Rapport de TP
FI3-INFO501 Année 2021-2022
Polytech Annecy-Chambéry - Université Savoie Mont-Blanc

MALACARNE Etienne
DASSEUX Damien
24/11/2021

TP3 :
Représentation visuelle des objets

Introduction

Pour ce TP3, nous devons représenter des objets en 3D dans une fenêtre graphique grâce à OpenGL et pygame.

I. Préparation

Utilisation de pygame

Question 1

On crée une fenêtre en utilisant pygame avec une hauteur de 200 et une largeur de 300. Mais elle disparaît immédiatement.

Question 2

Dans un premier temps, on crée la fenêtre.

Ensuite, on crée une boucle d'exécution while.

Dans cette boucle d'exécution, on récupère tous les évènements dans un tableau avec pygame.event.get().

On vérifie si l'un des évènements correspond à pygame.KEYDOWN.
Cet événement est activé lorsqu'une touche quelconque du clavier est enfoncé.

Si une touche est enfoncée, on arrête la boucle d'exécution et on ferme la fenêtre.

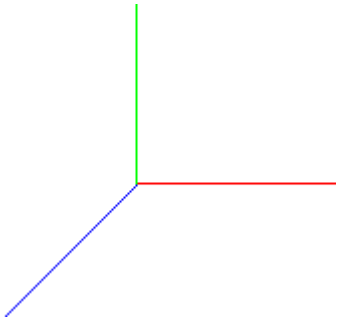
Utilisation PyOpengl

Question 1

On remarque que la couleur du fond est définie par gl.ClearColor.

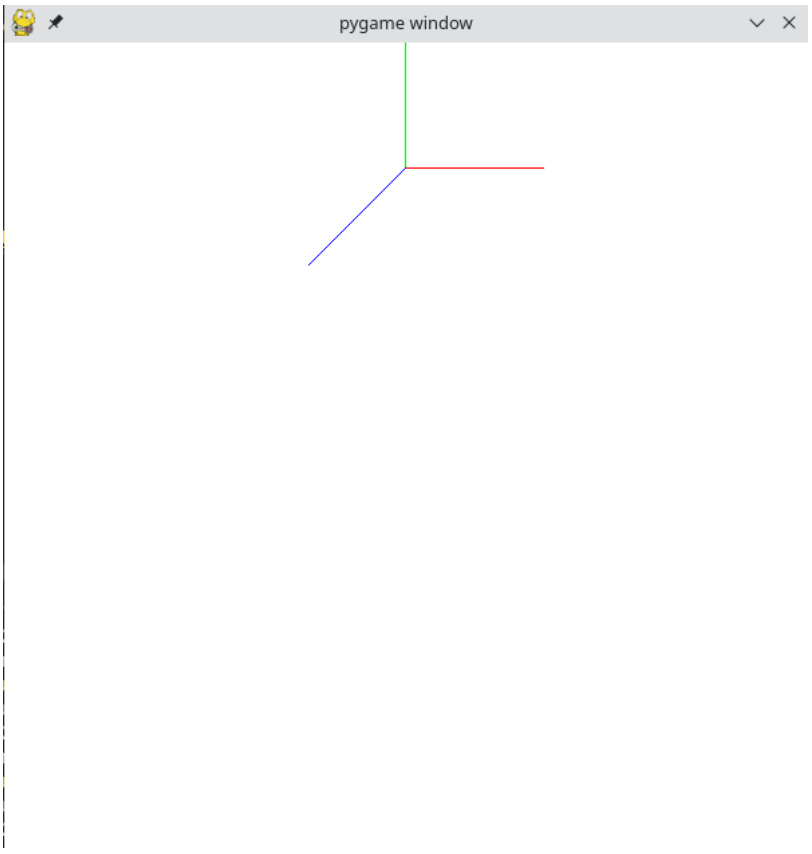
Question 2

On obtient ce genre d'image :

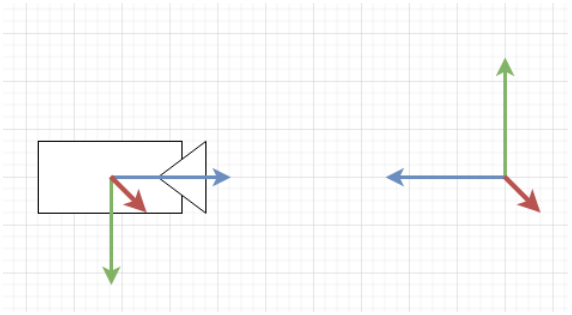


1.

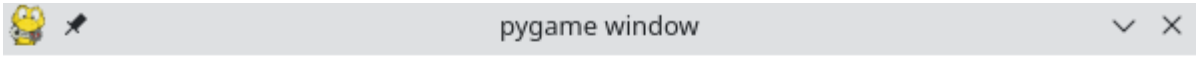
Lors de la translation, on obtient :



Ce résultat nous apprend beaucoup de choses sur le fonctionnement de la Caméra et les axes que la caméra suit.
En effet, si on veut retrouver le comportement de la caméra ci dessus, il faut que les axes soit comme cela :



Voici le résultat pour la rotation de 90° :



Ce résultat est logique, on effectue une rotation sur la caméra de 90°, lorsque l'on fait cela, c'est normal que l'on ne voit plus l'objet, on regarde le ciel :

Avant rotation :

Après rotation :

Rapport de TP
FI3-INFO501 Année 2021-2022
Polytech Annecy-Chambéry - Université Savoie Mont-Blanc

Découverte de l'environnement de travail de TP

Question 1

a. Question 1)a

Le fichier main.py récupère toutes les classes du dossier "src" afin de les utiliser. Les différentes configurations sont contenues dans des fonctions notées de Q1a à Q6. Elles sont appelées à la toute fin du fichier, dans la fonction main(). La fonction main est ensuite appelée lors de l'exécution du fichier.

Le fichier Configuration.py contient la classe "Configuration". Cette classe contient toutes les informations liées au GUI généré grâce à pygame. Elle contient également toutes les informations relatives à l'espace dans lequel les objets sont créés grâce à OpenGl.
Le constructeur permet de définir un bon nombre d'options :

- L'option d'afficher les axes ou non
- L'option de couleur pour chaque axe
- et enfin l'option de la position de la caméra par rapport au [0,0,0] de l'espace.

Durant la construction de l'objet, les éléments relatifs à Pygame et OpenGl sont initialisés.

initializePygame crée une fenêtre de 800 de largeur et 600 de hauteur et la configure pour fonctionner avec OpenGl.

initializeOpenGl définit la couleur utilisée en background et d'autres paramètres utilisés dans l'espace.

La classe contient également un getter(getParameter) et un setter(setParameter). Le setter a un comportement particulier par rapport à la position de la caméra dans l'espace.

La méthode add permet d'ajouter un élément à la collection d'objets à afficher à l'écran.

La méthode draw permet, comme son nom l'indique, de dessiner les objets contenus dans la collection.

b. Question 1)b

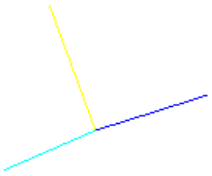
La modification de code : permet d'approcher la caméra de l'objet, car la position de base est -1. On met également la couleur de l'axe X en Jaune.

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

Pour ce morceau de code :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisCo lor', [0,1,1]). \

    display()
```

Les paramètres passés permettent de mettre la couleur de l'axe x en jaune et l'axe y en cyan



Ce chaînage d'appel est possible car la méthode setParameter retourne self.

Il y a un traitement particulier pour le paramètre : 'screenPosition' car il doit être appliqué à la matrice de translation à chaque modification.

c. Question 1)c

Je vois ce problème comme une simple rotation : on retourne la caméra de -90 degré autour de l'axe x. Cela permet de passer l'axe de cette position :

à cette position :

II. Mise en place des interactions avec l'utilisateur avec Pygame

Question 1)d

Nous devons faire en sorte qu'avec les touches "Page up" et "Page down", on puisse grossir ou réduire l'affichage.
Pour cela, nous avons rajouté ces lignes dans la fonction processKeyDownEvent :

```
        if self.event.key==pygame.K_PAGEUP:
            gl.glScalef(1.1,1.1,1.1)
        if self.event.key==pygame.K_PAGEDOWN:
            gl.glScalef(1/1.1,1/1.1,1/1.1)
```

Si l'événement quand j'appuie sur une clé correspond à la touche "Page up" , alors j'utilise une fonction d'OpenGL, glScalef, qui va me permettre de faire un zoom selon les données de l'énoncé .
On fait ensuite la même chose pour "Page Down", en changeant simplement les valeurs pour avoir un effet de " de-zoom".

Question 1)e

On veut la même chose que dans la question précédente, sauf que cette fois, cela va dépendre de la molette de la souris.
On obtient ce code :

```
    # 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)
```

Si on utilise la molette de la souris dans le sens "avant", alors on va effectuer un zoom.
Et si on utilise la molette dans le sens arrière, on va "de-zoom".

Question 1)f

Ici , on veut faire une rotation autour de x composée avec une rotation autour de z lorsque le bouton gauche est enfoncé et que la souris est déplacée et une translation selon l'axe x

composée à une translation selon l'axe z lorsque le bouton droit est enfoncé et que la souris est déplacée.
Pour la rotation, voici le code :

```
    # Processes the MOUSEMOTION event
    def processMouseMotionEvent(self):
        if pygame.mouse.get_pressed()[0]:
            gl.glRotatef(1,1,1,self.event.rel[0])
```

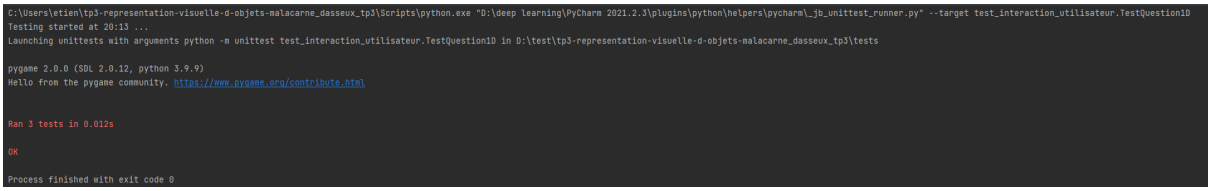
On a une condition pour vérifier si le bouton gauche est bien enfoncé.Puis, on utilise la fonction glRotatef(angle,x,y,z) pour effectuer la rotation en mettant l'angle à 1° (pour être plus précis). x et y sont à 1 et z va dépendre du déplacement horizontal de la souris

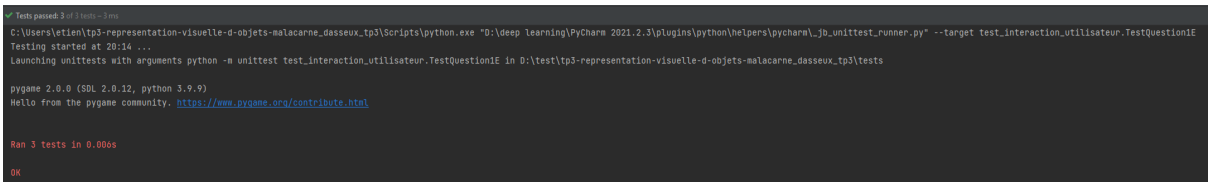
Pour la translation, voici le code :

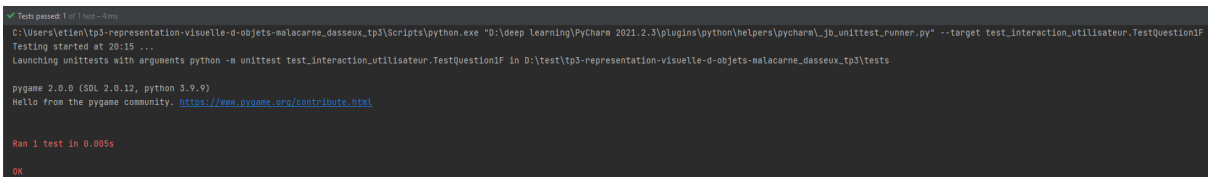
```
        if pygame.mouse.get_pressed()[2]:
            gl.glTranslatef(self.event.rel[0]/25,0,self.event.rel[1]/25)
```

On vérifie si on presse bien le bouton droit,puis on va effectuer une translation avec la fonction glTranslatef en utilisant le déplacement horizontal (x) et le déplacement vertical (y, qui sera ici z) de la souris.y,lui,sera à 0 car nous ne voulons pas faire de translation avec cet axe.

Voici aussi un screenshot des tests pour cette partie (qui sont tous validés) :







III. Création d'une section

1.

a.

Dans un premier temps, il faut déterminer comment les faces seront construites, c'est-à-dire qu'il faut finir le tableau "self.faces".

```
        self.faces = [
                [0, 3, 2, 1], # La face en face de l'utilisateur
                [1, 2, 6, 5], # la face en haut
                [5, 6, 7, 4],
                [0, 3, 7, 4],
                [2, 3, 7, 6],
                [1, 0, 4, 5]
                # définir ici les faces
                ]
```

Ensuite, il faut s'occuper de la création des vertices et surtout de leur positionnement.
Ici 2 approches sont possibles : soit on implémente les coordonnées de la section directement dans la méthode "generate" comme suit :

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [ # A revoir, il maque L'utilisation de self.position
            self.parameters['position'],
            [self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]],

            [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2] ],
            [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+ self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+ self.parameters['thickness'], self.parameters['position'][2]]

            # Définir ici les sommets
            ]
    self.faces = [
            [0, 3, 2, 1], # La face en face de l'utilisateur
            [1, 2, 6, 5], # la face en haut
            [5, 6, 7, 4],
            [0, 3, 7, 4],
            [2, 3, 7, 6],
            [1, 0, 4, 5]
            # définir ici les faces
            ]
```

L'autre approche ne demande l'addition que 3 lignes de code à l'étape du dessin de la section.
On utilise la méthode gl.glTranslatef afin de placer la section qui vient d'être créée à la bonne position :

```
        gl.glTranslatef(
            self.parameters['position'][0],
            self.parameters['position'][1],
            self.parameters['position'][2]
        )
```

On oublie pas de push une matrice et de la supprimé juste après.

La deuxième méthode a été retenue.

b.

Ce morceau de code affiche une face normal à l'axe y allant de x=0 à x=1 et de z=0 à z=1

Afin de créer toutes les faces de la section, il suffit de boucler sur self.faces et d'associer les différents sommets entre eux.

```
    # Draws the faces
    def draw(self):
        gl.glPushMatrix()
        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
        gl.glTranslatef(
            self.parameters['position'][0],
            self.parameters['position'][1],
            self.parameters['position'][2]
        )
        for f in self.faces:
            gl.glBegin(gl.GL_QUADS)
            gl.glColor3fv(self.parameters['color'])
            for sommet in f :
                gl.glVertex3fv(self.vertices[sommet])
            gl.glEnd()
        gl.glPopMatrix()
```

c.

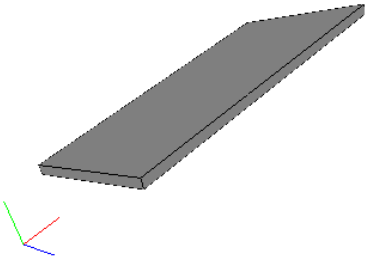
On a rencontré un problème : les arêtes ne sont pas stockées dans la classe, elles doivent être calculées.
Afin de trouver toutes les arretes, nous avions 2 approches, soit on marque toutes les possibilités d'arêtes en dur, ça fonctionne et ça ne demande aucun calcul, ou alors, on trouve toutes les arêtes de manière algorithmique, le codes est donc plus court et lisible.
Nous avons opté pour la seconde option.

```
    # Draws the edges
    def drawEdges(self):
        lines = []
        for x in self.faces :
            for i in range(len(x)):
                lines.append([x[i], x[i+1 if i+1 < len(x) else 0]])

        for l in lines :
            gl.glBegin(gl.GL_LINES)
            gl.glColor3fv([0,0,0])
            for sommet in l :
                gl.glVertex3fv(self.vertices[sommet])
            gl.glEnd()
```

Nous obtenons la liste des sommets pour les lignes en associant les sommets présents dans les faces 2 à 2. Cela revient à lire toutes les lignes dont les faces sont composées.

Ensuite, il suffit de parcourir la liste des lignes et de les tracer sur l'écran grâce à OpenGl.

Nous obtenons ce résultat :



L'option "edges" rentrées par l'utilisateur dans la section est prise en compte dans la fonction draw() de la classe "Section".

Voici les tests obtenus pour la partie section :

```
C:\Users\Damien\Desktop\temp\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3>C:/Users/Damie
/temp/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_section.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
.........
----------------------------------------------------------------------
Ran 9 tests in 0.002s

OK
```

IV. Création des murs

Cette partie a été bien plus compliquée : nous avons essayé de gérer la rotation des objets de manière mathématique, sans utiliser les fonctions de OpenGl. Cela n'a pas fonctionné, Nous avons adopté une nouvelle approche utilisant les méthodes glPushMatrix et glPopMatrix.

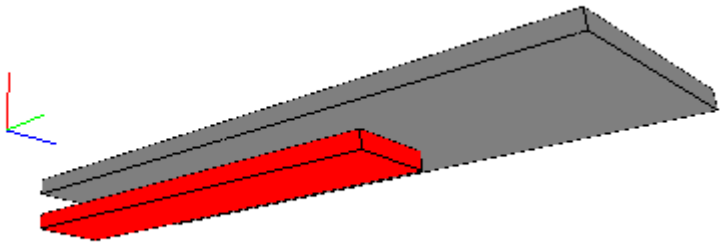
Ce que l'on fait c'est :

- On crée un matrice,
- On applique la rotation,
- On dessine les objets ,
- On enlève la matrice afin de ne plus travailler sur elle.

Afin de tester la méthode, On a ajouter ce code dans la fonction Q3a :

```
def Q3a():
    return Configuration().add(
            Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'edges': True, 'orientation':90}).\
            add(Section({'position':[1, 1.5, 0], 'width':5, 'height':1, 'edges': True, 'color':[1,0,0]}))
            )
```

Nous devons obtenir 2 sections : une grise et une rouge qui auront une rotation de 90 degrés autour de l'axe z.
Voici le résultat du test :



Nous avons ensuite enlevé la 2eme section (la barre rouge) afin de faire passer les tests unitaires :

```
38/python.exe c:/Users/Damien/Desktop/temp/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_wall.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
...
----------------------------------------------------------------------Ran 3 tests in 0.001s

OK
```

V. Création d'une maison

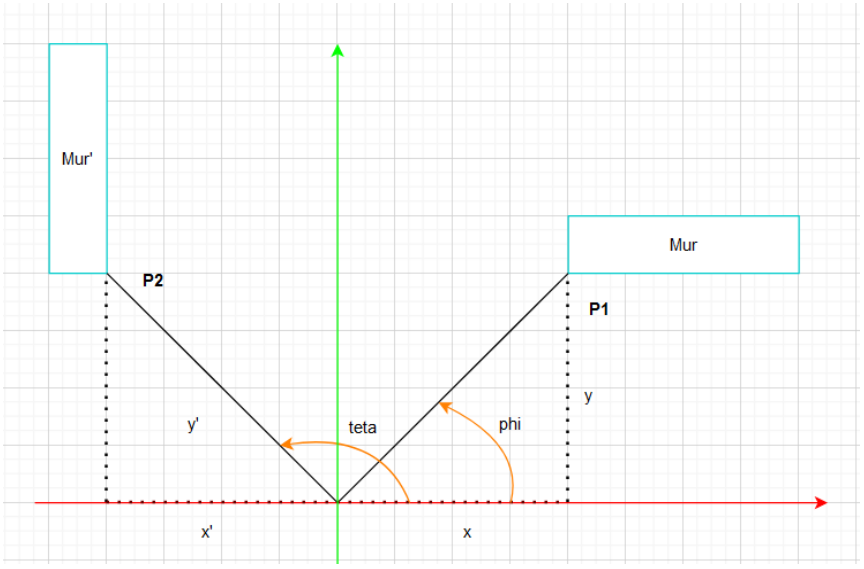
Pour créer la maison, on a commencé par la partie la plus simple : appliquer une translation pour positionner la maison comme elle devrait être et appliquer la rotation correspondant à l'orientation de la maison.

```
        gl.glPushMatrix()
        gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
        gl.glTranslatef(self.parameters['position'][0],
self.parameters['position'][1], self.parameters['position'][2])
        ... #Affichage objets
        gl.glPopMatrix()
```

Dans un second temps, il faut afficher tous les murs de la maison.
Cette partie a posé beaucoup de problèmes pour plusieurs raisons :

Le premier problème rencontré est la rotation des murs. En effet, lorsque l'on fait la rotation d'un mur grâce à la Classe Wall, la rotation est faite par rapport à l'axe Z, hors nous voulons faire une rotation sur place afin de placer tous les murs au bon endroit.

Après avoir fait des recherches nous avons déterminé qu'une solution possible était de faire une correction des mouvement fait par les murs lors de leur création.
On veut faire en sorte que la position du mur en coordonnées x et y ne change pas lors de la rotation. Ainsi il faut translater le mur de manière à ce qu'il reste sur place.
On peut représenter cette situation sous forme d'un schéma.



Sur le schéma ci-dessus, nous pouvons voir que Mur' est l'image de Mur après une rotation de 90 degrés. La solution au problème serait de déterminer la translation à faire pour passer de P2 à P1 et de l'appliquer.

On trouve cette translation grâce à cette suite d'instruction :

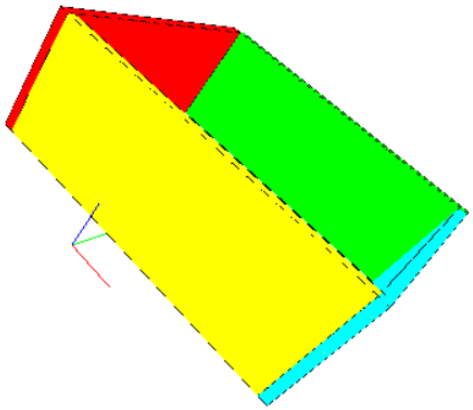
```
            x=o.parameters['position'][0]
```

```
            y=o.parameters['position'][1]

            phi = arctan(divide(y,x))*180/pi
            teta = o.parameters['orientation']+phi

            x_ = sqrt(pow(x,2)+ pow(y, 2)) * cos(teta*pi/180)
            y_ = sqrt(pow(x,2)+ pow(y, 2)) * sin(teta*pi/180)

            gl.glPushMatrix()
            gl.glTranslatef(float(x-x_), float(y-y_),0)
            o.draw()
            gl.glPopMatrix()
```

Le second problème rencontré est directement lié à la solution trouvée juste au-dessus : lorsque x et x sont à 0, la fonction numpy “divide” n’est pas capable de renvoyer inf pour l’arctangente.
Ce problème est facilement résolu :

```
            phi = arctan(divide(y,x))*180/pi if not(x==0) else 90
```

Ensuite nous avons obtenu ce résultat :



Voici les test unitaires pour cette partie :

```
tp3/tests/test_house.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
...
----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
```

VI. Création d’ouverture

VII. Pour finir …

Conclusion :

Nous nous sommes arrêtés au début de la partie 5, et tous les tests fonctionnaient jusqu'à là. Pour cette première partie de TP, nous n'avons pas eu de grandes difficultés (car nous avions déjà fait de la 3D sous Unity en C# et un peu en C/C++ avec SFML) même s'il fallait bien aller chercher dans la doc et lire le sujet attentivement pour bien comprendre, surtout quand on a pas fait de 3D avec OpenGL et python avant.
Il était donc intéressant de découvrir OpenGL et python dans ce TP.