BONNEFOND Quentin
BISCHOFF Eva
Date du TP (17/01/2222)

Rapport de TP3 – Lecture automatique de chiffres par analyse d'image

Version finale

POLYTECH ANNECY-CHAMBÉRY
UNIVERSITÉ SAVOIE MONT BLANC

I. Introduction

L'objectif de ce TP est de réussir à représenter des objets en 3D et de pouvoir manipuler cet objet. Par exemple, pouvoir zoomer, faire des rotations, etc... A la fin de ce TP, nous allons devoir construire une maison pas à pas en faisant tout d'abord les axes puis les murs, les portes, les fenêtres etc...

II. Travail préparatoire

1- Utilisation de Pygame

(1) Tout d'abord, nous avons codé les trois lignes demandées et le code fonctionne bien. On remarque qu'une page s'ouvre (en bas de l'écran) mais est directement fermée.

Le code que nous avons copié est le suivant :

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

Voici une explication de ce code:

import pygame
#Ici le module pygame a été importé

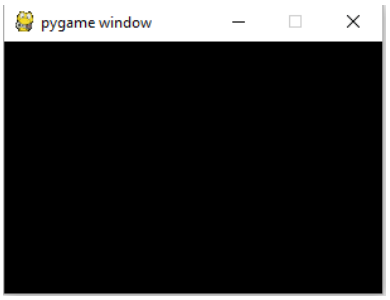
pygame.init()
#Cette commande initialise chacun des module de pygame

ecran = pygame.display.set_mode((300, 200))
#Cette ligne permet de créer une fenêtre de taille (300,200)

pygame.quit()
#Cette ligne permet de ne pas faire quitter le programme à la fin.

(2) Nous avons alors essayé le code suivant :

```
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()
```



Lorsque nous le faisons tourner une nouvelle page s'ouvre.
La voici :

Lorsque nous retouchons une touche de clavier cette page se ferme.

```
import pygame
pygame.init()
ecran = pygame.display.set_mode((300, 200))
#Ce début de code a été expliqué dans la question 1/.

continuer = True
while continuer:
   for event in pygame.event.get()
# obtenir des événements de la file d'attente

      if event.type == pygame.KEYDOWN:
         continuer = False
pygame.quit()
```

Pour conclure, si une touche du clavier est pressée alors la page se ferme.

2- Utilisation de Pyopengl pour représenter des objets 3D

(1)Tout d'abord nous avons entré le code donné mais rien ne s'est passé.

(2) Nous avons alors complété le code comme ceci :
Problème rencontré : Ici, nous ne savions pas quoi mettre pour "aspect", nous avons donc utilisé le code donné en exemple display[0]/display[1].

```
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.
    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([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()
```

Ici, un seul trait noir a été tracé. On a alors modifié le code comme ci-dessous afin de tracer les autres segments.

On a donc le code suivant :

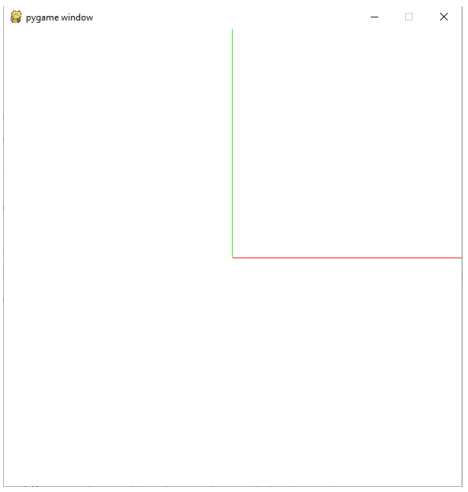
```python
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

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

    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()
```

Axe des X

Axe des Y

Axe des Z

Tout d'abord nous avons tracé les trois axes . On a obtient :



Ici l'axe des Z n'est pas visible car il est situé en face de nous.

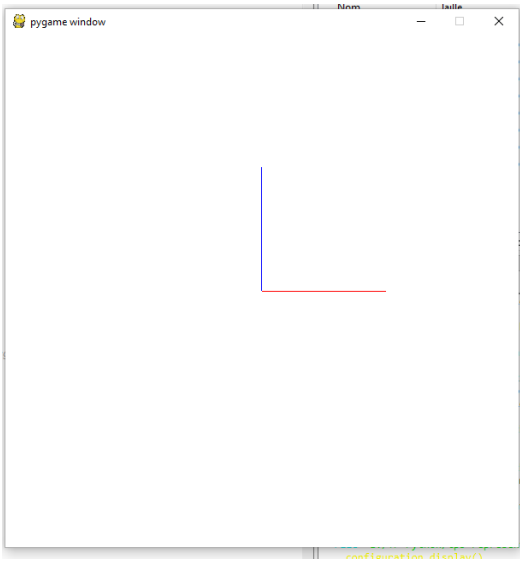
(3) On applique alors le code avec le carré rouge. On a donc effectué une rotation et une translation et on obtient:

```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)

    # Placer ici l'utilisation de gluPerspective.
    glu.gluPerspective(45,(display[0] / display[1]),0.1,50)
    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 lignes (segments)

    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()
```

Axe des X

Axe des Y

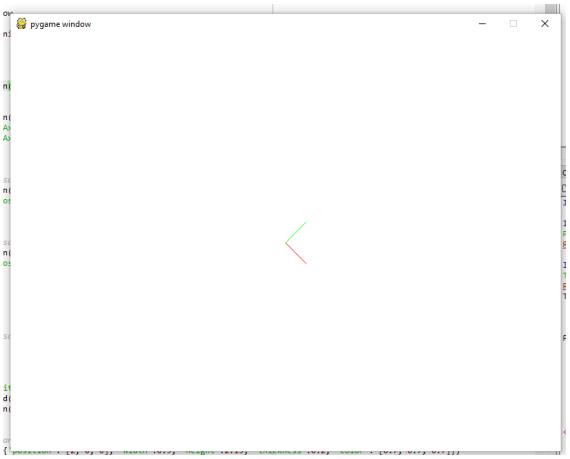
Axe des Z



3-Découverte de l'environnement du travail du TP

(1) Si on touche la touche A, les axes disparraissent et si on la retouche les axes réapparaissent.

Si on touche la touche z, on observe une rotation dans le sens horraire et si on touche "z en majuscule" on observe une rotation dans l'autre sens.



Tout cela est possible grâce ce code:

```
# Processes the KEYDOWN event
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)
```

Les deux premières parties (les deux if) de ce code permettent la rotation des axes de 2,5° grâce à la fonction glRotate, l'un dans le sens horaire et l'autre dans le sens anti horaire.
Le dernier if permet de faire disparaître les axes.

(1)b)
* Grâce à ce code, on peut changer la couleur des axes. Ici on remarque par exmple qu'on peut avoir des axes jaunes.

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

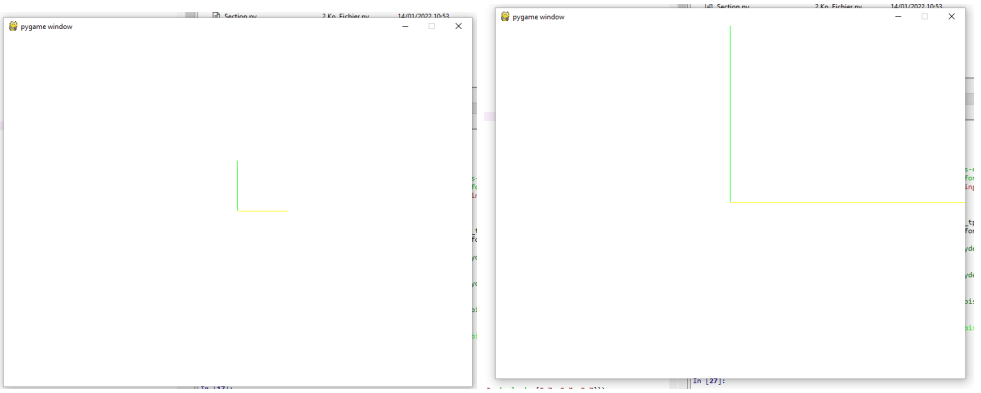
Remarque : ne pas oublier d'enlever # (pour mettre en commentaire) dans le main.py la méthode def main() : pour la ligne configuration = Q1b_f():

*

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

et

display() e*



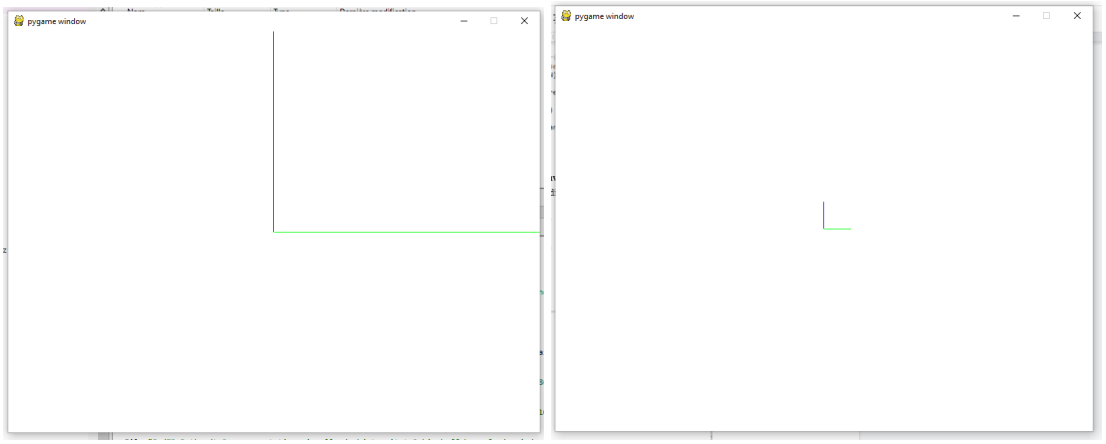
(1)c)
On cherche à faire une rotation pour faire passer l'axe z verticalement et l'axe y en profondeur. Pour cela il faut effectuer une rotation de 90° dans le sens trigonométrique autour de l'axe x.

```
# 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.glRotate(-90,1,0,0)
```

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

(1)d) Dans cette question on rajoute une condition dans le code suivant expliqué dans la question 1a). On obtient ainsi le grossissement de l'image.

```
# Processes the KEYDOWN event
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)

    elif self.event.dict['unicode'] == 'PAGEUP' or self.event.key == pygame.K_PAGEUP:
        gl.glScalef(1.1,1.1,1.1)

    elif self.event.dict['unicode'] == 'PAGEDOWN' or self.event.key == pygame.K_PAGEDOWN:
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```

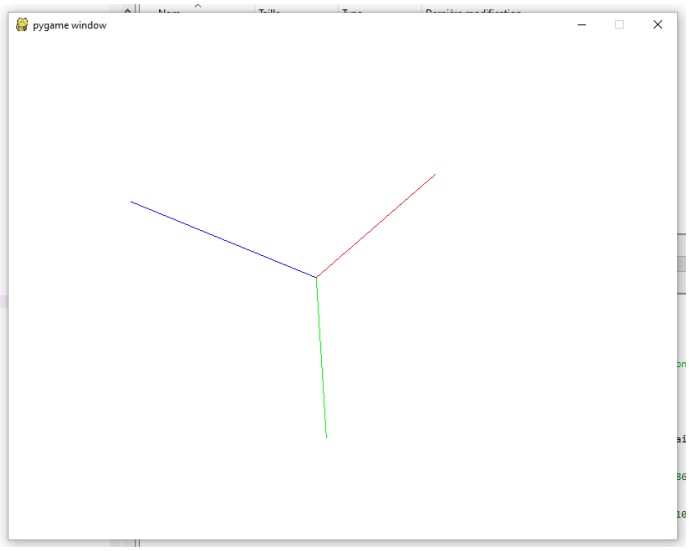


(1)e) On souhaite ici pouvoir zoomer et dézoomer grâce à la molette de la souris. Grâce au code suivant on obtient bien ce qui est demandé.

```
# 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)
```

(1)f) Dans cette question, on veut réussir à déplacer les axes en utilisant la souris pour cela, il faut que le bouton gauche pour faire des rotations et le bouton de droite pour faire des translation.

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[2]==1:
        gl.glTranslatef(self.event.rel[0]/10,0,-self.event.rel[1]/10)

    elif pygame.mouse.get_pressed()[0]==1:
        gl.glRotatef(self.event.rel[1],0,1,0)
        gl.glRotatef(self.event.rel[0],1,0,0)
```



IV. Création d'une section

(2)a) Dans cette question, nous devons écrire la méthode generate(self) de la Section qui va alors créer les sommets et les faces d'une section orientée selon l'axe x. Pour cela on définit que le coin bas gauche de la face externe est en (0,0,0). On utilise le code suivant :

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

(2)b)

- Tout d'abord analysons la fonction Q2b(). Cette fonction permet d'ajouter une section à la configuration initiale. Cette section est posistion au point [1, 1, 0] de largeur 7 et de hauteur 2.6.

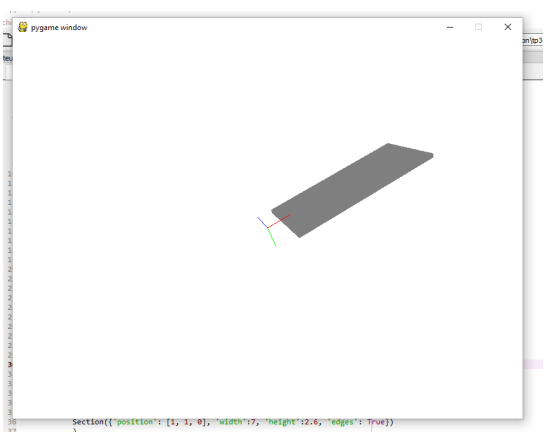
```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
            Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
            )
```

- L'instruction Configuration().add(section).display() permet de créer la section qui se situe dans la fonction add().
- L'objectif est ici d'afficher la première face. Pour ce faire on utilise le code suivant:

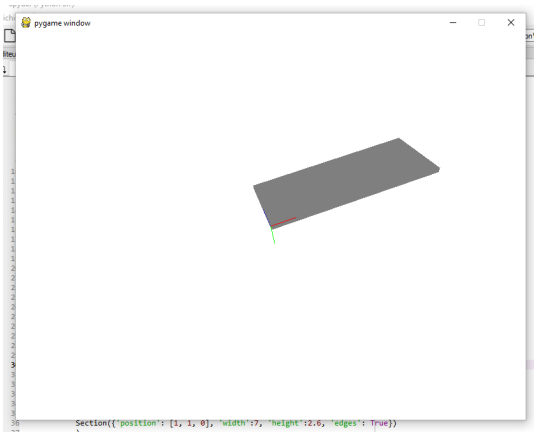
```
# Draws the edges
def drawEdges(self):
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5*0.5, 0.5*0.5, 0.5*0.5]) # Couleur gris moyen
    for face in self.faces:
        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()
```

PROBLÈME RENCONTRÉ : Nous avons rencontré un problème pour la fonction translate dans ce code. En effet, nous n'utilisons pas les bons paramètres dans cette fonction. Puis nous observions

directement un décalage. Nous avons alors compris qu'il modifier la fonction Q2(). Nous avons modifié alors position de [1,1,0] en [0,0,0].



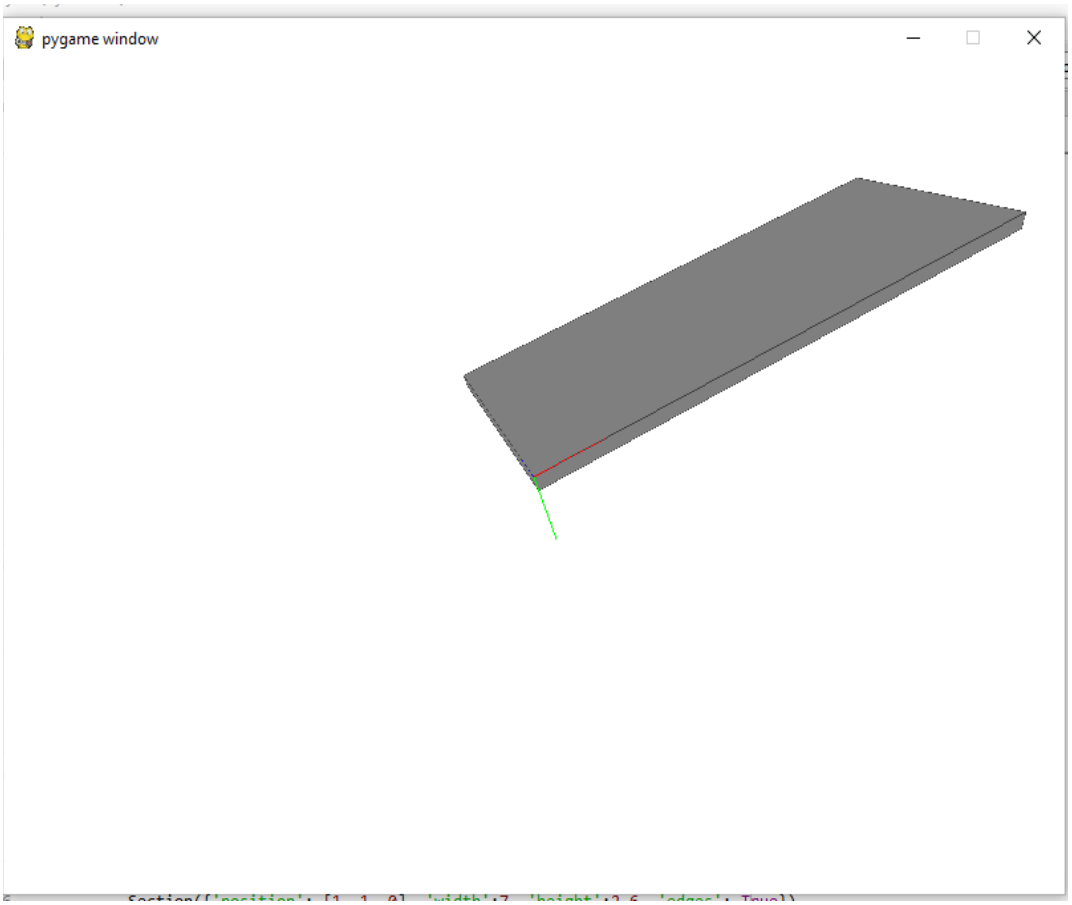
Problème :



Solution :

(2)c) Le but de cette question est de créer les arêtes de la section. Pour ce faire on re-utilise la fonction draw() mais en modifiant GL_FILL par GL_LINE.
On a donc :

```
# Draws the edges
def drawEdges(self):
    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 faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5*0.5, 0.5*0.5, 0.5*0.5]) # Couleur gris moyen
    for face in self.faces:
        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()

# Draws the faces
def draw(self):
    if self.parameters['edges']== True :
        self.drawEdges()
    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_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 face in self.faces:
        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()
```



V. Création des murs

(3)a)

- Analyse du fichier Wall.py : Dans ce fichier, la class Wall est construite. On utilise tout d'abord la fonction __init__().
  Cette fonction permet de remarquer s' il manque des paramètres grâce à plusieurs if. Ainsi s' il manque des paramètres, ils peuvent être re-demandés ou sont affectés par défaut.

- Le but ici est tout d'abord de construire un murs grâce aux fonctions draw() et Q3a(). Pour cela on utilise le code suivant :

```
# Adds an object
def add(self, x):
    # A compléter en remplaçant pass par votre code
    pass

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

   gl.glRotate(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()
```

```
def Q3a():
    return Configuration().add(Wall(({'position': [1, 1, 0], 'width':7, 'height':2.6,'orientation':90, 'edges': True})))
```

VI. Création d'une maison

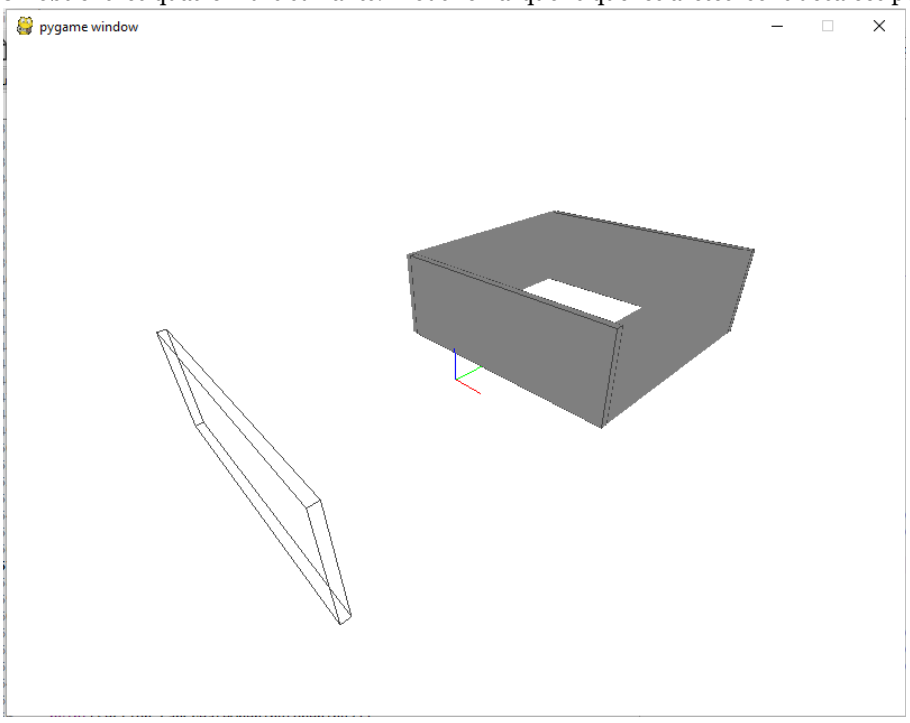
Toutd'abord nous nous sommes inspirés de la méthode draw() de la classe Configuration qui permet l'affichage de plusieurs objets. Dans la classe House, on utilise le code suivant.

```
# Draws the house
def draw(self):
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'],0,0,1)
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],0)

    for x in self.objects:
        x.draw()
    gl.glPopMatrix()
```

Puis on modifie la fonction Q4(). Ici on veut créer quatre murs. Nous avons eu des problèmes de position car nos murs n'étaient pas collés.

```
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, 6.8, 0], 'width':7, 'height':2.6,'orientation':0, 'edges': True})
    wall3 = Wall({'position': [0, 0, 0], 'width':7, 'height':2.6,'orientation':90, 'edges': True})
    wall4 = Wall({'position': [0, -7, 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)
```

On obtient les quatre murs suivants. Nous remarquons que les arêtes sont décalées pour certains murs.



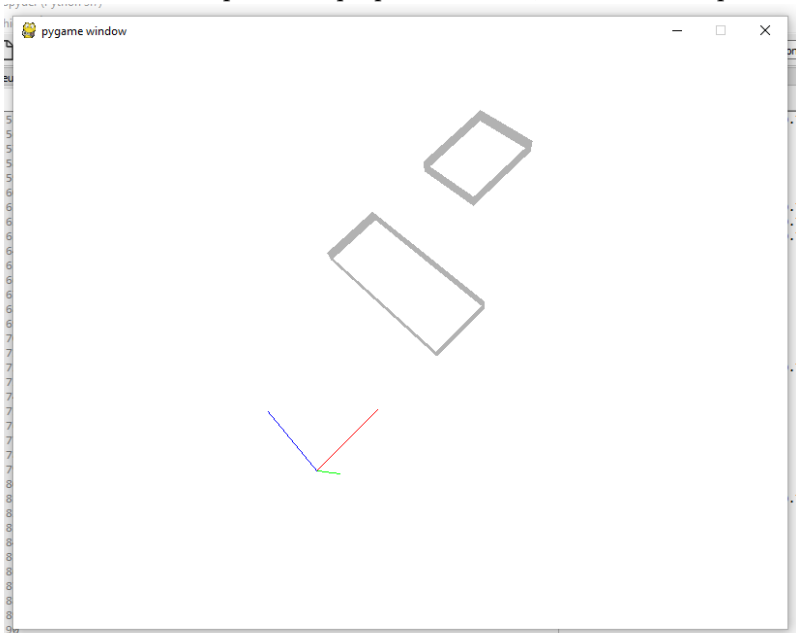
VII. Création d'ouvertures

(5)a) Dans cette question on veut tout d'abord représenter les contours de l'ouverture dont sa position sera définie par le coin inférieur gauche par rapport au repère de la section. Pour cela nous nous inspirons de ce que nous avons fait pour faire les sections.

On a donc comme code:

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

# Draws the faces
def draw(self):
   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_FILL)
   for x in self.faces:
       gl.glBegin(gl.GL_QUADS)
       gl.glColor3fv(self.parameters['color'])
       for y in x:
           gl.glVertex3fv(self.vertices[y])
       gl.glEnd()
   gl.glPopMatrix()
```

Problème : Ici nous avons le même problème que précédemment, nous n'arrivons pas à afficher les arêtes.



(5)b) Le but de cette question est de vérifier si une ouverture peut être ajoutée dans la section, c'est-à dire si la position de notre ouverture peut bien être présente DANS le mur.

```
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    if x.parameters['thickness'] == self.parameters['thickness']:
        if x.parameters['position'][0] >= self.parameters['position'][0]:
            if x.parameters['position'][1] >= self.parameters['position'][1]:
                if x.parameters['position'][2] >= self.parameters['position'][2]:
                    if x.parameters['position'][2] + x.parameters['height'] <= self.parameters['position'][2] + self.parameters['height'] :
                        if x.parameters['position'][0] + x.parameters['width'] <= self.parameters['position'][0] + self.parameters['width'] :
                            return True
    else:
        return False
```

Pour la fonction Q5(), nous obtenons le résultat **TRUE, TRUE, FALSE**

VIII. Pour finir

Nous n'avons pas eu le temps de faire cette partie.

IX. Conclusion

Dans ce TP , nous avons construit pas à pas une maison en 3D. Tout d'abord, nous avons manipulé les axes. La difficulté de cette partie était de ne pas se perdre dans la représentation des axes x,y,z et de ne pas les inverser. Puis nous avons modélisé des objets en 3D en commençant par des sections. Le principal problème que nous avons alors rencontré a été celui des arêtes que nous n'avons pas réussi à représenter correctement même avec l'aide de notre enseignant. Ce problème s'est alors répercuté dans la création des ouvertures.