FASKA Rachid
SAIDI Mokhliss
IDU3

Compte rendu TP3

I/ PRÉPARATION AVANT TP

a) Utilisation de Pygame

(1)

```
1    import pygame
2    pygame.init()
3    ecran = pygame.display.set_mode((300, 200))
4    pygame.quit()
```

La ligne 1 permet d'importer tous les modules pygame disponible. La ligne 2 permet d'initialiser chaque module importé. La fonction display.set_mode() de la ligne 3 crée un nouvel objet qui représente les graphiques affichés. La ligne 4 réinitialise tous les modules pygame. On voit qu'une fenêtre se ferme en moins d'une seconde.

(2)

```
1    import pygame
2
3
4    pygame.init()
5    ecran = pygame.display.set_mode((300, 200))
6
7    continuer = True
8    while continuer:
9        for event in pygame.event.get():
10           if event.type == pygame.KEYDOWN:
11               continuer = False
12
13   pygame.quit()
```

En exécutant ce programme, une fenêtre s'ouvre mais en appuyant sur un bouton quelconque du clavier, la fenêtre se ferme. La ligne 9 permet de « récupérer » tous les évènements. La ligne 11 fait passer la variable *continuer* à *false* si un évènement de type « appuyer sur une touche du clavier »(ligne 9) est réalisé. Donc tant que l'on n'a pas touché un bouton du clavier, la fenêtre restera ouverte.

b) Utilisation de Pyopengl pour représenter des objets 3D

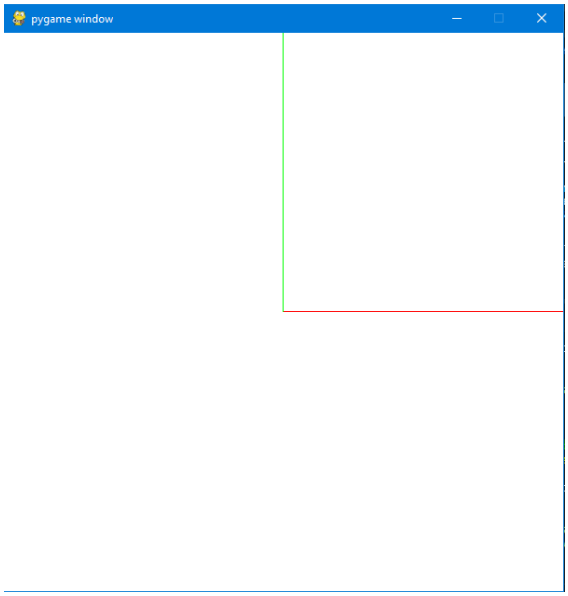
(1)

```
1    import pygame
2    import OpenGL.GL as gl
3    import OpenGL.GLU as glu
4
5    if __name__ == '__main__':
6        pygame.init()
7        display=(600,600)
8        pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
9
10       # Sets the screen color (white)
11       gl.glClearColor(1, 1, 1, 1)
12       # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
13       gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
14       gl.glEnable(gl.GL_DEPTH_TEST)
15
16       glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
17
18       while True:
19           for event in pygame.event.get():
20               if event.type == pygame.QUIT:
21                   pygame.quit()
```

(2)

```
gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes
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])
gl.glVertex3fv((0,0, -2))
gl.glVertex3fv((0, 1, -2))
gl.glColor3fv([0, 0, 1])
gl.glVertex3fv((0,0, -2))
gl.glVertex3fv((0, 0, -2))
gl.glEnd() # Find du tracé
pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

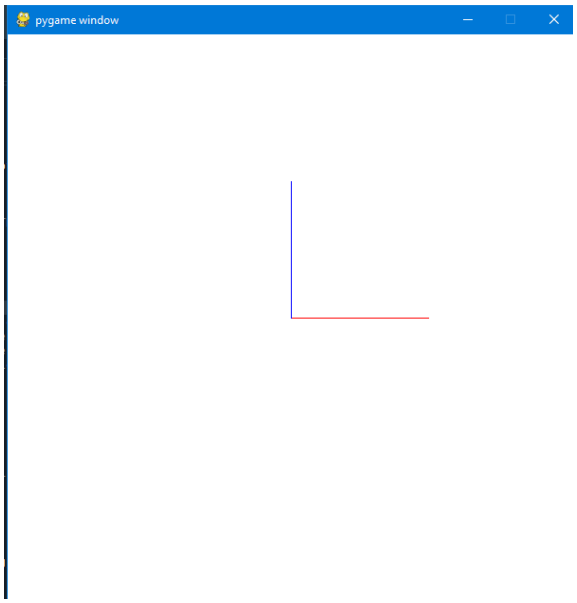
Résultat obtenu :



(3)

Résultat après translation :

Résultat après rotation :



c) Découverte de l'environnement du travail du TP

(1).c

```
def initializeTransformationMatrix(self):
    gl.glMatrixMode(gl.GL_PROJECTION)
    gl.glLoadIdentity()
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)
    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glLoadIdentity()
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
    gl.glRotatef(-90, 1, 0, 0)
```

On a rajouté la fonction *glRotatef* afin de pouvoir faire une rotation selon l'axe x.

Résultat obtenu :

II/ MISE EN PLACE DES INTÉRACTIONS AVEC L'UTILISATEUR AVEC PYGAME

1.d)

```
137     # Processes the KEYDOWN event
138     def processKeyDownEvent(self):
139         # Rotates around the z-axis
140         if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
141             gl.glRotate(-2.5, 0, 0, 1)
142         elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
143             gl.glRotate(2.5, 0, 0, 1)
144         # Zoom or dezoom
145         elif self.event.key==pygame.K_PAGEDOWN:
146             gl.glScalef(1/1.1,1/1.1,1/1.1)
147         elif self.event.key==pygame.K_PAGEUP:
148             gl.glScalef(1.1,1.1,1.1)
149         # Draws or suppresses the reference frame
150         elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
151             self.parameters['axes'] = not self.parameters['axes']
152             pygame.time.wait(300)
```

De la ligne 145 à 148 on permet le zoom ou le dezoom par les touches *PageUp* et *PageDown* du clavier.

1.e)

```
154     # Processes the MOUSEBUTTONDOWN event
155     def processMouseButtonDownEvent(self):
156         if self.event.button==4:
157             gl.glScalef(1.1,1.1,1.1)
158         if self.event.button==5:
159             gl.glScalef(1/1.1,1/1.1,1/1.1)
```

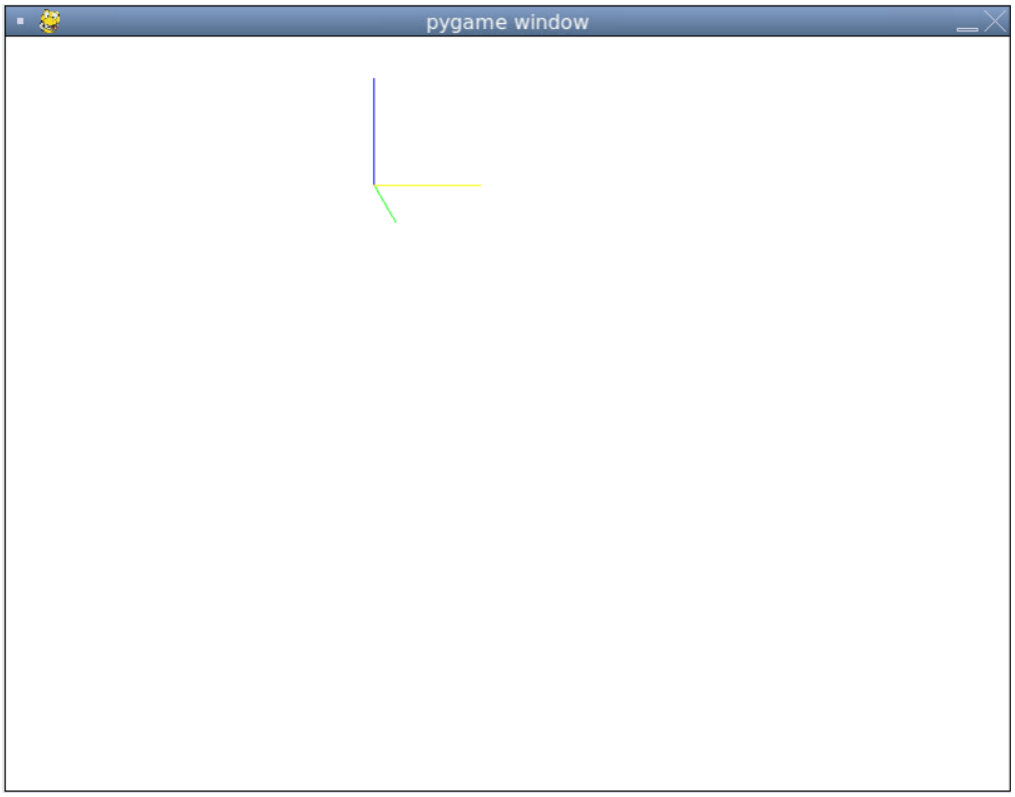
On veut obtenir un effet de zoom en utilisant la molette de la souris. Pour cela, on se sert de l'attribut *button* et du sens de l'action sur la molette associé à un évènement qui prend pour valeur 4 ou 5 selon le sens de l'action.

1.f)

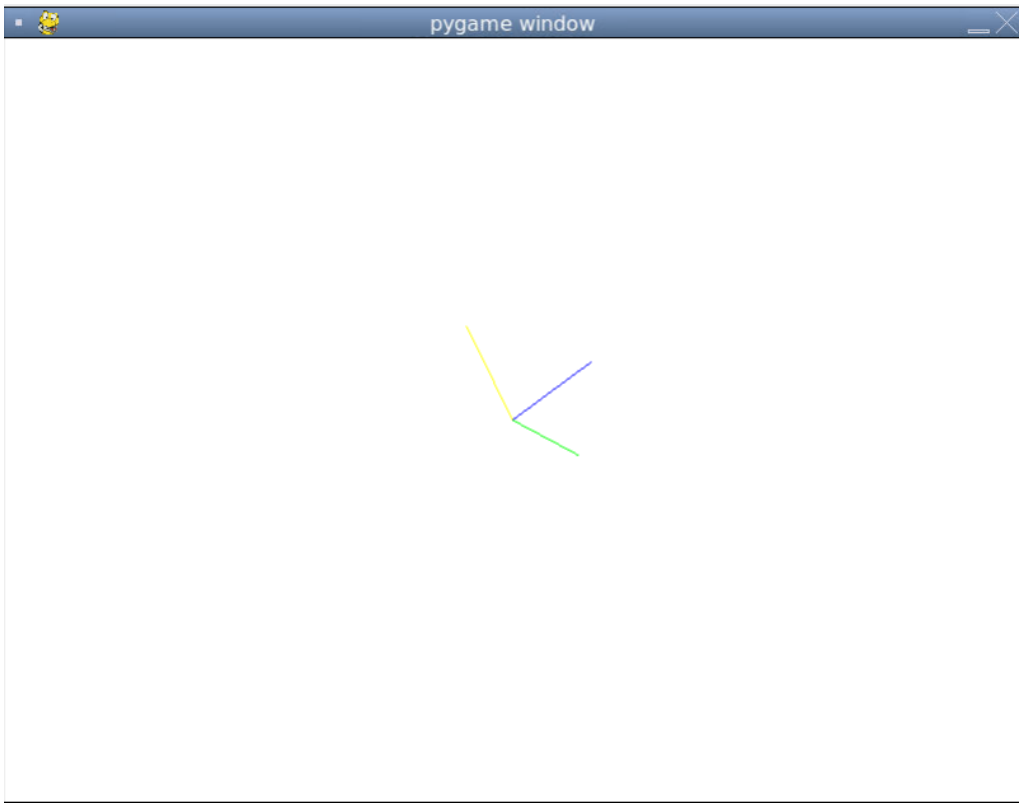
```
161     # Processes the MOUSEMOTION event
162     def processMouseMotionEvent(self):
163         if pygame.mouse.get_pressed()[0]==1:
164             x,z=self.event.rel
165             gl.glRotatef(x, 1, 0, 0)
166             gl.glRotatef(z, 0, 0, 1)
167         if pygame.mouse.get_pressed()[2]==1:
168             x,z=self.event.rel
169             gl.glTranslatef(x/25, 0, 0)
170             gl.glTranslatef(0,0,z/25)
```

La question 1)f a pour but de nous permettre de faire une rotation ou une translation quelconque par le déplacement de la souris et l'appui du bouton droit ou gauche. On utilise alors *pygame.mouse.get_pressed()* pour savoir sur quel bouton sommes nous en train d'appuyer. On utilise *self.event.rel* pour permettre un déplacement quelconque selon l'axe x et z. Ensuite, selon le bouton sur lequel on appuie, soit on fait une translation avec *gl.glTranslatef* soit on fait une rotation avec *gl.glRotatef*. On divise par 25 pour ne pas avoir une translation « trop rapide ».

Résultat obtenu après une translation quelconque :



Résultat obtenu après une rotation quelconque :



III/ CRÉATION D'UNE SECTION

2.a)

```
53      # Defines the vertices and faces
54      def generate(self):
55          self.vertices = [
56                      [0, 0, 0 ],
57                      [0, 0, self.parameters['height']],
58                      [self.parameters['width'], 0, self.parameters['height']],
59                      [self.parameters['width'], 0, 0],
60                      [0,self.parameters['thickness'],0],
61                      [0,self.parameters['thickness'],self.parameters['height']],
62                      [self.parameters['width'],self.parameters['thickness'],self.parameters['height']],
63                      [self.parameters['width'],self.parameters['thickness'],0]
64                      ]
65          self.faces = [
66                      [0, 3, 2, 1],
67                      [4,7,6,5],
68                      [0,3,7,4],
69                      [1,2,6,5],
70                      [3,7,6,2],
71                      [0,4,5,1]
72                      ]
```

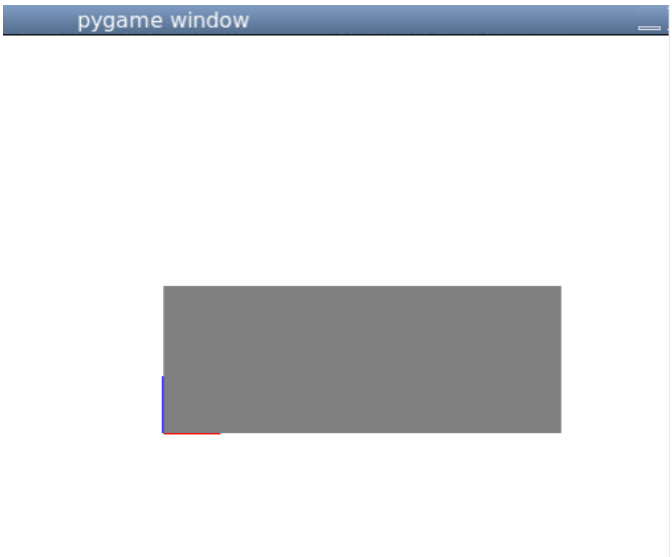
2.b)

- L'instruction *Configuration().add(section).display()* permet de rajouter des paramètres de la classe *Section* à la classe *Configuration* et de créer un objet de type *Configuration* avec les paramètres ajoutés.

```
89      # Draws the faces
90      def draw(self):
91          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
92          gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
93          gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
94          gl.glVertex3fv([0, 0, 0])
95          gl.glVertex3fv([self.parameters['width'], 0, 0])
96          gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
97          gl.glVertex3fv([0, 0, self.parameters['height']])
98          gl.glEnd()
```

- Le but de cette question est de pouvoir représenter une section. Pour cela, on réutilise le code donné en exemple mais on remplace les 1 par *self.parameters['width']* ou par *self.parameters['height']* selon la position du 1 dans *glVertex3fv* pour être en accord avec la fonction *Q2b()* du fichier *Main* où l'on a ajouté les paramètres *width* et *height*.

Résultat obtenu :



2.c)

Méthode *drawEdges* :

```
84      # Draws the edges
85      def drawEdges(self):
86          gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
87          gl.glBegin(gl.GL_QUADS)
88          gl.glColor3fv([0.25, 0.25, 0.25])
89          gl.glVertex3fv([0, 0, 0])
90          gl.glVertex3fv([self.parameters['width'], 0, 0])
91          gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
92          gl.glVertex3fv([0, 0, self.parameters['height']])
93          gl.glEnd()
```

C'est le même principe que la méthode précédente mais ici on utilise *gl.GL_LINE* et non pas *gl.GL_FILL* et on modifie la couleur (ici on a multiplié par 0.5).

Modification de *draw* :

Pour que la méthode *drawEdges()* soit exécutée en premier lorsque le paramètre *edges*, il faut rajouter ces deux lignes au début de la méthode *draw()* :

```
if self.setParameter('edges', True):
  self.drawEdges()
```

Résultat obtenu :

Des bords noirs sont apparus.

IV/ CRÉATION DES MURS

3.a)

On remarque que dans le constructeur de la classe *Wall*, un objet de type *Section* est associé aux objets de type *Wall* avec *self.parentSection = Section(…)* .

Méthode *draw()* :

```
71      # Draws the faces
72      def draw(self):
73          gl.glPushMatrix()
74          gl.glRotate(self.parameters['orientation'],0,0,1)
75          gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
76          for x in self.objects:
77              x.draw()
78          gl.glPopMatrix()
```

Dans la méthode *draw()*, on utilise une boucle for pour ajouter successivement les objets contenus dans l'attribut *objects* dans lequel chaque section parente est ajouté. De plus, dans le premier élément de *glRotate* on utilise *self.parameters['orientation']* pour pouvoir modifier l'orientation des murs.

```
40              # Adds a Section for this object
41              self.parentSection = Section({'width': self.parameters['width'], \
42                                            'height': self.parameters['height'], \
43                                            'thickness': self.parameters['thickness'], \
44                                            'color': self.parameters['color'],
45                                            'position': self.parameters['position'],
46                                            'edges': self.parameters['edges'],
47                                            'orientation': self.parameters['orientation']})
48              self.objects.append(self.parentSection)
```

Dans l'attribut *parentSection*, on ajoute le paramètre *edges* pour pouvoir représenter les contours des murs et le paramètre *orientation* pour pouvoir le modifier lorsque l'on construit un objet de type *Wall*.

```
39  def Q3a():
40      wall0 = Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'edges': True})
41      Configuration().add(wall0).display()
```

Dans la fonction *Q3a()* du *Main*, on crée une instance de type *Wall*.

IV/ CRÉATION D'UNE MAISON

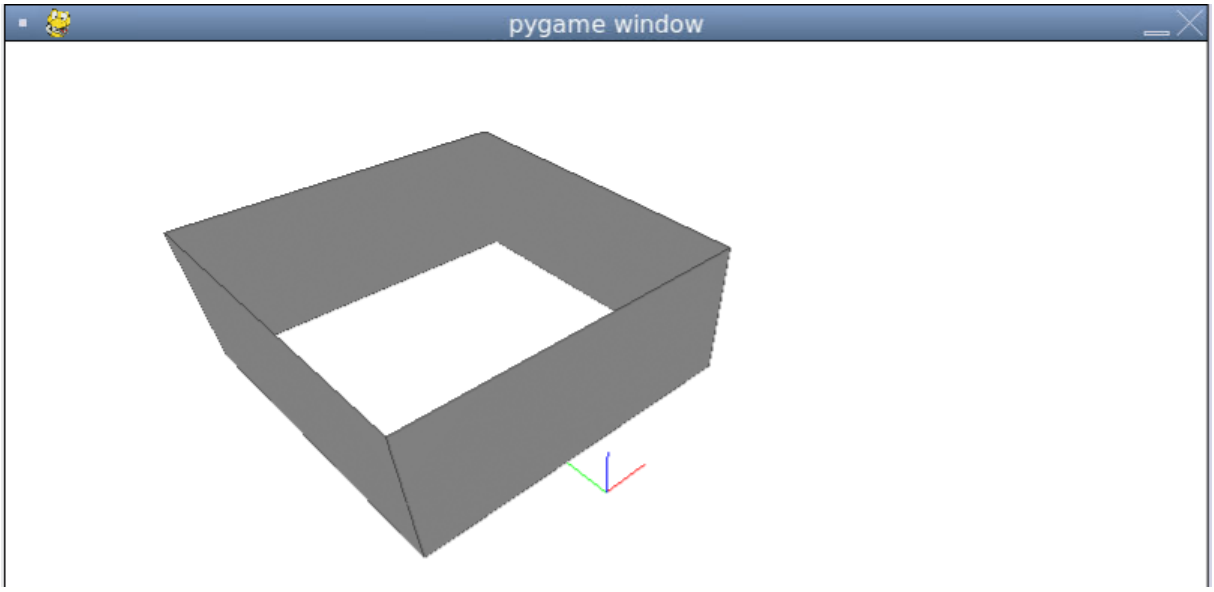
Méthode *draw()* :

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

Q4a) dans le main :

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

Résultat obtenu :



On obtient bien le résultat attendu avec le décalage demandé.

V/ CRÉATION D'OUVERTURES

5.a)

Méthode *generate()* :

```
48      def generate(self):
49          self.vertices = [
50                      [0, 0, 0 ],
51                      [0, 0, self.parameters['height']],
52                      [self.parameters['width'], 0, self.parameters['height']],
53                      [self.parameters['width'], 0, 0],
54                      [0,self.parameters['thickness'],0],
55                      [0,self.parameters['thickness'],self.parameters['height']],
56                      [self.parameters['width'],self.parameters['thickness'],self.parameters['height']],
57                      [self.parameters['width'],self.parameters['thickness'],0]
58                      ]
59          self.faces = [
60                      [0,4,5,1],
61                      [3,7,6,2],
62                      [0,3,7,4],
63                      [1,2,6,5]
64                      ]
```

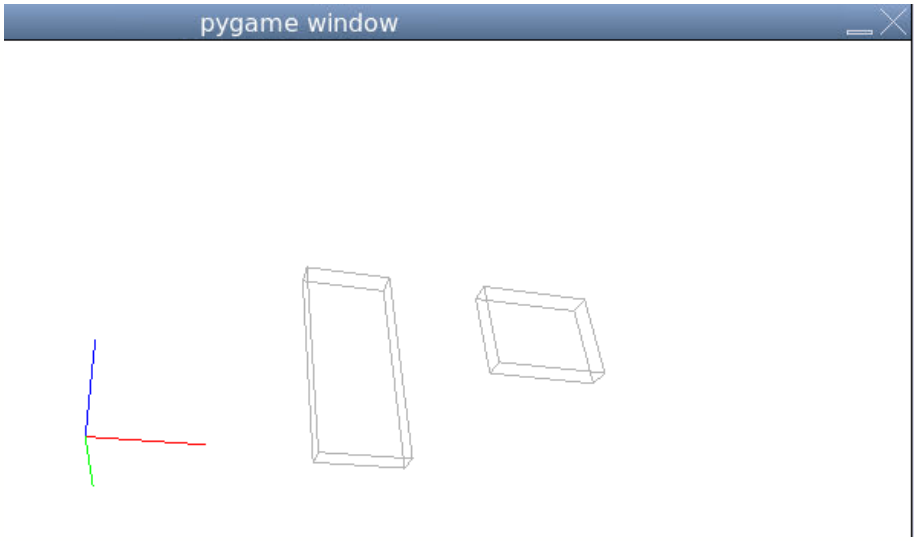
On définit *self.vertices* et *self.*faces de la même manière que la question 2)a).

Méthode *draw()* :

```
67      def draw(self):
68          gl.glPushMatrix()
69          gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
70          gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
71          gl.glBegin(gl.GL_QUADS)
72          gl.glColor3fv(self.parameters['color'])
73          for i in range (len(self.faces)):
74              for nombre in self.faces[i]:
75                  gl.glVertex3fv(self.vertices[nombre])
76          gl.glEnd()
77          gl.glPopMatrix()
```

Un sommet est spécifié par chaque élément de chacune des listes contenues dans self.faces et la double boucle *for* permet d'obtenir la figure demandée.

Résultat obtenu :



5.b)

Méthode *canCreateOpening()* :

```
75      def canCreateOpening(self, x):
76          L=x.parameters['width']+x.parameters['position'][0]
77          H=x.parameters['height']+x.parameters['position'][2]
78          return (L<=self.parameters['width']+self.parameters['position'][0] and H<=self.parameters['height']+self.parameters['position'][2])
```

Il suffit de comparer la hauteur totale et la largeur totale d'une ouverture représentée par un objet *x*, avec la hauteur et la largeur d'une section.

Résultat obtenu :

```
> 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
```

Le dernier affichage vaut *False* car la somme de la position en hauteur de l'ouverture avec la hauteur de l'ouverture dépasse 2.6 qui est la hauteur de la section.

5.c)

Méthode *createNewSections()* :

```
80      # Creates the new sections for the object x
81      def createNewSections(self, x):
82          #test si possibilité d'ouverture
83          if self.canCreateOpening(x):
84              #création liste de sortie
85              new_section=[]
86              #création des variables
87              xS,yS,zS=self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2]
88              w=self.parameters['width']
89              h=self.parameters['height']
90              xO,zO=x.parameters['position'][0],x.parameters['position'][2]
91              wO=x.parameters['width']
92              hO=x.parameters['height']
93
94              #test création de la section droite
95              if xO-xS>0 :
96                  new_section.append(Section({'position':[xS,yS,zS], 'width': xO-xS, 'height': h}))
97
98              #test création de la section gauche
99              if (xS+w)-(xO+wO)>0:
100                 position=[(xO+wO),yS,0]
101                 new_section.append(Section({'position':position, 'width': (xS+w)-(xO+wO), 'height': h}))
103             #test création de la section basse
104             if (zO)-(zS)>0:
105                 position=[xO,yS,0]
106                 new_section.append(Section({'position':position, 'width': wO, 'height': (zO)-(zS)}))
107
108             #test création de la section haute
109             if (zS+h)-(zO+hO)>0:
110                 position=[xO,yS,(zO)-(zS)+hO]
111                 new_section.append(Section({'position':position, 'width': wO, 'height':  (zS+h)-(zO+hO)}))
112         return new_section
```

Résultat obtenu :