NICOLAS Thomas
GUERIOT Benjamin

Rapport de TP3

Représentation visuelle d'objets

Sommaire

Introduction 2
I/ Travail préparatoire 2
Utilisation de Pygame 2
1) 2
2) 2
Utilisation de Pyopengl pour représenter des objets 3D 3
1) 3
2) 4
3) 5
Découverte de l'environnement du travail du TP 5
(1). a) 6
(1). b) 6
(1). c) 7
II/ Mise en place des interactions avec l'utilisateur avec Pygame 8
(1). d) 8
(1). e) 9
(1). f) 9
III/ Création d'une section 10
(2). a) 10
2). b) 12
(2). c) 13
IV/ Création des murs 15
(3). a) 15
V/ Création d'une maison 17
(4). a) 17
VI / Création d'ouvertures 18
(5). a) 18
(5). b) 20
(5). c) 21
(5). d) 24
VII/ Pour finir... 26
(6) 26
(7). Pour aller plus loin... 27
Conclusion 28

NICOLAS Thomas
GUERIOT Benjamin

Introduction

A travers ce TP, nous allons apprendre à manipuler des objets 3D en utilisant notamment la librairie **Pygame** ainsi que **OpenGL**. Ce TP est à réaliser en 2 séances.

Le but de ce TP est de représenter une maison à partir de différentes classes qui interagiront entre elles.

Le module Pygame permet, entre autres, de construire des jeux en *Python*. Pour traiter des jeux, le programme doit gérer des événements comme l'appui sur des boutons. Pygame sera utilisé afin de gérer l'affichage d'une fenêtre graphique et des interactions entre celle-ci et l'utilisateur.

Le module PyOpenGL permet d'accéder en *Python* aux très nombreuses fonctions de visualisation de la bibliothèque OpenGL dédiée à l'affichage de scènes tridimensionnelles à partir de simples primitives géométriques. Il sera par exemple utilisé pour construire et afficher les objets 3D sur la fenêtre graphique.

I/ Travail préparatoire

Cette partie est à réaliser à partir d'un fichier que l'on crée pour se familiariser avec les librairies citées précédemment.

Utilisation de Pygame

Abordons dans un premier temps la librairie Pygame.

1)

Nous devions d'abord utiliser une petite partie de code afin de comprendre comment l'affichage de Pygame fonctionne. La partie de code en question était celle-ci :

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

Premièrement, nous importons la librairie pygame. Puis on initialise tous les modules pygame importés. Ensuite, nous définissons une fenêtre pygame de taille 300 sur 200. (Ici nous l'appelons ecran). Enfin, nous désinitialisons tous les modules pygame.

Nous observons que la fenêtre ne s'affiche pas. Cependant nous pouvons apercevoir brièvement l'icône de la fenêtre pygame dans la barre de tâche.

2)

Dans un second temps, nous avons analysé un code plus complet :

```python
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()
```

Cette fois ci, la fenêtre s'ouvre correctement puisqu'il y a une condition de fermeture. Le code utilise une boucle while à partir d'un boolean *continuer* que nous définissons comme True. Celui-ci sera modifié comme False si nous appuyons sur n'importe quelle touche. En effet, le code vérifie le type d'évènement et s'il est bien égal à une pression de touche.

Utilisation de Pyopengl pour représenter des objets 3D

Puis dans un second temps, nous allons travailler sur la librairie OpenGL.

1)

Aussi, nous utilisons une partie de code afin de comprendre cette fois-ci le fonctionnement de OpenGL. La partie de code en question était celle-ci :

```python
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.

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

Nous avons ajouté la fonction **gluPerspective()** qui permet de projeter une matrice de perspective.

```python
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
```

Le 45 représente le champ de vision.
Le second paramètre représente le ratio d'aspect déterminant le champ dans la direction x.
Le paramètre suivant est le plan de clipping near Enfin la
valeur 50.0 représente le plan de clipping far.

2)

Pour pouvoir tracer une ligne à l'aide de OpenGL, il faut indiquer que nous débutons le tracer en mode ligne grâce à **glBegin()**.

Ensuite, **glColor3fv()** indique la couleur des vertices suivants.

Puis, nous indiquons 2 vertices symbolisant le point de départ et le point d'arrivé, ce qui créer le segment. (Nous utilisons la fonction **glVertex3fv()**) Enfin, il faut terminer le tracer avec **glEnd()**.

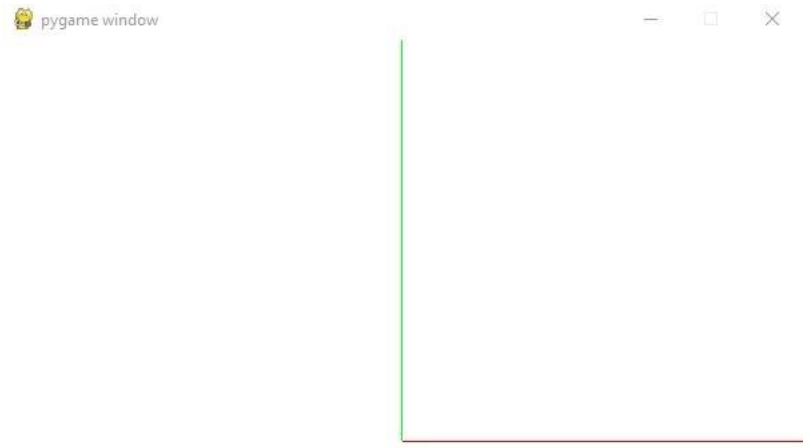
Voici le code d'exemple que nous avons utilisé :

```
gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
gl.glColor3fv([0, 0, 0]) # 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.glEnd() # Find du tracé
pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

Il fallait utiliser ce code pour tracer les axes x, y et z avec des couleurs définies.

```
# Axe des X
gl.glColor3fv([1, 0, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne

# Axe des Y
gl.glColor3fv([0, 1, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne

# Axe des Z
gl.glColor3fv([0, 0, 1]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 0, -3)) # Deuxième vertice : fin de la ligne
```

Voici l'affichage obtenu à l'aide de la fenêtre :



Nous observons l'axe des x en rouge ainsi que l'axe des y en vert.

Il faut noter que nous ne pouvons pas voir l'axe des z actuellement car il est en face du point du vu que nous avons.

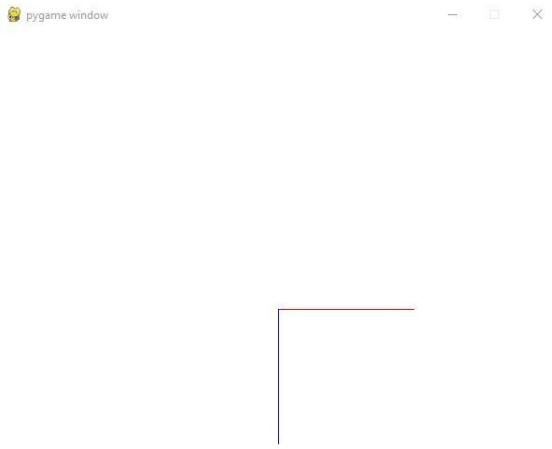
3)

Nous allons maintenant translater et effectuer un mouvement de rotation sur nos axes afin de pouvoir les observer sous différents angles. Pour cela, nous utilisons les fonctions **glTranslatef()** et **glRotatef()**.

Le code d'exemple mis à disposition était :

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
gl.glTranslatef(0.0, 2, -5)
gl.glRotatef(-90, 1, 0, 0)
```

Nous avons copié cette partie et l'avons collé dans notre code. Nous avons alors observé l'affichage graphique obtenu :



Nous remarquons que grâce à la rotation de 90°, nous pouvons désormais observer l'axe z (en bleu).

<u>Découverte de l'environnement du travail du TP</u>

Nous utilisons maintenant les fichiers fournis pour réaliser le TP. Ceux-ci contiennent les différentes classes que nous allons utiliser dans la suite de ce TP.

NICOLAS Thomas
GUERIOT Benjamin

(1). a)

Nous devions ici, d'ajouter la commande return Configuration() dans une fonction du fichier main. Lorsque nous exécutons notre programme, la fenêtre de Pygame nous affiche les 3 axes x, y et z. Nous obtenons le même affichage que la question 2) de la partie précédente. Cependant les touches « a », « z » et « Z » nous permettent respectivement d'afficher ou non les axes, de tourner les axes dans un sens et enfin de tourner les axes dans l'autre sens.

Le fichier main contient une fonction **main()** qui instancie un objet à partir de différentes fonctions (ici comme nous somme à la première partie, c'est Q1a()). Ces fonctions retournent des objets de types de Configuration. Puis nous utilisons la fonction **display()** de l'objet instancié.

A noter que nous importons toutes futures classes qui servirons à la réalisation de ce TP.

Passons désormais au fichier Configuration. Celui-ci contient la classe Configuration qui possède de nombreuses fonctions que nous traiterons par la suite. Il faut noter cependant, que pour initialiser un objet Configuration, il faut ajouter en paramètre un dictionnaire comprenant des informations comme les axes, la couleur de ceux-ci ainsi que la position de l'écran. En outre, toujours dans l'initialisation, on initialise Pygame, OpenGL, TransformationMatrix et une liste d'objets comme vide. Enfin, on génère la liste des coordonnées des axes.

```
# Initializes PyGame
self.initializePyGame()

# Initializes OpenGl
self.initializeOpenGL()

# Initializes the tranformation matrix
self.initializeTransformationMatrix()

# Initializes the object list
self.objects = []

# Generates coordinates
self.generateCoordinates()
```

Aussi, la fonction **draw()** permet de tracer les axes x, y et z suivants les paramètres renseignés lors de l'instanciation.

De plus, l'une des fonctions de cette classe est display. D'abord celle-ci appelle la fonction draw() pour tracer les axes. Ensuite, elle affiche dans une fenêtre pygame ceux-ci. Une boucle infinie lui permet de vérifier les différents types d'évènements qui peuvent survenir.

(1). b)

Nous avons réalisé une modification dans notre main afin de remarquer des changements possibles.

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

Ici, lors de l'instanciation, nous définissions les paramètres screenPosition et xAxisColor ce que changement la position de l'écran ainsi que la couleur de l'axe x.

Puis nous avons utilisé les setters **setParameter()** :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisCo lor', [0,1,1]). \
    display()
```

Un setter permet de modifier les valeurs d'un paramètre sans pour autant l'avoir définit lors de l'instanciation.

Dans ce cas-ci, le premier setParameter() était inutile puisque nous avions déjà spécifié la couleur de l'axe x dans les paramètres d'instanciation.

Aussi, le chaînage des méthodes ***setParameter()*** et ***display()*** est possible car ces fonctions se situent dans la classe Configuration. Donc ceci revient à instancier un objet et utiliser les fonctions mis à disposition.

Cela revient par exemple à faire :

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

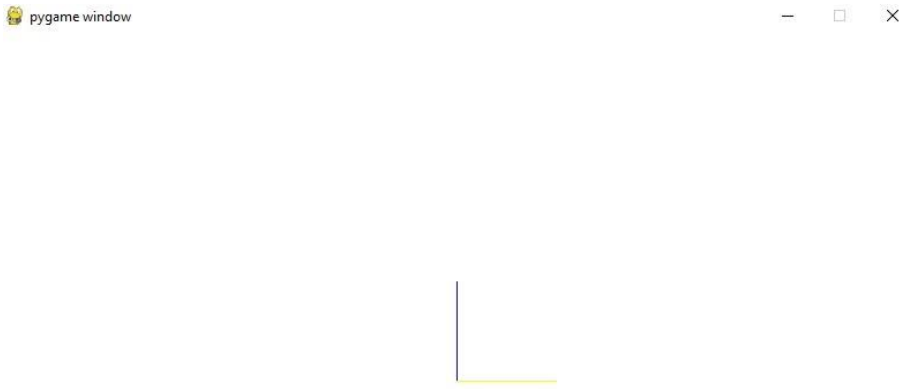
Un traitement particulier est effectué dans le « setter » pour le paramètre ***screenPosition***. En effet, si nous changeons la distance de l'écran, il faut recalculer la nouvelle perspective. C'est pourquoi, cette fonction appelle *initializeTransformationMatrix*.

(1). c)

Afin de pouvoir représenter l'axe z verticalement et l'axe x horizontalement, nous devons effectuer une rotation autour de l'axe x. Par conséquent, nous ajoutons la ligne de code suivante dans la méthode ***initializeTransformationMatrix().***

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

Voici le résultat obtenu lors de l'affichage graphique :



Nous observons bien x (en jaune) horizontalement et z (en bleu) verticalement.

NICOLAS Thomas
GUERIOT Benjamin

II/ Mise en place des interactions avec l'utilisateur avec Pygame

Dans cette partie, nous allons gérer les interactions avec l'utilisateur.

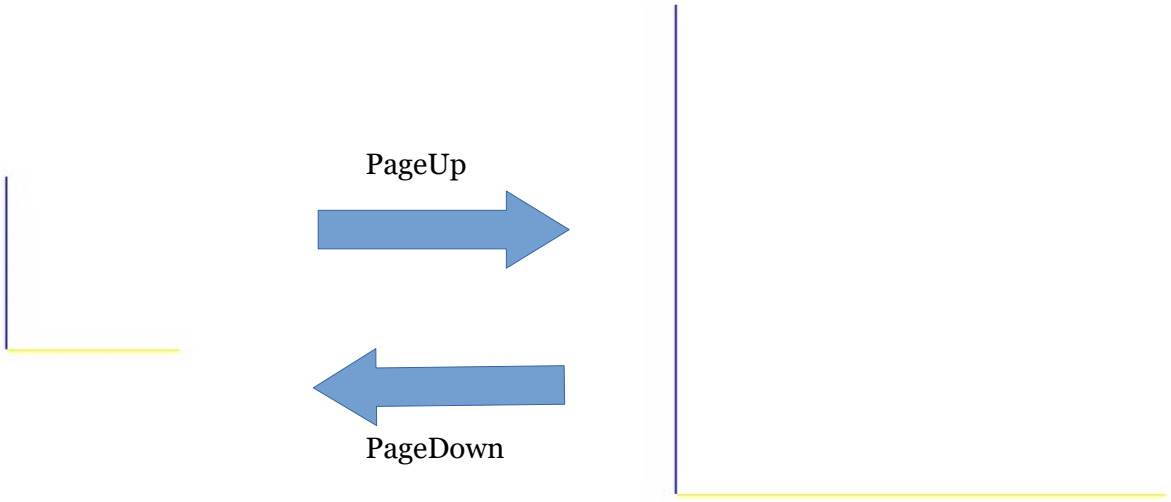
(1). d)

Nous avons ajouté les gestions des touches « Page Up », « Page Down » pour, respectivement, grossir ou réduire l'affichage. Lorsque la pression sur une touche est détectée, la méthode **processKeyDownEvent()** est appelée. Nous vérifions ainsi si la touche appuyée est « Page Up » ou « Page Down ». Si celle-ci est bel est bien pressée, nous changeons l'échelle.

La fonction devient ainsi :

```
def processKeyDownEvent(self):

    # Fait une rotation autour de z
    # Vérification des touches
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key ==
        gl.glRotate(-2.5, 0, 0, 1)   # Rotation
    # Vérification des touches
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)    # Rotation

    # Trace ou supprime les axes de référence
    # Vérification des touches
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        # Vérification que les axes ne sont pas nuls
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)

    # Vérification de la touche PageUp
    elif (self.event.key == pygame.K_PAGEUP) :
        # Mise à l'échelle de facteur 1.1
        gl.glScalef(1.1, 1.1, 1.1)

    # Vérification de la touche PageDown
    elif (self.event.key == pygame.K_PAGEDOWN) :
        # Mise à l'échelle de facteur 1/1.1
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Et voici un exemple du changement observé avec l'interface graphique :



(1). e)

Dans cette partie, nous nous intéressons à l'effet de zoom. Cette fois-ci, il fallait détecter lorsque l'utilisateur utilise la mollette de la souris. Pour cela, on utilise l'attribut button qui prend la valeur 4 lorsque le mouvement de la mollette est vers le haut et 5 lorsque celui-ci est vers le bas.

Voici la fonction que nous avons réalisé :

```
def processMouseButtonDownEvent(self):

    # Verifie si le mouvement de la mollette est vers le haut
    if self.event.button == 4 :
        # Mise à l'échelle de facteur 1.1
        gl.glScalef(1.1, 1.1, 1.1)

    # Verifie si le mouvement de la mollette est vers le bas
    elif self.event.button == 5 :
        # Mise à l'échelle de facteur 1/1.1
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

(1). f)

Nous souhaitons maintenant gérer le déplacement des objets. Le bouton gauche de la souris nous permettra de réaliser une rotation alors que le droit réalisera une translation.

Voici le code de notre fonction :

```python
def processMouseMotionEvent(self):

    # On verifie si le pouton pressé est le gauche
    if pygame.mouse.get_pressed()[0] == 1 :

        # On récupère le déplacement en x et y
        x = self.event.rel[0]
        y = self.event.rel[1]
        # On effectue une rotation
        gl.glRotate(1, x, 1, y)

    # On verifie si le pouton pressé est le droit
    elif pygame.mouse.get_pressed()[2] == 1 :
        # On récupère le déplacement en x et y puis on divise
        # par 100 pour que le changement ne soit pas trop brutal
        x = self.event.rel[0]/100
        y = self.event.rel[1]/100
        # On effectue une translation en x et y
        gl.glTranslatef(x, 0, y)
```

III/ Création d'une section

Nous nous intéressons désormais à la création d'une section. Pour ce faire, nous allons utiliser le fichier section qui contient la classe Section.

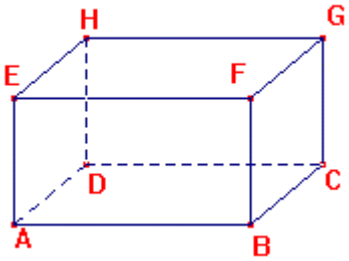
(2). a)

Nous avons écrit la méthode generate(self) qui crée les sommets et les faces d'une section orientée selon l'axe x.

```python
def generate(self):
    # Définie tous les points à partir des paramètres
    self.vertices = [
            # Point A
            self.parameters['position'],
            # Point E
            [self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2] + self.parameters['height']],
            # Point F
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2] + self.parameters['height']],
            # Point B
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]],
            # Point D
            [self.parameters['position'][0], self.parameters['position'][1] + self.parameters['thickness'], self.parameters['position'][2]],
            # Point H
            [self.parameters['position'][0], self.parameters['position'][1] + self.parameters['thickness'], self.parameters['position'][2] + self.parameters['height']],
            # Point G
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2] + sel
            # Point C
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'],self.parameters['position'][2] ]
            ]

    # Définie toutes les faces
    self.faces = [
            [0,3,2,1], # Face AEFB
            [0,4,5,1], # Face ADHE
            [4,7,6,5], # Face DCGH
            [3,7,6,2], # Face BCGF
            [0,3,7,4], # Face ABCD
            [1,2,6,5]  # Face EFGH
            ]
```

Pour représenter la section que nous allons réaliser, voici un schéma sur lequel notre méthode se base.

NICOLAS Thomas
GUERIOT Benjamin



Voici désormais le code correspondant :

FIN DE SEANCE 1

NICOLAS Thomas
GUERIOT Benjamin

2). b)

Nous nous intéressons désormais au dessin d'une section.

Nous devions expliquer l'instruction **Configuration().add(section).display().**

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

Nous ajoutons à la configuration une section de position 1, 1, 0 avec une largeur de 7 et une hauteur de 2,6.
A noter que **display()** permet d'afficher la fenêtre pygame contenant les objets créés.

Nous avons par la suite créé la méthode **draw()** :

```
def draw(self):

    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

    for y in self.faces:

        gl.glColor3fv(self.getParameter("color")) # Couleur

        for i in range(0,4):
            x = y[i]
            gl.glVertex3fv(self.vertices[x]) #Tracé des vertices


    gl.glEnd()
```

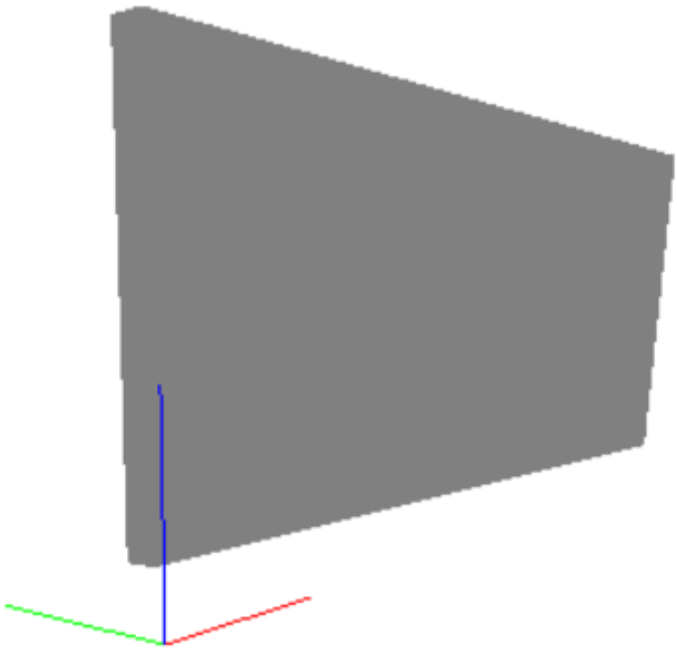
Comme nous pouvons le remarquer, cette fonction permet de tracer les différentes faces formant ainsi une section.

```
if 'color' not in self.parameters:
    self.parameters['color'] = [0.5, 0.5, 0.5]
```

Lors de l'initialisation de l'objet, la couleur par défaut est le gris.

NICOLAS Thomas
GUERIOT Benjamin

Voici à quoi ressemble la section :



(2). c)

Nous dessinons alors les arêtes de la section. Pour ce faire, nous avons écrit la méthode **drawEdges()** dans la classe **Section**.

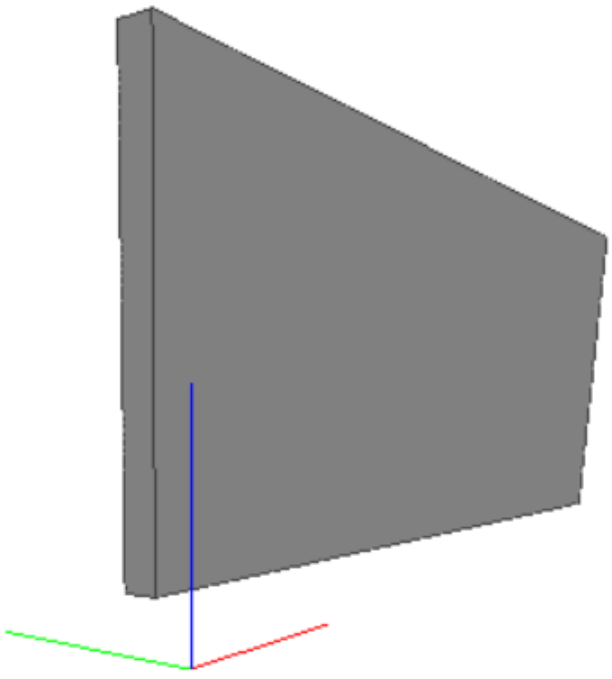
```
def drawEdges(self):

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE) # on Trace des arêtes : GL_LINE
    gl.glBegin(gl.GL_QUADS)# Tracé d'un quadrilatère


    for y in self.faces:

        facteur = 0.5 #défini la couleur des arêtes

        gl.glColor3fv([0.5 * facteur, 0.5 * facteur, 0.5 * facteur]) # Couleur

        for i in range(0,4):

            x = y[i]
            gl.glVertex3fv(self.vertices[x])    #Tracé des vertices



    gl.glEnd()
```

NICOLAS Thomas
GUERIOT Benjamin

Aussi, nous avons modifié la méthode **draw()** de la question précédente pour que la méthode **drawEdges()** soit exécutée en premier lorsque le paramètre edges, fourni au constructeur ou via le « setter » **setParameter()**, prend la valeur **True**.

```
def draw(self):

    if self.parameters['edges'] == True :
        self.drawEdges()    # Tracé des bordures

    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

    for y in self.faces:

        gl.glColor3fv(self.getParameter("color")) # Couleur

        for i in range(0,4):
            x = y[i]
            gl.glVertex3fv(self.vertices[x]) #Tracé des vertices


    gl.glEnd()
```

Nous visualisons ensuite la section avec arrêtes :



IV/ Création des murs

(3). a)

Abordons désormais la classe **Wall**. Celle-ci permet de générer des murs à partir de section.

Dans le constructeur de la classe, des paramètres sont indiqués. Ensuite, un objet **Section** est créé puis cette section est ajouté aux objets à afficher.

```
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)
```

NICOLAS Thomas
GUERIOT Benjamin

Nous avons ensuite écrit la méthode **draw()** de cette même classe.

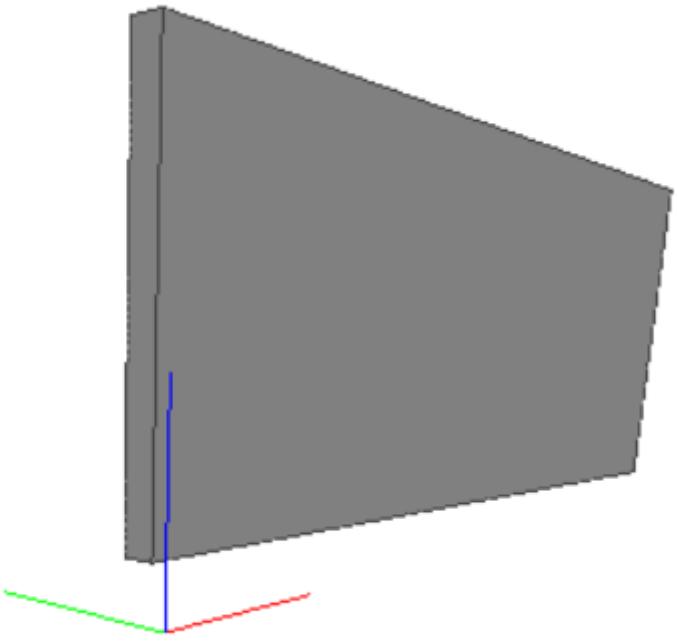
```
def draw(self):

    gl.glPushMatrix() # Crée une matrice de projection temporaire
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)   # Oriente le mur
    self.parentSection.drawEdges() # Trace les arêtes
    for x in self.objects:
        x.draw()    # Appele la méthode draw de section
    gl.glPopMatrix() # Termine la matrice de projection temporaire
```

Cette fonction permet de tracer les différentes sections à l'aide de la fonction **draw()** de la classe Section.

Nous écrivons alors dans la fonction **Q3a()** du fichier **Main.py** des instructions pour créer un mur constitué d'une section parente.

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

Puis nous affichons ce mur :



NICOLAS Thomas
GUERIOT Benjamin

V/ Création d'une maison

(4). a)

Nous nous intéressons alors à la classe **House**. Celle-ci permet d'afficher une maison crée à partir de plusieurs murs.

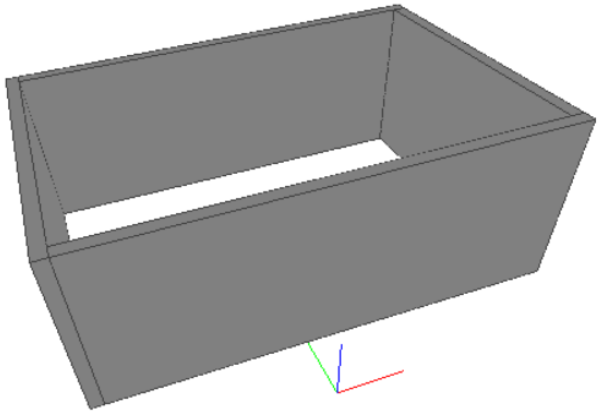
Nous avons programmé la méthode **draw()** de cette classe.

```
def draw(self):

    gl.glPushMatrix() # Crée une matrice de projection temporaire
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1) # Oriente la maison
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2]) # On translate la maison

    for x in self.objects:

        gl.glPushMatrix() # Crée une matrice de projection temporaire
        x.draw() #On trace l'objet
        gl.glPopMatrix() # Termine la matrice de projection temporaire
    gl.glPopMatrix() # Termine la matrice de projection temporaire
```

Nous écrivons alors dans la fonction **Q4a()** du fichier **Main.py** des instructions pour créer une maison, constituée de 4 murs.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0]   , 'width':7, 'height':2.6,'orientation': 0, 'edges': True})
    wall2 = Wall({'position': [0, 4.8, 0]    , 'width':7, 'height':2.6,'orientation': 0, 'edges': True})
    wall3 = Wall({'position': [0, 0, 0]   , 'width':5, 'height':2.6,'orientation': 90, 'edges': True})
    wall4 = Wall({'position': [0.2, -7, 0]   , 'width':4.6, 'height':2.6,'orientation': 90,'edges': True})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

Enfin, nous traçons la maison et voici l'affichage obtenu :



NICOLAS Thomas
GUERIOT Benjamin

VI / Création d'ouvertures

L'objectif est maintenant de créer des ouvertures dans un mur pour pouvoir mettre des portes et des fenêtres. On rappelle que, pour le moment, un mur est une section. Pour créer une ouverture dans un mur, nous allons décomposer une section en plusieurs sections.

(5). a)

Nous nous intéressons d'abord à la représentation graphique du contour de l'ouverture, constitué de 4 faces. Nous avons alors utilisé la classe **Opening** qui permet de réaliser cette tâche.

Nous avons alors complété cette classe.

D'abord en créant la méthode **generate()** permettant de définir les vertices et les 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],

            [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, 1, 5, 4],
            [0, 3, 7, 4],
            [1, 2, 6, 5],
            [3, 2, 6, 7]
            ]
```

Puis, en programmant la fonction **drawEdges()** :

```
def drawEdges(self):

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE) # On trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé de ligne


    for y in self.faces:

        facteur = 0.5 #défini la couleur des arêtes

        gl.glColor3fv([0.5 * facteur, 0.5 * facteur, 0.5 * facteur]) # Couleur

        for i in range(0,4):

            x = y[i]
            gl.glVertex3fv(self.vertices[x]) # Tracé des vertices


    gl.glEnd()
```

Cette méthode permet de tracer les arêtes des ouvertures.
Enfin, nous avons modifié la méthode **draw()** :

```
def draw(self):

    gl.glPushMatrix() # Crée une matrice de projection temporaire
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # On trace les faces : GL_FILL
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2]) # Translate l'ouverture

    for f in self.faces:
        gl.glBegin(gl.GL_QUADS) # Trace un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur
        for e in f :
            gl.glVertex3fv(self.vertices[e]) # Trace les vertices
        gl.glEnd()
    self.drawEdges()
    gl.glPopMatrix() # Termine la matrice de projection temporaire
```

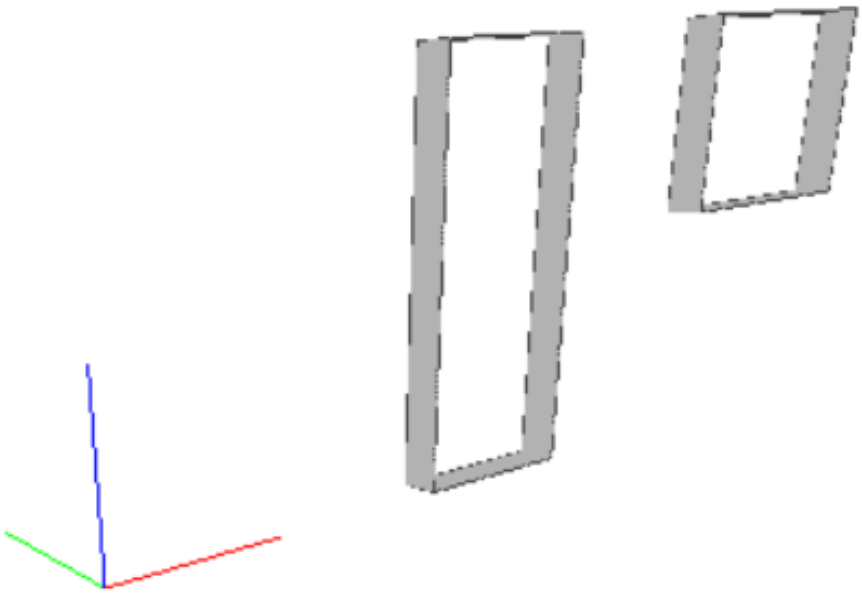
Nous pouvons observer que cette fonction permet de tracer les ouvertures.

Par la suite, nous avons exécuté la fonction **Q5a()** du fichier **Main.py** et avons observé le résultat sur l'affichage graphique :

```
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)
```

NICOLAS Thomas
GUERIOT Benjamin

(5). b)



Nous modifions alors la classe Section en ajoutant une 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.

Voici la méthode que nous avons programmé :

```
def canCreateOpening(self, x):

    # compare la largeur total de l'ouverture à la largeur total de la section
    if x.getParameter("position")[0] + x.getParameter("width") > self.parameters["position"][0] + self.parameters["width"]:
        return False

    # compare la hauteur total de l'ouverture à la largeur hauteur de la section
    elif x.getParameter("position")[2] + x.getParameter("height") > self.parameters["position"][2] + self.parameters["height"]:
        return False

    # compare la position en x de l'ouverture à la position en x de la section
    elif x.getParameter("position")[0]< self.parameters["position"][0] :
        return False

    # compare la position en z de l'ouverture à la position en z de la section
    elif x.getParameter("position")[2] < self.parameters["position"][2]:
        return False

    # sinon on retourne TRue
    else:
        return True
```

Nous vérifions alors le bon fonctionnement de notre méthode en exécutant dans le main la fonction **Q5b()** :

NICOLAS Thomas
GUERIOT Benjamin

```python
def Q5b():
    # Ecriture avec mélange de variable et de chaînage
    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]})
    opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    opening3 = Opening({'position': [4, 0, 1.7], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})

    print(section.canCreateOpening(opening1))
    print(section.canCreateOpening(opening2))
    print(section.canCreateOpening(opening3))
    return Configuration()
```

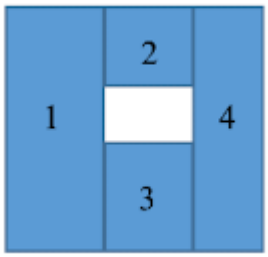
Et voici le résultat dans la console :

```
True
True
False
```

Le troisième objet **Opening** retourne faut car sa hauteur est de 2,7 alors que la hauteur de la section est de 2.6. Par conséquent, cela dépasse la hauteur de la section.

(5). c)

Une ouverture dans une section engendre 4 sections comme le montre la figure suivante :



Nous avons alors créé la fonction **createNewSections(self, x)** de la classe Section où x est un objet, par exemple une ouverture, porte ou une fenêtre. Cette fonction retourne une liste de sections engendrées par la création de l'ouverture.

NICOLAS Thomas
GUERIOT Benjamin

Voici notre méthode :

```
def createNewSections(self, x):

    sections= [] # Creation d'une liste de section

    # Creation de la section 1 en indiquant les paramètres
    section1 = Section(
        {
            "position":     self.parameters["position"],
            "width":        x.getParameter("position")[0],
            "height":       self.parameters["height"],
            "thickness":    self.parameters["thickness"],
            "color" :       self.parameters["color"],
            "edges":        self.parameters['edges'],
            "orientation":  self.parameters["orientation"]
        })

    # Vérifie si la taille n'est pas nulle
    if section1.parameters["width"] > 0 :
        sections.append(section1)

    # Creation de la section 2 en indiquant les paramètres
    section2 = Section(
        {
            "position":     [self.parameters['position'][0] + x.getParameter("position")[0],self.parameters['position'][1],self.parameters['position'
            "width":        x.getParameter("width"),
            "height":       self.parameters["height"]-x.getParameter("height")-x.getParameter("position")[2],
            "thickness":    self.parameters["thickness"],
            "color" :       self.parameters["color"],
            "edges":        self.parameters['edges'],
            "orientation":  self.parameters["orientation"]
        })

    # Vérifie si la taille n'est pas nulle
    if section2.parameters["height"] > 0 :
        sections.append(section2)

# Creation de la section 3 en indiquant les paramètres
section3 = Section(
    {
        "position":     [self.parameters['position'][0] + x.getParameter("position")[0],self.parameters['position'][1],self.parameters['pos
        "width":        x.getParameter("width"),
        "height":       x.getParameter("position")[2],
        "thickness":    self.parameters["thickness"],
        "color" :       self.parameters["color"],
        "edges":        self.parameters['edges'],
        "orientation":  self.parameters["orientation"]
    })

# Vérifie si la taille n'est pas nulle
if section3.parameters["height"] > 0 :
    sections.append(section3)


# Creation de la section 4 en indiquant les paramètres
section4 = Section(
    {
        "position":     [self.parameters['position'][0] + x.getParameter("position")[0]+x.getParameter("width"),self.parameters['position'
        "width":        self.parameters["width"]-x.getParameter("width")-x.getParameter("position")[0],
        "height":       self.parameters["height"],
        "thickness":    self.parameters["thickness"],
        "color" :       self.parameters["color"],
        "edges":        self.parameters['edges'],
        "orientation":  self.parameters["orientation"]
    })

# Vérifie si la taille n'est pas nulle
if section4.parameters["width"] > 0 :
    sections.append(section4)

return sections
```

NICOLAS Thomas
GUERIOT Benjamin

Nous vérifions donc le bon fonctionnement de notre méthode en exécutant **Q5c1()** puis en exécutant Q**5c2()** dans Main.py.

```
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
```

(5). d)

La méthode **findSection(self, x)** de la classe **Wall** permet de pouvoir retrouver, parmi toutes les sections qui composent le mur, celle dans laquelle l'insertion va se faire.

```
def findSection(self, x):
    for item in enumerate(self.objects):
        if isinstance(item[1], Section) and item[1].canCreateOpening(x):
            return item
    return None
```

Le rôle de la fonction **enumerate()** est de parcourir une collection d'éléments tout en gardant une trace de l'index de l'élément actuel dans une variable de compteur.

Les section[0] et section[1] valent 0 et 1 car ceux-ci représentent leurs indices.

Nous écrivons alors la méthode **add***(self, x)* dans la classe **Wall**.

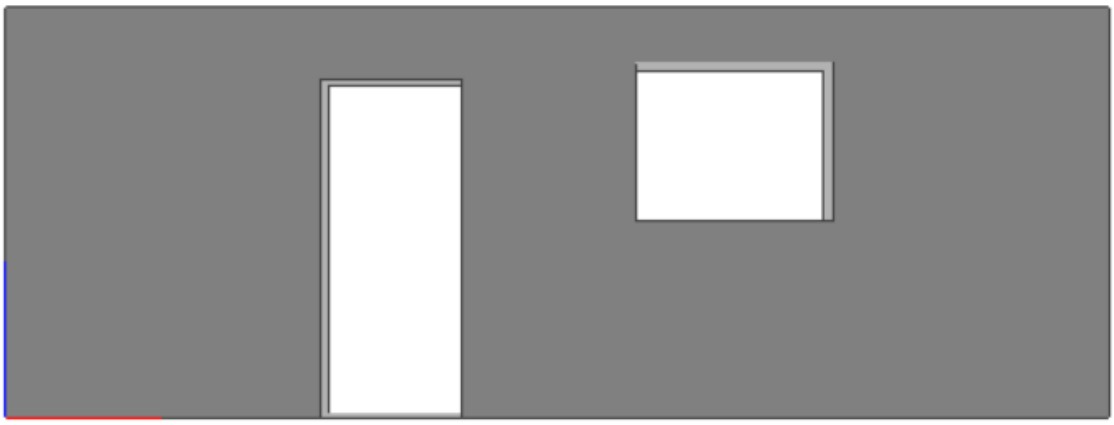
```
def add(self, x):
    section = self.findSection(x) # Cherche la section

    # Calcule de la position relative entre l'ouverture et la section
    position_Relative = [
        x.parameters["position"][0] - (section[1].getParameter("position")[0]),
        x.parameters["position"][1] - (section[1].getParameter("position")[1]),
        x.parameters["position"][2] - (section[1].getParameter("position")[2]),
    ]


    self.objects.append(x)  # Ajoute l'ouverture

    newopening= deepcopy(x) # Copie profonde afin de ne pas modifier x

    # On change la position de la nouvelle ouverture en fonction de la position relative
    newopening.setParameter("position", position_Relative)

    #On crée une nouvelle liste de sections contenant les sections nouvellement crées
    new_Sections = section[1].createNewSections(newopening)

    self.objects.pop(section[0]) # On supprime la section original

    # On ajoute toutes les nouvelles sections
    for i in new_Sections:
        self.objects.append(i)
    return self
```

NICOLAS Thomas
GUERIOT Benjamin

Nous vérifions notre méthode en exécutant la fonction **Q5d()** que nous avons modifié dans le fichier **Main.py**.

```
def Q5d():

    wall1 = Wall({"position":[0,0,0], 'width':7, 'height':2.6})

    doorFrame1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    windowFrame1 = Opening({'position': [4, 0, 1.25], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})

    wall1.add(doorFrame1)
    wall1.add(windowFrame1)

    configuration = Configuration()
    configuration.add(wall1)

    return configuration
```

Voici l'affichage obtenu :



NICOLAS Thomas
GUERIOT Benjamin

VII/ Pour finir…

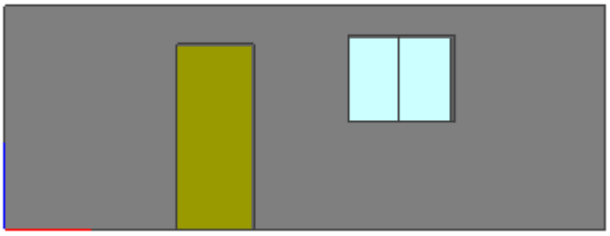
(6).

Nous avons par la suite analysé les classes **Door.py** et **Wall.py**. Celles-ci permettent de gérer les portes et les fenêtres.

Nous avons écrit la fonction **Q6()** du fichier **Main.py** :

```
def Q6():

    wall1 = Wall({'position': [0, 0, 0] , 'width':7, 'height':2.6,'orientation': 0})

    door1 = Door({'position': [2, 0.1, 0],'color': [0.6, 0.6, 0]})
    window1 = Window({'position': [4, 0.1, 1.25],'color': [0.8, 1, 1]})

    doorFrame1 = Opening({'position': [0, -0.1, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.4, 0.4, 0.4]})
    windowFrame1 = Opening({'position': [0, -0.1, 0], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.4, 0.4, 0.4]})
    door1.add(doorFrame1)
    window1.add(windowFrame1)

    wall1.add(door1)
    wall1.add(window1)

    wall2 = Wall({'position': [0, 4.8, 0]   , 'width':7, 'height':2.6,'orientation': 0, 'edges': True})
    wall3 = Wall({'position': [0.2, -0.2, 0]   , 'width':4.6, 'height':2.6,'orientation': 90, 'edges': True})
    wall4 = Wall({'position': [0.2, -7, 0]   , 'width':4.6, 'height':2.6,'orientation': 90,'edges': True})
    house = House({'position': [0, 0, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)

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

Voici le résultat obtenu à travers l'affichage graphique :



NICOLAS Thomas
GUERIOT Benjamin

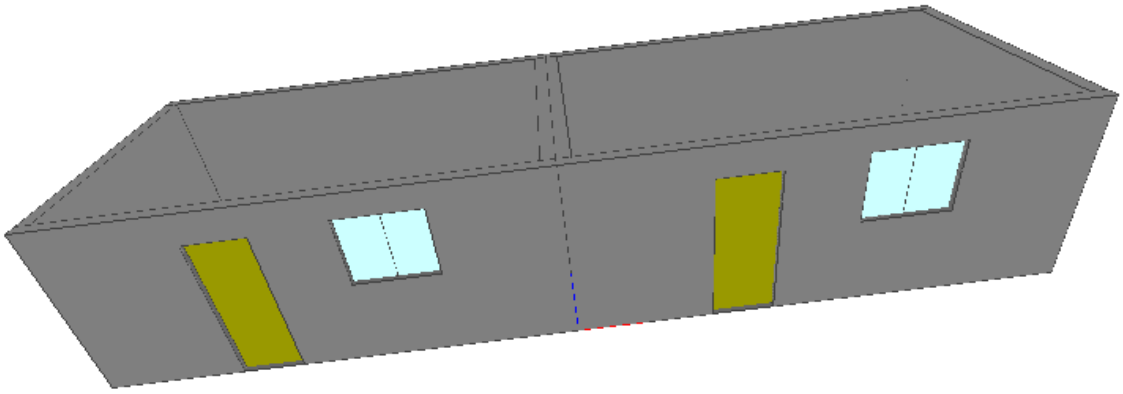
(7). Pour aller plus loin…

Pour conclure ce TP, nous avons dupliqué notre maison pour avoir un affichage semblable au gif au début du TP.

Voici notre partie du code permettant cela :

```
def Q7():

    wall1 = Wall({'position': [0, 0, 0] , 'width':7, 'height':2.6,'orientation': 0})

    door1 = Door({'position': [2, 0.1, 0],'color': [0.6, 0.6, 0]})
    window1 = Window({'position': [4, 0.1, 1.25],'color': [0.8, 1, 1]})

    doorFrame1 = Opening({'position': [0, -0.1, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.4, 0.4, 0.4]})
    windowFrame1 = Opening({'position': [0, -0.1, 0], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.4, 0.4, 0.4]})
    door1.add(doorFrame1)
    window1.add(windowFrame1)

    wall1.add(door1)
    wall1.add(window1)

    wall2 = Wall({'position': [0, 4.8, 0]   , 'width':7, 'height':2.6,'orientation': 0, 'edges': True})
    wall3 = Wall({'position': [0.2, -0.2, 0]   , 'width':4.6, 'height':2.6,'orientation': 90, 'edges': True})
    wall4 = Wall({'position': [0.2, -7, 0]   , 'width':4.6, 'height':2.6,'orientation': 90,'edges': True})
    house = House({'position': [0, 0, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    house2= House({'position': [-7, 0, 0], 'orientation':0})
    house2.add(wall1).add(wall3).add(wall4).add(wall2)

    return Configuration().add(house).add(house2)
```

Et enfin, voici notre résultat final pour l'affichage de la maison :



NICOLAS Thomas
GUERIOT Benjamin

Conclusion

A travers ce TP, nous avons appris à utiliser un grand nombre de classes objets pour un seul et même projet. Nous avons également manipulé une interface graphique pour représenter notre maison en 3D.