SARRE Valentin
MAZZOCHI Léo
Date du TP (06/12/2021)

Rapport de TP3 - Représentation visuelle d'objets.

Comme convenu avec notre prof, nous vous rendons le TP3 avec un peu de retard suite à la perte de notre premier document, car nous attendu la réponse du SOS polytech

I) Préparation du TP

1) Test du premier code

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

1ere ligne : importation du module pygame
2eme ligne : initialisation de pygame
3eme ligne : création d'une fenêtre pygame de dimension 300x200
4eme ligne : désinitialise le module pygame

Lorsqu'on exécute le programme, cela ouvre une fenêtre de 300*200 qui se referme automatiquement.

2) Test du deuxième code

```
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()
```

Une variable booléenne continuer est initialisée à True, qui permet de sortir de pygame à partir du moment où elle passe à False. La variable continuer passe à False dès que l'évènement Keydown est réalisé
La fenêtre reste ouverte et se ferme uniquement si on appuie sur une touche du clavier.

Utilisation de Pyopengl :

1) Ecriture de la ligne glu

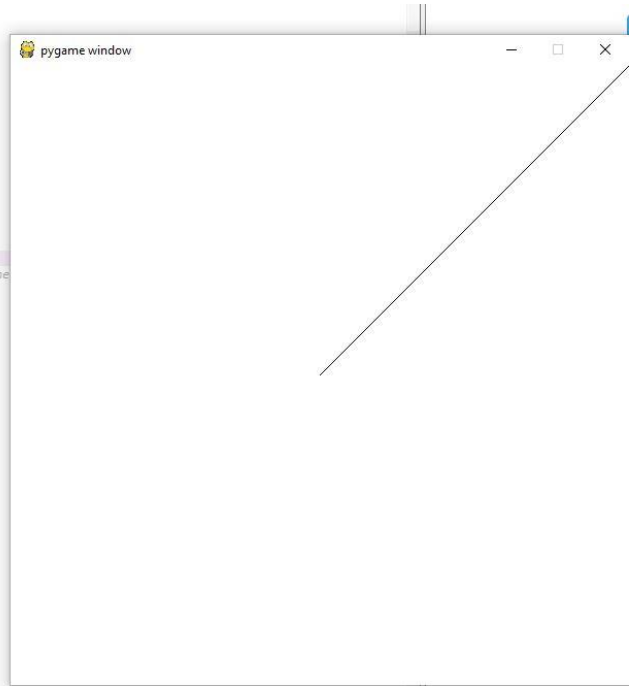
Avec ce code, une fenêtre vide s'ouvre, la seule interraction que nous pouvons faire avec est de la fermer en cliquant sur la croix grâce à ligne if event.type == pygame.QUIT

La ligne rajoutée correspond à la méthode (fovy, aspect, zNear, zFar) qui initialise une matrice de perspective.

```
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)

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

2) Tracer

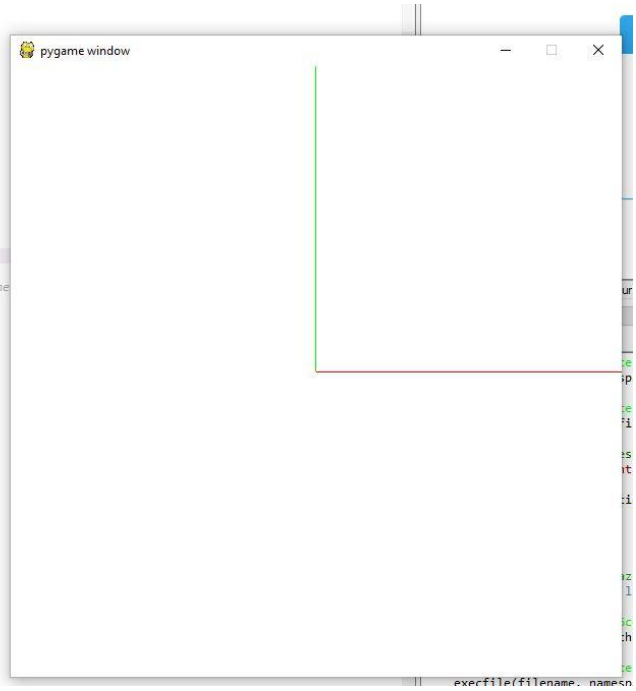
```python
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

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)

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

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

    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

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



On obtient un trait noir car la fonction glColor3fv est réglée au code couleur [0, 0, 0], le trait est en diagonale car la fonction glVertex3fv définit le départ du trait et la fin.

Grâce à cet exemple, on trace l'axe X en rouge, Y en vert et Z en bleu.

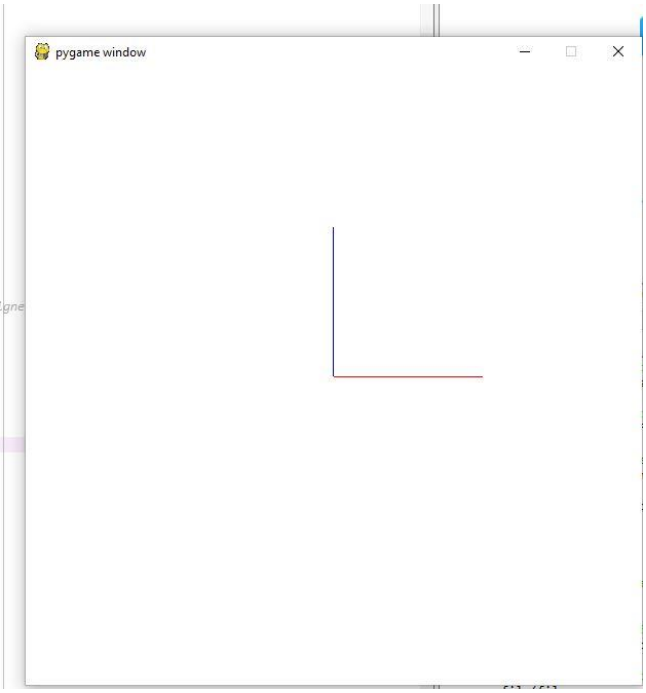
```python
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

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)

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


    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode ligne
    gl.glColor3fv([255, 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, 255, 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

    gl.glColor3fv([0, 2, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0, -1)) # Deuxième vertice : fin de la ligne

    gl.glEnd() # Find du tracé

    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique

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



3) Rotation de l'image

On effectue une rotation avec la fonction glRotate d'un angle de 90 autour de l'axe X afin de faire apparaître l'axe Z en bleu.

```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)

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

    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode ligne
    gl.glColor3fv([255, 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, 255, 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

    gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0, -1)) # Deuxième vertice : fin de la ligne

    gl.glEnd() # Find du tracé

    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique

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



Découverte de l'environnement du travail du TP

1)a) Premier contrôle

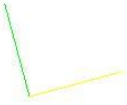
La touche « a » permet de faire apparaitre et disparaitre les axes.
La touche « z » permet de faire pivoter les axes dans le sens antihoraire.
La touche « Z » permet de faire pivoter les axes dans le sens horaire.

Dans la classe Configuration() du fichier « configuration.py », nous avons appelé la fonction ci-dessous qui permet d'agir sur les touches précédemment décrient.

```python
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)
```

Nous Pouvons bien constater que l'appui sur la touche « Z » provoque une rotation autour de l'axe Z de 2.5°. Et que la touche « a » fait disparaitre ou apparaitre la fenêtre de référence c'est-à-dire les axes.

1)b) Changement position de l'écran et de la couleur



Avec la fonction Q1b_f, on remarque que les axes ont changé de couleurs selon le code couleur que l'on a indiqué. Ici, on configure l'axe des x avec le code RGB [1,1,0] qui corresponds à du jaune(Rouge + Vert)

```python
def Q1b_f():
    return Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

Par la suite, on effectue le code ci-dessous qui fait un autre changement de couleur du même principe que le précédent code.

```
def Q1b_f():
    return Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisColor', [0,1,1]). \
    display()
```

L'axe des y à été coloré en cyan à l'aide du code « Configuration({…}).setParameter('yaxisColor', [0,1,1]).display ».

Le premier setParameter('xaxisColor' …) n'est pas visible car il attribue la couleur verte a l'axe X qui était déjà vert.

//Une erreur d'espace était présente dans le code à copier, il faut donc bien penser à l'enlever pour que le code fonctionne.

Le chainage de l'appel des méthodes setParameter() et display() est possible car setParametrer va permettre de changer la couleur des axes dans configuration. Les setParameter vont s'exécuter avant le .display() et le display va être exécuté sur "l'objet" retourné par configuration

Un traitement particulier est fait dans le « setter » pour le paramètre screenPosition pour définir la partie à afficher.

1)c) Représenter l'axe x horizontalement et l'axe z verticalement

On ajoute une seule instruction à la méthode initializeTransformationMatrix().
L'axe x est déjà représenté horizontalement et y est représenté verticalement. Pour que l'axe z soit représenté verticalement il faut donc effectuer une rotation de 90° ou -90° autour de l'axe x. Par soucis d'esthétique on choisit -90° (sinon l'axe z pointe vers le bas)

Code rajouté en bleu :

```
# Initializes the tranformation matrix
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)
```

II) Mise en place des interactions avec l'utilisateur avec Pygama

1)d) Gestion des touches « PageUP et PageDown »

Nous souhaitons changer d'échelle avec les touches PageUp(+1.1 pour augmenter de 10%) et PageDown (0.9 pour diminuer de 10%). On rajoute donc les 2 lignes de code suivantes dans la méthode processKeyDownEvent :

Code rajouté en Bleu

```
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)

    #Zoom et dézoom avec les touches haut et bas
    elif self.event.key == pygame.K_UP:
        gl.glScalef(1.1, 1.1, 1.1)
    elif self.event.key == pygame.K_DOWN:
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Ces 2 lignes fonctionnent pour les 2 types de clavier (Qwerty ou azerty) grâce à la constante « pygame.K_ »

1)e) Affecter un changement d'échelle à la molette de la souris

self.event.button == 4 qui signifie « lorsque l'on roule la molette vers l'avant pour zoomer)
self.event.button == 5 qui signifie « lorsque l'on roule la molette vers l'arrière pour dézoomer)

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

1)f)

```python
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0]==1:
      gl.glRotate(self.event.rel[0]/2, 0, 0, 1)
      gl.glRotate(self.event.rel[1]/2, 1, 0, 0)

    if pygame.mouse.get_pressed()[2]==1:
      gl.glTranslatef(self.event.rel[0]/40, 0, 0)
      gl.glTranslatef( 0, 0,-self.event.rel[1]/40)
```

III) Création d'une section

Une section est constituée du dessin de ses arrêtes et du remplissage des faces par une couleur.

Les paramètres d'une section sont :
• La largeur dans des x (width)
• La hauteur dans l'axe des z (height)
• La profondeur dans l'axe des y (thickness)
• La position du coin inférieur gauche à l'aide de trois coordonnées stockées dans une liste position

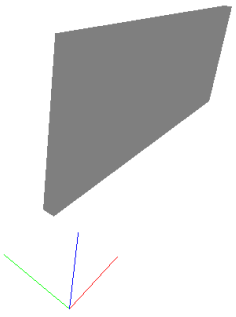
Nous allons compléter le fichier section.py :

2)a)

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


    return self
```

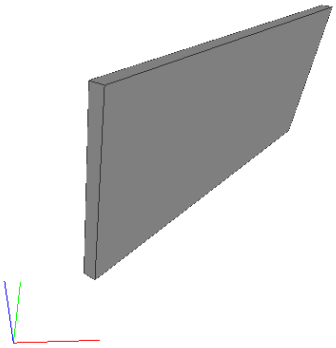
2)b)

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

    if self.parameters['edges']==True:
      self.drawEdges()
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()

    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1], self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)


    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']) # Couleur gris moyen


    for i in range(0,len(self.faces)):
      face=self.faces[i]
      gl.glVertex3fv(self.vertices[face[0]])
      gl.glVertex3fv(self.vertices[face[1]])
      gl.glVertex3fv(self.vertices[face[2]])
      gl.glVertex3fv(self.vertices[face[3]])

    gl.glEnd()
    gl.glPopMatrix()

    return self
```

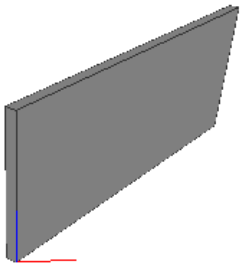


2)c)

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

    gl.glMatrixMode(gl.GL_MODELVIEW)
    #gl.glLoadIdentity()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1], self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)


    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.2, 0.2, 0.2]) # Couleur gris moyen
    for i in range(0,len(self.faces)):
      face=self.faces[i]
      gl.glVertex3fv(self.vertices[face[0]])
      gl.glVertex3fv(self.vertices[face[1]])
      gl.glVertex3fv(self.vertices[face[2]])
      gl.glVertex3fv(self.vertices[face[3]])

    gl.glEnd()
    gl.glPopMatrix()
    return self
```



IV) Création des Murs

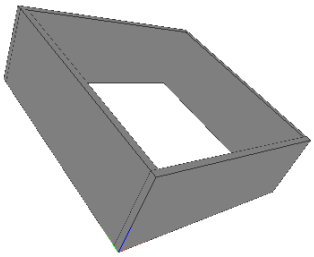


```
def Q3a():
    return Configuration().add(
        Wall({
            'position': [0, 0, 0],
            'width': 7,
            'height': 2.6,
            'orientation': 45
        }))
```

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    for i in self.objects:
      i.draw()
```

V) Création d'une Maison

Pour créer une maison, nous avons créé dans le fichier main quatre murs en ajustant leurs positions et leurs orientations afin qu'ils se rejoignent. Puis nous les avons ajoutés à la classe house que nous avons ajouté à la classe configuration.

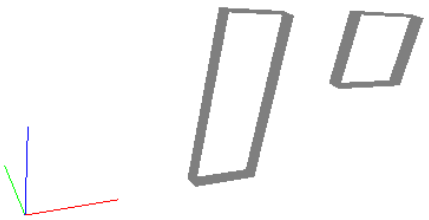


```
# Adds an object
def add(self, x):
    self.objects.append(x)
    return self

# Draws the house
def draw(self):
    # A compléter en remplaçant pass par votre code
    for x in self.objects:
        x.draw()
```

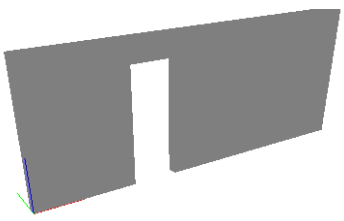
```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
    wall2 = Wall({
        'position': [0.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
    wall3 = Wall({
        'position': [0, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
    wall4 = Wall({
        'position': [7, 0.2, 0],
        'width': 7,
        '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)
```

VI) Création d'Ouvertures

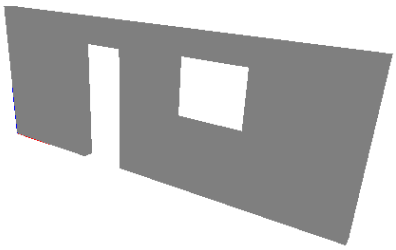
Le but de cette partie est de créer des ouvertures dans un mur afin d'y mettre des portes ou des fenêtres. Afin de créer une ouverture, nous allons devoir transformer notre section initiale en plusieurs sections



```
def draw(self):

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


    gl.glPushMatrix()

    gl.glMatrixMode(gl.GL_MODELVIEW)
    #gl.glLoadIdentity()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1], self.parameters['position'][2])



    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


    for i in range(0,len(self.faces)):
      face=self.faces[i]
      gl.glVertex3fv(self.vertices[face[0]])
      gl.glVertex3fv(self.vertices[face[1]])
      gl.glVertex3fv(self.vertices[face[2]])
      gl.glVertex3fv(self.vertices[face[3]])

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

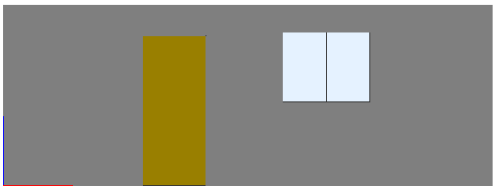


```
def canCreateOpening(self, x):

    test=True
    if (self.parameters['height']>=x.getParameter('height')+x.getParameter('position')[2]-self.parameters['position'][2])==False:
      test=False
    if (x.getParameter('position')[2]>=self.parameters['position'][2])==False:
      test = False
    if (self.parameters['width']>=x.getParameter('width')+x.getParameter('position')[0]-self.parameters['position'][0])==False:
      test=False
    if (x.getParameter('position')[0]>=self.parameters['position'][0])==False:
      test=False

    return test
```



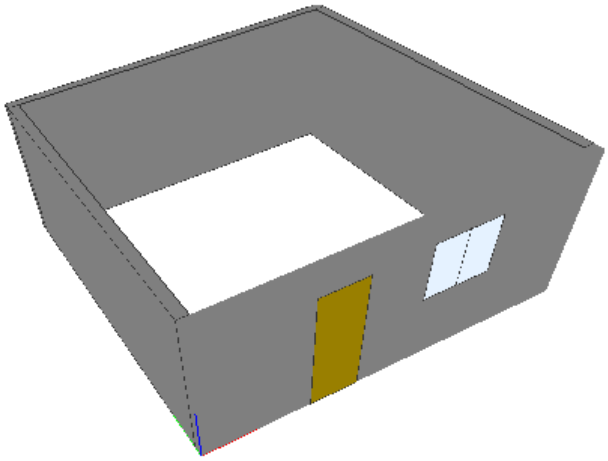
```
def add(self, x):
    self.objects.append(x)
    return self
```

VII – Pour finir



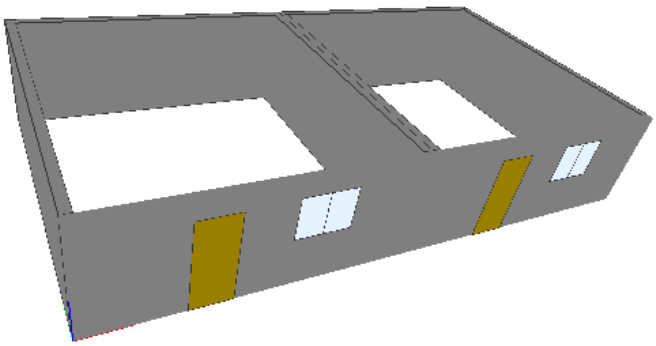
```
def Q6():
    wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True

    })

    porte = Door({
        'position': [2, 0, 0]
        })

    fenetre= Window({
        'position': [4, 0, 1.2],
        'width': 1.25,
        'height': 1})

    wall1.add(porte).add(fenetre)
    configuration=Configuration()
    configuration.add(wall1).add(porte).add(fenetre)
    return configuration
```

```
def Q7():

  wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall2 = Wall({
        'position': [0.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  wall3 = Wall({
        'position': [0, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall4 = Wall({
        'position': [7, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  house = House({'position': [-3, 1, 0], 'orientation': 0})
  n=0;
  i=0.1;


  house.add(wall1).add(wall3).add(wall4).add(wall2)

  porte = Door({
        'position': [2, 0, 0]
       })

  fenetre= Window({
        'position': [4, 0, 1.2],
        'width': 1.25,
        'height': 1})

  wall1.add(porte).add(fenetre)
  configuration=Configuration()
  configuration.add(house).add(porte).add(fenetre)
  return configuration
```



Duplication :

```
def Q8():

#parti 1
  wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall2 = Wall({
        'position': [0.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  wall3 = Wall({
        'position': [0, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall4 = Wall({
        'position': [7, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  house1 = House({'position': [-3, 1, 0], 'orientation': 0})
  n=0;
  i=0.1;


  house1.add(wall1).add(wall3).add(wall4).add(wall2)

  porte1 = Door({
        'position': [2, 0, 0]
       })

  fenetre1= Window({
        'position': [4, 0, 1.2],
        'width': 1.25,
        'height': 1})

  wall1.add(porte1).add(fenetre1)
  configuration1=Configuration()
  configuration1.add(house1).add(porte1).add(fenetre1)

#parti 2
  wall5 = Wall({
        'position': [7, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall6 = Wall({
        'position': [7.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  wall7 = Wall({
        'position': [7, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
  wall8 = Wall({
        'position': [14, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
  house2 = House({'position': [-3, 1, 0], 'orientation': 0})
  n=0;
  i=0.1;


  house2.add(wall5).add(wall7).add(wall8).add(wall6)

  porte2 = Door({
        'position': [9, 0, 0]
       })

  fenetre2= Window({
        'position': [11, 0, 1.2],
        'width': 1.25,
        'height': 1})

  wall5.add(porte2).add(fenetre2)
  configuration1.add(house2).add(porte2).add(fenetre2)


  return configuration1
```



Question bonus : ajout d'un toit :

```
def Q9():


    #parti 1
wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
wall2 = Wall({
        'position': [0.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
wall3 = Wall({
        'position': [0, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
wall4 = Wall({
        'position': [7, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
house1 = House({'position': [-3, 1, 0], 'orientation': 0})
n=0;
i=0.1;

while n*i<1.5:

    house1.add(Wall({
          'position': [n*i/2, n*i/2, 2.6+n*i] ,
          'width': 7-n*i,
          'height': 0.1,
          'thickness':7.4-n*i,
          'orientation': 0,
          'edges': True,
          'color': [2.0,0.1,0.2]}))
    n+=1

house1.add(wall1).add(wall3).add(wall4).add(wall2)

porte1 = Door({
      'position': [2, 0, 0]
      })

fenetre1= Window({
      'position': [4, 0, 1.2],
      'width': 1.25,
      'height': 1})

wall1.add(porte1).add(fenetre1)
configuration1=Configuration()
configuration1.add(house1).add(porte1).add(fenetre1)
```

```
#parti 2
wall5 = Wall({
        'position': [7, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
wall6 = Wall({
        'position': [7.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
wall7 = Wall({
        'position': [7, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
wall8 = Wall({
        'position': [14, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
house2 = House({'position': [-3, 1, 0], 'orientation': 0})
n=0;
i=0.1;

while n*i<1.5:

    house2.add(Wall({
          'position': [n*i/2+7, n*i/2, 2.6+n*i] ,
          'width': 7-n*i,
          'height': 0.1,
          'thickness':7.4-n*i,
          'orientation': 0,
          'edges': True,
          'color': [2.0,0.1,0.2]}))
    n+=1

house2.add(wall5).add(wall7).add(wall8).add(wall6)

porte2 = Door({
      'position': [9, 0, 0]
      })

fenetre2= Window({
      'position': [11, 0, 1.2],
      'width': 1.25,
      'height': 1})

wall5.add(porte2).add(fenetre2)
configuration1.add(house2).add(porte2).add(fenetre2)


return configuration1
```