LACROIX Clara
LAUBERT Théoxane
ROJON Florian
Date du TP (01/12/2021)

Rapport de TPX – Lecture automatique de chiffres par analyse d'image

I. Présentation du TP

II. Préparation à faire avant le TP

(1)

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

Lorsqu'on execute ce programme, il y a une fenêtre (de 300 x 200) qui s'ouvre et qui se referme immédiatement après.

Dans ce programme, on importe la fonction Pygame qu'on execute.
Cette fonction est un moteur de jeu.

(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()
```

Lorsque la fonction s'execute, une fenêtre (de 300 x 200) s'ouvre et ne se ferme pas. Si on sélectionne la fenêtre et on appuie sur une touche du clavier, la fenêtre se ferme.

Dans ce programme, on fait appel à la fonction pygame.event, celle-ci permet de paramétrer les événements entrants et sortants.

Utilisation de Pyopengl pour représente des objet 3D

(1) Lorsqu'on exécute ce code, une fenêtre pygame s'ouvre, et on peut la fermer uniquement si on appuie sur la croix.
La fonction gluPerspective permet d'initialiser une matrice de perspective avec des paramètres donnés dans le sujet : fovy = 45°, aspect = display[0] / display [1], zNear = 0,1, zFar = 50

```
glu.gluPerspective(45, (display[0]/display[1]), 0.1, 50)
```

(2)

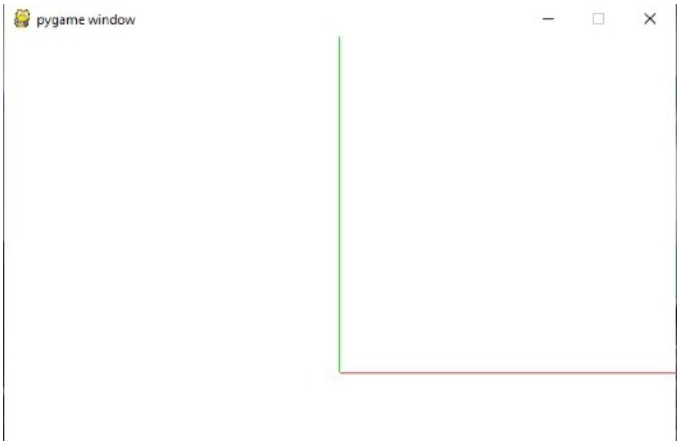
```
glu.gluPerspective(45, (display[0]/display[1]), 0.1, 50)

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

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

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))

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, -1))

gl.glEnd() # Find du tracé
pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

premier segment (rouge)

deuxième segment (vert)

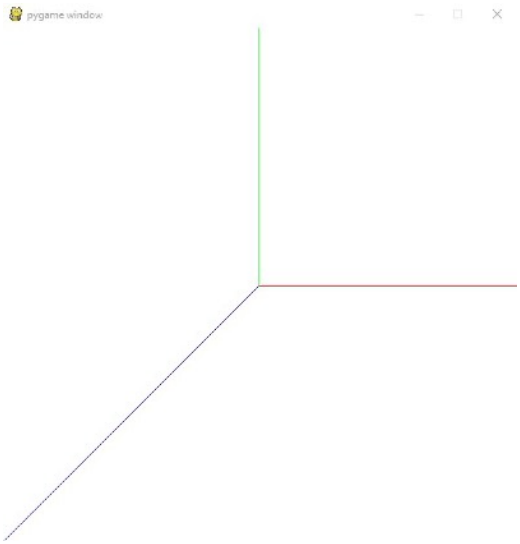
troisième segment (bleu)

Ce programme affiche le graphique suivant :

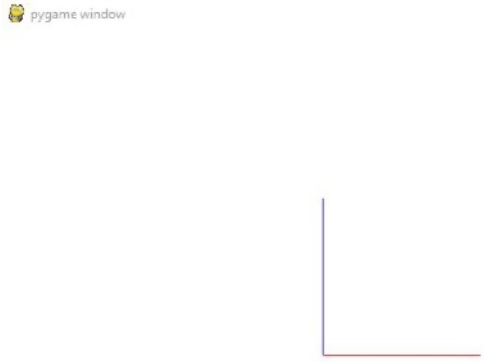


Avec ce choix de glVertex3fv au troisième segment on ne voit pas le segment bleu.
Pour le faire apparaitre, il faut modifié : gl.glVertex3fv(-1, -1, -1)

Avec se changement voici le nouveau graphique :



(3) On a ajouté la fonction gluTranslatef et glRotatef après la fonction gluPerspective, elle sert à faire une translation et une rotation, on a ensuite affiché la translation et la rotation :



Découverte de l'environnement du travail du TP

(1)

a. On analyse le fichier configuration.py :

0-14 : Les premières lignes sont des importations de modules.

14-50 : la classe est initialisée: avec la paramétrisation des axes (selon la direction, leur couleur n'est pas la même) et de la fenêtre d'affichage.
ensuite les modules pygame, OpenGL, les fonctions de transformations et la liste des objets sont initialisés.
Des coordonnées sont générées.

53-58 : définition de la fonction d'initialisation de pygame : position, lancement et affichage.

61-67 : définition de glOpen : la couleur blanche est initialisée. L'utilisateur ne va voir que les surfaces apparentes.

70-77 : les fonctions openGL initialisent les matrices de transformations

80-81 : définition de la fonction permettant d'afficher les paramètres (sans passer par le constructeur).

84-88 : fonction qui effectue les réglages des paramètres dans l'espace et qui réalise une transformation.

90-102 : fonction qui génère les coordonnées dans l'espace (vertices -> arrêtes, edges -> sommets).

105-107 : fonction de surcharge de l'opérateur addition

110-135 : fonction qui réalise l'affichage des axes.

138-148 : définition de la fonction qui prend en compte les différentes possibilités de mouvements des axes. Chacune de ces interactions est régit par une une touche du clavier : 'z' fait tourner l'axe dans un sens, 'Z' dans l'autre, 'a' montre ou cache l'axe.

151-156 : fonction qui résulte sur les événements liés à l'appui du bouton sur la souris. (Fonction vierge)

159-194 : fonction qui paramètre les événements entrants et sortants, (affichage et coordination)

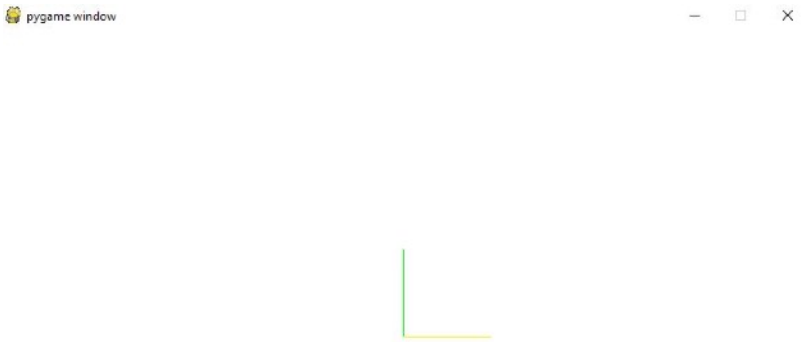
On analyse le fichier main.py :

Le fichier importe les modules nécessaires et comporte les fonctions des questions.

Avec la ligne rajouter, l'affichage est beaucoup plus éloigné, ce qui permet d'avoir une vue d'ensemble.

b.

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



L'axe des « x » est modifié, initialement il était rouge, on lui rajoute du vert ce qui résulte sur du jaune.

Le chaînage de l'appel des méthodes setParameter() et display() est possible car les deux méthodes appartiennent à la meme classe : configuration.

« \ » permet de sauter une ligne plutôt que d'écrire tout le code à la suite.

c.

On veut un changement d'axe : donc il faut utiliser une rotation : afin que l'axe y soit en profondeur

```
# Initializes the tranformation matrix
self.initializeTransformationMatrix()
gl.glRotatef(-90, 1, 0, 0)
```

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

d. On utilise la fonction glScalef(a, b, c) avec a, b et c les coefficient d'agrandissement ou de réduction des axes x, y et z.

Les codes correspondants à page up et page down respectivement sont K_PAGEUP et K_PAGEDOWN

```
148     # Processes the KEYDOWN event
149     def processKeyDownEvent(self):
150         # Rotates around the z-axis
151         if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
152             gl.glRotate(-2.5, 0, 0, 1)
153         elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
154             gl.glRotate(2.5, 0, 0, 1)
155 
156         # Draws or suppresses the reference frame
157         elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
158             self.parameters['axes'] = not self.parameters['axes']
159             pygame.time.wait(300)
160 
161 
162         elif self.event.dict['unicode'] == 'K_PAGEUP' or self.event.key == pygame.K_PAGEUP :
163             gl.glScalef(1.1, 1.1, 1.1)
164 
165         elif self.event.dict['unicode'] == 'K_PAGEDOWN' or self.event.key == pygame.K_PAGEDOWN :
166             gl.glScalef(1/1.1, 1/1.1, 1/1.1)
167 
```

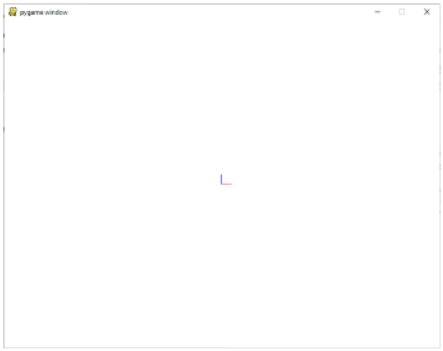
Les lignes de code 162-163 correspondent au zoom et les lignes 165-166 correspondent au dé-zoom.

e. Même principe que précédemment, mais le zoom pourra se régler à l'aide la molette de la souris.

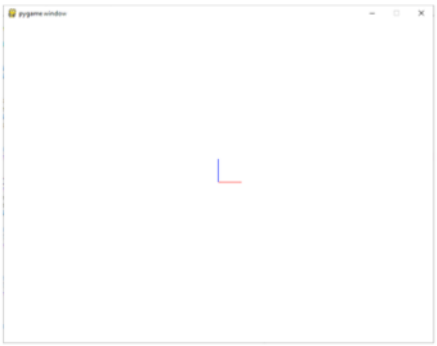
```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.button == 4 :
        gl.glScalef(1.1, 1.1, 1.1)
    elif self.event.button == 5 :
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Résultat du code :

Mode dé-zoomé :



Mode normal :



Mode zoomé :

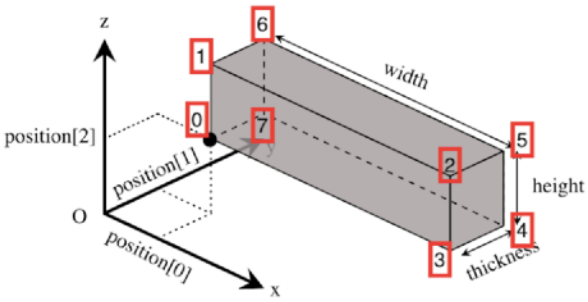
f. Dans la fonction glTranslatef on ajoute “-“ devant self afin d’aller dans le même sens que la souris (empirique)

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

Résultat du code :

IV. Création de section

(2) .a) Schéma annoté :



Code :

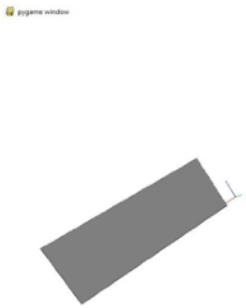
```
# Defines the vertices and faces
def generate(self):
    self.vertices = [
            [0, 0, 0 ],
            [0, 0, self.parameters['height']],
            [self.parameters['width'], 0, self.parameters['height']],
            [self.parameters['width'], 0, 0],
            [self.parameters['width'],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

(2) .b) Code :

Configuration().add(Section(...))
On récupère la fonction add dans le dossier configuration, on va lui faire prendre en paramètre une fonction définie, où on pourra faire apparaitre le résultat à l'aide de la fonction display().

```
130     # Draws the faces
131     def draw(self):
132         # A compléter en remplaçant pass par votre code
133         self.drawEdges()
134         gl.glPushMatrix()
135         gl.glTranslatef(self.parameters['position'][1], self.parameters['position'][0], self.parameters['position'][2])
136
137         gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
138
139         for i in self.faces :
140             gl.glBegin(gl.GL_QUADS)
141             gl.glColor3fv([0.5, 0.5, 0.5])
142             for j in i:
143                 gl.glVertex3fv(self.vertices[j])
144             gl.glEnd()
145
146         gl.glPopMatrix()
```

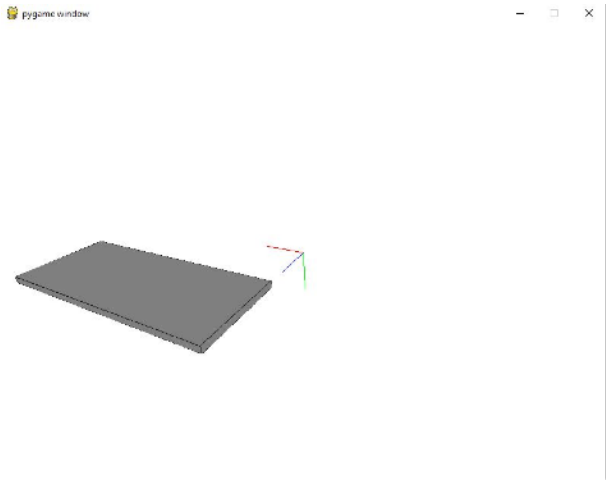
Résultat du code :



(2) .c) Code :

```
113     # Draws the edges
114     def drawEdges(self):
115         # A compléter en remplaçant pass par votre code
116         gl.glPushMatrix()
117         gl.glTranslatef(self.parameters['position'][1], self.parameters['position'][0], self.parameters['position'][2])
118
119         gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les lignes : GL_LINE
120
121         for i in self.faces :
122             gl.glBegin(gl.GL_QUADS)
123             gl.glColor3fv([0.1, 0.1, 0.1])
124             for j in i:
125                 gl.glVertex3fv(self.vertices[j])
126             gl.glEnd()
127
128         gl.glPopMatrix()
```

Résultat du code :



V. Création des murs

(3) .a)
Analyse du fichier Wall.py :

12 - 46 : fonction d'initialisation
La fonction définit une valeur initiale pour la position, la largeur, la hauteur, l'orientation, l'épaisseur et la couleur.

38 : La fonction crée une liste objet.

41 - 45 : élabore la section principale (partie du mur).

46 : ajout de la section principale (précédemment créée) dans la liste d'objet.

49 - 50 : surcharge de l'opérateur getParameter dans le but d'avoir accès aux paramètres du mur.

53 - 55 : surcharge de l'opérateur setParameter afin de changer les paramètres de la fonction rapidement.

58 - 62 : renvoie l'emplacement où un objet peut être inséré.

65 - 70 : fonctions supplémentaires non finies, dont la finalité est de permettre l'ajout de l'objet à un certain emplacement et dessine les faces.

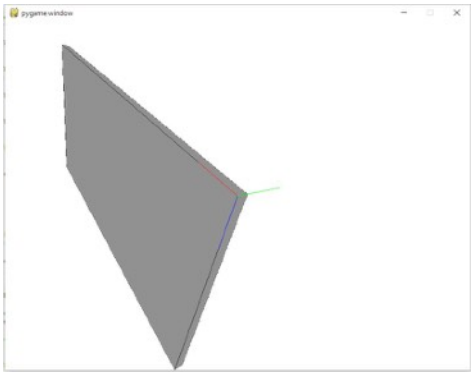
Code dans Wall.py :

```
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)

    for x in self.objects:
        x.draw()

    gl.glPopMatrix()
```

Code dans main.py :

```
def Q3a():
    return Configuration().add(
            Wall({'position' : [0, 0, 0], 'width' : 10, 'height' : 5})
            )
```

Résultat des codes :



VI. Création d'une maison

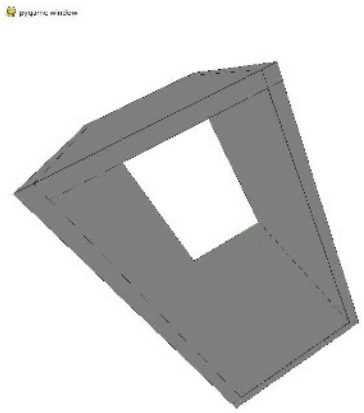
Code dans House.py :

```
41      # Draws the house
42      def draw(self):
43          # A compléter en remplaçant pass par votre code
44          gl.glPushMatrix()
45          gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
46          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
47
48          for x in self.objects:
49              x.draw()
50
51          gl.glPopMatrix()
```

Code dans main.py :

```
48 def Q4a():
49     # Ecriture en utilisant des variables : A compléter
50     wall1 = Wall({'position' : [0, 0, 0], 'width' : 5, 'height' : 5, 'orientation' : 0})
51     wall2 = Wall({'position' : [0, 0, 0], 'width' : 3, 'height' : 5, 'orientation' : 90})
52     wall3 = Wall({'position' : [2.8, 0, 0], 'width' : 5, 'height' : 5, 'orientation' : 0})
53     wall4 = Wall({'position' : [-5, 0, 0], 'width' : 2.8, 'height' : 5, 'orientation' : 90})
54     house = House({'position': [-3, 1, 0], 'orientation':0})
55     house.add(wall1).add(wall3).add(wall4).add(wall2)
56     return Configuration().add(house)
```

Résultat des codes :



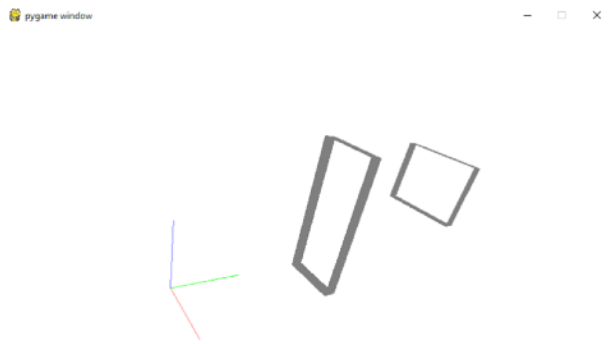
VII. Création des ouvertures

(5) .a)

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]
            ]
```

Résultat du code :



(5) .b)

Code :

```
72    # Checks if the opening can be created for the object x
73    def canCreateOpening(self, x):
74        # A compléter en remplaçant pass par votre code
75        if x.parameters['width']+x.parameters['position'][0] <= self.parameters['position'][0] + self.parameters['width']:
76            if x.parameters['height'] + x.parameters['position'][2] <= self.parameters['position'][2] + self.parameters['height']:
77                return True
78        return False
```

Explication du code :

75 : on test afin que le trou dans le mur ne dépasse pas la largeur du mur.

76 : on test que la hauteur du trou n'est pas plus haute que le mur.

77 : résultat des tests, si le trou est bien plus petit que le mur dans les dimensions, retourne True.

78 : dans le cas contraire renvoie False.

Résultat du code :

```
In [30]: runfile('E:/INFO TP/tp3/src1/Main.py', wdir='E:/INFO TP/tp3/src1')
Reloaded modules: Configuration, Section, Wall, Door, Window, Opening, House
True
True
False
```

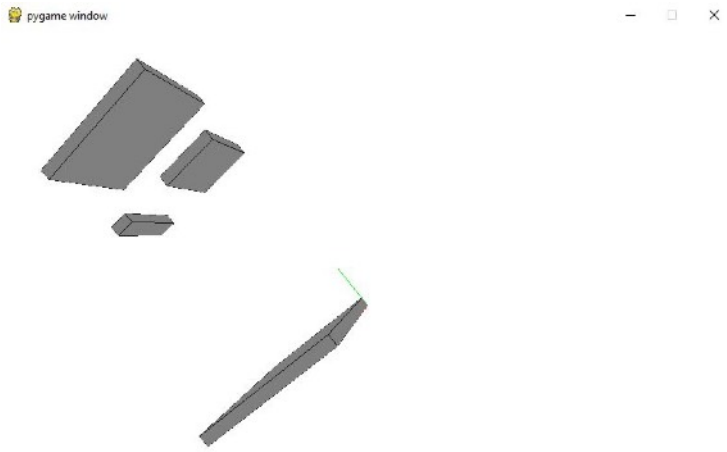
Ici nous avons un 3eme résultat : car le trou dans le mur appartient à celui-ci en partie mais pas totalement.

(5) .c)

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):
        L = []
        h_0 = self.parameters['height']
        w_1 = self.parameters['width'] - x.parameters['position'][0]
        x_2 = x.parameters['position'][0]
        z_2 = x.parameters['position'][2] + x.parameters['height']
        w_2 = x.parameters['width']
        h_2 = self.parameters['height'] - z_2
        x_3 = x_2
        w_3 = w_2
        h_3 = x.parameters['position'][2]
        x_4 = x.parameters['width'] + x.parameters['position'][0]
        w_4 = self.parameters['width'] - x.parameters['position'][0] - x.parameters['width']
        if w_1 != 0:
            section1 = Section({'position' : [0, 0, 0], 'width' : w_1, 'height' : h_0})
            L.append(section1)
        if h_2 != 0 :
            section2 = Section({'position' : [x_2, 0, z_2], 'width' : w_2, 'height' : h_2})
            L.append(section2)
        if h_3 != 0 :
            section3 = Section({'position' : [x_3, 0, 0], 'width' : w_3, 'height' : h_3})
            L.append(section3)
        if w_4 != 0 :
            section4 = Section({'position' : [x_4, 0, 0], 'width' : w_4, 'height' : h_0})
            L.append(section4)
        return L
```

Résultat du code :



(5). d)

La fonction enumerate() sert à compter le nombre de sections disponibles. Ainsi on pourra réaliser un certain nombre d'insertion.