BENHIMA Mehdi

SALH Hamza
Date du TP (14/12/2021)

Rapport TP3 – Représentation visuelle d'objets

I. Introduction

Le TP-3 est dans le but de la représentation d'objets 3D à l'écran dans une fenêtre graphique qui permet des opérations de zoom, rotation est translations. On a choisi la représentation de maisons à partir d'objets simples que l'on va construire progressivement comme les murs, les portes, les fenêtres…

II - Préparation à faire avant le TP

1)

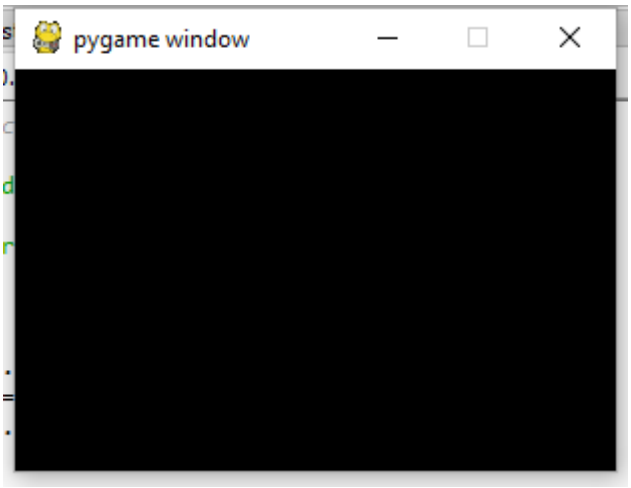
```
import pygame
pygame.init()
ecran = pygame.display.set_mode((300, 200))
pygame.quit()
```

On a importé Pygame tout d'abord, après on l'a initialisé, et après on crée un écran de dimension (300,200) à l'aide du module Pygame, la dernière ligne sert à quitter Pygame.

2)

```
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()
```

Après exécution du programme, une fenêtre Pygame apparait, et après avoir touché un bouton du clavier ou un clic souris, la fenêtre disparait.



On peut justifier cela suite aux 4 lignes qu'on a ajouté dans le programme:

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

Dans ces 4 lignes, on a initialisé une variable nommée « continuer », au début, cette variable est initialisée à « True », après on effectue une boucle While en sortant de la boucle si seulement si un évènement de type Keydown apparait, et cet évènement se manifeste en appuyant sur une touche du clavier ou en appuyant sur la souris, une fois cet évènement apparait, la variable « continuer » prend la valeur False et on sort de la boucle, et la fenêtre Pygame disparait.

Utilisation de Pyopengl pour représenter des objets 3D

1)

En initialisant le nouveau programme :

```
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()
```

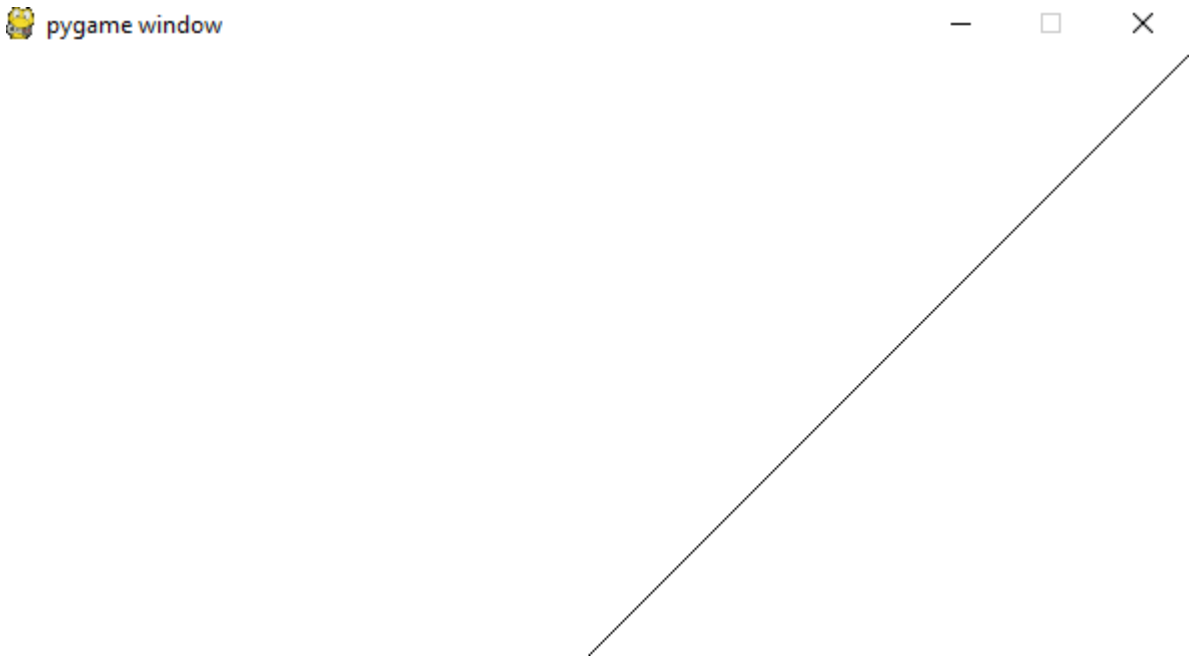
Une nouvelle fenêtre Pygame apparait, et cette fois grâce aux 4 derniers lignes du programme, on peut fermer la fenêtre Pygame en appuyant sur la croix.

En ajoutant la fonction gluPerspective de la librairie glu, et on l'initialisant à (45, 1, 0.1, 50).

```
glu.gluPerspective(45, 1, 0.1, 50)
```

On exécute le programme, il n y'a pas d'erreurs.

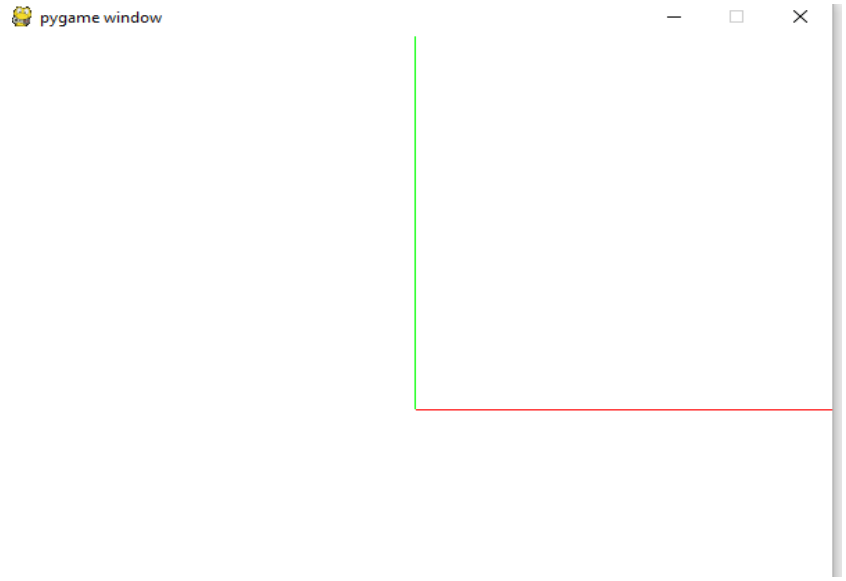
2)



Après on modifie les paramètres de chaque fonction pour tracer 3 axes rouge, vert, bleu respectivement suivant les axes x, y, z, en prenant le code RGB de chaque couleur :

```
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)

    glu.gluPerspective(45,  (display[0] / display[1]), 0.1, 50.0)
    gl.glBegin(gl.GL_LINES)
    gl.glColor3fv([254, 0, 0])
    gl.glVertex3fv((0,0, -2))
    gl.glVertex3fv((1, 0, -2))
    gl.glColor3fv([0, 255, 0])
    gl.glVertex3fv((0,0, -2))
    gl.glVertex3fv((0, 1, -2))
    gl.glColor3fv([0, 0, 255])
    gl.glVertex3fv((0,0, -2))
    gl.glVertex3fv((0, 0,0))
    gl.glEnd()
    pygame.display.flip()

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

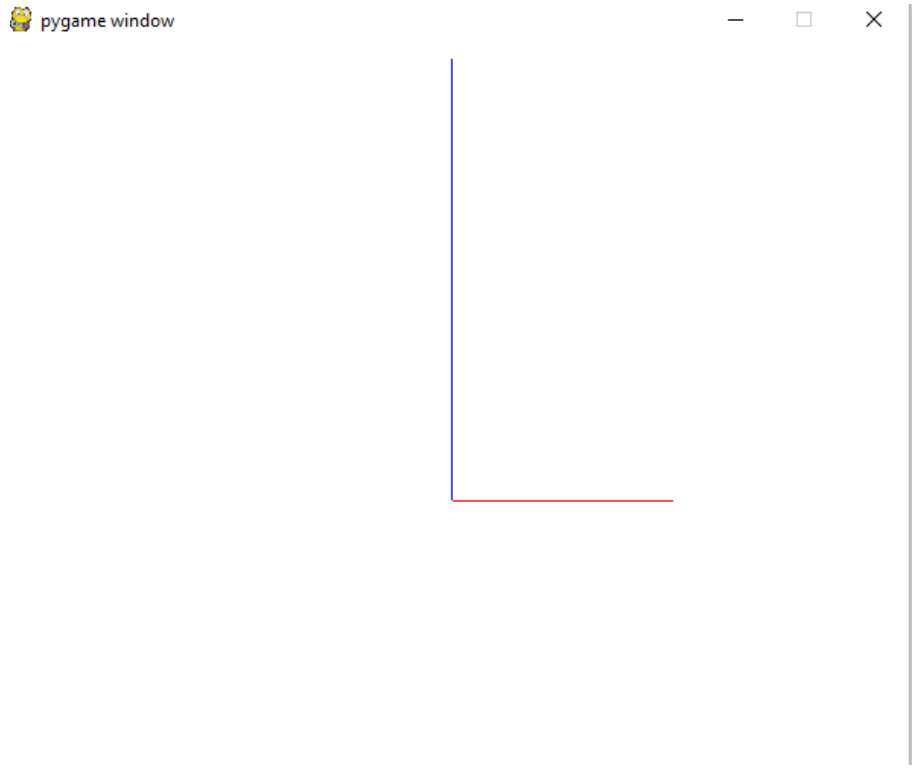
On obtient la fenêtre suivante :



3) on ajoute au programme ces 3 lignes :

```
gl.glTranslatef(0.0, 2, -5)
gl.glRotatef(-90, 1, 0, 0)
```

Qui permettent de faire une translation suivant l'axe z et faire une rotation de 90 degré, on exécute et on trouve la figure ci-dessous, on remarque qu'on peut voir maintenant l'axe z qu'on a pas pu voir lors de la question précédente.



Découverte de l'environnement du travail du TP

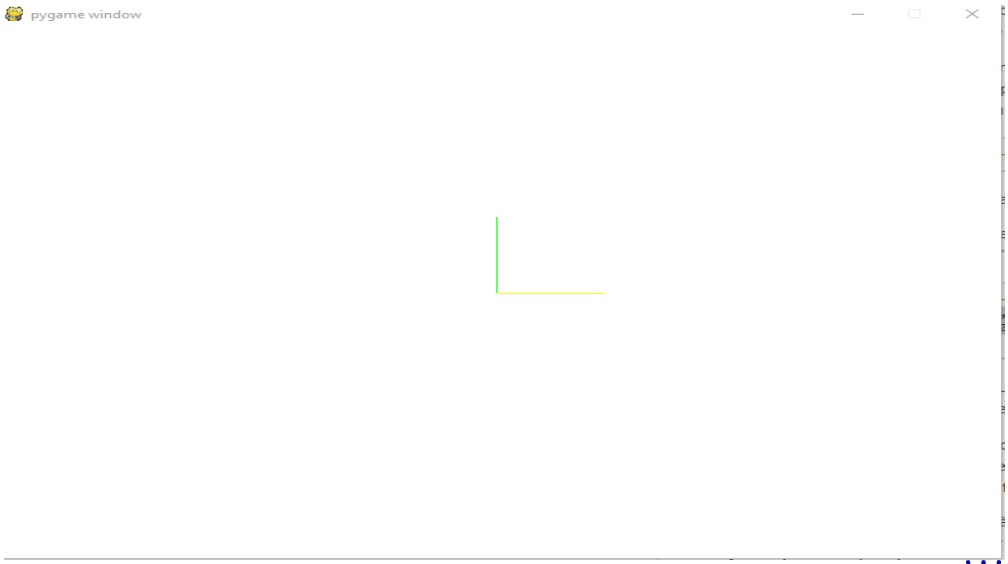
1)a)

```
def Q1a():
    return Configuration()
```

Après qu'on a ajouté la ligne dans la question 1a) le code rend la fenêtre comme avant tout les segments selon les 3 axes en colleur précise.

1)b)

On a transmis les paramètres dans la classe de configuration afin de faire des modifications de la position de l'écran et la couleur de l'axe *x* par le constructeur par rapport aux paramètres et on obtient :

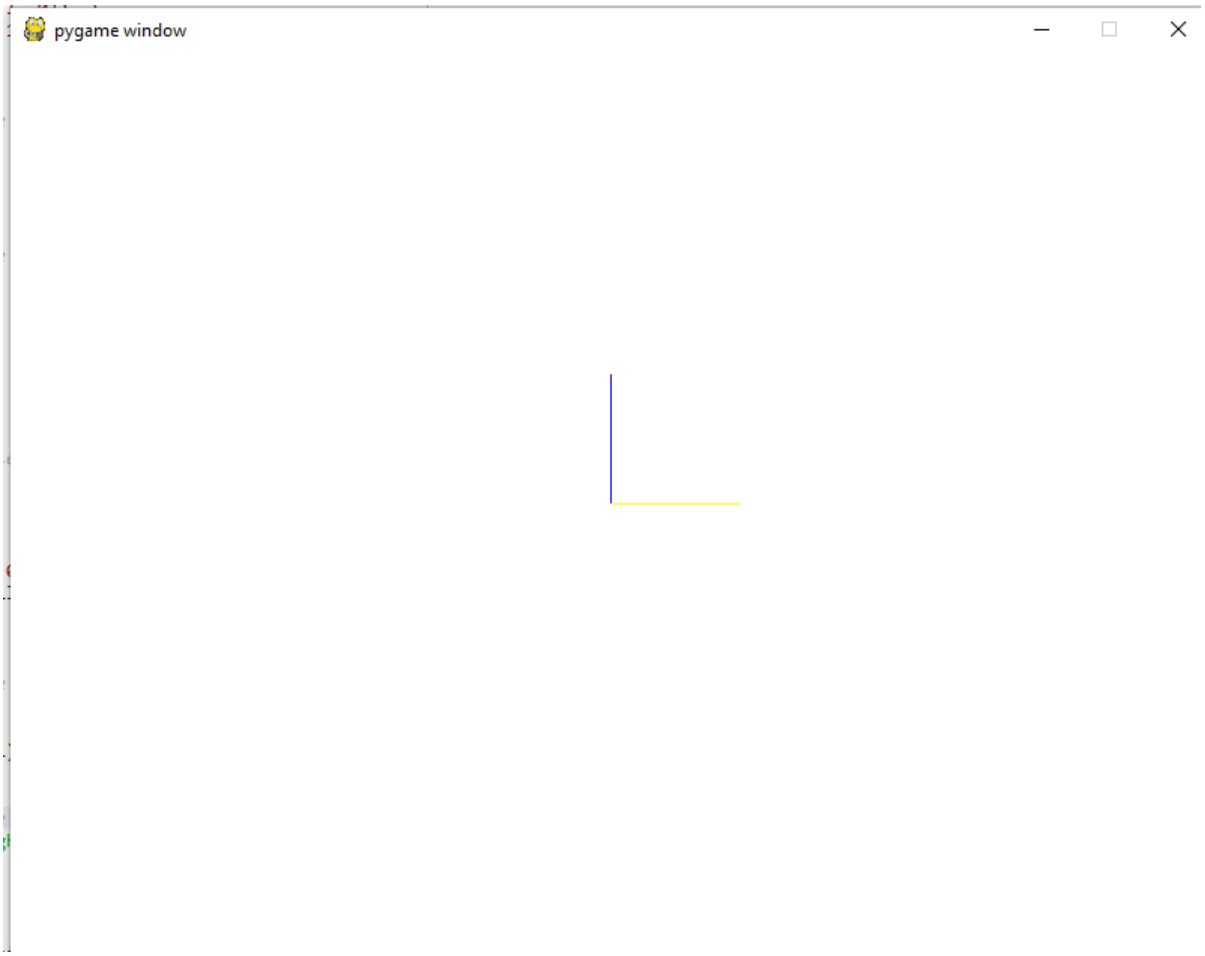


1)c)

Dans cette question on essaie de changer les axes z et x (rotaion de 90 degres)

```
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)
```

Et on obtient la fenetre suivant :



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

(1). d)

Dans cette partie on s'intéresse à gérer les interactions possibles dans application avec pygame

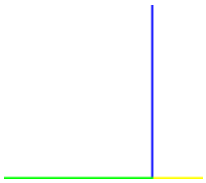
```
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
       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

Avec cette on peut gérer trois touches suivantes :

- « z » cette touche permet de faire une rotation de 2.5 degrés autour de l'axe z

- « a » pour afficher ou supprimer les axes

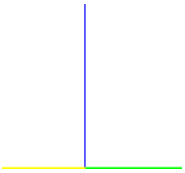
- « Z » afin de faire une rotation de -2.5 autour de l'axe z



1)e)

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
        if  self.event.button == 4:
            gl.glRotate(-2.5, 0, 0, 1)
        elif self.event.button == 5:
            gl.glRotate(2.5, 0, 0, 1)
```

Avec le code on a réalisé le changement d'échelle (effet de zoom) en utilisant la mollette de la souris



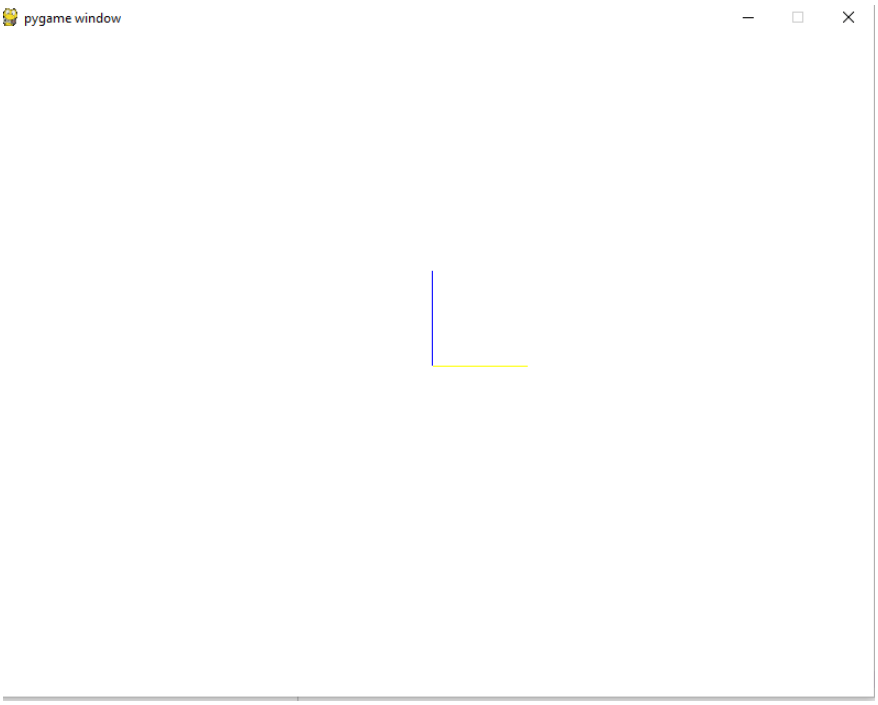
Avant après (zoom positive avec la souris

(1). f)

Le but de cette question est de déplacer les objets affichés en bougeant la souris lorsque le bouton de droite ou de gauche est activé. Le bouton de gauche servira aux rotations des objets et le bouton de droite aux translations des objets

```
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0]==1:
         if  self.event.rel[0]==1:
                gl.glRotate(-2.5, 0, 0, 1)
    if pygame.mouse.get_pressed()[2]==1:
            if self.event.rel[0]==0:
                gl.glTranslatef(0.0, 2, 1)
```

Au début on a fait Une rotation autour de x composée avec une rotation autour de z avec le bouton gauche et déplacer la souris.



Après on fait une translation selon l'axe x composée à une translation selon l'axe z avec le bouton droit et le déplacement de la souris.

IV - Création d'une section

Dans cette partie on essaie de créer une section, d'abord une section correspond à un parallélépipède à 8 sommets et 6 faces et il est constituée du dessin de ses arrêtes et du remplissage des faces par une couleur.

(2). a)

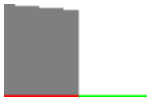
Dans la classe on écrit la méthode generate(self) comme suivant :

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

(2). b)

On commence avec le traçage d'une seule face comme ce qui est sonné

On fait une rotation pour vérifier le fonctionnement des touches et de souris



On trace les 6 faces avec le code suivant :

```
##     Draws the faces
    def draw(self):
        gl.glPushMatrix()
        gl.glTranslatef(self.parameters['position'][0],self.parameters['position']
[1],self.parameters['position'][2])
        gl.glRotate(self.parameters['orientation'], 0, 0, 1)
        if self.parameters['edges'] == True:
            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
        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']])

        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.glVertex3fv([self.parameters['width'], self.parameters['thickness'], 0])
        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],
self.parameters['height']])
        gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
        gl.glVertex3fv([0, self.parameters['thickness'], 0])

        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, self.parameters['thickness'], 0])
        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])

        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        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([0, 0, 0])
        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
        gl.glVertex3fv([0, self.parameters['thickness'], 0])
```

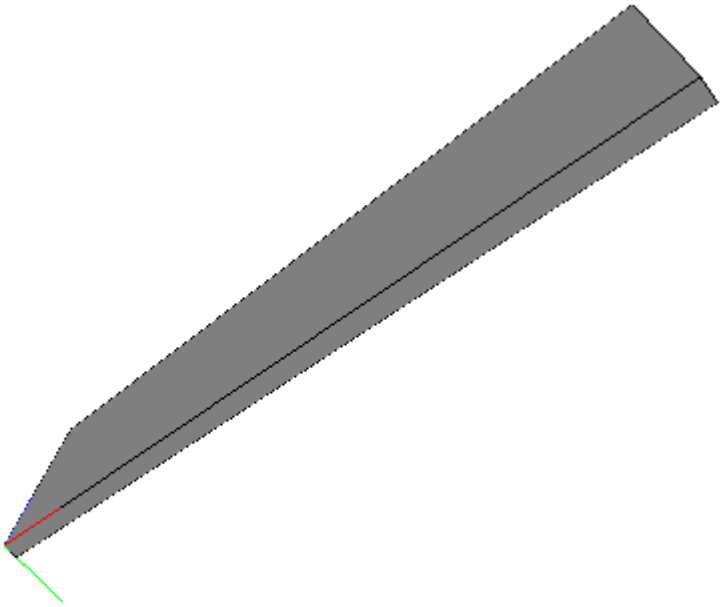
2)c)on Programme le tracé des arretes :

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

        gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE) # on trace les arretes :
GL_LINE
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0, 0, 0])
        # Couleur NOIR
        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([0, 0, self.parameters['height']])

        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, 0, self.parameters['height']])

        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([self.parameters['width'], 0, 0])

        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], 0, 0])

        gl.glVertex3fv([0, self.parameters['thickness'], 0])
        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']])
        gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])

        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],
self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], 0])

        gl.glVertex3fv([0, 0, 0])
        gl.glVertex3fv([0, self.parameters['thickness'], 0])

        gl.glVertex3fv([0, 0, self.parameters['height']])
        gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])

        gl.glVertex3fv([self.parameters['width'], 0, 0])
        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], 0])

        gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
        gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],
self.parameters['height']])

        gl.glEnd()
```

On ajoute à 2)b) la condition pour laquelle la méthode **drawEdges()** soit exécutée en premier :

```
    if self.parameters['edges'] == True:
        self.drawEdges()
```

On exécute le programme, on obtient un mur avec ses arretes en noir :



V - Création des murs :

La classe Wall correspond à un mur : les différents paramètres sont : la position du mur, l'épaisseur, l'orientation, la couleur.

On modifie le fichier 'Wall' et 'main' et on exécute :

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

        gl.glRotate(self.parameters['orientation'], 0, 0, 1)
        # Draws the objects if any
        for x in self.objects:
            x.draw()

        gl.glPopMatrix()
```

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

On exécute, on aura le résultat suivant :

VI - Création d'une maison :

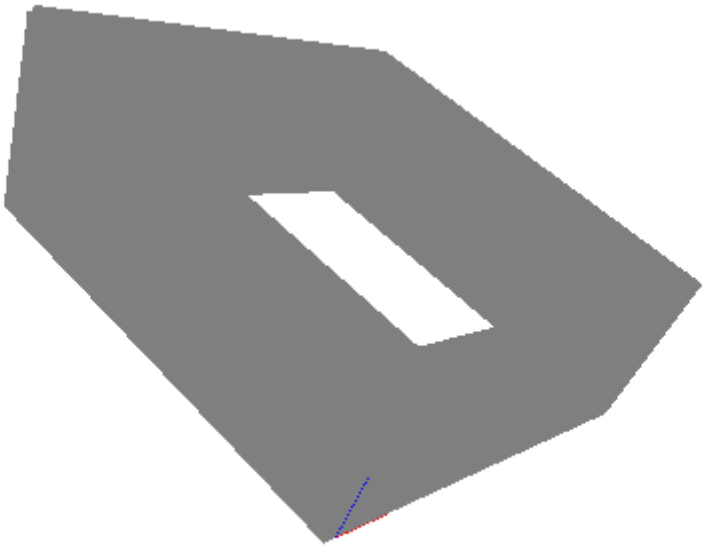
On fait de méme pour la classe House :

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

        gl.glRotate(self.parameters['orientation'], 0, 0, 1)
        # Draws the objects if any
        for x in self.objects:
            x.draw()

        gl.glPopMatrix()
```

Et on modifie la question de 4)a) du fichier Main.py en réglant les paramètres des quatres murs de la maison :

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

On exécute, ; on aura le résultat suivant :



VII - Création d'ouvertures :

5)a)

```
def generate(self):
        self.vertices = [
        [0, 0, 0 ],
        [0, 0, self.parameters['height']],
        [self.parameters['width'], 0, self.parameters['height']],
        [self.parameters['width'], 0, 0],
        [0, self.parameters['thickness'], 0 ],
        [0, self.parameters['thickness'], self.parameters['height']],
        [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
        [self.parameters['width'], self.parameters['thickness'], 0]
        ]

        self.faces = [
        [0,4,5,1],
        [3,7,6,2],
        [0,4,7,3],
        [1,5,6,2]
        ]
        # 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.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
        for i in self.faces:
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv(self.parameters['color'])
        for j in i:
        gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
        gl.glPopMatrix()
```

On exécute, on aura :

5)b)

```
def canCreateOpening(self, x):
        return (self.parameters['height']>=x.getParameter('height')
+x.getParameter('position')[2]-self.parameters['position'][2]
        and x.getParameter('position')[2]>=self.parameters['position'][2]
        and self.parameters['width']>=x.getParameter('width')+x.getParameter('position')[0]-
self.parameters['position'][0]
        and x.getParameter('position')[0]>=self.parameters['position'][0]
        )
```

Conclusion :

On a utilisé lors de ce TP des différentes méthodes pour la représentation visuelle des objets, en parcourant des méthodes dans des différentes classes pour construire des différents objets afin de pouvoir visualiser la représentation 3D d'objets et effectuer des différentes opérations.

On a du mal à programmer plusieurs fonctions, mais on a pu finalement gérer la situation et on a appris à bien maitriser les différents outils pour la représentation visuelles des objets.