

# TP 3 : Représentation visuelle d'objet

## Introduction

Ce tp a pour but de nous faire créer une maison, on va apprendre à se familiariser avec les classes pygame et OpenGL. On va aussi comprendre comment intégrer des petites fonctions dans la fonction principale avec l'aide de paramètre.add. On apprend aussi à faire des initialisation de position, d'orientation et couleurs afin de bien différencier les éléments.

## 1.Préparation avant le Tp

### A. Utilisation de Pygame

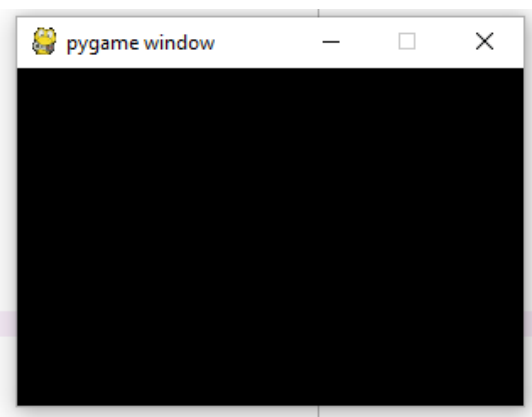
#### Question 1 :

```
8 import pygame
9 pygame.init()
10 ecran = pygame.display.set_mode((300, 200))
11 pygame.quit()
```

Ces quelques lignes permettent d'importer le module pygame. Celui-ci est directement initialisé afin d'utiliser la méthode `pygame.display.set_mode()`. Cette méthode permet de définir l'écran utilisé. L'origine de l'affichage est en haut à gauche et augmente vers le bas et la droite avec les valeurs saisies. Cette méthode supprime tous types d'écran déjà présents. De plus, cette méthode ne nous restreint pas sur la modification des pixels plus tard.

#### Question 2 :

```
8 import pygame
9
10
11 pygame.init()
12 ecran = pygame.display.set_mode((300, 200))
13
14 continuer = True
15 while continuer:
16     for event in pygame.event.get():
17         if event.type == pygame.KEYDOWN:
18             continuer = False
19
20 pygame.quit()
```



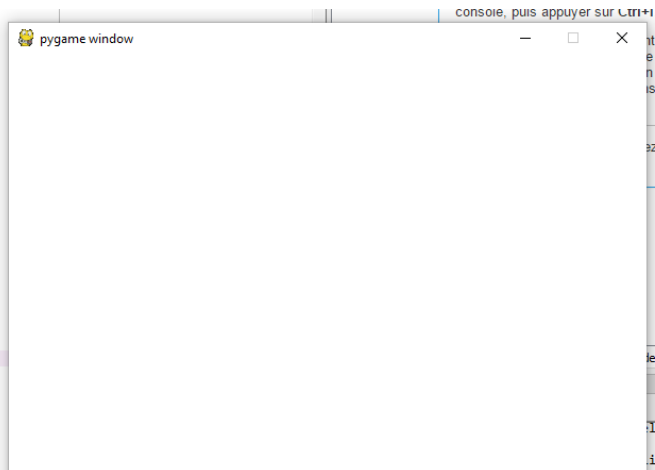
On constate effectivement l'ouverture d'une fenêtre. La méthode `pygame.event.get` crée un événement. Celui est complété avec un booléen qui est tout le temps sur vrai. La fonction

pygame.KEYDOWN permet de définir une action. En effet, en appuyant sur n'importe quelle touche du clavier la fenêtre contenant notre écran de ferme.

## B. Utilisation de Pyopengl pour représenter des objets 3D

### Question 1 :

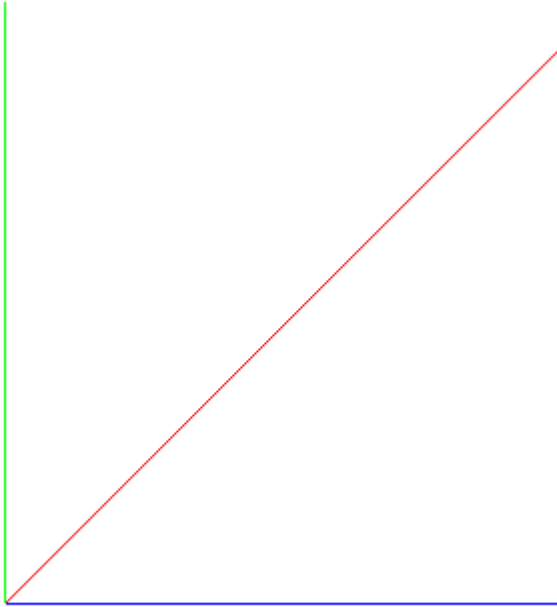
```
9 import pygame
10 import OpenGL.GL as gl
11 import OpenGL.GLU as glu
12
13 if __name__ == '__main__':
14     pygame.init()
15     display=(600,600)
16     pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
17
18     # Sets the screen color (white)
19     gl.glClearColor(1, 1, 1, 1)
20     # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
21     gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
22     gl.glEnable(gl.GL_DEPTH_TEST)
23
24     # Placer ici l'utilisation de gluPerspective.
25
26     while True:
27         for event in pygame.event.get():
28             if event.type == pygame.QUIT:
29                 pygame.quit()
30                 exit()
31
```

The image shows a code editor on the left with Python code for initializing a 3D window using Pygame and OpenGL. The code includes imports for pygame, OpenGL.GL, and OpenGL.GLU, followed by a main function that initializes pygame, sets the display mode to DOUBLEBUF and OPENGL, clears the screen to white, and enables depth testing. A while loop is present for handling events. On the right, a window titled 'pygame window' is shown, which is currently blank. A status bar at the top right of the window says 'console, puis appuyer sur Ctrl+I'.

Premièrement les modules OpenGL.GL et OpenGL.GLU sont importés en plus de pygame. Ensuite tout se déroule main dans le main. La fenêtre est agrandie et transformée en blanc. Les méthodes du module gl. Permettent de manipuler directement l'écran comme par exemple sa couleur et les surfaces.

### Question 2 :

Ces lignes de codes permettent d'afficher les lignes de graphiques, on répète 3 fois l'opération afin d'afficher les 3 axes.



On trace chaque axe avec une couleur différente.

Question 3 :

### C. Découverte de l'environnement du travail du TP

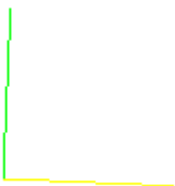
Question 1.a :

Le code dans Q1 Configuration.py crée des événements lorsqu'on appuie sur des touches est appuyée. Quand on appuie sur la touche "Z" l'axe fait une rotation autour de l'axe z dans le sens positif. Alors que "Z" l'axe effectue une rotation dans le sens négatif. La touche "a" permet de faire disparaître et apparaître les éléments.

Le code permet d'afficher 2 axes : l'axe en rouge et l'axe y en vert.

Question 1.b :

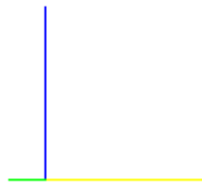
La modification du code rend l'axe x qui était rouge en vert. Screen position : -5, décale le graphique de -5.



Les `setParameter()` initialise le graphique et définit les axes, on finit avec un `display` pour les intégrer dans la fenêtre.

#### Question 1.c :

Pour que l'axe z apparaisse verticalement, on ajoute le code qui suit à la méthode `initializeTransformationMatrix()` dans le programme `Configuration.py`. A l'aide de la méthode `rotate` appartenant au module `gl` nous réalisons une rotation de  $90^\circ$  dans le sens antihoraire sur x :



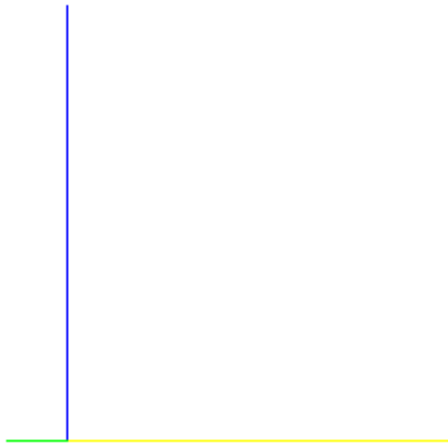
```
gl.glRotate(-90,1,0,0)
```

## 2. Mise en place des interactions avec l'utilisateur avec Pygame

#### Question 1.d :

On ajoute à la gestion des touches via la méthode `processKeyDownEvent(self)`, la gestion des touches permettant de zoomer et dézoomer. La touche "U" (Up) correspond à un zoom de  $\times 1.5$  et "D" (down) à un dézoom de  $\times 0.5$ . Cela est géré grâce à une boucle `elif` permettant de proposer un cas par touche. Ensuite on associe un événement à chaque touche. Enfin le zoom se gère via la méthode `glScale(...,...,....)`.

```
# Grossi l'affichage-Page Up
elif self.event.dict['unicode'] == 'U':
    gl.glScale(1.5,1.5,1.5)
# Réduit l'affichage-Page Down
elif self.event.dict['unicode'] == 'D':
    gl.glScale(0.5,0.5,0.5)
```



En appuyant sur “U”



En appuyant sur “D”

#### Question 1.e :

Il faut maintenant réaliser la même opération mais en utilisant la molette de la souris. Il faut maintenant créer une autre méthode `processMouseButtonDownEvent(self)`. Il faut ensuite créer un événement pour le zoom et un pour le dézoom. La molette avant correspond à la touche 4 de la souris et la molette arrière à la touche 5 de la souris.

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.type == pygame.MOUSEBUTTONDOWN:
        # zoom avant avec la molette de la souris
        if self.event.button == 4:
            gl.glScale(1.5,1.5,1.5)
        # zoom avant avec la molette de la souris
        elif self.event.button == 5:
            gl.glScale(0.5,0.5,0.5)
```

#### Question 1.f :

Cette question va nous permettre de pouvoir observer nos axes dans tous les angles possible en déplaçant celui grâce à notre curseur de souris.

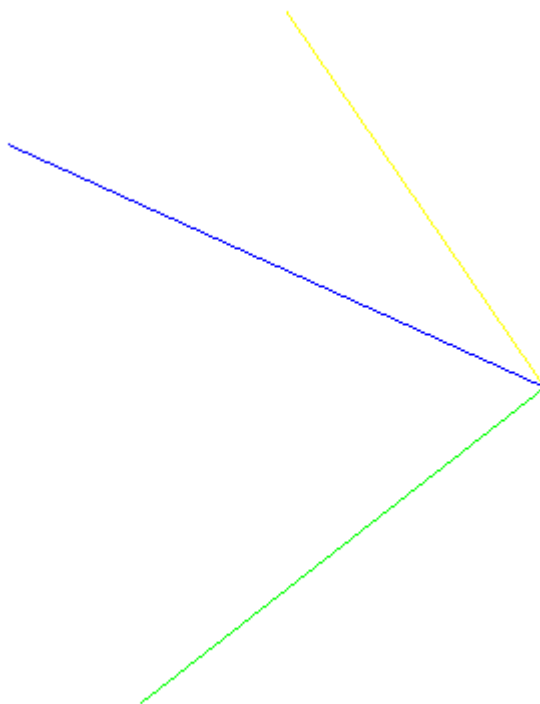
Ici le clic gauche correspond à la rotation, on a donc `get_pressed()[0]` qui correspond au clic gauche de la souris, avec le `if` on autorise les rotations faites par le clic de la souris, le `sinon` correspond au non clic et donc à la rotation = 0.

Alors que le clic droit correspond à la translation `get_pressed()[2]` qui correspond au clic droit de la souris, si les conditions du `if` sont respectées alors on peut traduire ici un clic droit sinon pas de translation quand il n'y a pas de clic.

```

def processMouseEvent(self):
    if self.event.type == pygame.MOUSEMOTION:
        # rotation clic gauche souris et translation clic droit souris
        if pygame.mouse.get_pressed()[0]==1:
            gl.glRotate(self.event.rel[0],1,0,0)
            gl.glRotate(self.event.rel[1],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)

```



rotation

Ici on voit qu'on peut effectuer une

### 3. Création d'une section

#### Question 2.a :

La création d'une section se réalise en plusieurs étapes: le dessin et le remplissage par une couleur. La section se divise en 3 paramètres: width (largeur), height (hauteur), thickness (profondeur) et position (position du coin inférieur gauche). Il faut compléter la méthode def generate(self): dans le fichier section.py.

Il faut d'abord créer toutes les arêtes puis toutes les faces avec 'vertices' et 'faces'.

```
# 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]
    ]
```

### Question 2.b :

Ici, on ajoute une section afin d'avoir un objet section avec ses sommets et ses faces défini par glVertex, on définit aussi une couleur avec glColor et 0.5 pour la couleur grise, on instancie la forme qui est un quadrilatère.

```
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)

    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
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([1, 0, 0])
    gl.glVertex3fv([1, 0, 1])
    gl.glVertex3fv([0, 0, 1])
    gl.glEnd()
```

Cette consigne est configurée en ajoutant le code de section. On rentre cette commande dans la console ainsi on peut voir la section de mur rentré dans le code de section.

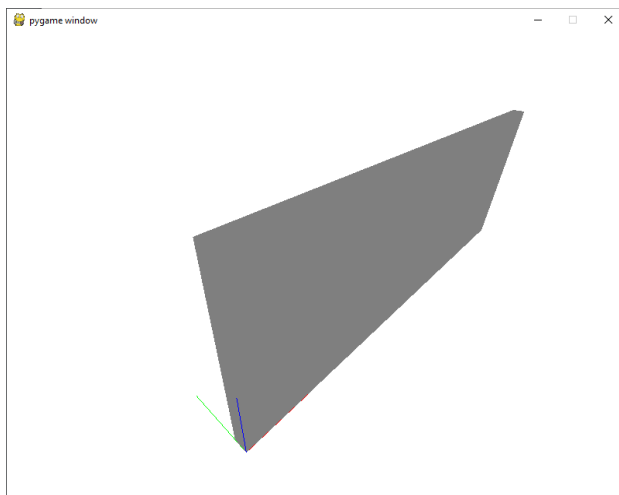
```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
        Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
    ).display()
```

Il faut ensuite colorier les faces à l'aide de la méthode draw. Il faudra aussi paramétrer les conditions des arêtes.

```
# Draws the faces
def draw(self):

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

    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],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
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # TracÃ© d'un quadrilatÃ¨re
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for j in i:
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```



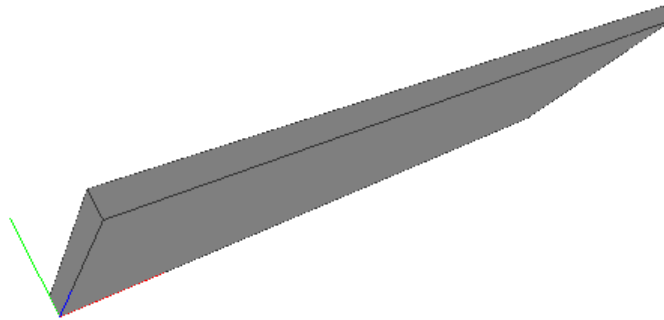
### Question 2.c :

La méthode drawEdges nous permet de colorier les arêtes afin de les rendre visibles. Cette méthode est similaire à la précédente sauf que vertex devient face et donc des modifications s'imposent.



```
# Draws the edges
def drawEdges(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # TracÃ© d'un quadrilatÃ¨re
        gl.glColor3fv([1,0,0]) # Couleur gris moyen
        for j in i:
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

Le résultat de la création de la section est :



## 4. Création des murs

### Question 3.a :

Le constructeur permet d'initialiser l'ensemble des paramètres tels que la position, la taille, la couleur, etc... La variable paramètre est d'initialiser de manière classique avec self. Les autres paramètres sont initialisés à condition de ne pas faire partie de "paramètre" à l'aide d'une boucle if. Chaque paramètre ressort initialisé avec une valeur de base correspondant au type de chacun.

```

class Wall:
    # Constructor
    def __init__(self, parameters = {}):
        # Parameters
        # position: position of the wall
        # width: width of the wall - mandatory
        # height: height of the wall - mandatory
        # thickness: thickness of the wall
        # color: color of the wall

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'position' not in self.parameters:
            self.parameters['position'] = [0, 0, 0]
        if 'width' not in self.parameters:
            raise Exception('Parameter "width" required.')
        if 'height' not in self.parameters:
            raise Exception('Parameter "height" required.')
        if 'orientation' not in self.parameters:
            self.parameters['orientation'] = 0
        if 'thickness' not in self.parameters:
            self.parameters['thickness'] = 0.2
        if 'color' not in self.parameters:
            self.parameters['color'] = [0.5, 0.5, 0.5]

        # Objects list
        self.objects = []

        # Adds a Section for this object
        self.parentSection = Section({'width': self.parameters['width'], \
                                     'height': self.parameters['height'], \
                                     'thickness': self.parameters['thickness'], \
                                     'color': self.parameters['color'], \
                                     'position': self.parameters['position']})
        self.objects.append(self.parentSection)

```

Pour réaliser un mur, le procédé est la même qu'une section. C'est pour cela que nous utilisons seulement un objet qui a l'orientation en plus pour pouvoir bien placer et orienter le mur afin de créer notre maison.

```

# Draws the faces
def draw(self):

    gl.glPushMatrix()
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for i in self.objects:
        i.draw()
    gl.glPopMatrix()

def Q3a():
    return Configuration().add(
        Wall({'position' : [1, 1, 0], 'width':7, 'height':2.6,'edges': True})
    ).display()

```

Ici on intègre la classe Wall dans la classe principale avec return configuration().add, et tout en définissant les paramètres de la classe Wall qui sont la position, x, z et l’affichage des bordures.

## 5. Création d’une maison

### Question 4.a :

Ici on initialise la classe house avec des paramètres par défaut.

A chaque fois que le paramètre est vide on l’initialise :

On initialise donc la position, l’orientation

Ensuite on crée un autre objet parentWall afin d’enregistrer les données du mur afin de ne pas le modifier et de garder la rotation.

```

class House:
    # Constructor
    def __init__(self, parameters = {}):
        # Parameters
        # position: position of the house
        # orientation: orientation of the house

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'position' not in self.parameters:
            self.parameters['position'] = [0, 0, 0]
        if 'orientation' not in self.parameters:
            self.parameters['orientation'] = 0

        # Objects list
        self.objects = []

    # Getter
    def getParameter(self, parameterKey):
        return self.parameters[parameterKey]

    # Setter
    def setParameter(self, parameterKey, parameterValue):
        self.parameters[parameterKey] = parameterValue
        return self

    # Adds an object
    def add(self, x):
        self.objects.append(x)
        return self

```

Ensuite avec add on ajoute des objets pour créer les différents murs de la maison. Enfin draw dessine les murs, ici on mets des position et des rotations différentes afin que les murs ne soit pas confondu

```

# Draws the house
def draw(self):
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for i in self.objects :
        i.draw()
    gl.glPopMatrix()

```

Le draw permet de créer les variables pour la translation ainsi que pour la rotation des murs. Ici, translate contient 3 paramètres, 3 positions différentes initialisées dans un tableau.

Quand à rotate, 4 paramètres sont nécessaires : un de rotation, les 3 autres sont de simples chiffres pour positionner la rotation.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position' : [0, 0, 0], 'width':7, 'height':3, 'edges': True, 'orientation' : 0 })
    wall2 = Wall({'position' : [0, -7, 0], 'width':7, 'height':3, 'edges': True, 'orientation' : 90 })
    wall3 = Wall({'position' : [0, 7, 0], 'width':7, 'height':3, 'edges': True, 'orientation' : 0 })
    wall4 = Wall({'position' : [0, 0, 0], 'width':7, 'height':3, 'edges': True, 'orientation' : 90 })
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house).display()
```

Dans la classe principale, on envoie la classe house par le biais de la création de nouvelle variable composée de la classe wall et de ces différents paramètres (position et orientation varie pas les 3 autres) pour chaque mur pour qu'ils ne soient pas confondus. Enfin house permet de bien positionner la maison par rapport à l'origine.

## 6. Création d'ouvertures

### Question 5.a :

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

# Draws the faces
def draw(self):
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # Trace un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for j in i:
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

La méthode `generates` est semblable à celles présentes dans les autres classes. Il faut seulement gérer 4 faces car seul le contour du rectangle est rempli pour laisser une ouverture. Pour colorier les faces, on s'aide de `gl` afin d'impliquer la couleur aux surfaces.

### Question 5.b :

Ici, on modifie la classe `section` et plus particulièrement la méthode `canCreateOpening(self,x)`. Cette méthode retourne *True* si une ouverture, représentée par `x`, peut être ajoutée dans la section, représentée par `self`, et *False* sinon.

```
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    return (self.parameters['height']>=x.getParameter('height') + x.getParameter('position')[2] - self.parameters['position'][2]
            and x.getParameter('position')[2]>= self.parameters['position'][2]
            and self.parameters['width']>= x.getParameter('width') + x.getParameter('position')[0] - self.parameters['position'][0]
            and x.getParameter('position')[0] >= self.parameters['position'][0])
```

### Question 5.c :

On décompose la section afin de créer l'ouverture voulue comme une porte ou fenêtre. Chaque section a ses paramètres c'est pour cela que l'on crée 1 variable par section que l'on définit avec des initialisations des différents paramètres les composants.

```
def createNewSections(self, x):
    if self.canCreateOpening(x):
        section = [Section(copy.copy(self.parameters))
                    .setParameter('width', x.getParameter('position')[0] - self.getParameter('position')[0]),
                    Section(copy.copy(self.parameters))
                    .setParameter('height', self.getParameter('height') - x.getParameter('height') - x.getParameter('position')[2] + self.getParameter('position')[2])
                    .setParameter('width', x.getParameter('width'))
                    .setParameter('position', [x.getParameter('position')[0], self.getParameter('position')[1], x.getParameter('position')[2] + x.getParameter('height')
                    ]),
                    Section(copy.copy(self.parameters))
                    .setParameter('height', x.getParameter('position')[2] - self.getParameter('position')[2])
                    .setParameter('width', x.getParameter('width'))
                    .setParameter('position', [x.getParameter('position')[0], self.getParameter('position')[1], self.getParameter('position')[2]
                    ]),
                    Section(copy.copy(self.parameters))
                    .setParameter('width', self.getParameter('width') - x.getParameter('width') - x.getParameter('position')[0] + self.getParameter('position')[0])
                    .setParameter('position', [x.getParameter('position')[0] + x.getParameter('width'), self.getParameter('position')[1], self.getParameter('position')[2]
                    ])
        ]
        res=[]
        for i in sections:
            print(i.getParameter('width'))
            print(i.getParameter('height'))
            print(i.getParameter('position'))
            if (i.getParameter('width')!= 0.0 and i.getParameter('height')!=0.0):
                i.generate()
                res.append(i)
        return res
```

```

def Q5c1():
    section = Section({'width':7, 'height':2.6})
    opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createOpening(opening1)
    configuration = Configuration()
    for x in sections:
        configuration.add(x)
    return configuration

def Q5c2():
    section = Section({'width':7, 'height':2.6})
    opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createNewSections(opening2)
    configuration = Configuration()
    for section in sections:
        configuration.add(section)
    return configuration

```

### Question 5.d :

Maintenant il suffit d'insérer un objet vide pour créer l'ouverture, l'objet est inséré là où il n'y a aucune section. D'où le balayage avec le for afin de s'apercevoir des dimensions du trou et ainsi de la taille de l'objet à créer.

```

# Finds the section where the object x can be inserted
def findSection(self, x):
    for item in enumerate(self.objects):
        if isinstance(item[1], Section) and item[1].canCreateOpening(x):
            return item
    return None

```

## Conclusion

Ce Tp nous a appris de nombreuses choses, à commencer par les différentes bibliothèques tel que pygame ou bien OpenGL. Nous avons pu apprendre à créer de sous classes telle que wall.py ou House.py et donc après de les réunir dans un la classe main, avec la commande return Configuration().add() qui signifie ajouter une classe, ensuite il suffit juste de l'instancier avec un ligne de paramètre.

Malheureusement par manque de temps notre code pour la création d'ouverture n'est pas parfait, nous n'avons pas réussi à afficher ce que nous voulions. Nous avons tout de même commenter ce que nous avons produit.