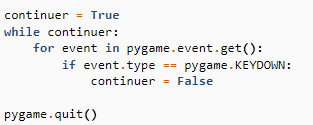
TP info 3

# II - Préparation à faire avant le TP

## Utilisation de Pygame

1) Lorsque que nous copions le premier code, il ne se passe rien car nous ne donnons pas d’instruction. La fenêtre pygame se ferme instantanément.

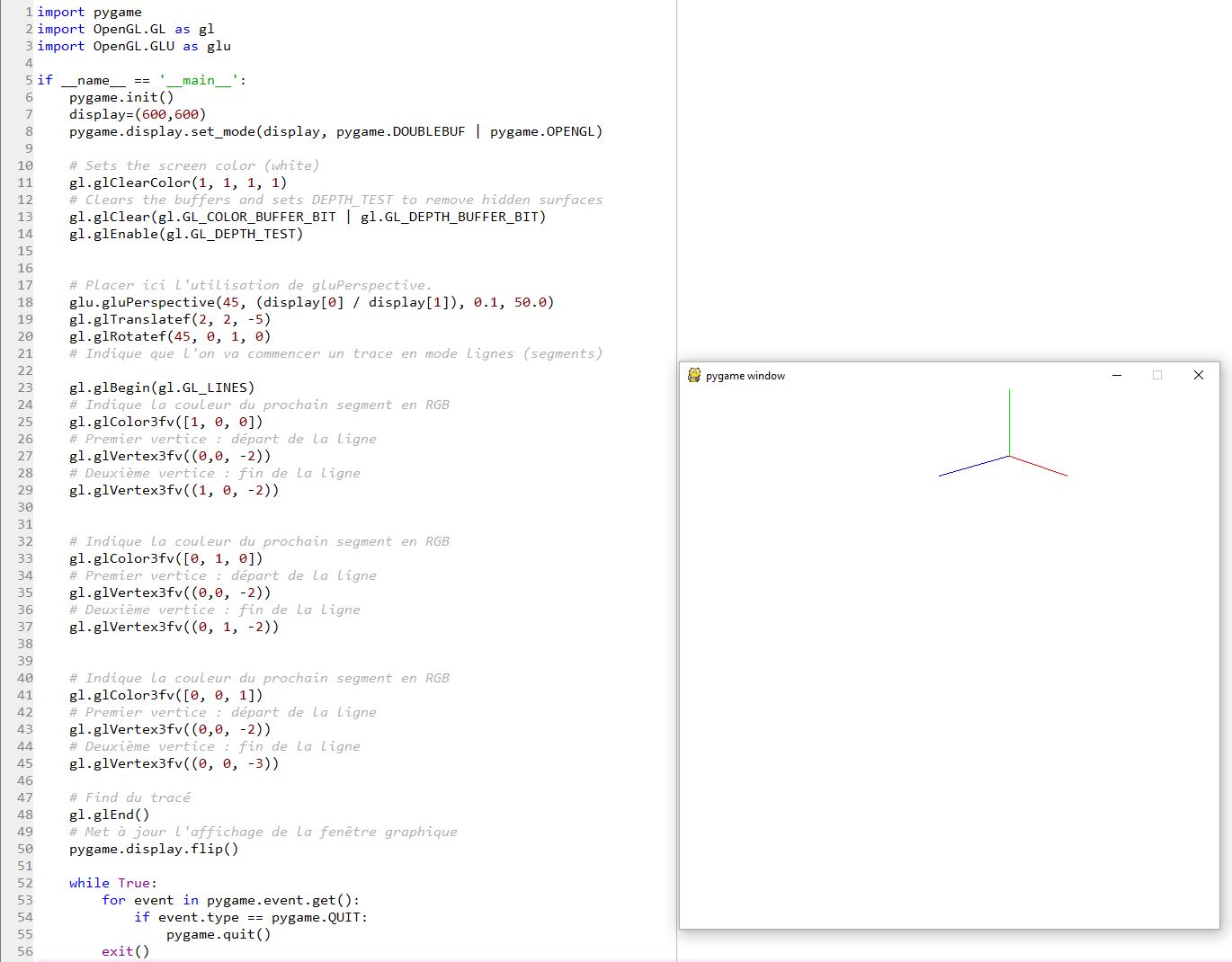
2) Pour le second code la fenêtre reste ouverte grâce aux nouvelles instructions. Cette partie du code permet à la fenètre de rester ouverte tant que nous clicons sur rien.

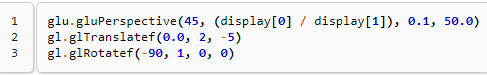
3) Le troisième code permet d’avoir l’affichage de pygame en blanc. Puis maintenant pour fermer la fenêtre pygame il faut appuyer sur la croix. Appuyer sur n’importe quel autre bouton ne ferme plus la fenêtre.

## Utilisation de Pyopengl pour représenter des objets 3D

2) Nous avons donc tracé 3 segments de couleurs différents et orthogonaux entre eux. Puis pour voir les trois segments lors de l’affichage nous avons dû choisir une certaine rotation et une certaine translation de façon à pouvoir voir les trois axes.

Pour tracer les axes nous avons tapé le code suivant :



3) Cette commande permet de faire des translations et des rotations selon les différents axes. 

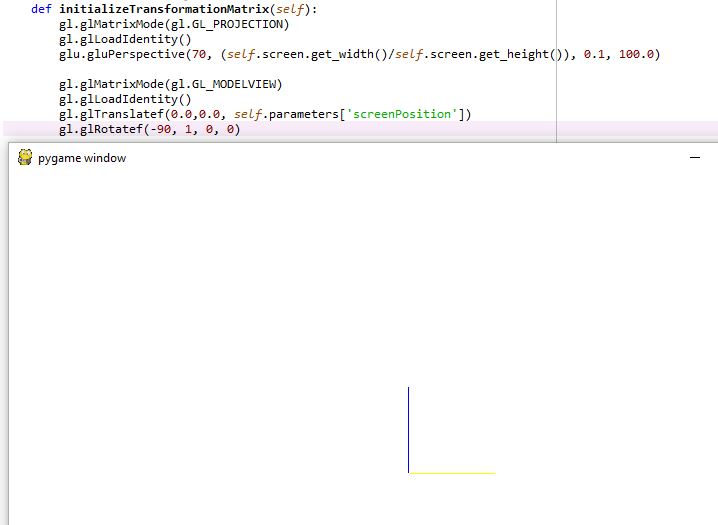
## Découverte de l’environnement du travail du TP

(1). a) La taille des axes est petite, les couleurs sont vert et rouge et la vitesse de rotation est lente par rapport à (1).b)

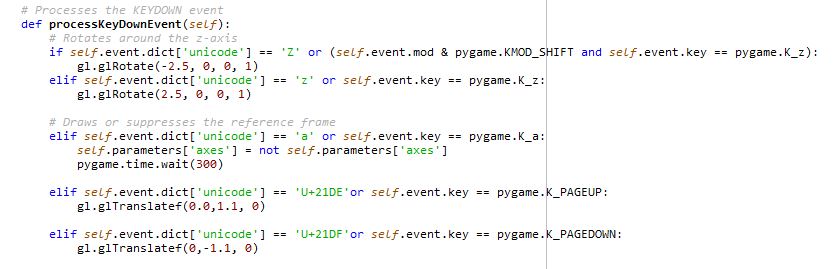
(1). b)Ici nous avons changé la couleur des axes, accéléré la vitesse de rotation. On a rapproché le point de vue donc les axes paraissent plus grands. On est passé d’un point de vue de -10 à -5.

Plutôt que d’écrire .display() à chaqueligne de laconfiguration, on utilise . \ puis un display() à la fin.

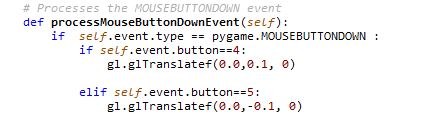
(1). c)Pour mettre l’axe z de façon verticale nous avons dû faire une rotation de -90 selon l’axe x. Le code et les résultats sont ci-dessous :

****

# III- Mise en place des interactions avec l’utilisateur avec Pygame

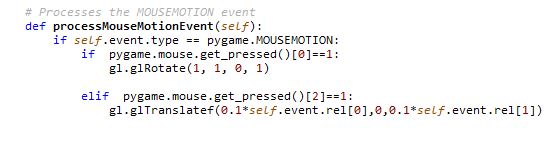
(1). d) Le code pour réduire ou zoomer l’image est le suivant : ****

Comme vous avons maintenant l’axe y qui correspond à la profondeur, nous avons dû faire une translation selon cet axe pour pouvoir faire une zoome avant et arrière**.**

(1). e) Le code pour réduire ou zoomer l’image grâce à la souris est le suivant :

Tout d’abord on regarde si nous avons un élément de type ‘ la mollette bouge’. Si oui on regarde si on l’a tourne vers le haut ou le bas. En effet la valeur 4 correspond à un mouvement vers le haut. Si c’est le cas alors on fait une translation selon y qui correspond à un zoome avant. Si la valeur correspond à 5 alors on a un mouvement de la mollette vers le bas. Dans ce cas on fait un zoome arrière.

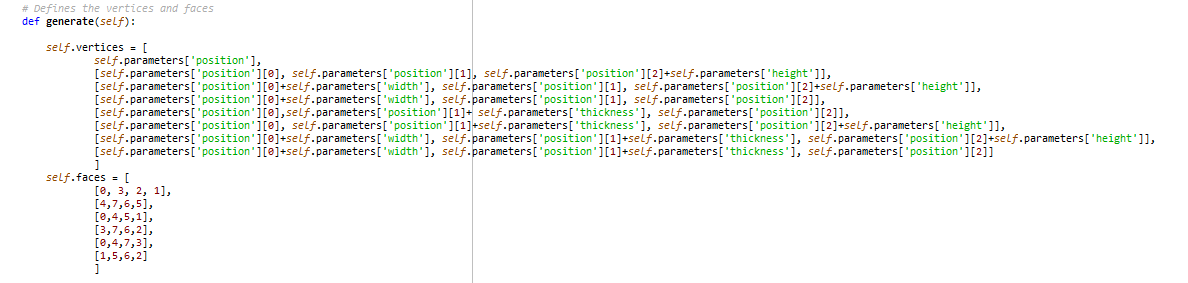
(1). f) Voici le code pour déplacer et tourné notre repère :

****

Tout d’abord nous regardons si nous avons un évènement du type ‘la souris est en mouvement’. Si oui alors on regarde si durant cet évènement le bouton gauche ou droit est pressé.

Si le bouton gauche est pressé on fait une rotation selon x et z. Si c’est le bouton droit qui est pressé on fait une translation.

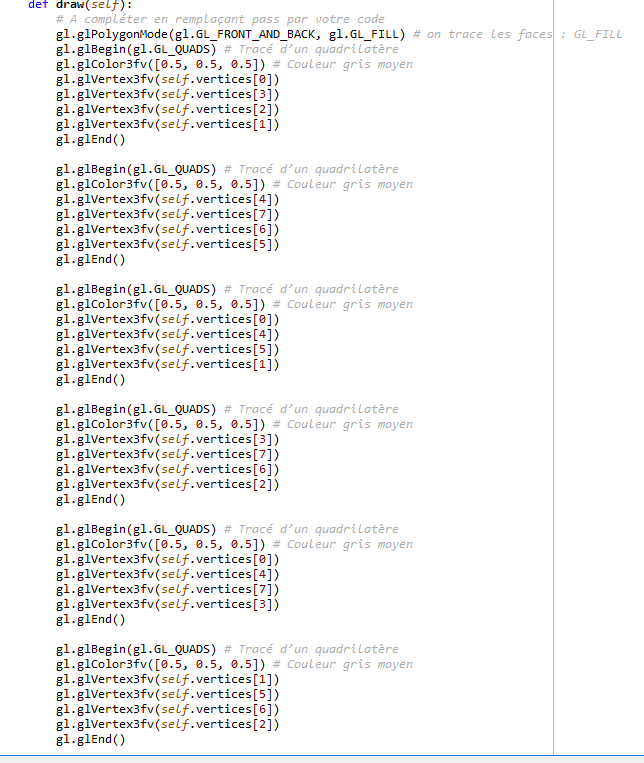
# lV création de section

(2)a) Voici le code pour générer des sommets et des faces : 

Dans un premier temps on crée 8 vertices (sommets), puis dans un deuxième temps on crée des faces. Les faces sont des rectangles délimités par les vertices créé précédemment.

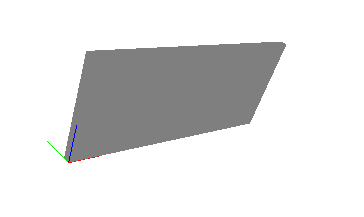
(2)b) La méthode Q2b permet d’ajouter une section et de la représenter. Sa position initiale à pour coordonner [0, 0, 0], longueur 7 et hauteur 2.6 et par défaut les autre paramètres sont définies. Par exemple on aura une largeur de 0.2.

Voici le code de la class draw () :

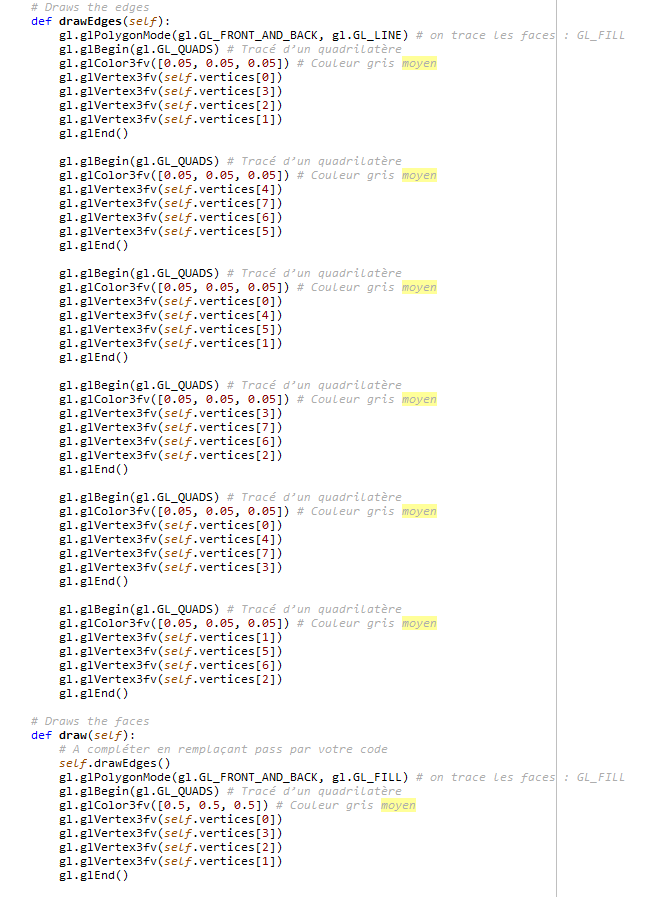


Dans cette classe nous créons toutes les faces à partir des points dans le bon ordre. En effet si nous ne mettons pas les points dans le bon ordre, l’affichage n’est plus un rectangle mais 2 triangles.

Remarque : on aurait pu faire une boucle pour sélectionner automatiquement les valeurs des vertices, allégeant le code.



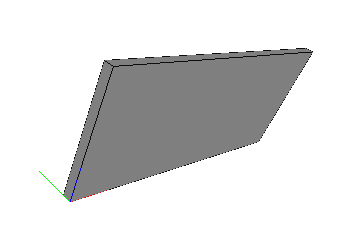
L’exécution de la méthode Q2b nous donne cette représentation dans pygame.



(2)c) Voici le code de la méthode drawEdge :

Pour écrire ce code nous avons eu la même logique de pour le code de la classe ‘draw’ sauf que la fonction change. En effet nous avons un ‘gl.GL\_LINE’. Ce code permet de tracer et de colorer seulement les arrêtes des quadrilatères. Nous avons bien fait attention à changer la couleur. Pour cela nous avons dû mettre des composantes proches de 0 dans les 3 coordonnées de couleurs. Cela nous donne une teinte sombre visible.

Remarque : on aurait pu faire une boucle pour sélectionner automatiquement les valeurs des vertices, allégeant le code.



# V Création des murs

(3)a) Le constructeur de la class Wall prend 6 paramètres :

Position : qui permet de placé le premier vertices du quadrilatère dans le plan

Width : qui est la longueur du mur (il est impératif de la remplir)

Height : qui est la hauteur du mur (il est impératif de la remplir)

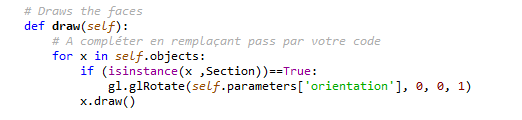
Thickness : qui est la largeur du mur (il est impératif de la remplir)

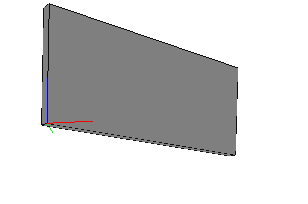
Orientation : qui permet la rotation d’un mur autour de l’axe Z (permet d’avoir des murs facilement perpendiculaire)

Color : qui donne une certaine couleur aux murs.

Un mur pour l’instant est une section.

Voici le code de la méthode draw :

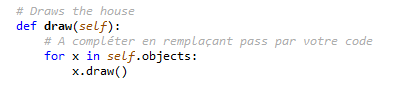




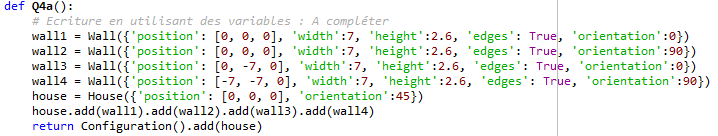
Tout t’abord on parcourt la liste ‘objects’ qui est composé de section. Puis si notre mur à une orientation alors on fait une rotation de cette valeur selon l’axe Z. Puis on dessine notre section dans la console pygame.

# Vl création d’une maison

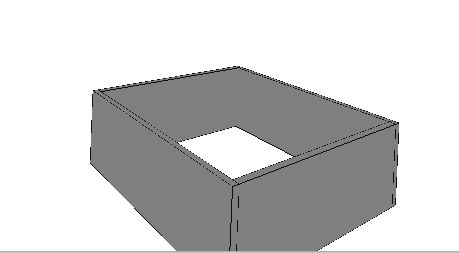
Voici le code de draw dans la class wall :



Les modifications de la méthode Q4a() :



Le résultat dans pygame.



Pour former notre maison nous avons créé 2 murs d’orientation 0 et deux autre mur d’orientation 90. Pour nous avons joué sur la position de nos murs de façon à avoir une maison ‘fermé’.

# Création d’ouverture

Nous n’avons pas eu le temps de finir, ayant rencontré des difficultés avec l’orientation.