FDHILA Amine
ADJADJ Amaury
SILVA Leo
07/12/2021

Rapport de TP3 – Représentation visuelle d'objets.

I.Introduction

Ce TP a pour but 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.Travaille Préparatoire

1) *Utilisation de pygame*

a. *Code simple pygame*

```
import pygame #on import les modules de pygames
pygame.init() #on initialise les modules de pygame
ecran = pygame.display.set_mode((300, 200)) #on ouvre un écran de 300 par 200
pygame.quit() #on désinitialise tout les modules pygame qui ont éé initialisés
```

b.Code complexe pygame

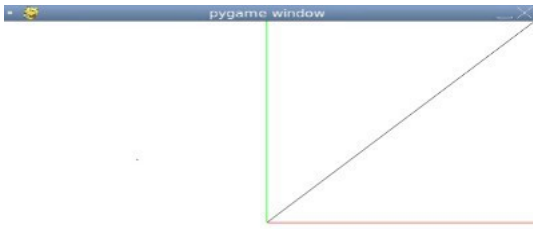
```
import pygame


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

continuer = True
while continuer:     # si continuer est vrai
    for event in pygame.event.get(): #on fait une boucle qui prend en comptes des evenements extérieurs
        if event.type == pygame.KEYDOWN: # si l'evenements est qu'une touche est relaché on quitte la boucle
            continuer = False

pygame.quit()
```

L'écran reste ouvert temps que l'on n'appuie pas sur une touche du clavier.

2) Utilisation de pyopengl pour représenter des objets 3D

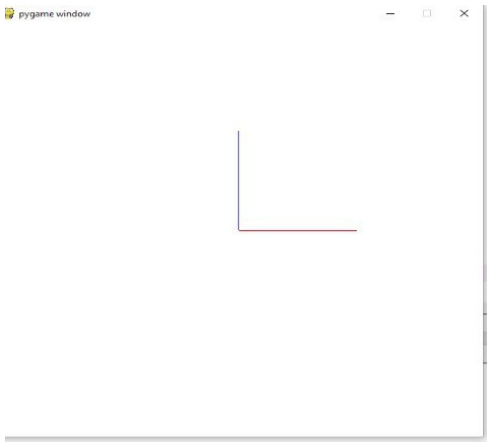
a.Affichage d'axes

En rouge : axe x
En vert : axe y
En noir axe z (normal au plan Oxy)
En bleu : axe définit dans l'énoncé



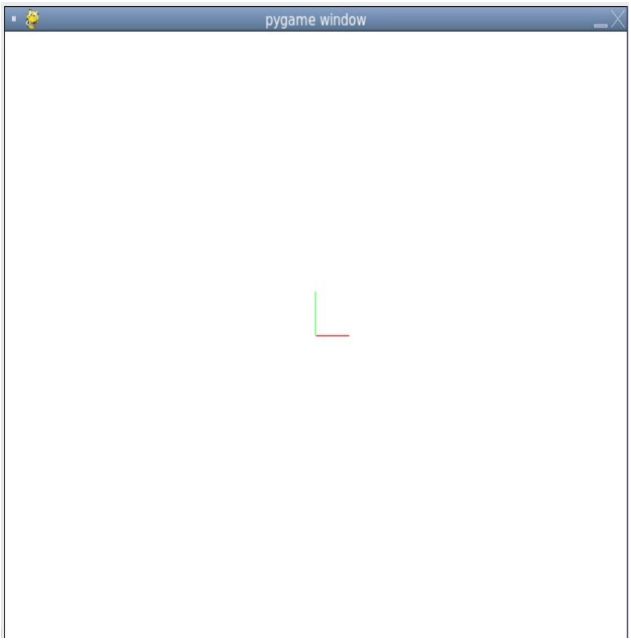
b. Rotation et translation

Rotation de -90 par rapport à l'axe x Translation de -5 en z et 2 en y



3) Découverte de l'environnement du travail de TP

Q1).a)
On a rentré dans main.py return Configuration() ce qui nous affiche les axes x (rouge) et y (vert) et z (bleu) par défaut, lorsque aucun paramètre n'est rentré comme indiqué dans la feuille de programme .



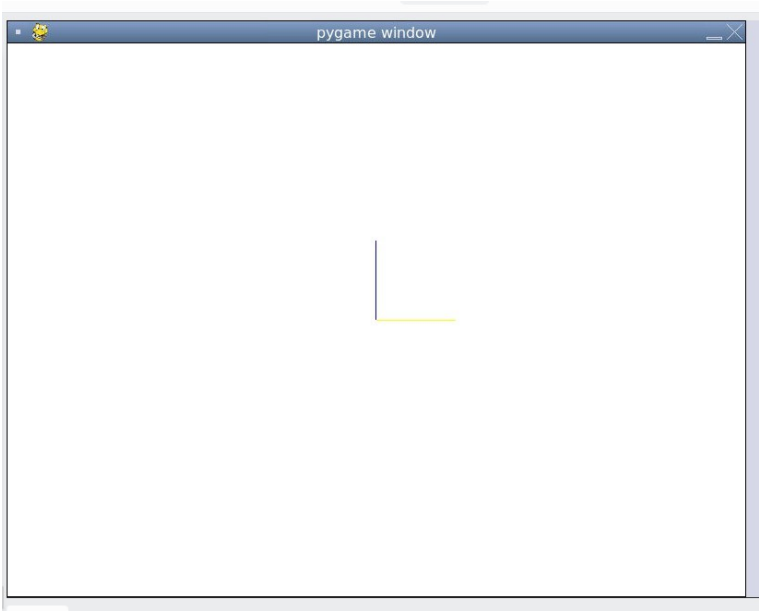

```python
# Constructor
def __init__(self, parameters = {}) :
    # Parameters
    # axes: if True the axis is displayed
    # xAxisColor: color for the x-axis
    # yAxisColor: color for the y-axis
    # zAxisColor: color for the z-axis
    # screenPosition: position of the screen on the z-axis (negative value) - on

    # Sets the parameters
    self.parameters = parameters

    # Sets the default parameters
    if 'axes' not in self.parameters:
        self.parameters['axes'] = True
    if 'xAxisColor' not in self.parameters:
        self.parameters['xAxisColor'] = [1, 0, 0]
    if 'yAxisColor' not in self.parameters:
        self.parameters['yAxisColor'] = [0, 1, 0]
    if 'zAxisColor' not in self.parameters:
        self.parameters['zAxisColor'] = [0, 0, 1]
    if 'screenPosition' not in self.parameters:
        self.parameters['screenPosition'] = -10

    # Initializes PyGame
    self.initializePyGame()

    # Initializes OpenGl
    self.initializeOpenGL()

    # Initializes the tranformation matrix
    self.initializeTransformationMatrix()

    # Initializes the object list
    self.objects = []

    # Generates coordinates
    self.generateCoordinates()
```

Q1).b)

La fonction rentré permet de modifier la couleur de l’axe x en jaune ainsi que la profondeur des axes.

Q1).c)

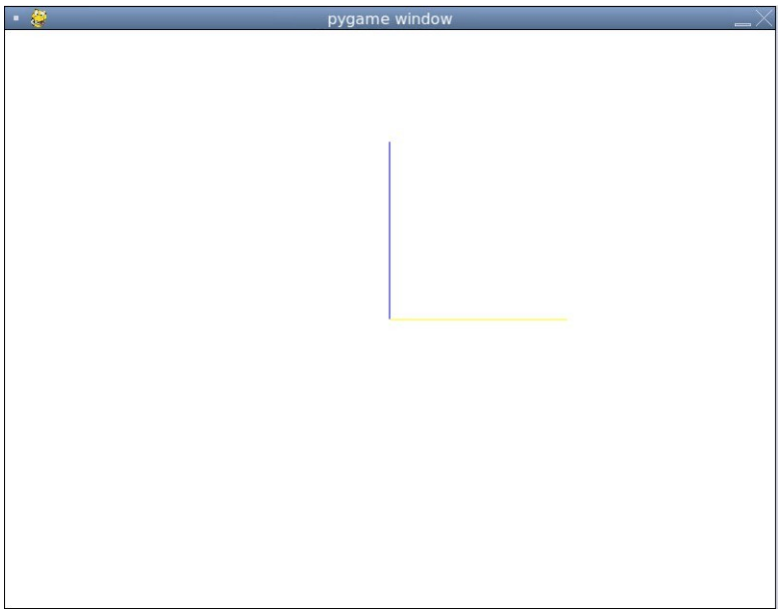
L’instruction gl.glRotatef(-90, 1, 0, 0) ajouté à la méthode initializeTransformationMatrix() permet d’effectuer une rotation des axes (-90 autour de l’axe x).



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

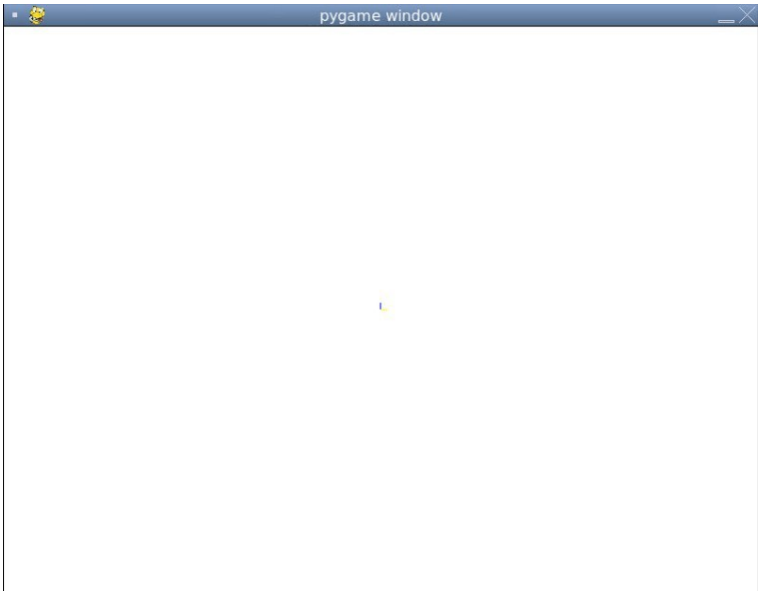
Q1).d)

le programme permet de faire une rotation et également de zoomer les axes.



Q1).e)

La fonction processMouseButtonDownEvent(self) permet d'effectuer un zoom via la molette.



Q1).f)

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
  if pygame.mouse.get_pressed()[2]:
    gl.glTranslatef(self.event.rel[0]/100,0.0,-self.event.rel[1]/100)
    #Translation suivant l'axe x et z. On met un - devant self.event.rel[1]
     car la translation sur l'axe z est inversé de base avec la
    translation de la souris. On a souhaité diviser par 100 car la vitesse
    de translation était trop rapide à notre gout.

  if pygame.mouse.get_pressed()[0]:
    gl.glRotatef(self.event.rel[0], 0, 0, 1) #rotation suivant l'axe z
    gl.glRotatef(self.event.rel[1], 1, 0, 0) #rotation suivant l'axe x
```

On définit la méthode processMouseMotionEvent qui permet de déplacer avec la sourie.
On utilise la méthode getpressed() qui renvoie si les boutons gauche[0] ou droit [2]sont appuyés. Les fonctions gl.glTranslatef et gl.glRotatef permettent d'effectuer des translations et rotation autour des axes x et z.
On prendra le bouton de droite pour translater notre objet sur les axes x et z en réduisant la vitesse de déplacement par 100, le bouton de gauche sera pour les rotations.

Nous avons rajouté un moins devant le **self.event.rel** sur l'axe z car sans le déplacement est inversé par rapport au mouvement de la sourie.

IV. Création d'une section

Dans cette partie nous allons créer un parallélépipède à travers plusieurs parties.

Q2).a)

Pour commencer la création du parallélépipède nous définissons les 8 sommets.

```
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] #sommets de la face y=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] #sommets de la face y="thickness"
```

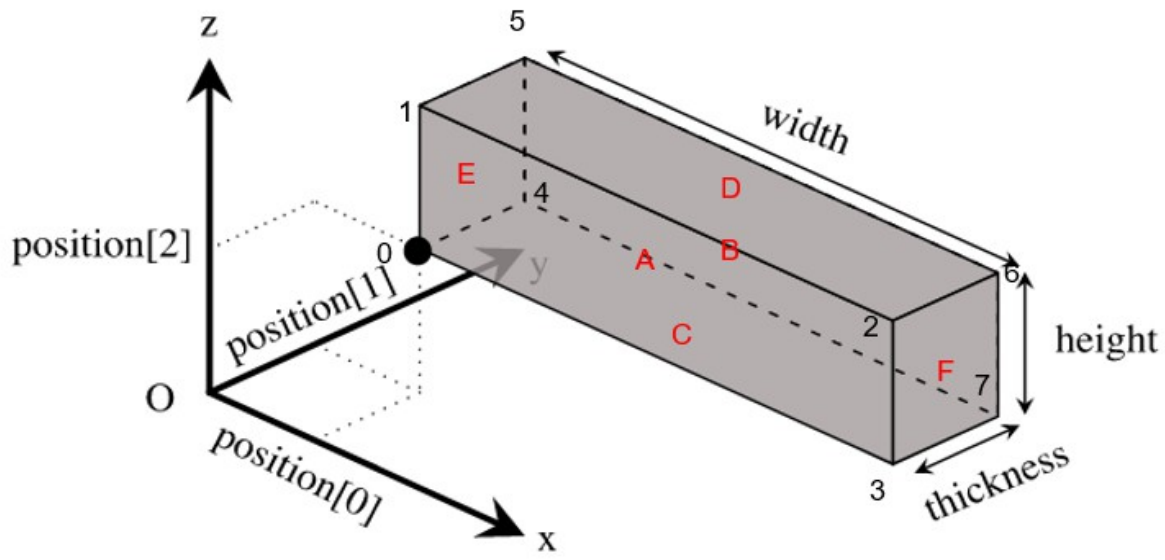


figure section 1

Les coordonnées des sommets sont définit dans self.vertices :

Sommet 0 : [0, 0, 0] Représente l'origine associé au parallélépipède. 1

Sommet 1 : [0, 0, self.parameters['height']]

Sommet 2 : [self.parameters['width'], 0, self.parameters['height']]

Sommet 3 : [self.parameters['width'], 0, 0]

Sommet 4 : [0, self.parameters['thickness'],0]

Sommet 5 : [0, self.parameters['thickness'] , self.parameters['height']]

Sommet 6 : [self.parameters['width'], self.parameters['thickness'], self.parameters['height']]

Sommet 7 : [self.parameters['width'], self.parameters['thickness'], 0]

On associe ensuite à chaque face leurs 4 sommets respectifs. Les coordonnées des sommets utilisés pour définir les faces sont leur position dans la liste self.vertices.

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

Face A : [0,3,2,1]

Face B : [4,7,6,5]

Face C : [0,3,7,4]

Face D : [1,2,6,5]

Face E : [0,4,5,1]

Face F : [3,2,6,7]

Q2).b)

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

Cette instruction permet d'ajouter une section qui sera positionée dans le repère principale en [1,1,0] de largeur 7 et de hauteur 2,6. Cette section est ajoutée à la liste object de configuration.py.

La méthode draw() permet de tracer les faces de la section en gris comme visualisé sur la figure suivante.

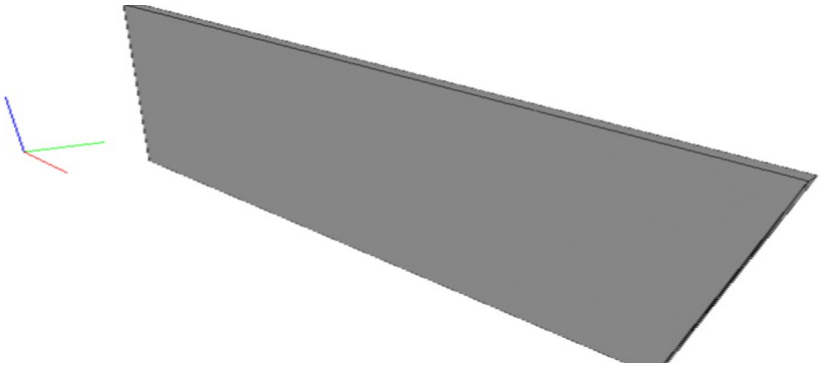
```
# Draws the faces
def draw(self):

 if self.parameters['edges'] :
   self.drawEdges()#commande rajouter q2c


 gl.glPushMatrix()#permet de stocker la matrice de projection global puisque on crée une matrice de projection propre
 à la section
 gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2]) #on
 crée la section à partir de sa position dans setparameter
 gl.glRotatef(self.parameters['orientation'],0,0,1)#on oriente la section à partir de son orientation defini dans les
 parametres. l'orientation se fait autour de l'axe verticale z

 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(self.parameters['color']) # on prend la couleur defini dans setparameter. Par defaut si la couleur
 n'est pas definie les sections sont grises.



 #création des surfaces
 for i in self.faces: #On parcours la liste self.faces
   for j in i: #on parcourt l'indice des sommets des faces de la liste self.faces. cela permettra de récuperer dans la
   commande suivante les données de chaques sommets de chaques faces.
     gl.glVertex3fv(self.vertices[j]) #on trace les sections de chaques faces (coordonnées des sections récupéré dans
     la liste vertices)

 gl.glEnd()
 gl.glPopMatrix()
```

Q2).c)

On a placé en premier la méthode drawEdges avant la méthode draw de cette manière la fonction drawEdges est excuté avant si la condition edges=True est vraie.

```
# Draws the edges
def drawEdges(self):
  #comme draw
  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_LINE) # on trace les lignes
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv([0.1,0.1,0.1]) # couleur plus foncé.

  #création des lignes comme pour la methode draw
  for i in self.faces:
    for j in i:
      gl.glVertex3fv(self.vertices[j])
  gl.glEnd()
  gl.glPopMatrix()
```

Il s'agit du même code que la question précédente à l'exeption de GL_FILL qui devient GL_LINE et la couleur qui est plus foncé.



V. Création des murs

On s'intéresse maintenant à la création d'un mur. Celui-ci correspond à un objet contenant une liste d'éléments à tracer (fenêtres, portes et sections) dans notre application. Dans un premier temps, nous allons considérer qu'un mur n'est constitué que d'une seule section.

Q3).a)

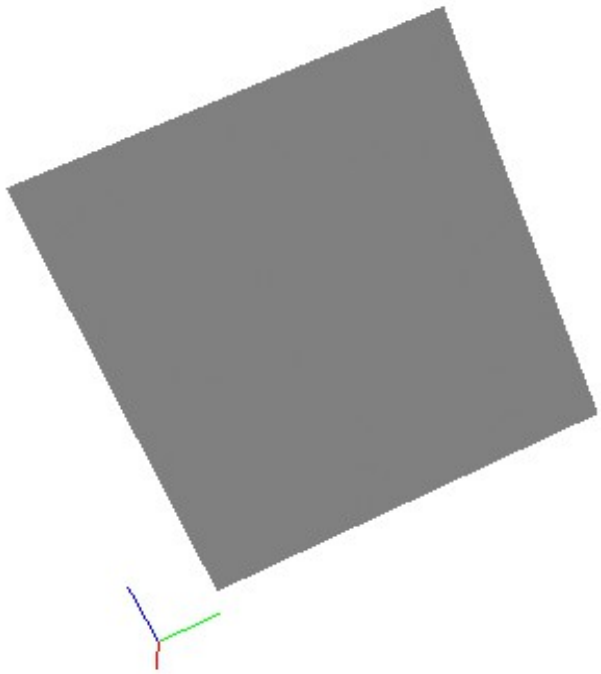
```
class Wall:
    # Constructor
    def __init__(self, parameters = {}) :
        # Parameters
        # position: position of the wall
        # width: width of the wall - mandatory
        # height: height of the wall - mandatory
        # thickness: thickness of the wall
        # color: color of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

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'position' not in self.parameters:
            self.parameters['position'] = [0, 0, 0]   #si pas definit se met a
            0,0,0
        if 'width' not in self.parameters:
            raise Exception('Parameter "width" required.')   #obligé d'etre
            defini
        if 'height' not in self.parameters:
            raise Exception('Parameter "height" required.')   #obligé d'etre
            defini
        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]
```

Le constructeur de cette classe va rentrer des valeurs standards pour créer le mur, dans le cas où elles ne sont pas déjà rentrées par l'opérateur.

```
def draw(self):
  gl.glPushMatrix() #meme chose que dans le code draw de section
  gl.glRotatef(self.parameters['orientation'],0,0,1)
  for objet in self.objects: #les objets qui forment la face sont des
  sections du mur
    objet.draw()
  gl.glPopMatrix()
```

```
def Q3a():
    return Configuration().add(Wall({'position': [1, 1, 0],
    'width':7, 'height':7,'orientation':90})) #mur de taille 7*7*0.2
    orienté de 90 degré
```

Dans cette fonction on définit un mur situé en [1,1,0] par rapport au repère principale de hauteur et de largeur 7 avec une orientation de 90° comme dans la question 2)b). Le paramètre edges n'étant pas mis à 1 les arrêtes n'apparaissent pas.



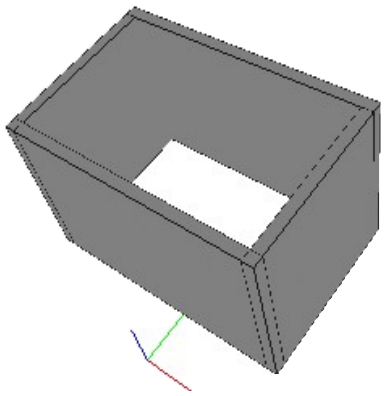
VI. Création d'une maison

Q4).a)

```
def draw(self):
  gl.glPushMatrix()
  gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],
  self.parameters['position'][2]) #pour translater la maison
  gl.glRotatef(self.parameters['orientation'],0,0,1) #pour orienter la maison

  for objet in self.objects: #les objets sont les murs de la maison
    objet.draw()
  gl.glPopMatrix()
```

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':5, 'height':3,'orientation':0})
    wall2 = Wall({'position': [0, 3, 0], 'width':5, 'height':3,'orientation':0})
    wall3 = Wall({'position': [0, -0.2, 0], 'width':3, 'height':3,'orientation':90}) #mis à
    -0.2 sur y car il faut prendre en compte l'epaisseur du mur lateral pour fermer la maison.
    la rotation de 90° fait que l'on modifie la position sur y alors que le deplacement est
    sur x (par rapport à une position [0,0,0])
    wall4 = Wall({'position': [0, -5, 0], 'width':3, 'height':3,'orientation':90})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

Dans Main.py on construit notre en positionnant les faces du mur en fonction du repère du mur en leur donnant leurs tailles, leurs largeurs et leurs orientation respectifs.
Le code nous affiche :

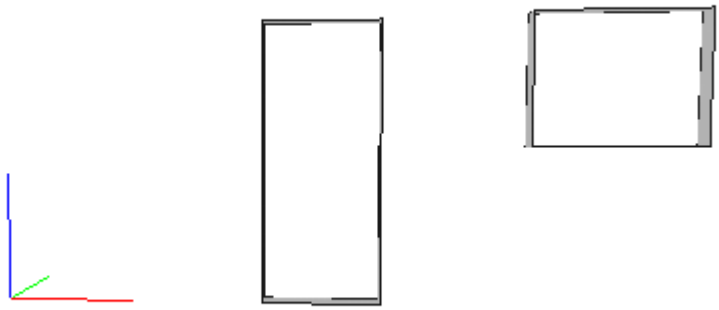


VII. Création d'ouvertures

Q5).a)

```
# Draws the faces
def draw(self):

  # condensé des draw et drawedges de la partie section. Sinon c'est le meme
  methode.
  gl.glPushMatrix()
  gl.glTranslatef(self.parameters['position'][0] , self.parameters['position']
  [1], self.parameters['position'][2])#on defini les translations possibles

  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) #pour créer les surfaces
  des ouvertures
  gl.glBegin(gl.GL_QUADS)
  gl.glColor3fv(self.parameters['color'])
  for i in self.faces:
    for j in i:
      gl.glVertex3fv(self.vertices[j])
  gl.glEnd()

  gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE) # on trace les aretes des
  ouvertures
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv([0.1,0.1,0.1]) # couleur plus foncé.

  #création des lignes comme pour la methode draw
  for i in self.faces:
    for j in i:
      gl.glVertex3fv(self.vertices[j])
  gl.glEnd()
  gl.glPopMatrix()
```

Avec le programme on compare les positions et les dimensions de la section et des ouvertures. La première et la seconde ouverture rentre dans la section que ce soit en hauteur et largeur. Mais la 3ieme ouverture qui à une position en z de 1.7 et une hauteur de 1 dépasse de la hauteur de la section. Ce pourquoi on nous renvoie True pour la première et la seconde et False pour la troisième.
Avec le programme on compare les positions et les dimensions de la section et des ouvertures.
Lorsque la méthode fonctionne correctement on nous affiche :



Q5)b)

```
def canCreateOpening(self, x):
    if x.parameters['position'][0]<self.parameters['position'][0] or x.parameters['width']
    +x.parameters['position'][0]>self.parameters['width']+self.parameters['position'][0]:
        return False #sur x: on renvoie faux si la position de la fenetre est avant la
        position du mur ou si la position de la fin de la fenetre est apres celle du mur.
    if x.parameters['position'][2]<self.parameters['position'][2] or x.parameters['height']
    +x.parameters['position'][2]>self.parameters['height']+self.parameters['position'][2]:
        return False #sur z: on renvoie faux si la position du bas de la fenetre est en
        dessous de la position du mur ou si la position du haut de la fenetre est au dessus
        de celle du mur.
    return True #si on a pas trouvé de chose à reprocher
```

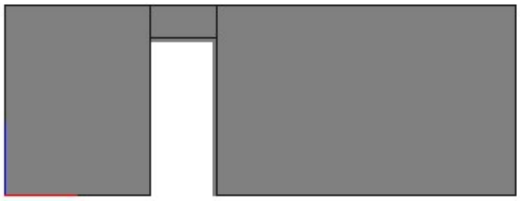
```
Console  Shell  Markdown

> python ./src/Main.py
pygame 2.0.0 (SDL 2.0.12, python 3.8.12)
Hello from the pygame community. https://
www.pygame.org/contribute.html
True
True
False
```

Les deux premières ouvertures sont dans la section mais la troisième ouverture dépasse en hauteur (1.7+1=2.7 >2.6).

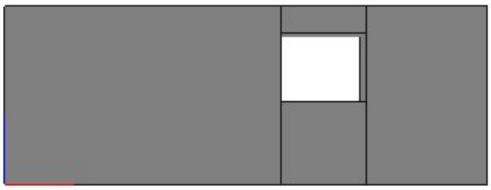
Q5)c)

```
# Creates the new sections for the object x
def createNewSections(self, x):
  liste=[]

  if (x.parameters['position'][0]-self.parameters['position'][0])>0 :

    s1=Section({'position':self.parameters['position'],'width':x.parameters['position']
    [0]-self.parameters['position'][0], 'height':self.parameters['height']})
    liste.append(s1)
    #section 1: uniquement si sa largeur >0. sa longeur et hauteur sont defini dans
    l'enoncé

  if (self.parameters['height']-x.parameters['height']-x.parameters['position'][2])>0:

    s2=Section({'position':[x.parameters['position'][0],self.parameters['position'][1],
    x.parameters['position'][2]+x.parameters['height']],'width':x.parameters['width'],
    'height':self.parameters['height']-x.parameters['height']-x.parameters['position']
    [2]})
    liste.append(s2)
    #section 2: sa largeur vaut la largeur de l'ouverture, sa hauteur vaut la hauteur
    de la section - la position de l'ouverture en y - sa hauteur. existe que si sa
    hauteur >0

  if (x.parameters['position'][2]-self.parameters['position'][2])>0:

    s3=Section({'position':[x.parameters['position'][0],self.parameters['position'][1],
    self.parameters['position'][2]],'width':x.parameters['width'], 'height':x.parameters
    ['position'][2]-self.parameters['position'][2]})
    liste.append(s3)
    #section 3: sa largeur vaut la largeur de l'ouverture, sa hauteur vaut la hauteur
    de la section - la position de l'ouverture en y - la hauteur de l'ouverture.existe
    que si sa hauteur >0

  if (self.parameters['width']+self.parameters['position'][0]-x.parameters['width']
  +x.parameters['position'][0])>0:

    s4= Section({'position':[x.parameters['position'][0]+x.parameters['width'],
    self.parameters['position'][1],self.parameters['position'][2]],
    'width':self.parameters['width']-x.parameters['position'][0]-x.parameters['width'],
    'height':self.parameters['height']})
    liste.append(s4)
    #section 4:Sa position vaut la position de l'ouverture + sa largeur. Sa hauteur
    vaut la hauteur de la section sa largeur vaut la largeur de la section moins la
    largeur de l'ouverture moins la position en x de l'ouverture. existe que si sa
    largeur > 0.

  return liste #on renvoie la liste
```

c)1



c)2)



Q5).d)

La fonction enumerate() permet d'associer un numéro à chaque objet dans enumerate. Le compteur commence à 0 par défaut si on ne précise pas. Cela permet de créer un tuple contenant le compteur et l'objet.