DELAMAIN Maïa
POTHIER Mona
Date du TP (02/12/2021)

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

II. Travail préparatoire

1. Question (1)

```
:linenos:

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

On initialise les modules de *pygame*
Puis on associe l'affiche de la surface (*pygame.display)* à écran.
Avec *set.mode* on définie la taille de la fenêtre : 300x200

Lorsqu'on exécute le code, une fenêtre s'ouvre et se ferme aussitôt.

2. Question (2)

Une fenêtre s'ouvre et reste à l'écran, lorsqu'on appuie sur une touche du clavier (n'importe laquelle), la fenêtre se ferme.

```
:linenos:

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()
```

Lorsqu'une touche du clavier est appuyée :
On reconnait un evenement de type *pygame.KEYDOWN*
Donc *continuer = False* et la fenetre se ferme

III. Utilisation de Pyopengl pour représenter des objets 3D

1. Question (1)

On utilise la fonction *gluPerspective :*

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

On rentre en paramètre les valeurs données dans l'énoncé. Quand on lance le programme une fenêtre blanche s'ouvre :

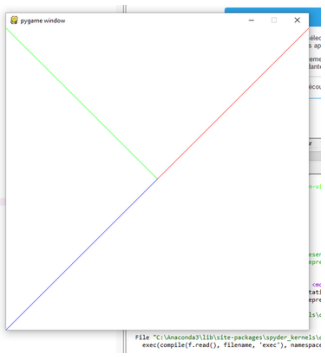
2. Question (2)

On cherche maintenant à afficher les vecteurs dans la fenêtre d'affichage :

```
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, 1, -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((-2, 2, -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((-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
```

Pour cela on effectue 3 fois le tracé de vecteur comme ci-dessous, en changeant à chaque fois *glColor* pour changer la couleur et les coordonnées de *glVertex3fv* pour changer de position.

Cela affiche :

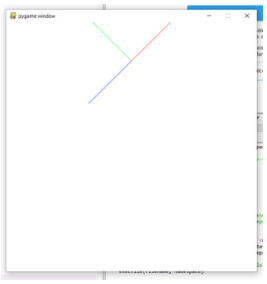


3. Question (3)

Pour déplacer l'écran on utilise la fonction *gl.glTranslatef*

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

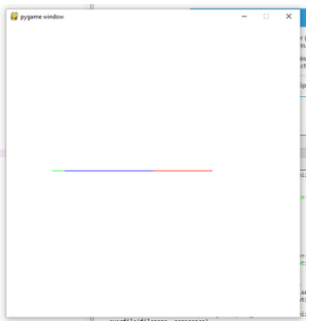
Cela affiche la fenêtre suivante :



Pour faire une rotation de 90] autour de l'axe on utilise la fonction *gl.glroratef*

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

Cela affiche la fenêtre suivante :



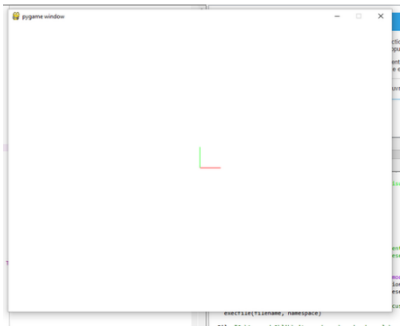
IV. Découverte de l'environnement du travail du TP

1. Question (1a)

On ajoute la fonction Q1a() à Configuration :

```
8 def Q1a():
9     return Configuration()
```

Cela affiche



En appuyant sur des touches définies du clavier, cela modifie la fenêtre :

- La touche a fait apparaitre et disparaitre la figure
- La touche z fait tourner la figure dans le sens trigonométrique
- La touche Z fait tourner la figure dans le sens inverse trigonométrique

2. Question (1b)

Dans Configuration.py :

```
def generateCoordinates(self):
    self.vertices = [
            [0, 0, 0 ],
            [1, 0, 0 ],
            [0, 1, 0],
            [0, 0, 1]
            ]
    self.edges = [
            [0, 1],
            [0, 2],
            [0, 3]
            ]
```

generateCoordonates génère des coordonnées pour les vertices et les points

```
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)
```

processKeyDownEvent associe des actions aux évènements :
par exemple il associe les rotations aux touches z et Z et la disparition des axes à la touche a.

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

La taille de la figure et la couleur des axes à changé

Dans main.py
On retrouve :

```
def main():
    # Enlever un des commentaires pour la question traitée

   # configuration = Q1a()
    configuration = Q1b_f()
    # configuration = Q2b()
    # configuration = Q2c()
    # configuration = Q3a()
    # configuration = Q4a()
    # configuration = Q5a()
    # configuration = Q5b()
    # configuration = Q5c1()
    # configuration = Q5c2()
    # configuration = Q5d()
    # configuration = Q6()
    configuration.display()
```

Cela effectue seulement les configurations correspondant aux questions qui ne sont pas commentées. Il faut donc décommenter la configuration de la fonction d'une question et commenter la précédente pour pouvoir l'utiliser.

Le chainage des méthodes setParameter et dislay est possible car ces méthodes n'agissent que sur ce qu'on leur donne en paramètre et non pas sur ce qu'on peut mettre avant l'appel de la fonction. En effet ici cette commande agit comme une suite de commande plutôt que comme une seule grosse commande. Ce traitement particulier concernant screenposition doit être effectué car c'est dans la méthode initializeTransformationMatrix que la screenposition est utilisé et « créer ».

1.c)

Pour avoir le z vertical, le x horizontal et le y en profondeur : on decide de faire une rotation de 90° avec la fonction *gl.glRotatef* de telle sorte que :

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

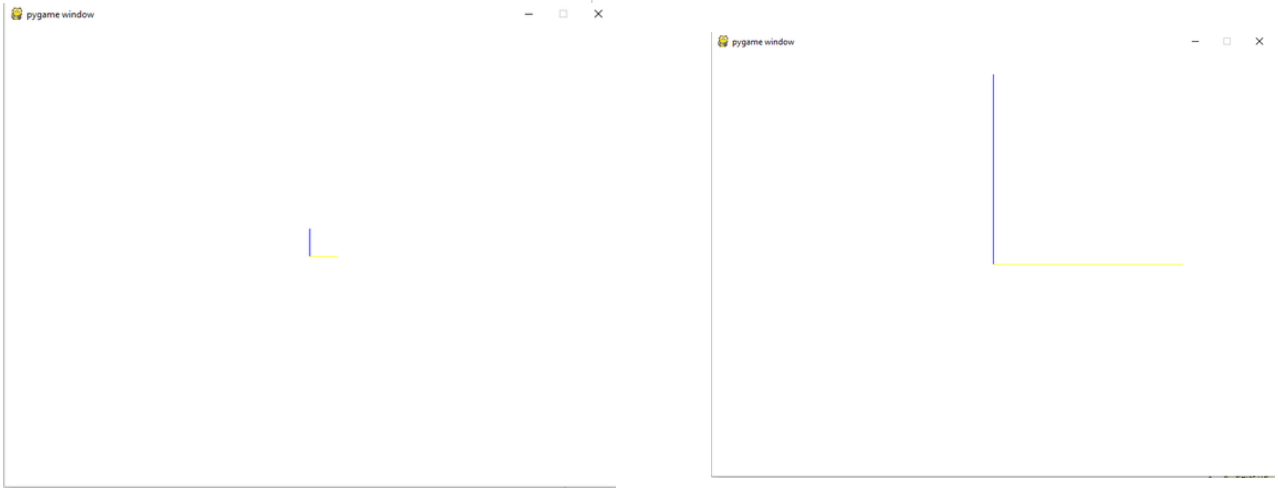
Et cela affiche bien :

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

1.d)

On cherche à ajouter à la fonction **processKeyDownEvent** une fonction qui zoom : Pour cela on utilise les touche PAGEUP et PAGEDOWN

```
# 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'] == 'page up' or self.event.key == pygame.K_PAGEUP :
       gl.glScalef(1.1, 1.1, 1.1)


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



1.e)

```
# 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)
```

On utilise la methode *processMouseButtonDownEvent* pour zoomer et dezoomer avec la molette de la souris.

Comme la molette est représentée par *button*, on appelle *self.event.button* qui prend pour valeur 4 ou 5 et pour chaque cas on associe la fonction zoom *glScalef* avec les mêmes facteurs de formes que dans les questions précédentes .

1.f)

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

Cette méthode permet de bouger et d'effectuer une rotation grâce à la combinaison des fonction gl.gl Rotatef et gl.glTranslatef.

IV- Création d'une section

2.a)

On souhaite ici créer une face, pour cela on créer d'abord les sommets avec les coordonnées, et ensuite les face qui prennent en paramètre chacun des sommets qui les définissent.

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

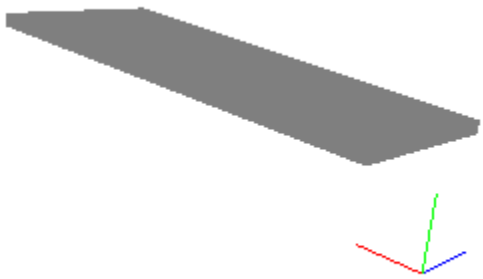
2.b)

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

Configuration().add(section).display()
Avec le add(section) cette instruction ajoute la section définie à notre configuration existante à la position (1,1,0), une section de longueur 7 et de hauteur 2,6
Avec le display() cela permet d'afficher la configuration
Ensuite on écrit la fonction draw dans section.py pour initialiser la section :

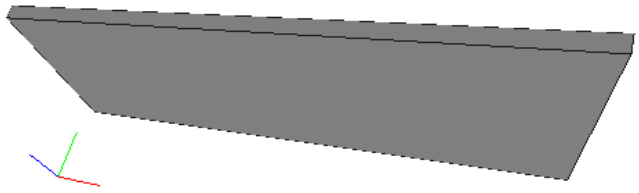
```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][1],self.parameters['position'][0],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    for i in self.faces:

        for j in i:
            gl.glVertex3fv( self.vertices[j])
    gl.glEnd()
    gl.glPopMatrix()
```

Et cela affiche :



2.c)

On reprend le squelette de draw pour la fonction edges
Cela affiche :



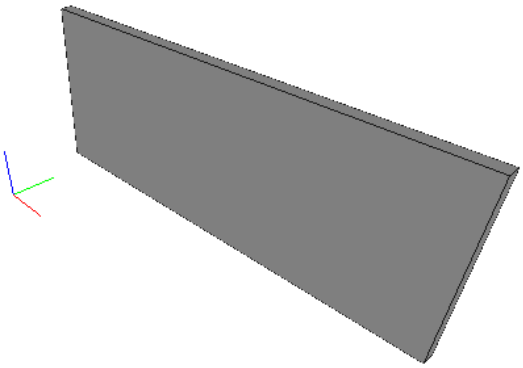
V - Création de mur

3.a)

En analysant le constructeur on remarque qu'il faut impérativement une longueur sur x et z sur le mur. Si le reste des paramètres n'est pas donné en paramètre alors l'origine est 0,0,0, l'épaisseur est de 0,2, l'orientation est nulle et une couleur de base est défini. On ajoute à la liste des objets une section de mêmes paramètres que le mur donné. On réussit à créer un mur à partir des modifications à voir dans les programmes.
On reprend la fonction draw dans Wall :

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

# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
   # gl.glTranslate(self.parameters['position'][1],self.parameters['position'][1],self.parameters['position'][0])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for objects in self.objects:
        objects.draw()

    gl.glPopMatrix()
```

Cela affiche :



VI – Création d'une maison

4.a)

De même manière que dans la question précédente on définit la méthode draw() dans la classe House qui va nous permettre de dessiner notre maison.
On définit 4 murs dans le main, cela affiche les murs extérieurs de la maison.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':5, 'height':2.6, 'edges': True,'orientation':0})
    wall2 = Wall({'position': [0, 0, 0], 'width':5, 'height':2.6, 'edges': True,'orientation':90})
    wall3 = Wall({'position': [0, 5, 0], 'width':5, 'height':2.6, 'edges': True, 'orientation':0})
    wall4 = Wall({'position': [0,-5, -5], 'width':5, 'height':2.6, 'edges': True, 'orientation':90})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall2).add(wall3).add(wall4).add(wall1)
    return Configuration().add(house).display()
```

Nous avons rencontré un problème pour la relation entre les murs. Nous n'avons pas réussi à formé le carré en changeant les coordonnées. En modifiant tour à tour toutes les cordonnées nous avons remarqué que même en changeant les coordonnées en z, cela n'avait aucune incidence sur la position des murs.
Nous en avons conclu que nous avons un problème lors de l'initialisation de la section ou du mur mais nous n'avons pas réussi à l'identifier.