Compte rendu TP3, PART 1

Préparation avant TP :

1) On teste le code suivant :

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

import python → importation du module *pygame*
pygame.init() → initialisation de pygame
ecran = pygame.displlay.set_mode((300,200)) → création d'une fenêtre pygame de dimension 300x200
pygame.quit() → désinitialise le module pygame

Quand on exécute le programme on a une fenêtre de 300x200 qui est créée puis quittée directement.

2) On teste un autre 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()
```

Les 3 premières lignes ont déjà été commentés précédemment.

continuer = True → initialisation d'une variable continuer booléenne à True. Cette variable permet de sortir de pygame à partir du moment où elle passe à False.

***while** continuer:*
→ initialisation de la boucle while. Si on sort de cette boucle, on quitte pygame puisque le code « pygame.quit() » sera exécuté.

for event in pygame.event.get():
→ boucle for permettant d'attendre un évènement

if event.type == pygame.KEYDOWN:
continuer = False
→ On passe la variable continuer à False dès que l'évènement KEYDOWN est réalisé. Cet évènement correspond à l'appuie sur la touche « flèche du bas ».

Concrètement le code ci-dessus initialise une fenêtre puis la ferme dès que la touche « flèche du bas » est pressé.

Utilisation de PyOpenGl pour représenter des objets 3D :

1) Essai d'un code

```
16  import pygame
17  import OpenGL.GL as gl
18  import OpenGL.GLU as glu
19
20  if __name__ == '__main__':
21      pygame.init()
22      display=(600,600)
23      pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
24
25      # Sets the screen color (white)
26      gl.glClearColor(1, 1, 1, 1)
27      # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
28      gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
29      gl.glEnable(gl.GL_DEPTH_TEST)
30
31      glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
32      while True:
33          for event in pygame.event.get():
34              if event.type == pygame.QUIT:
35                  pygame.quit()
36                  exit()
```

Lorsque nous utilisons ce code, une fenêtre vide s'ouvre et reste ouverte. Nous ne pouvons pas vraiment interagir avec, nous pouvons cependant la fermer en cliquant sur la croix grâce aux lignes de code 34 et 35.

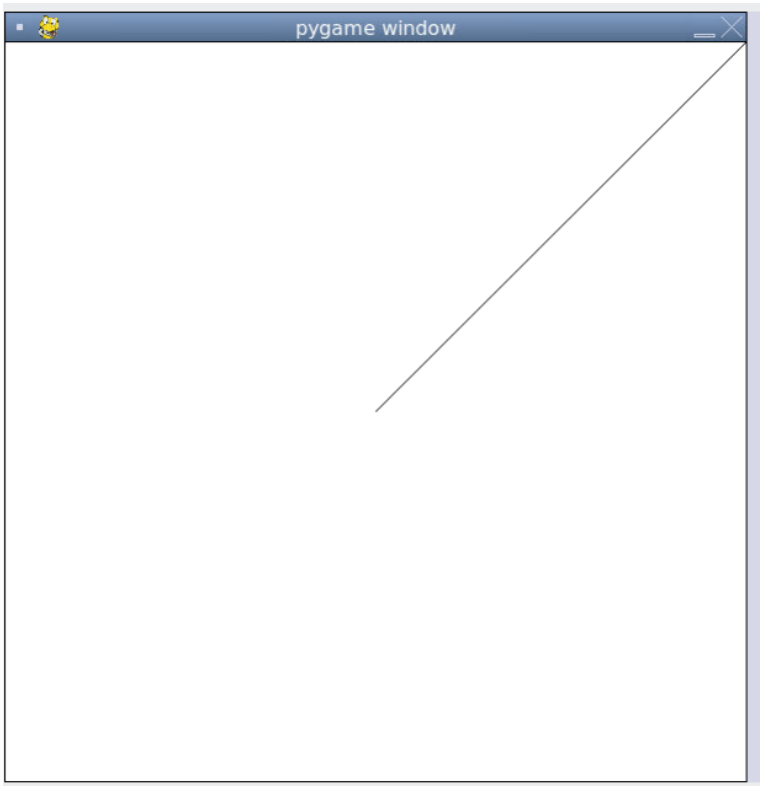
La ligne 31 corresponds à la méthode (fovy, aspect, zNear, zFar) qui initialise une matrice de perspective

2) On trace les axes :

Lorsque l'on copie ce code :

```
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
```

Nous obtenons le tracé suivant :

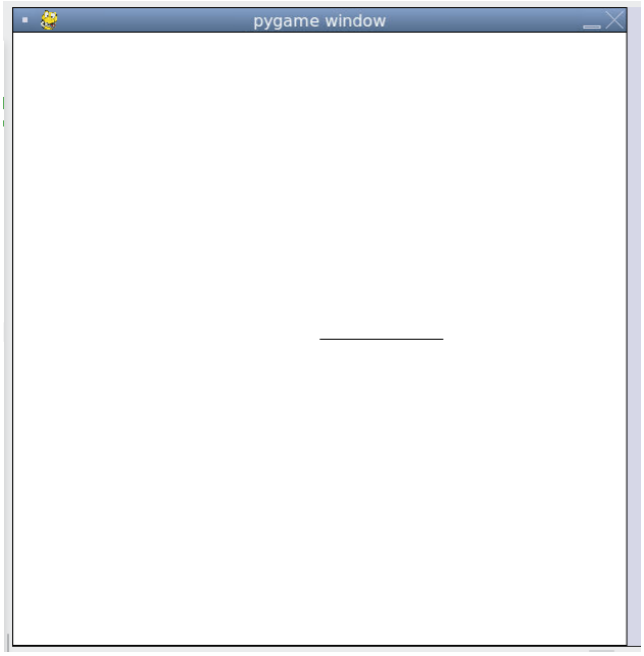


3) Utilisation d'une transformation de translation et rotation :

Nous souhaitons faire pivoter l'axe de -90° autour de x et effectuer une rotation de facteur 0,2 et -5. On exécute donc le code suivant :

```
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 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
```

Ce qui nous donne le résultat suivant :



Découverte de l'environnement de travail du TP :

1/a) Ajout d'une commande à la fonction Q1a() :

Nous complétons la fonction avec un « return Configuration() » pour appeler la classe Configuration du fichier « configuration.py ».

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

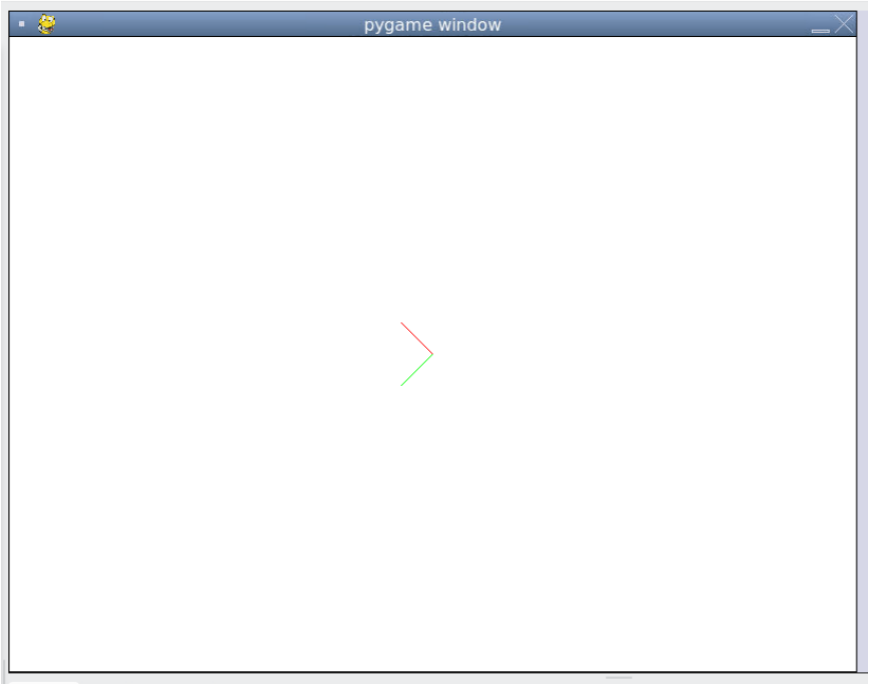
La fonction Q1a() sera par la suite appelée par la fonction main du même fichier (main.py)

```
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()

# Calls the main function
if __name__ == "__main__":
    main()
```

Cela permet de retourner le résultat suivant :



L'appuie sur la touche « a » fait apparaitre ou disparaitre les axes, les touche Z et z font respectivement pivoter l'axe Z vers la gauche et la droite.

En effet, dans la classe Configuration() du fichier « configuration.py », nous avons appelé la fonction ci-dessous qui nous permets d'agir avec les touches précédemment décrites :

```
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)

# Processes the MOUSEBUTTONDOWN event
```

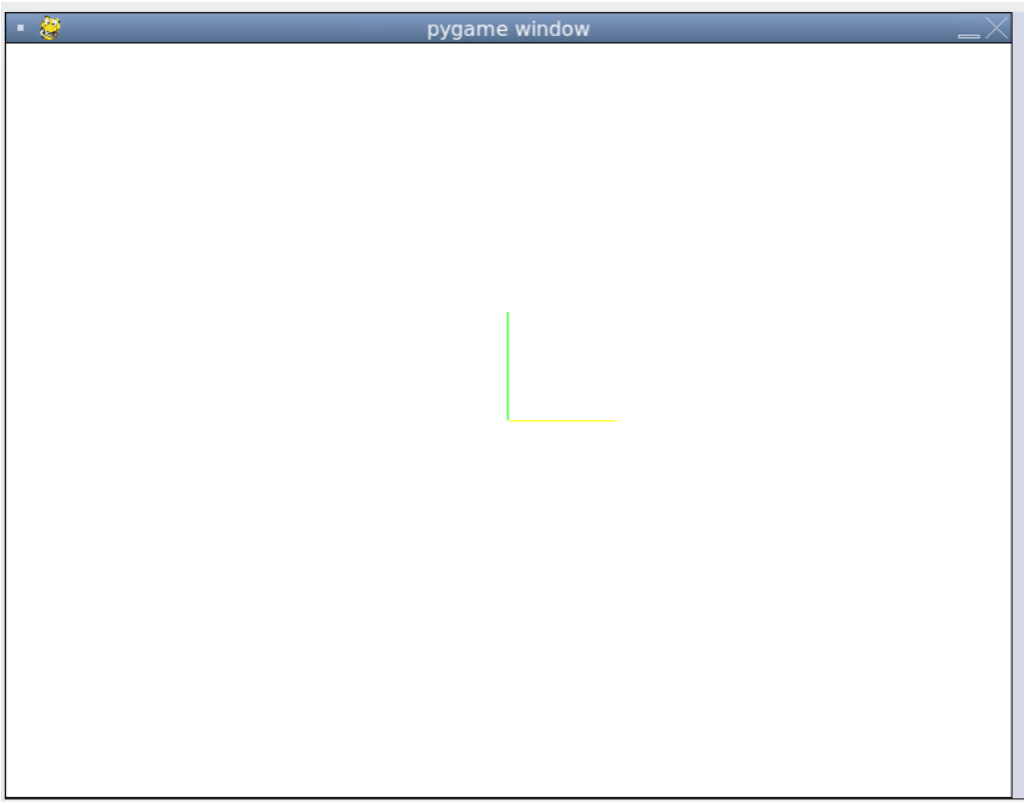
Nous constatons bien que l'appui sur la touche « Z » provoque une rotation autour de l'axe Z de -2.5°. (+2.5° pour « z »). Et que l'appui sur la touche « a » fait disparaitre ou apparaitre la fenêtre de référence c'est-à-dire les axes.

1)b) Modification de la position de l'écran et de la couleur de l'axe x :

Nous exécutons le code suivant :

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

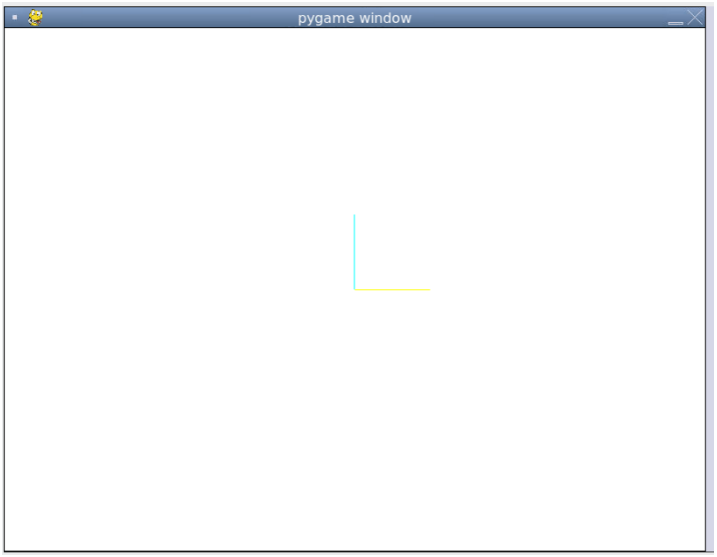
On remarque que les axes ont changé de couleurs selon le code couleur que l'on a indiqué. Ici, on configure l'axe des x avec le code RGB [1,1,0] qui corresponds à du jaune(Rouge + Vert).



Pour continuer, nous effectuons ce code :

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

Qui nous donne le résultat suivant :



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 de setParameter() et .display() est possible car les setParameter vont s’exécuter avant le .display() et le display va être exécuté sur “l’objet” retourné par configuration.

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 choisi -90° (sinon l’axe z pointe vers le bas)
Cf code sur l’image ci-dessous

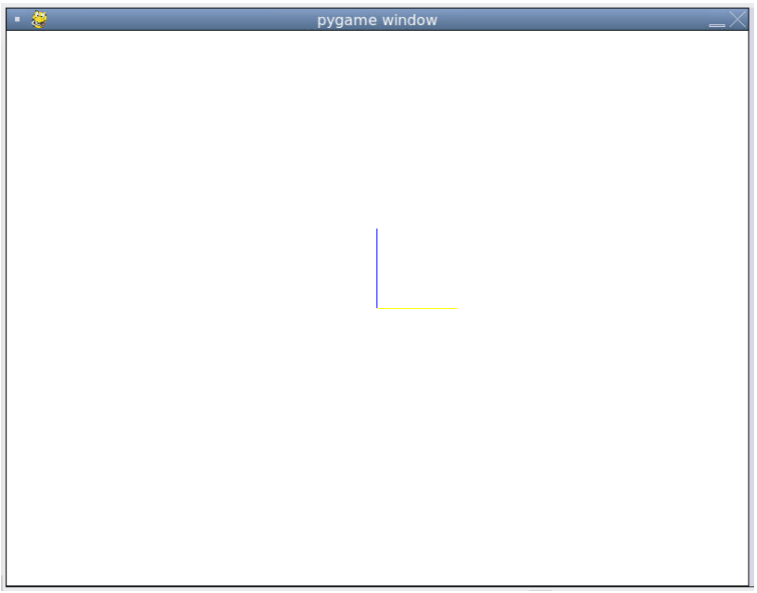
```
# Initializes OpenGL
def initializeOpenGL(self):
    # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)

    # 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)

# 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)
```

Résultat obtenu :



Mise en place des interactions avec l'utilisateur avec Pygame :

1)d) Ajout de la 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 :

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

elif self.event.key == pygame.K_PAGEDOWN:
    gl.glScalef(0.9, 0.9, 0.9)
```

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 :

```
def processMouseButtonDownEvent(self):
    if self.event.type == pygame.MOUSEBUTTONDOWN:
        if self.event.button == 4:
            gl.glScalef(1.1, 1.1, 1.1)
        elif self.event.button == 5:
            gl.glScalef(0.9, 0.9, 0.9)
```

Nous avons :

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

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

1)f) Déplacement des objets affichés par rotation ou translation :

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0] == 1: #= Si le bouton gauche de la souris est préssé then ...
      gl.glRotate(self.event.rel[0], 1, 0, 0) # Rotation autour de x quand la souris se déplace sur l'axe x
      gl.glRotate(self.event.rel[1], 0, 0, 1) # Rotation autour de z quand la souris se déplace sur l'axe y

    if pygame.mouse.get_pressed()[2] == 1: #= Si le bouton droit de la souris est préssé then ...
      gl.glTranslatef(self.event.rel[0], 0, self.event.rel[1]) #translation selon l'axe x composé à une
      translation selon l'axe z
```

Création d'une section

Nous avons rencontré plusieurs difficultés lors de ce TP, comme des problèmes de mises à jour Spyder puis des problèmes avec le logiciel Spyder. Aussi, certaines petites erreurs comme des espaces en trop nous ont fait perdre un peu de temps (parfois les erreurs évidentes sont les plus difficiles à trouver).

De cette façon, même si nous avancions bien plus vite sur la fin nous n'avons pas pu aller au-delà de « Création d'une section ». Nous comptons donc sur la 2ème partie du TP pour rattraper notre retard et le finir. Nous rattraperons aussi un peu de notre retard avant le prochain TP.