Dubois Lucas
Laroudie Vincent
09/12/2021

Rapport de TP3 – Représentation visuelle d'objets

I/ Introduction

L'objectif de ce TP est de s'approprier la représentation d'objets 3D à l'écran dans une fenêtre graphique qui permet des opérations de zoom, rotation est translations.

II/ Travail préparatoire

Utilisation de Pygame

1)

Explication du code :

La première ligne « import pygame » permet d'importer le package pygame. La deuxième ligne permet d'initialiser. A la troisième ligne « ecran = pygame.display.set_mode((300, 200)) » permet de créer une fenêtre en lui passant des paramètres de dimension, ces paramètres indiquent que le fenêtre fera 200 pixels de haut et 300 pixels de large. La quatrième ligne permet de nettoyer, c'est-à-dire libérer les ressources dont nous n'avons plus besoin.

Lorsque l'on exécute le code, une fenêtre noire apparaît qui se referme instantanément.

2)

Lorsque nous lançons le second programme, la fenêtre ne se referme plus immédiatement. Mais dès que l'on appuie sur une touche, la fenêtre se ferme immédiatement.

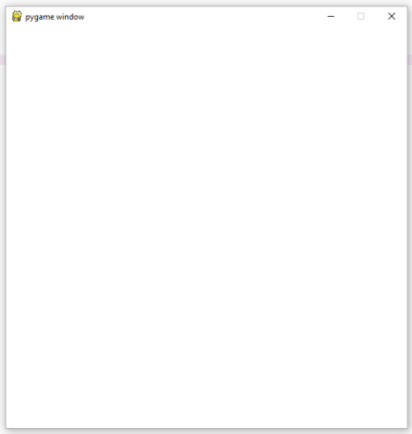
Explication du code :

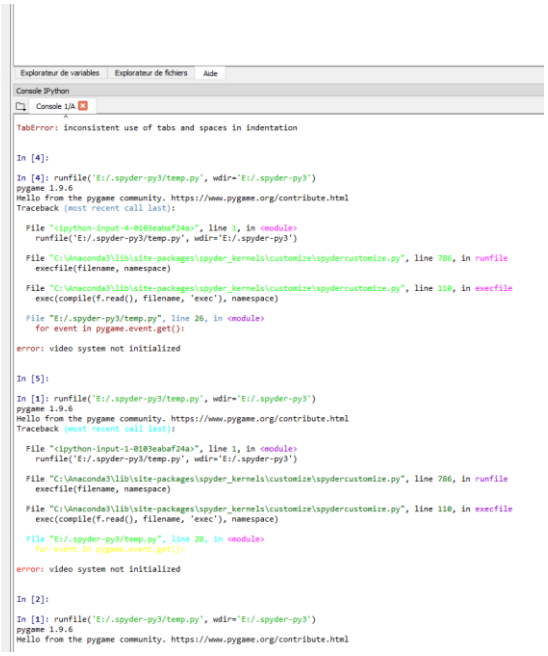
La première ligne permet de garder la fenêtre ouverte. Les boucles while et for permettent de récupérer tous les fichiers créés de pygame. Les lignes 4 et 5 vérifient si l'on appuie sur une touche du clavier (KEYDOWN) et si c'est le cas, permettent de stopper la fonction continue. La boucle s'arrête et la dernière ligne permet de fermer la fenêtre.

Utilisation de Pyopengl pour représenter des objets 3D

1)

Nous avons effectué le code suivant afin de gérer la perspective :
perspective = glu.gluPerspective (45, 1, 0.1, 50)



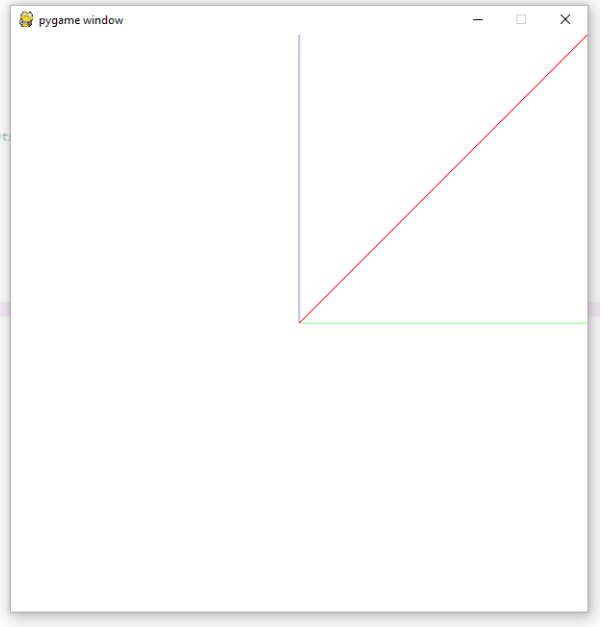


Il n'y a pas d'erreurs.

2)

Voici le code que nous avons effectué ainsi que notre résultat obtenu :

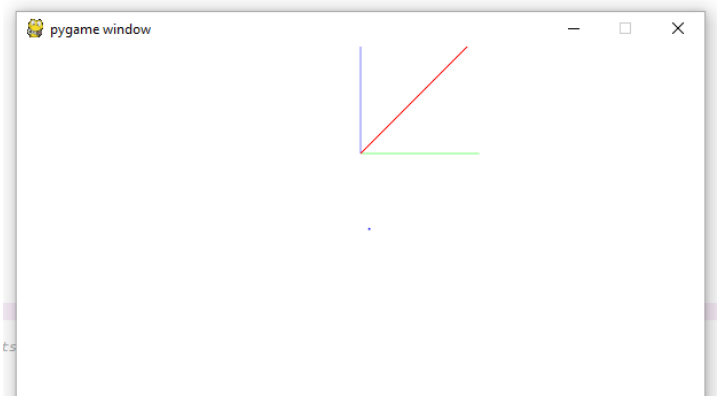
```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

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, 1, 0.1, 50)

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

    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((1, 0, -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, 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()
```



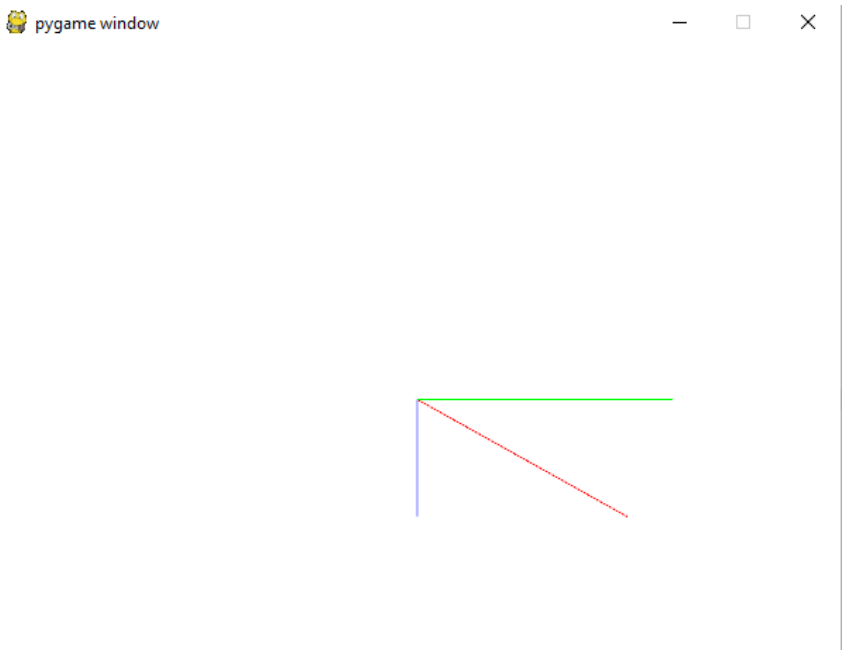
Dans la ligne « gl.glColor3fv([0, 0, 0]) », le 1er chiffre correspond au rouge, le 2e correspond au vert et le 3e chiffre correspond au bleu.

3)

Voici la translation que nous avons effectué, Nous avons décalé y de 2 et z de -5. Voici notre résultat :



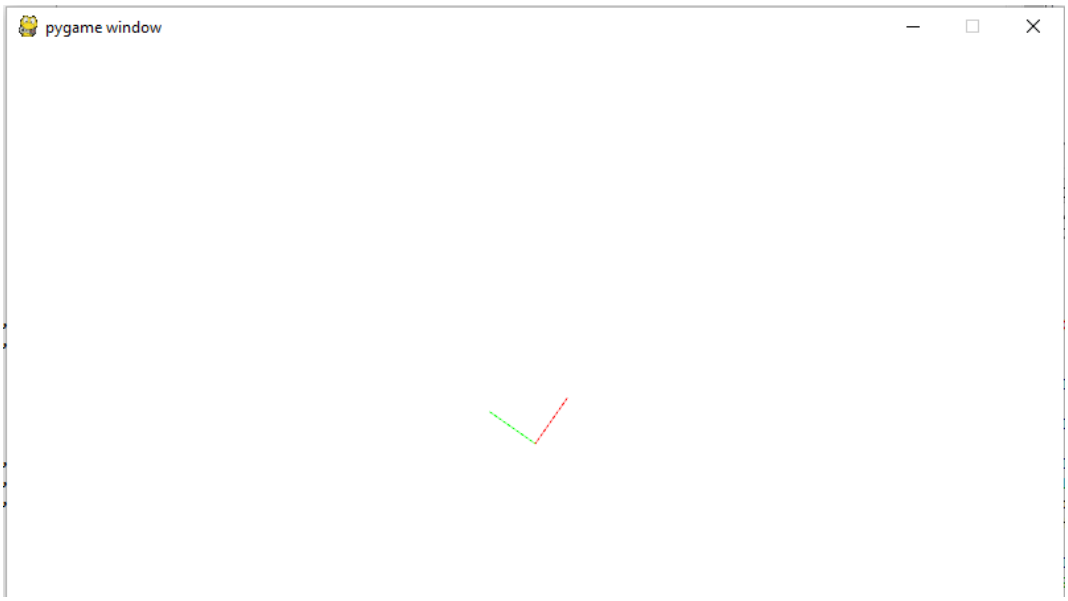
Voici la rotation que nous avons effectué. Nous avons effectué une rotation de 120 degrés autour de l'axe x. Voici notre résultat :



Découverte de l'environnement de travail

1) a)

Nous remarquons que lorsque nous appuyons sur la touche a, le graphique disparaît, lorsque l'on réappuie sur la touche, le graphique réapparait. Lorsque l'on appuie sur la touche z, le graphique tourne dans le sens trigonométrique et sur la touche Z (maj+z), le graphique tourne dans le sens horaire.



Dans le Main, on importe toutes les classes de chaque module. On peut exécuter l'ensemble des modules avec tous les paramètres inscrit dans le Main.

1) b)

Le code en question permet de motifier la couleur des axes

Voici ce que nous obtenons :

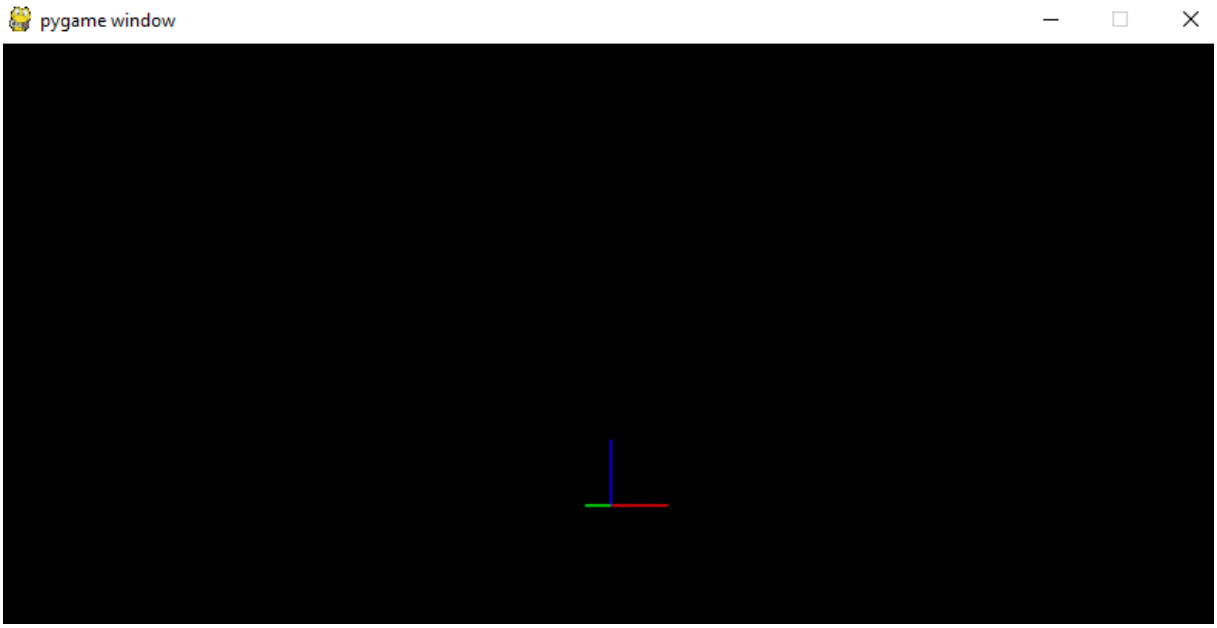
Le chainage de l'appel des méthodes SetParameter() et display est disponible car pour utiliser display et donc afficher les axes, il faut les paramétrer à l'aide de SetParameter.

1) c)

Pour que l'axe z soit représenté verticalement sur l'écran et que l'axe x soit représenté horizontalement, nous avons fait une rotation de 90° autour de l'axe x. Voici notre code que nous avons modifié :

```
gl.glMatrixMode(gl.GL_MODELVIEW)
gl.glLoadIdentity()
gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
gl.glRotatef(-90,1,0,0)
```

Voici ce que nous obtenons :



On a bien l'axe z qui est représenté verticalement sur l'écran et l'axe qui est représenté horizontalement.
De plus, nous avons changé le fond d'écran pour plus de visibilité.

III/ Mise en place des interactions avec l'utilisateur avec Pygame¶

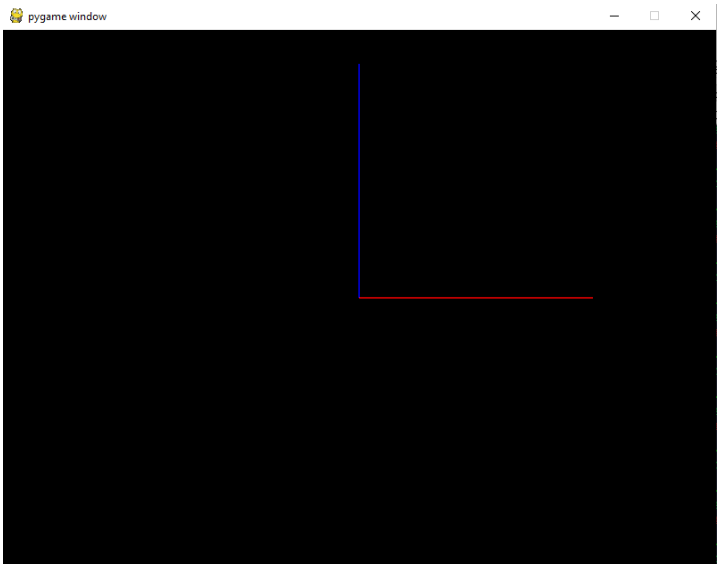
1) d)

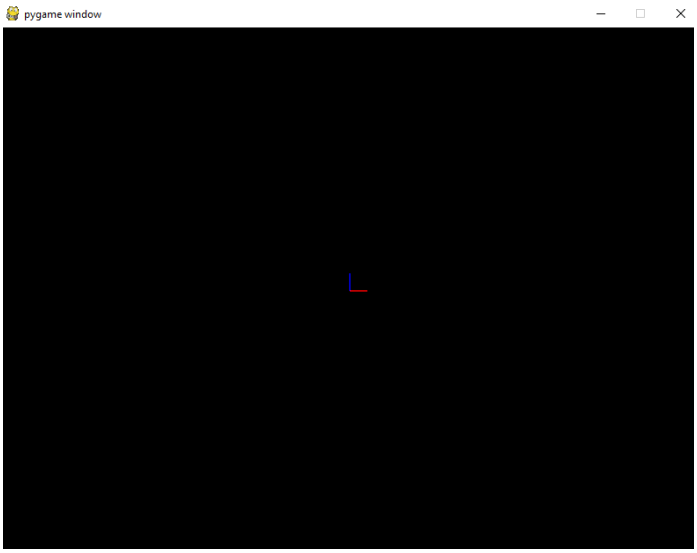
Voici le code que nous avons effectué :

```
    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 pour les trois axes un facteur d'échelle pour un zoom positif de 1.1 et pour un zoom négatif de 1/1.1.

Voici les résultats que nous avons obtenu :





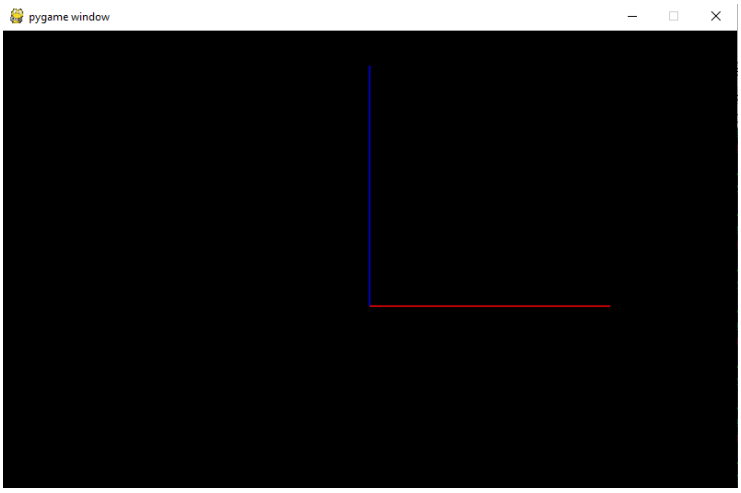
1) e)

L'obejectif est maintenant de pouvoir zoomer à l'aide de la roulette de la souris. Voici le code que nous avons réalisé :

```
# 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)
```

Comme pour la question précédente, nous avons ajouté pour les trois axes un facteur d'échelle pour un zoom positif de 1.1 et pour un zoom négatif de 1/1.1.

Voici un exemple que nous avons obtenue :



1) f)

Voici le code que nous avons effectué :

```python
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if self.event.type == pygame.MOUSEMOTION:
        if pygame.mouse.get_pressed()[0]==1:
            gl.glRotate(self.event.rel[1], -1, 0, 0)
            gl.glRotate(self.event.rel[0], 0, 0, -1)
        else:
            gl.glRotate(0,0,0,0)
        if pygame.mouse.get_pressed()[2] == 1:
            gl.glTranslate(0.1*self.event.rel[0], 0, 0.1*self.event.rel[1])
        else:
            gl.glTranslate(0,0,0)
```

Quand on clique et que l'on bouge avec la sourie, tous les axes bougent. Le code de fonctionne donc bien correctement.

IV/ Création d'une section

2) a)

Voici notre méthode **generate(self) :**

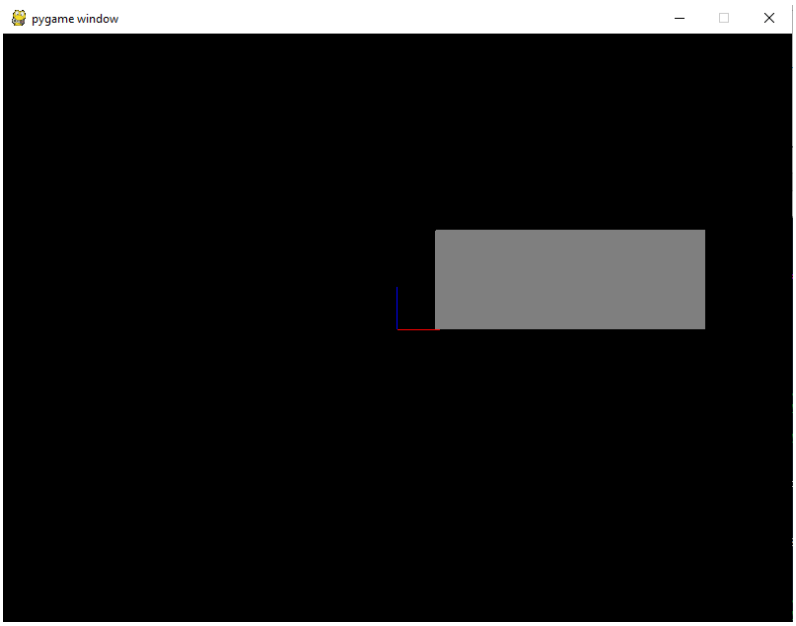
```python
# Defines the vertices and faces
def generate(self):
    self.vertices =[
            [0, 0, 0 ],
            [0, 0, self.parameters['height']],
            [self.parameters['width'], 0, self.parameters['height']],
            [self.parameters['width'], 0, 0],
            [0, self.parameters['thickness'],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

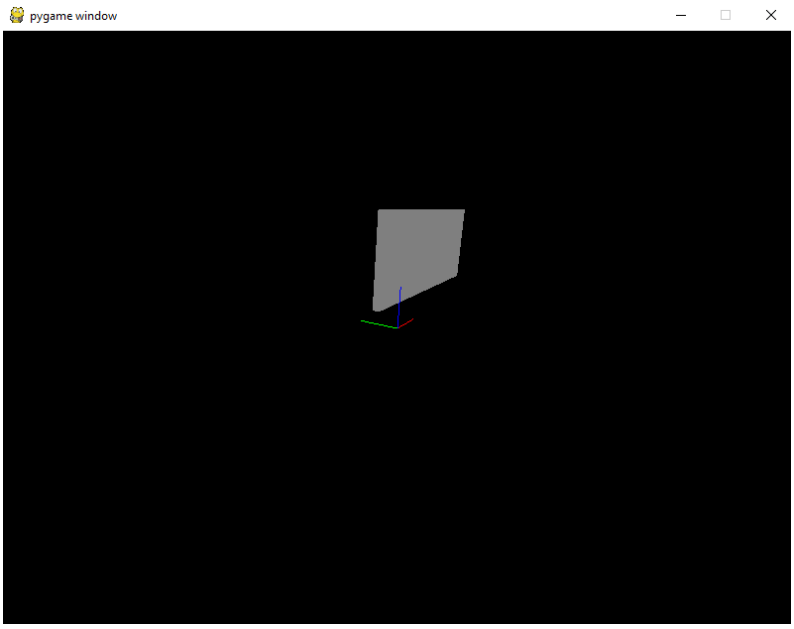
Dans cette méthode, on génère tout d'abord les 8 sommets puis ensuite une matrice qui génère les faces correspondantes à ses sommets.

2) b)

La fonction Q2b () crée une configuration et ainsi l'espace disponible afin de pouvoir créer la face. Ensuite on ajoute une section en lui donnant une position ainsi qu'une dimension.
Le .display permet d'afficher une section.

Voici ce que nous obtenons :

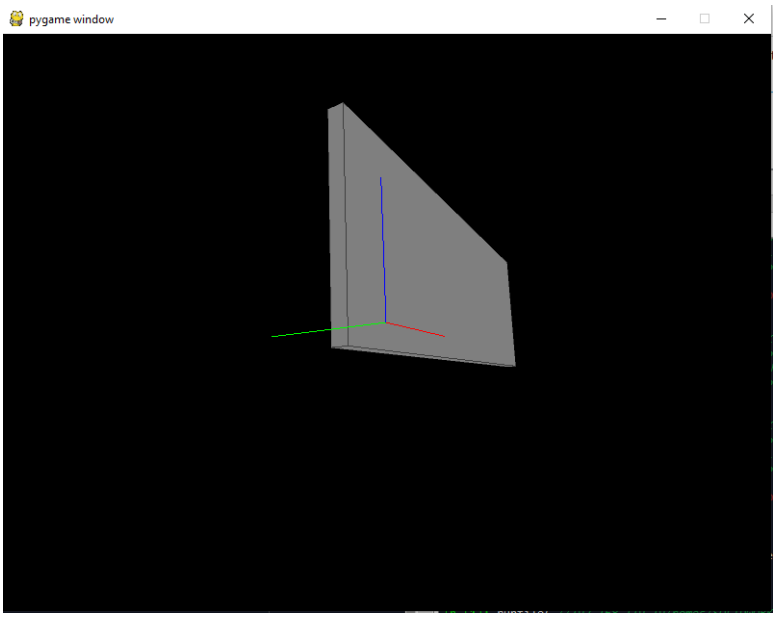




Nous obtenons donc bien une face en 3D.

2) c)

Voici ce que nous obtenons :

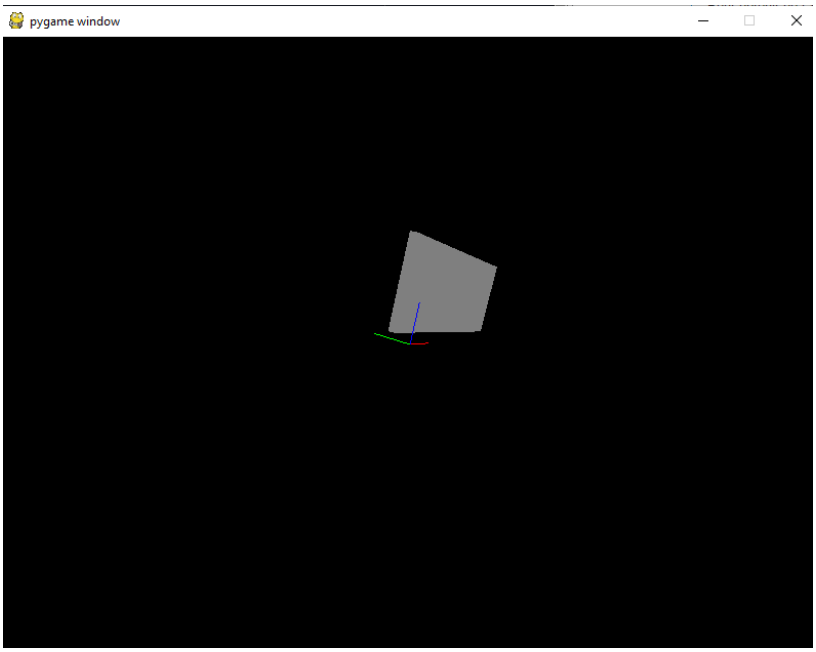


Nous arrivons bien à visualiser les arêtes.

V/ Création des murs

Dans le fichier Wall et notamment dans son constructeur, il y a une succession de tests qui vérifient si les paramètres de parameters sont bien présents. S'il en manque, il va en fixer par « défaut » avec des valeurs simples.

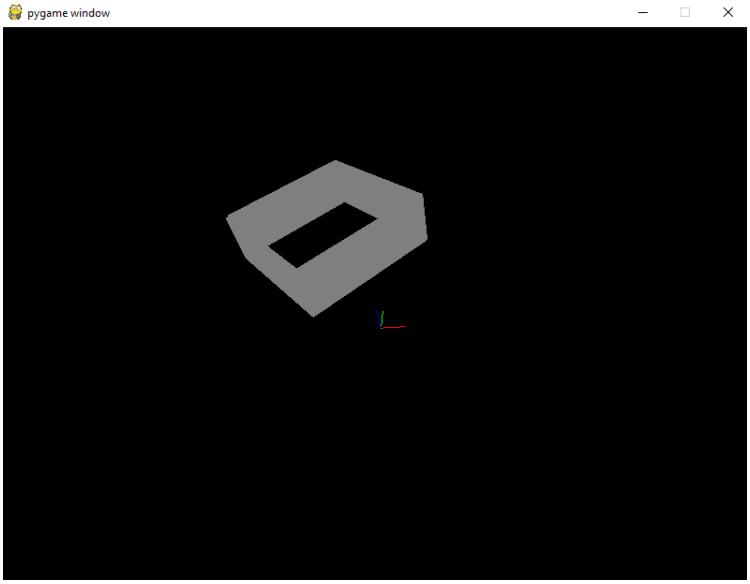
Voici ce que nous obtenons :



On créé également une liste d'objets auxquels on va assigner une section selon des paramètres donnés et requis dans la classe section. Pour finir, nous ajoutons les objets créés à la liste des objets dans l'objectif de pouvoir les représenter plus tard.

VI/ Création d'une maison

Voici la maison que nous obtenons :



Pour cela on utilise les 4 murs que nous avons créé dans le Main, que nous positionnons de manière à former une maison.

VII/ Création d'ouverture

5) a)

Ici, on utilise approximativement le même principe que pour générer des murs, sauf que l'on veut afficher que les faces extérieures. Donc on créé les 8 points dans l'espace et en suite les 4 faces du bord de la fonction.

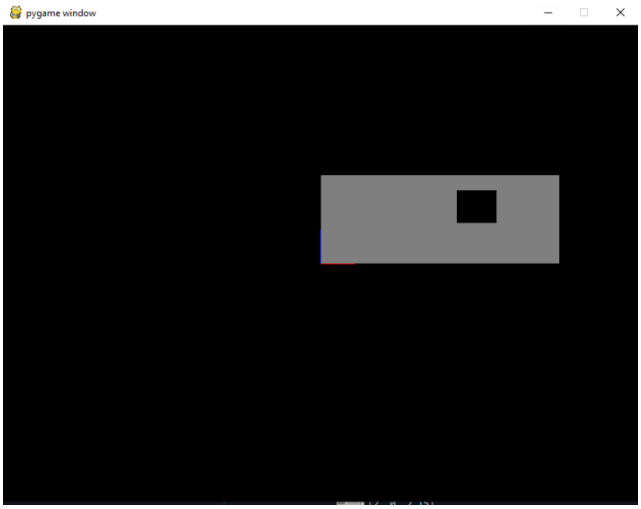
5) b)

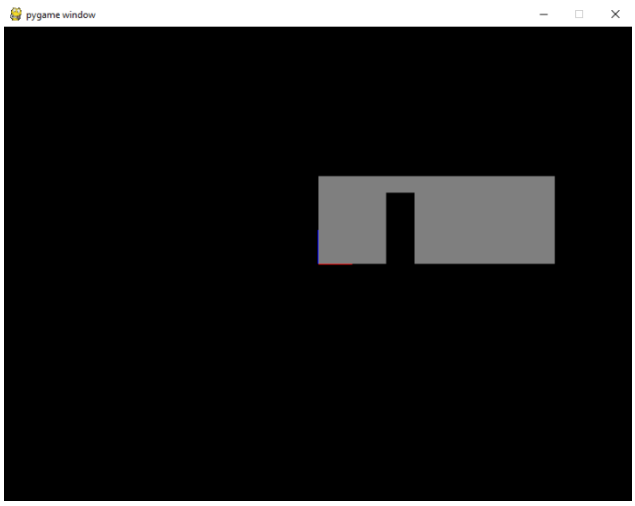
La méthode canCreateOpening permet donc de vérifier si une ouverture peut être créée. Donc, on vérifie si la position et les dimensions de l'ouverture ne dépasse pas celle du mur.
Le dernier affichage vaut False car on a une section qui fait 7 de width et 2.6 de height, cependant dans opening3, la position de la hauteur qui faut 1.6 plus la hauteur de l'ouverture qui est de 1 dépasse les 2.6 de hauteur de la section.

5) c)

A partir de la position et des dimensions de l'ouverture, on calcul 4 nouvelles sections qui sont créées en réutilisant les méthodes de création d'une section.
Ensuite, dans le Main on test la possibilité et on va renvoyer les coordonnées et les dimensions des sections créée par notre ouverture.

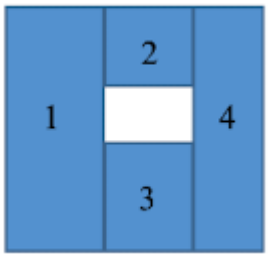
Voici les deux ouvertures que nous arrivons à créer :





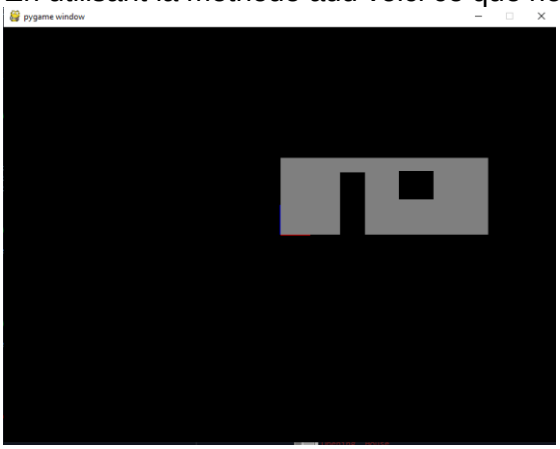
5) d)

Enumarate () sert à compter le nombre d'objets que l'on veut parcourir, donc ici le nombre de section afin de les parcourir 1 à 1.
Dans section 0, on aura les informations de la partie 1 et dans section 1 on aura les informations de la partie 2…



On a donc une liste de petite section qui si on les ajoute entre elles cela crée une nouvelle section avec la fenêtre au milieu.

En utilisant la méthode add voici ce que nous obtenons :



VII/ Conclusion

En conclusion de ce tp qui a duré 2 séances, nous l'avons trouvé très intéressant. D'une part car c'était quelque chose de très nouveau pour nous et ne pensions pas que nous pouvions visualiser des objets visuels. Ce tp montre bien une autre forme d'application du langage python ce que nous trouvons assez impressionnant. D'autre part, le fait de rester 2 séances sur ce tp permettait bien de se familiariser avec le sujet, ce qui nous a aider à comprendre au mieux le but et le fonctionnement de ce tp.