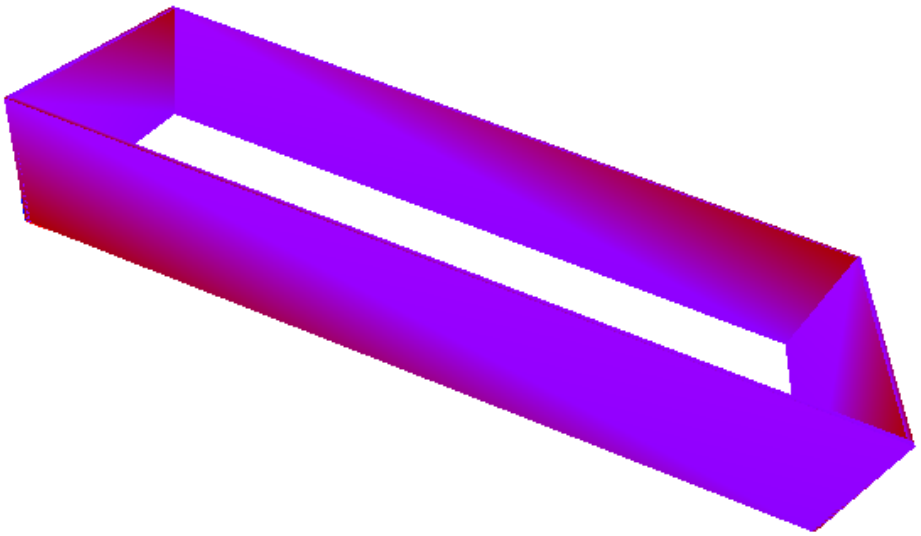
COMPTE-RENDU TP3

Andrew Mary Huet de Barochez / Calvin Ngor

VI Création d’une maison

```
# Draws the house
def draw(self):
    for elem in self.objects:
        elem.draw()
```

Pour dessiner la maison il suffit de parcourir tous les objets qui constituent cette dernière et on appelle leur fonction draw().



```python
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'width': 30, 'height': 5, 'orientation': 0, 'position': [0, 0, 0]})
    wall2 = Wall({'width': 10, 'height': 5, 'orientation': 90, 'position': [30, 0, 0]})
    wall3 = Wall({'width': 30, 'height': 5, 'orientation': 180, 'position': [30, 10, 0]})
    wall4 = Wall({'width': 10, 'height': 5, 'orientation': 270, 'position': [0, 10, 0]})
    house = House({'position': [-3, 1, 0], 'orientation': 0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

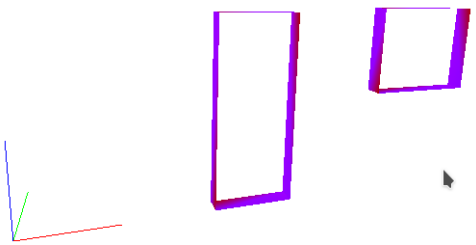
Afin de créer une maison constituée de 4 murs, nous avons défini la longueur, la hauteur et l'orientation de ces murs. De plus, nous avons déterminé le point à l'origine d'un angle de ces murs afin de s'en servir de base pour le tracer de ce mur. Et nous avons ajouté à l'objet maison ces murs.

VII Création d'ouvertures

5)a)

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

    self.faces = [
        [1, 5, 6, 2],  # Top
        [4, 7, 3, 0],  # Bottom
        [0, 1, 5, 4],  # Left
        [2, 6, 7, 3]  # Right
    ]
```

La complétion de la classe Opening, définie dans le fichier Opening.py afin de produire un affichage comme celui de la figure a été obtenue par le tracer de sections mais sans prendre en compte la face avant et la face arrière.



5)b)

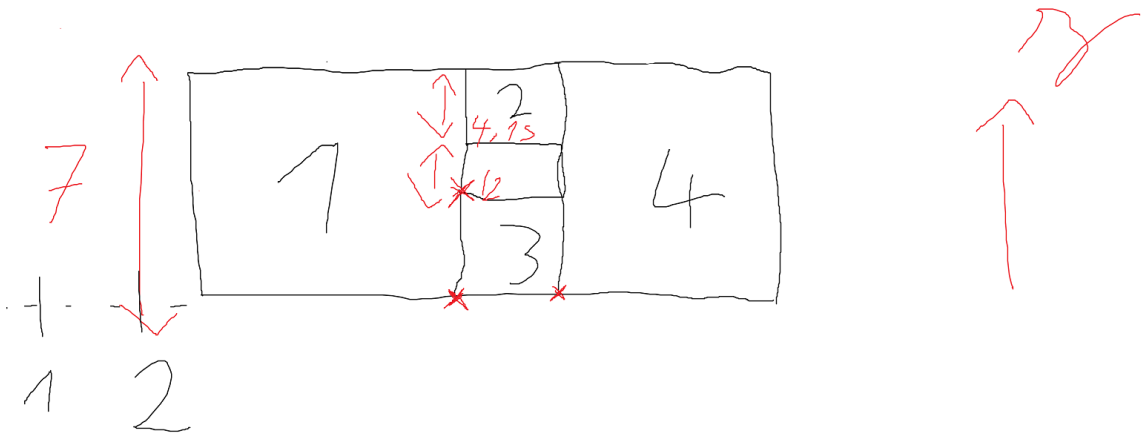
```
    def canCreateOpening(self, x: Opening):
        # A compléter en remplaçant pass par votre code
        x_pos = x.getParameter("position")
        x_width = x.getParameter("width")
        x_height = x.getParameter("height")
        self_pos = self.getParameter("position")
        self_height = self.getParameter("height")
        self_width = self.getParameter("width")

        if self_pos[0] > x_pos[0] or self_pos[0] + self_width < x_pos[0] + x_width:
            return False
        if self_pos[2] > x_pos[2] or self_pos[2] + self_height < x_pos[2] + x_height:
            return False
        return True
```

Pour la fonction canCreateOpening on vérifie que l'ouverture est comprise dans la section aux coordonnées X et Z. Pour cela il faut regarder si chaque arête de l'ouverture est bien à l'intérieur de la section.

```
def Q5b():
    # Ecriture avec mélange de variable et de chaînage
    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]})
    opening2 = Opening(
        {'position': [4, 0, 1.2], 'width': 1.25, 'height': 1, 'thickness': 0.2, 'color': [0.7, 0.7, 0.7]})
    opening3 = Opening(
        {'position': [4, 0, 1.7], 'width': 1.25, 'height': 1, 'thickness': 0.2, 'color': [0.7, 0.7, 0.7]})

    print(section.canCreateOpening(opening1))
    print(section.canCreateOpening(opening2))
    print(section.canCreateOpening(opening3))
    return Configuration()
```

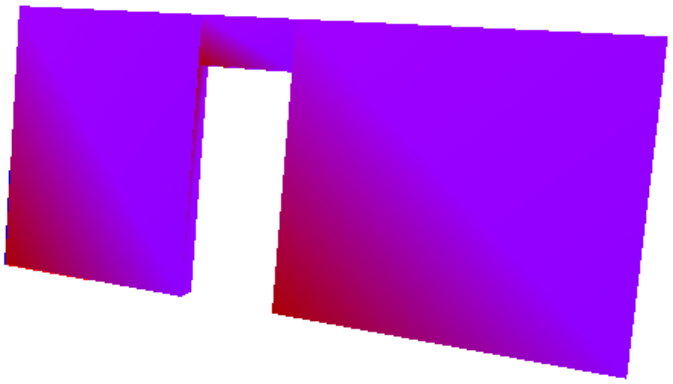
5)c)

```python
def createNewSections(self, x: Opening):
    # A compléter en remplaçant pass par votre code
    if not self.canCreateOpening(x):
        raise Exception("Can't create this opening")
    sec1 = Section({'position': self.parameters['position'],
                    'width': x.getParameter('position')[0] - self.parameters['position'][0],
                    'height': self.parameters['height'],
                    'thickness': self.parameters['thickness']
                    })
    sec2 = Section({'position': [x.getParameter('position')[0],
                                 self.parameters['position'][1],
                                 x.getParameter('position')[2] + x.getParameter('height')],
                    'height': self.parameters['height'] - (
                           x.getParameter('position')[2] + x.getParameter('height')),
                    'width': x.getParameter('width'),
                    'thickness': self.parameters['thickness']
                    })

    sec3 = Section({'position': [
        sec2.getParameter('position')[0],
        self.parameters['position'][1],
        self.parameters['position'][2]],
        'height': x.getParameter('position')[2] - self.parameters['position'][2],
        'width': x.getParameter('width'),
        'thickness': self.parameters['thickness']
    })
    sec4 = Section(
        {'position': [sec2.parameters['position'][0] + sec2.parameters['width'],
                      self.parameters['position'][1],
                      self.parameters['position'][2]],
         'height': self.parameters['height'],
         'width': self.parameters['width'] - (sec1.parameters['width'] + sec2.parameters['width']),
         'thickness': self.parameters['thickness']
         })
    sections = [sec1, sec2, sec3, sec4]

    if sec3.parameters['height'] == 0:
        sections.remove(sec3)
    if sec2.parameters['height'] == 0:
        sections.remove(sec2)
    if sec1.parameters['width'] == 0:
        sections.remove(sec1)
    if sec4.parameters['width'] == 0:
        sections.remove(sec4)
    return sections
```



La méthode createNewSections(self,x) de la classe Section où x est un objet, qui retourne la liste des sections engendrées par la création de l'ouverture à été réalisée en prenant en compte de calculs mathématiques. En effet, lorsque nous enlevions une partie du mur, il fallait déterminer les dimensions des sections que cette découpe entraînait. Nous étions confrontés à de nombreuses erreurs de calculs dû à l'interversion entre position et longueurs.

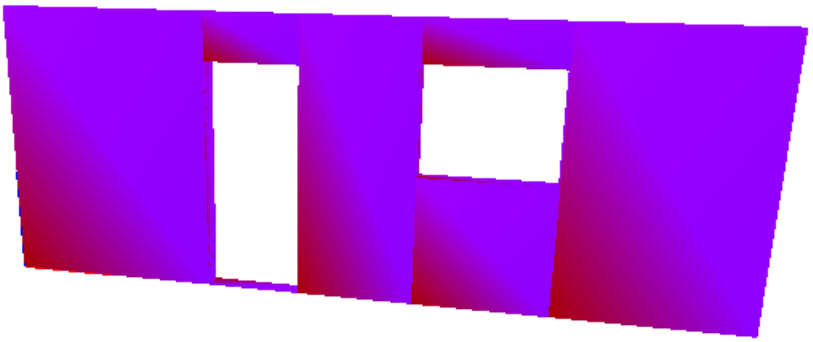
```
def Q5d():
    wall = Wall({'width': 7, 'height': 2.6, })
    opening1 = Opening(
        {'position': [2, 0, 0], 'width': 0.9, 'height': 2.15, 'thickness': 0.2, 'color': [0.7, 0.7, 0.7]})

    opening2 = Opening(
        {'position': [4, 0, 1.2], 'width': 1.25, 'height': 1, 'thickness': 0.2, 'color': [0.7, 0.7, 0.7]})

    wall.add(opening1)
    wall.add(opening2)

    configuration = Configuration()
    configuration.add(wall)
    return configuration
```



5)d) La fonction enumerate permet de générer des tuples contenant l'indice et les éléments d'une liste. Ainsi la variable section[0] contient l'indice de la section et section[1] contient la référence de la section actuelle dans le parcours de boucle.

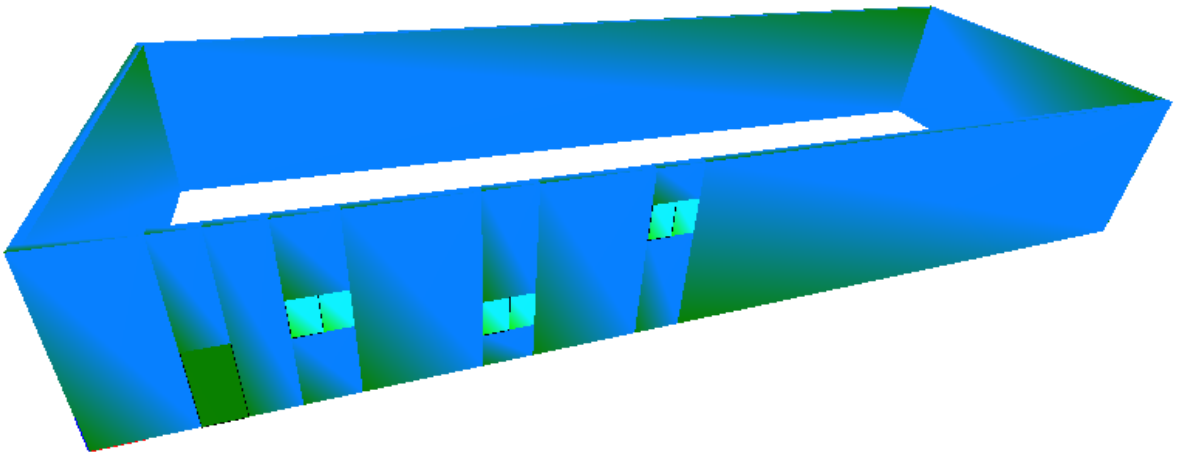
```
def add(self, x):
    section = self.findSection(x)
    if section is None:
        raise Exception("Ajout impossible de l'ouverture")

    sections = section[1].createNewSections(x)

    self.objects.pop(section[0])
    self.objects.extend(sections)
    self.objects.append(x)
    return self
```

Pour la fonction add on commence par récupérer la section que l'on va devoir modifier à l'aide de la fonction findSection. Ensuite on peut appeler notre méthode createNewSections sur la section à diviser. On peut donc pop la dernière section de notre mur, pour ensuite la remplacer par les nouvelles sections générées, en les ajoutant avec la méthode extend. On finit par ajouter l'objet d'ouverture.



```python
def Q6():
    configuration = Configuration()

    wall1 = Wall({'width': 30, 'height': 5, 'orientation': 0, 'position': [0, 0, 0]})
    wall2 = Wall({'width': 10, 'height': 5, 'orientation': 90, 'position': [30, 0, 0]})
    wall3 = Wall({'width': 30, 'height': 5, 'orientation': 180, 'position': [30, 10, 0]})
    wall4 = Wall({'width': 10, 'height': 5, 'orientation': 270, 'position': [0, 10, 0]})

    house = House({'position': [-3, 1, 0], 'orientation': 0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)

    door1 = Door({'position': [2, 0, 0]})

    window1 = Window({'position': [4, 0, 2]})

    wall1.add(door1)
    wall1.add(window1)

    configuration.add(house)

    return configuration
```

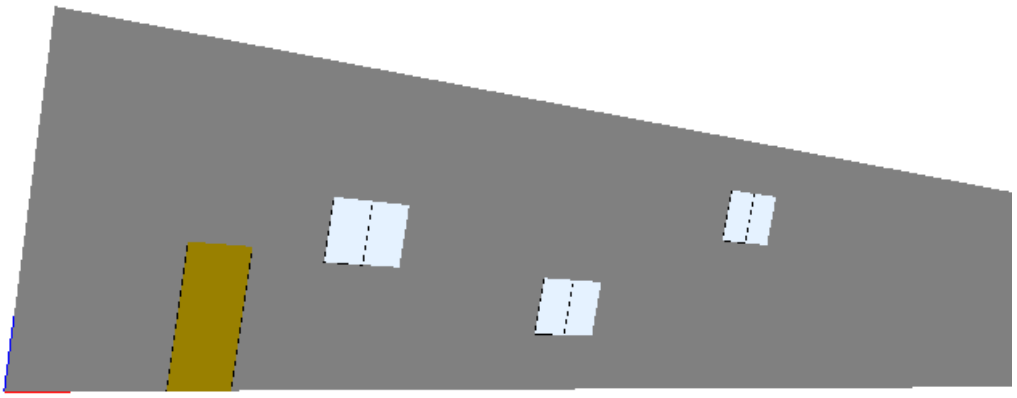
6) Les fichiers Door.py et Wall.py nous permettent de gérer les portes et les fenêtres. En effet, en regardant les classes qui leurs sont associées nous avons pu écrire le programme permettant de tracer la maison avec une fenêtre avec la porte. La méthode utilisé est de tout d'abord commencer par tracer les 4 murs, puis nous avons attribué ces murs à l'objet maison, ensuite nous avons positionné la porte sur une face de la maison en l'occurrence celle de devant, après nous avons mis la fenêtre sur une des sections du mur qui possédaient suffisamment d'espace.



On constate très bien les différentes section grâce (ou à cause) de notre dégradé de couleur.

```
configuration = Configuration()

wall1 = Wall({'width': 30, 'height': 5, 'orientation': 0, 'position': [0, 0, 0]})
wall2 = Wall({'width': 10, 'height': 5, 'orientation': 90, 'position': [30, 0, 0]})
wall3 = Wall({'width': 30, 'height': 5, 'orientation': 180, 'position': [30, 10, 0]})
wall4 = Wall({'width': 10, 'height': 5, 'orientation': 270, 'position': [0, 10, 0]})

house = House({'position': [-3, 1, 0], 'orientation': 0})
house.add(wall1).add(wall3).add(wall4).add(wall2)

door1 = Door({'position': [2, 0, 0]})

window1 = Window({'position': [4, 0, 2]})
window2 = Window({'position': [8, 0, 1]})
window3 = Window({'position': [12, 0, 3]})

wall1.add(door1)
wall1.add(window1)
wall1.add(window2)
wall1.add(window3)

configuration.add(house)
```

Pour définir la maison il suffit donc d'instancier des murs, des fenêtres, des portes, et ensuite de les ajouter au murs et le mûrs à la maison.



Voici une version avec les couleurs de base.

7)

Pour afficher deux maisons on peut tout simplement copier collé le code pour une maison et changer les valeurs. Évidemment, faire une fonction pour créer une maison et l'appeler deux fois serait plus propre.

Tests: