous avons pensé que ce TP était très compliqué et demandait beaucoup de temps. Mais grâce au résultat de ce TP qui nous motivait, nous avons réussi à nous surpasser et arriver à la dernière question après de longues heures de travail. Nous avons réussi finalement à représenter une partie d'une maison sur Python ce qui nous a satisfait.