HUMBERT Ophélie
GERNIGANT Elsa
Date du TP (14/01/2022)

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Le but de ce TP est de réaliser un objet en 3D. Pour finaliser ce TP, nous allons devoir réaliser une maison dans l'espace.
Pour cela, il va falloir apprendre à utiliser de nouvelles conditions dans des fonctions.
Nous allons utiliser la librairie Pygame afin d'importer des fonctions qui vont nous permettre de pouvoir modéliser et changer l'objet en 3D.
On va pouvoir faire bouger l'objet dans l'espace (déplacements en ligne, rotation).
On utilisera encore une fois les classes pour écrire la majorité du code.
Le menu main et les tests au long du TP nous permettrons de vérifier si les fonctions que nous écrivons fonctionnent.

II. Présentation à faire avant le TP :

Question (1).

On importe le programme pygame puis on initialise chacun des modules de pygame.
On crée ensuite une fenêtre de 300 sur 200, et on quitte le programme.
On ne voit rien s'afficher dans la console.
Néanmoins on voit une nouvelle fenêtre se créer et se fermer directement dans la barre d'icône en bas de l'écran de l'ordinateur.

Question (2).

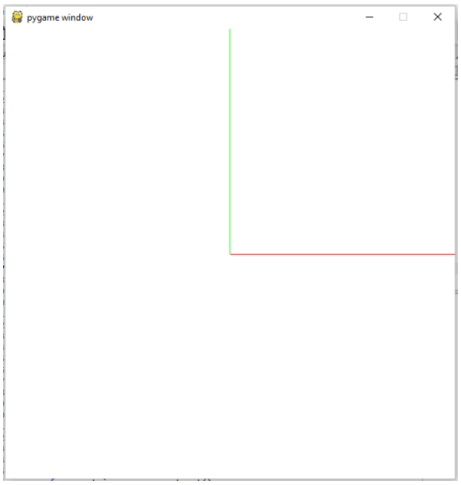
Le code va permettre d'afficher une fenêtre d'après les dimensions voulues.
Ensuite la boucle « while » permet de dire à Pygame de garder la fenêtre ouverte si rien ne se passe.
Puis le « if » introduit le fait que si l'on appuie sur une touche du clavier alors la fenêtre doit se fermer.
Continuer == False → la fenêtre se ferme.

```
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()
```

Utilisation de Pyopengl pour représenter des objets 3D

Question (1).

Nous avons ajouté la fonction glu.gluPerspective (45, 1, 0.1, 50) en utilisant les chiffres donnés dans l'énoncé.
Le fichier ne renvoie pas d'erreurs.



Question (2).

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 la couleur du prochain segment ici c'est l'axe des x qui sera en rouge
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne à l'origine du repère (0,0,-2)
gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne au point de coordoneés (1,0 ;-2)

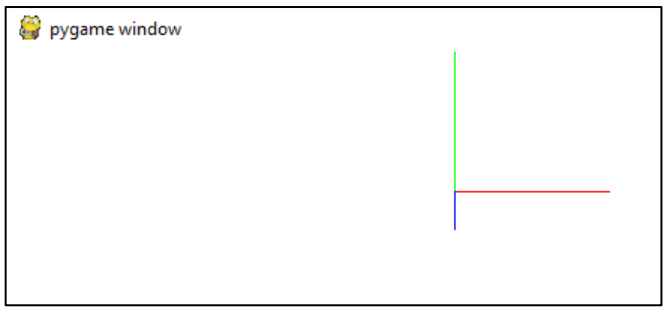
gl.glColor3fv([0, 1, 0]) # Indique la couleur du prochian segment ici c'est l'axe des y qui sera en vert
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne à l'origine du repère (0,0,-2)
gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne au point de coordonnées (0,0 ,-2)

gl.glColor3fv([0, 0, 1]) # Indique la couleur du prochian segment ici c'est l'axe des z qui sera en bleu
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne à l'origine du repère (0,0,-2)
gl.glVertex3fv((0, 0, -3)) # Deuxième vertice : fin de la ligne au point de coordonnées (0,0,-3)
gl.glEnd() # Find du tracé

Question (3).

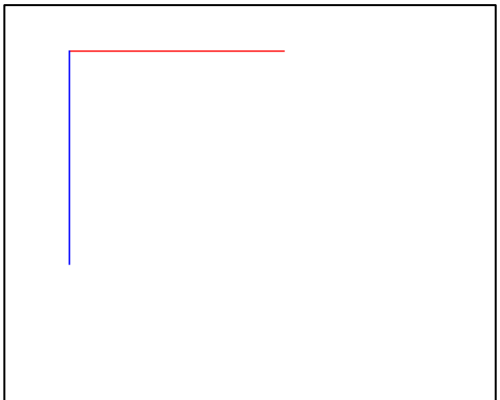
Après translation :

```
gl.glTranslatef(0.0, 2, -5)
```



Après rotation :

```
gl.glRotatef(-90, 1, 0, 0)
```



Découverte de l'environnement du travail du TP

Question (1.a)

La touche a nous permet de faire disparaître les axes, tandis que les touches Z et z nous permettent de faire des rotations (respectivement dans le sens horaire ou dans le sens contraire).

Voir le code ci-dessous :

```
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)
```

If introduit une condition, si on appuie sur la touche « z » du clavier alors cela effectue une rotation de 2.5 degrés (dans le sens trigonométrique) autour de l'axe z.

Si c'est la touche « Z » qui est actionnée alors cela se traduira une rotation de -2,5 degrés autour de l'axe z.

Enfin si on appuie sur la touche « a » cela entraine la pause du programme, cela permet d'afficher ou de supprimer les axes.

```
    def display(self):

        # Displays on screen
        self.draw()
        pygame.display.flip()

        # Allows keybord events to be repeated
        pygame.key.set_repeat(1, 100)

        # Infinite loop
        while True:

            # Waits for an event
            self.event = pygame.event.wait()

            # Processes the event

            # Quit pygame (compatibility with pygame1.9.6 and 2.0.0)
            if self.event.type == pygame.QUIT or (self.event.type == pygame.WINDOWEVENT and pygame.event.wait(100).type == pygame.QUIT):
                pygame.quit()
                break

            elif self.event.type == pygame.KEYDOWN:
                self.processKeyDownEvent()

            elif self.event.type == pygame.MOUSEBUTTONDOWN:
                self.processMouseButtonDownEvent()

            elif self.event.type == pygame.MOUSEMOTION:
                self.processMouseMotionEvent()

            # Clears the buffer and displays on screen the result of the keybord action
            gl.glClear(gl.GL_COLOR_BUFFER_BIT|gl.GL_DEPTH_BUFFER_BIT)
            self.draw()
            pygame.event.clear()
            pygame.display.flip()
```

La fonction display va permettre d'afficher un objet 3D dans une fenêtre.

L'affichage va se modifier en fonction des évènements apparus dans les fonctions définies précédemment (« processKeyDownEvent() », « processMouseButtonDownEvent() », « processMouseMotionEvent() »)

Le fichier « Main py » commence d'abord par importer le code « Configuation » :
from Configuration import Configuration
Puis il retourne le résultat de la fonction Configuration() : return Configuration()

Question (1.b)

- La couleur des axes est changée et la position de l'objet 3D est modifiée également
- Le chaînage fonctionne car les deux fonctions ***setParameter()*** et ***display()*** sont définies par le paramètre « self ».
- Ce paramètre permet de fixer l'objet 3D de façon à toujours le voir et le faire bouger dans la fenêtre définie.

Question (1.c)

Au départ on a x en abscisse et y en ordonnée. Donc pour avoir z en ordonnée et x en abscisse, il nous faut effectuer une rotation de 90 degrés autour de l'axe x de l'objet 3D.
On obtient donc le code suivant :

```
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)
```

L'image obtenue est :

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

Question (1.d)

```
elif self.event.key == pygame.K_PAGEUP :
     gl.glScalef(1.1, 1.1, 1.1)
elif self.event.key == pygame.K_PAGEDOWN :
     gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Nous avons ajouté deux nouvelles conditions afin de modifier l'échelle, selon la documentation pygame et les informations en début de TP.
Si on appuie sur la touche « PAGEUP » alors le facteur d'échelle pour chaque dimension x,y et z sera de 1.1. Et si on appuis sur la touche « PAGEDOWN » alors le facteur d'échelle vaudra 1/1.1.
C'est ce changement d'échelle qui va permettre de grossir ou réduire l'affichage.

Question (1.e)

```
def processMouseButtonDownEvent(self):

    if self.event.button == 5 :
         gl.glScalef(1/1.1, 1/1.1, 1/1.1)
    elif self.event.button == 4 :
         gl.glScalef(1.1, 1.1, 1.1)
```

Nous avons créé des conditions if dans la nouvelle fontion « processMouseButtonDownEvent() ». Cela nous a permis de zoomer lorsque l'on pousse la roulette de la souris vers le haut (bouton = 4). Et le code nous permet de dézoomer lorsque le button est égal à 5.

Question (1.f)

```
def processMouseMotionEvent(self):
     if pygame.mouse.get_pressed()[2] == 1:
         gl.glTranslate( self.event.rel[0]/100,0, 0)
         gl.glTranslate( 0,0, - self.event.rel[1]/100)
     elif pygame.mouse.get_pressed()[0] == 1:
         gl.glRotate(self.event.rel[0], 1, 0, 0)
         gl.glRotate(self.event.rel[1], 0, 0, 1)
```

Il y a deux conditions une pour le bouton de droite et une pour celui de gauche.
Dans le premier if on regarde l'état du bouton de droite (pygame.mouse.get_pressed()[2]) si c'est =1 c'est qu'il est enfoncé, dans ce cas on veut effectuer une translation selon x et z en fonction du mouvement de la souris. C'est ce que nous avons écrit :

```
gl.glTranslate( self.event.rel[0]/100,0, 0)
gl.glTranslate( 0,0, - self.event.rel[1]/100)
```

Nous avons ajouté /100 afin que la translation soit plus maitrisée.

La deuxième condition permet de faire une rotation autour de l'axe x et z si le bouton de gauche de la souris est enfoncé.

IV. Création d'une section

Question (2.a)

```
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'], self.parameters['height']],
            [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
            [0, self.parameters['thickness'],0],
            [self.parameters['width'], self.parameters['thickness'],0]
            ]
   self.faces = [
            [0, 3, 2, 1],
            [0,1,4,6],
            [1,4,5,2],
            [6,7,5,4],
            [5,2,3,7],
            [7,6,0,3]
            ]
```

Self.vertices permet d'entrer les coordonnées des somments du parallélépipède rectangle.
Self.faces permet d'entre les faces du pavé droit qui sont définies avec les numéros de ces sommets.

Question (2.b)

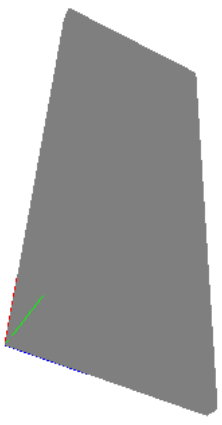
- **Configuration().add(section).display()** permet d'afficher les axes (depuis la classe Configuration) et le parallépipède (depuis la classe Section)

- Pour écrire la méthode draw(), on utilise les coordonnées des sommets définis à la question précédente afin de tracer un polygone qui relie les quatre points.

On réitère l'opération 6 fois avec les coordonnées des différentes faces.

```
def draw(self):

    gl.glPushMatrix()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv( [self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'],0])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, self.parameters['thickness'],0])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv( [self.parameters['width'], self.parameters['thickness'],0])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv( [self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv( [self.parameters['width'], self.parameters['thickness'],0])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],0])
    gl.glVertex3fv( [0, self.parameters['thickness'],0])
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glEnd()

    gl.glPopMatrix()
```

- On obtient :



Question (2.c)

- Les arrêtes sont tracés de manière analogue aux faces, seulement on remplace `gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)` par `gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)`

Cela donne :

```
def drawEdges(self):

    gl.glPushMatrix()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv( [self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'],0])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([0, self.parameters['thickness'],0])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],0])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv( [self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv( [self.parameters['width'], self.parameters['thickness'],0])
    gl.glEnd()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.1, 0.1, 0.1])
    gl.glVertex3fv([self.parameters['width'], self.parameters['thickness'],0])
    gl.glVertex3fv( [0, self.parameters['thickness'],0])
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glEnd()

    gl.glPopMatrix()
```

- On a ajouté une condition à la fonction « draw » de façon à ce que les arrêtes soient également dessinées sur l'objet 3D.

```
def draw(self):

    if self.parameters['edges']:
        self.drawEdges()
```

- On obtient :