Rapport de TP3 – REPRÉSENTATION VISUELLE d'OBJETS

II - Préparation à faire avant le TP

Pour que ça fonctionne sur les PC de l'école (pb de librairie), on utilise https://replit.com/@fallingtree/TP3-base

1

Il nous est demandé d'expliquer les lignes de code suivantes :

```
import pygame #On importe la fonction pygame dans Spyder


pygame.init() #On initialise pygame pour dire que nous allons utiliser pygame

ecran = pygame.display.set_mode((300, 200)) #On crée un écran de taille 300, 200

pygame.quit() #On quitte pygame

# En compilant le programme, nous ne voyons rien car l'écran s'ouvre puis se ferme presque simultanément.
```

2

Il nous est demandé d'observer ce que fait ce code et de voir ce qu'il se passe lorsqu'on appuie sur un bouton quelconque. Ensuite, nous devons expliquer comment nous avons obtenu ce résultat.

```
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()

# Quand nous lançons le deuxième programme, l'écran (avec la taille que nous avons définie) apparaît et reste à l'écran jusqu'à ce que nous appuyons sur la barre espace.

En effet, on utilise la même ligne de code que dans le 1 pour créer un écran. Cependant, maintenant on met des lignes de code en dessous pour lui dire de continuer jusqu'à ce qu'une touche du clavier soit touchée par l'utilisateur.
```

Découverte de l'environnement du travail du TP

1.a

Nous devons ajouter à la fonction `Q1a()` la commande **`return`** `Configuration()` et tester notre code. On nous demande de l'analyser ci-dessous :

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

class Configuration:

    # Constructor
    def __init__(self, parameters = {}) :
        # Cette fonction permet d'initialiser les variables à l'aide d'un dico nommé "parameter"
        # Elle initialise les bibliothèques nécessaires (pygame et OpenGL)
        # Parameters
        # axes: if True the axis is displayed
        # xAxisColor: color for the x-axis
        # yAxisColor: color for the y-axis
        # zAxisColor: color for the z-axis
        # screenPosition: position of the screen on the z-axis (negative value) - on

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'axes' not in self.parameters:
            self.parameters['axes'] = True
        if 'xAxisColor' not in self.parameters:
            self.parameters['xAxisColor'] = [1, 0, 0]
        if 'yAxisColor' not in self.parameters:
            self.parameters['yAxisColor'] = [0, 1, 0]
        if 'zAxisColor' not in self.parameters:
            self.parameters['zAxisColor'] = [0, 0, 1]
        if 'screenPosition' not in self.parameters:
            self.parameters['screenPosition'] = -10

        # Initializes PyGame
        self.initializePyGame()

        # Initializes OpenGl
        self.initializeOpenGL()

        # Initializes the transformation matrix
        self.initializeTransformationMatrix()

        # Initializes the object list
        self.objects = []

        # Generates coordinates
```

```
    self.generateCoordinates()

# Initializes Pygame
def initializePyGame(self):
    # Sert à initialiser la fenêtre Pygame
    pygame.init()
    # Sets the screen size.
    pygame.display.set_mode((800, 600), pygame.DOUBLEBUF|pygame.OPENGL)
    # Gets pygame screen
    self.screen = pygame.display.get_surface()

# Initializes OpenGL
def initializeOpenGL(self):
    # Sert à initialiser OpenGL et et la couleur de l'écran
    # 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 transformation 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.glRotate(-90, 1, 0, 0) # on fait tourner le repère de -90° par rapport à x
# Getter
def getParameter(self, parameterKey):
    #Renvoie un paramètre en fonction de la clé renseignée
    return self.parameters[parameterKey]

# Setter
def setParameter(self, parameterKey, parameterValue):
    # Cette fonction permet de changer la valeur d'un paramètre
    self.parameters[parameterKey] = parameterValue
    if parameterKey == 'screenPosition':
        self.initializeTransformationMatrix()
    return self

# Generates the axis coordinates in the real space
def generateCoordinates(self):
    # Sert à initialiser les coordonnées des axes
    self.vertices = [
            [0, 0, 0 ],
            [1, 0, 0 ],
            [0, 1, 0],
            [0, 0, 1]
            ]
    self.edges = [
            [0, 1],
            [0, 2],
            [0, 3]
            ]

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

# Draws the axes and objects
def draw(self):
    # Sert à dessiner les axes

    # Draws the axes
    if self.parameters['axes']:
        gl.glBegin(gl.GL_LINES)

        # x-axis
        gl.glColor3fv(self.parameters['xAxisColor'])
        gl.glVertex3fv(self.vertices[0])
        gl.glVertex3fv(self.vertices[1])

        # y-axis
        gl.glColor3fv(self.parameters['yAxisColor'])
        gl.glVertex3fv(self.vertices[0])
        gl.glVertex3fv(self.vertices[2])

        # z-axis
        gl.glColor3fv(self.parameters['zAxisColor'])
        gl.glVertex3fv(self.vertices[0])
        gl.glVertex3fv(self.vertices[3])

        gl.glEnd()
```

```
        # Draws the objects if any
        for x in self.objects:
            x.draw()

    # 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)

    # Processes the MOUSEBUTTONDOWN event
    def processMouseButtonDownEvent(self):
        pass

    # Processes the MOUSEMOTION event
    def processMouseMotionEvent(self):
        pass

    # Displays on screen and processes events
    def display(self):
        # Sert à afficher l'écran

        # Displays on screen
        self.draw()
        pygame.display.flip()

        # Allows keybord events to be repeated
        pygame.key.set_repeat(1, 100)

        # Infinite loop
        while True:

            # Waits for an event
            self.event = pygame.event.wait()

            # Processes the event

            # Quit pygame (compatibility with pygame1.9.6 and 2.0.0)
                        if self.event.type == pygame.QUIT or (self.event.type == pygame.WINDOWEVENT and
pygame.event.wait(100).type == pygame.QUIT):
                pygame.quit()
                break

            elif self.event.type == pygame.KEYDOWN:
                self.processKeyDownEvent()

            elif self.event.type == pygame.MOUSEBUTTONDOWN:
                self.processMouseButtonDownEvent()

            elif self.event.type == pygame.MOUSEMOTION:
                self.processMouseMotionEvent()

            # Clears the buffer and displays on screen the result of the keybord action
            gl.glClear(gl.GL_COLOR_BUFFER_BIT|gl.GL_DEPTH_BUFFER_BIT)
            self.draw()
            pygame.event.clear()
            pygame.display.flip()
```

1.b

- **On nous demande** d'analyser l'effet de la modification suivante :

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

Lorsque nous exécutons cette ligne, nous remarquons que l'axe des x s'affiche en bleu avec une longueur de 1 à une position -5 par rapport au centre de l'écran.

- Nous devons ensuite expliquer pourquoi le chaînage de l'appel des méthodes

`setParameter()` et `display()` est possible.

Il est possible d'écrire le chaînage de `setParameter()` car il renvoie un objet du même type. Il peut donc utiliser `display()`.

- **On nous demande** ensuite d'expliquer pourquoi `screenPosition` doit être effectué dans le setter.

Nous l'utilisons pour afficher notre environnement 3D sur un plan 2D. La commande `screenPosition` détermine la distance entre "l'oeil" et le plan 2D.

1.c

Nous devons ajouter une seule instruction à la méthode

`initializeTransformationMatrix()` pour afficher l'axe x soit à l'horizontale sur l'écran, l'axe z à la verticale, et l'axe y soit dans la profondeur.

Nous ajoutons la ligne `gl.glRoratef(-90, 1, 0, 0)` et nous remarquons que les trois axes s'affichent comme on le souhaite sur l'écran.

III - Mise en place des interactions avec l'utilisateur de Pygame

1.d

On nous demande de chercher dans la documentation de Pygame les codes à utiliser pour zoomer et dézoomer sur les 3 axes, avec un facteur d'échelle de 1.1 pour le zoom et 1/1.1 pour le dézoom grâce aux touches `page up` et `page down`.

Nous trouvons dans la documentation que les commandes pour le zoom et le dézoom sont effectuées par

```
142         # Draws or suppresses the reference frame
143         elif self.event.dict['unicode'] == 'a' or self.event.key
    == pygame.K_a:
144             self.parameters['axes'] = not self.parameters['axes']
145             pygame.time.wait(300)
146         elif self.event.key == pygame.K_PAGEUP:
147             gl.glScalef(1.1, 1.1, 1.1)
148         elif self.event.key == pygame.K_PAGEDOWN:
149             gl.glScalef(1 / 1.1, 1 / 1.1, 1 / 1.1)
150
```

1.e

Nous devons maintenant écrire une commande qui permet à l'utilisateur d'utiliser la molette de sa souris pour le zoom. On nous demande d'écrire ans la méthode `processMouseButtonDownEvent()` pour zoomer avec la souris.

On écrit

```
150
151     # Processes the MOUSEBUTTONDOWN event
152     def processMouseButtonDownEvent(self):
153         if self.event.button == 4:
154             gl.glScalef(1.1, 1.1, 1.1)
155         elif self.event.button == 5:
156             gl.glScalef(1 / 1.1, 1 / 1.1, 1 / 1.1)
157
```

1.f

Nous devons déplacer les objets affichés en rotation en faisant un clique gauche et en translation en faisant un clique droit `processMouseMotionEvent()`.

- Pour effectuer une rotation autour de x et de z lorsque le bouton gauche est enfoncé et que l'on bouge la souris, on écrit (partie en vert).
- Pour effectuer une translation selon de x et selon z lorsque le bouton droit est enfoncé et que l'on bouge la souris, on écrit (partie en bleu).

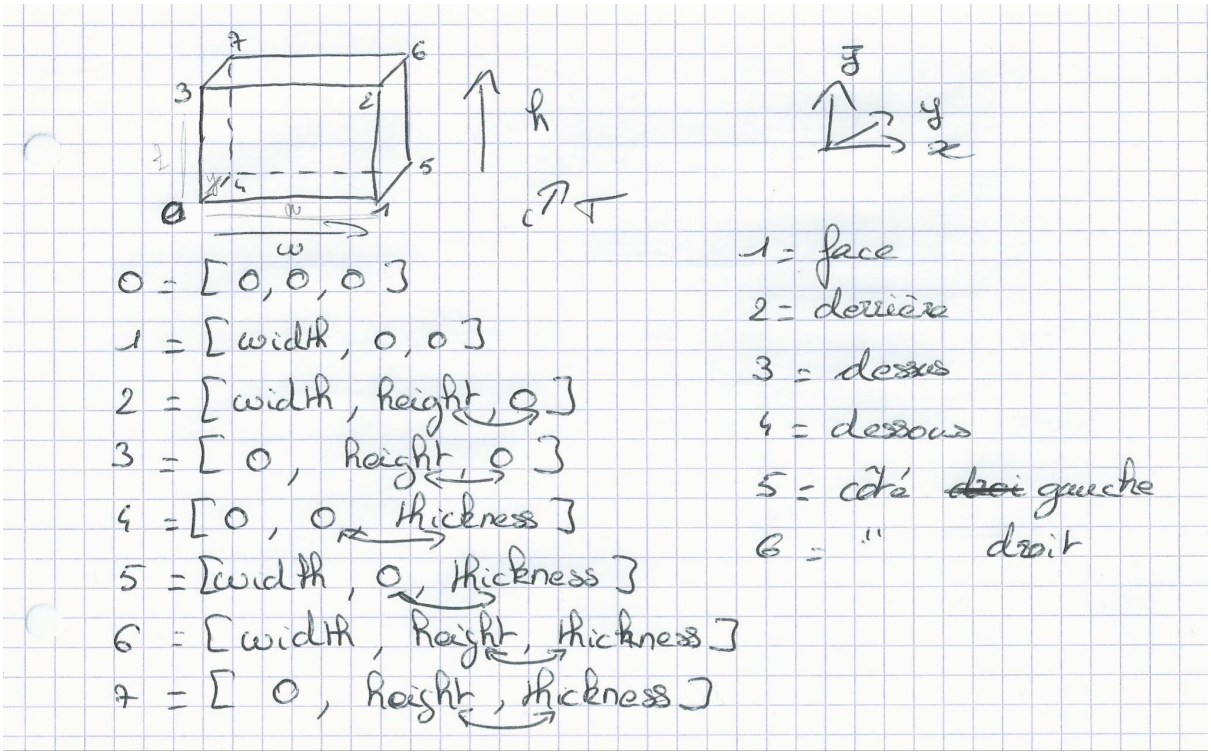
```
157
158     # Processes the MOUSEMOTION event
159     def processMouseMotionEvent(self):
160         if pygame.mouse.get_pressed()[0] == 1:
161             gl.glRotatef(self.event.rel[0], 0, 0, 1)
162             gl.glRotatef(self.event.rel[1], 1, 0, 0)
163         elif pygame.mouse.get_pressed()[2] == 1:
164             gl.glTranslatef(self.event.rel[0]/10, 0, 0)
165             gl.glTranslatef(0, 0, -self.event.rel[1]/10)
166
```

IV - Création d'une section

2.a

On nous demande d'écrire la méthode generate(self) dans la classe Section.

Nous dessinons un rectangle pour nous aider à écrire les coordonnées des points.



Cela nous permet de visualiser ce que l'on devra écrire dans le code :

```
52
53      # Defines the vertices and faces
54      def generate(self):
55          self.vertices = [
56                  [0,0,0],#0
57                  [self.parameters['width'],0,0],#1
58
    [self.parameters['width'],0,self.parameters['height']],#2
59                  [0,0,self.parameters['height']],#3
60                  [0,self.parameters['thickness'],0],#4
61
    [self.parameters['width'],self.parameters['thickness'],0],#5
62
    [self.parameters['width'],self.parameters['thickness'],self.parame
    ters['height']],#6
63
    [0,self.parameters['thickness'],self.parameters['height']]#7
64                  ]
65          self.faces = [
66                  [0,1,2,3],
67                  [4,5,6,7],
68                  [3,2,6,7],
69                  [0,1,5,4],
70                  [0,4,7,3],
71                  [1,5,6,2]
72                  ]
73
```

2.b & **2.c**

- **On nous demande** d'analyser la fonction `Q2b()` du `main.py` et d'expliquer l'instruction qu'elle contient.

```
26
27 def Q2b():
28     # Ecriture en utilisant le chaînage
29     return Configuration().add(
30             Section({'position': [1, 1, 0], 'width':7,
    'height':2.6})
31             )
32
```

Cette instruction permet d'ajouter une section de largeur 7 et de hauteur 2,6. Elle commencera sur le point [1,1,0] du repère 3D.

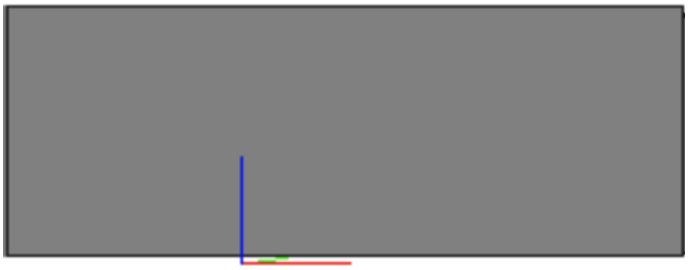
- **Nous devons** ensuite écrire la méthode `draw()` pour la classe `Section` pour tracer les faces du mur en gris. (Ligne pour ajouter les arêtes : `101`).

```
95
96      # Draws the faces
97      def draw(self):
98          # A compléter en remplaçant pass par votre code
99          gl.glPushMatrix()
100         gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],
            self.parameters['position'][2])
101         self.drawEdges()
102         for i in range (len(self.faces)):
103             gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
104             gl.glBegin(gl.GL_QUADS)
105             gl.glColor3fv([0.5, 0.5, 0.5])
106             for k in range (len(self.faces[0])):
107                 gl.glVertex3fv(self.vertices[self.faces[i][k]])
108             gl.glEnd()
109         gl.glPopMatrix()
110
```

- **Nous écrivons** la fonction `drawEdges()` avant `draw` pour pouvoir l'utiliser dans `draw`.

```
83
84      # Draws the edges
85      def drawEdges(self):
86          # A compléter en remplaçant pass par votre code
87          for i in range (len(self.faces)):
88              gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
89              gl.glBegin(gl.GL_QUADS)
90              gl.glColor3fv([0.1,0.1,0.1])
91              for k in ran Loading... elf.faces[i])):
92                  gl.glVertex3fv(self.vertices[k])
93              gl.glEnd()
94
```

- **Enfin, nous devons** exécuter la fonction `Q2b()` et obtenir une image de la section sur l'écran. Nous obtiendrons une section grise de ce type, sans arêtes. Avec la fonction `Q2c()`, nous obtenons une section grise de ce type avec des arêtes en noir.



V - Création des murs

3.a

- **Nous devons** analyser le fichier `Wall.py` et expliquer ce que fait le constructeur.

```
9
10  class Wall:
11      # Constructor
12      def __init__(self, parameters = {}) :
13          # Parameters
14          # position: position of the wall
15          # width: width of the wall - mandatory
16          # height: height of the wall - mandatory
17          # thickness: thickness of the wall
18          # color: color of the wall
19
20          # Sets the parameters
21          self.parameters = parameters
22
23          # Sets the default parameters
24          if 'position' not in self.parameters:
25              self.parameters['position'] = [0, 0, 0]
26          if 'width' not in self.parameters:
27              raise Exception('Parameter "width" required.')
28          if 'height' not in self.parameters:
29              raise Exception('Parameter "height" required.')
30          if 'orientation' not in self.parameters:
31              self.parameters['orientation'] = 0
32          if 'thickness' not in self.parameters:
33              self.parameters['thickness'] = 0.2
34          if 'color' not in self.parameters:
35              self.parameters['color'] = [0.5, 0.5, 0.5]
36
```

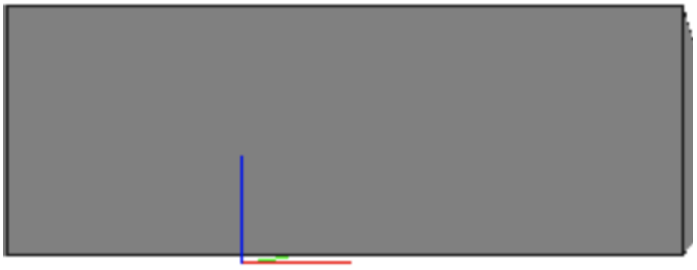
Le constructeur de cette classe rentre des valeurs standard pour créer le mur si elles ne sont pas déjà rentrées dans les paramètres. On utilise des if pour tester si les valeurs sont paramétrées comme on le désire ou non.

- **Nous devons** ensuite écrire des instructions permettant de dessiner le mur dans la méthode `draw`. Il faut ensuite écrire une matrice de projection pour la rotation du mur autour de l'axe `z` (Ligne 73).

```
67
68      # Draws the faces
69      def draw(self):
70          # A compléter en remplaçant pass par votre code
71          gl.glPushMatrix()
72          gl.glTranslatef(self.parameters['position'][0],self.parameters
            ['position'][1],self.parameters['position'][2])
73          gl.glRotatef(self.parameters['orientation'],0,0,1)
74          for wall in self.objects :
75              wall.draw()
76          gl.glPopMatrix()
77
```

- **Finalement, nous devons** écrire la fonction `Q3a` du `Main.py` et créer un mur à partir de la section que nous avions créé.

```
38
39  def Q3a():
40      return Configuration().add(Wall({'position':[1,1,0]
        ,'width':7,'height':2.6}))
41
```



VI - Création d'une maison

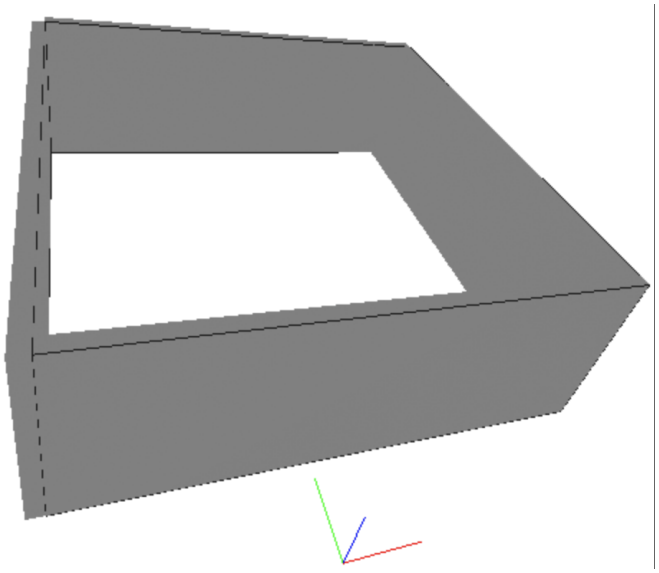
- **Nous devons** écrire une méthode draw pour dessiner plusieurs murs.

```
41
42      # Draws the house
43      def draw(self):
44          # A compléter en remplaçant pass par votre code
45          for H in self.objects :
46              H.draw()
47
```

- **Ensuite, il faudra** modifier `Q4a()` du `Main.py` pour créer une maison à 4 murs.

```
41
42  def Q4a():
43      # Ecriture en utilisant des variables : A compléter
44      wall1 = Wall({'position':[-3,1.2,0], 'width':7,
        'height':2.6})
45      wall2 = Wall({'position':[4,1.2,0], 'width':7,
        'height':2.6,'orientation':90})
46      wall3 = Wall({'position':[-3,8.2,0],'width':7,
        'height':2.6, 'orientation':0})
47      wall4 = Wall({'position':[-3,1.2,0],'width':7,
        'height':2.6, 'orientation':90})
48      house = House({'position': [-3, 1, 0],
        'orientation':0})
49      house.add(wall1).add(wall3).add(wall4).add(wall2)
50      return Configuration().add(house)
51
```

Nous ajustons la position de chaque mur créé et nous faisons tourner les murs nécessaires pour former notre maison. Finalement, nous obtenons :



VII - Création d'ouvertures

Nous devons créer des ouvertures pour les portes et les fenêtres de la maison.

5.a

```
51
52  def Q5a():
53      # Ecriture avec mélange de variable et de chaînage
54      opening1 = Opening({'position': [2, 0, 0],
        'width':0.9, 'height':2.15, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
55      opening2 = Opening({'position': [4, 0, 1.2],
        'width':1.25, 'height':1, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
56      return Configuration().add(opening1).add(opening2)
57
```

5.b

```
57
58  def Q5b():
59      # Ecriture avec mélange de variable et de chaînage
60      section = Section({'width':7, 'height':2.6})
61      opening1 = Opening({'position': [2, 0, 0],
        'width':0.9, 'height':2.15, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
62      opening2 = Opening({'position': [4, 0, 1.2],
        'width':1.25, 'height':1, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
63      opening3 = Opening({'position': [4, 0, 1.7],
        'width':1.25, 'height':1, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
64
65      print(section.canCreateOpening(opening1))
66      print(section.canCreateOpening(opening2))
67      print(section.canCreateOpening(opening3))
68      return Configuration()
69
```

5.c

```
70  def Q5c1():
71      section = Section({'width':7, 'height':2.6})
72      opening1 = Opening({'position': [2, 0, 0],
        'width':0.9, 'height':2.15, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
73      sections = section.createOpening(opening1)
74      configuration = Configuration()
75      for x in sections:
76          configuration.add(x)
77      return configuration
78
78
79  def Q5c2():
80      section = Section({'width':7, 'height':2.6})
81      opening2 = Opening({'position': [4, 0, 1.2],
        'width':1.25, 'height':1, 'thickness':0.2,
        'color': [0.7, 0.7, 0.7]})
82      sections = section.createNewSections(opening2)
83      configuration = Configuration()
84      for section in sections:
85          configuration.add(section)
86      return configuration
87
```

Conclusion

Nous nous sommes arrêtés à la partie **V** car nous n'avons pas eu le temps de finir le TP. Cependant, nous avons apprécié utiliser pygame pour pouvoir utiliser de nouvelles fonctions dans nos programmes. Ce TP nous a permis de voir ce que nous avions créé dans la fenêtre, ce qui était agréable et nous a permis d'expérimenter les différents paramètres pour créer le mur, comme les positions, les couleurs ou encore les mouvements que l'utilisateur était capable de faire pour voir la maison en 3D. De plus, ce TP nécessite de nombreux fichiers pour s'y retrouver, ce qui nous a permis de se familiariser avec l'utilisation de plusieurs fichiers différents que nous utilisions dans le `Main.py`.