DALENC Nathan
BENITO Enzo
Date du TP (24/11/2021)

Rapport de TP3 – Visualisation de fractales : Ensemble de Mandelbrot

I. Introduction

Le but de ce TP est de concevoir une représentation d'objets 3D à l'écran dans une fenêtre graphique

II. Préliminaires : les nombres complexes

1. Question (1).

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

La 1ere ligne importe tous les modules pygame disponibles dans le package pygame. Puis la 2ème initialisera tous les modules pygame. ET enfin crée un écran de dimension 300 x 200. Qui disparait aussitôt.

2. 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()
```

La fenêtre précédemment créer apparaît et reste afficher jusqu'à l'appui d'une touche du clavier. Tout cela grâce à la création d'une variable continuer à vrai. Puis d'une boucle while qui continue d'afficher l'écran jusqu'à ce que dans la seconde boucle for un évènement passe la variable à faux.

Découverte de l'environnement du travail du TP

3. Question (1_b).

```
20  def Q1b_f():
21      return Configuration({'screenPosition': -5, 'xAxisColor': [0, 1, 0]}). \
22          setParameter('xAxisColor', [0, 1, 0]). \
23          setParameter('yAxisCo lor', [0,1,0]). \
24          display()
```

Dans ce code le screenposition correspond au zoom Xaxiscolor modifie la couleur de l'axe x en fonction des valeur mise dans la liste. Ces valeurs correspondent au dosage de couleur (Rouge Vert Bleu)

Le chainage des méthodes setparameter et display est possible car à la fin de la méthode setparameter elle renvoie l'objet sur lequel elle a été appelé.

Une traitement particulier est fait pour screenposition car il s'agit de la position à laquelle on observe l'objet donc si celle-ci varie nous devons changer l'échelle et donc modifier la matrice d'échelle.

4. Question (1_c).

```
78          gl.glRotatef(-90, 1, 0, 0)
```

Cette ligne de code permet d'effectuer une rotation de -90° par rapport à l'axe x. Permettant d'afficher le repère dans la position axe x largeur, axe y profondeur et axe z hauteur.

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

1. Question (1_d).

```
148         elif self.event.key == pygame.K_PAGEDOWN:
149             gl.glScalef(1/1.1, 1/1.1, 1/1.1)
150         elif self.event.key == pygame.K_PAGEUP:
151             gl.glScalef(1.1, 1.1, 1.1)
152
```

Dans ce code nous paramétrons le fait que lorsque l'on appuie sur la touche page down (pygame.k_PAGEDOWN), on dezoom la figure. De même pour page up utilisé pour zoomer.
Scale correspond au pas de l'échelle.

2. Question (1_e).

```
156
157         if self.event.button == 4:
158            gl.glScalef(1/1.1, 1/1.1, 1/1.1)
159         elif self.event.button == 5:
160            gl.glScalef(1.1, 1.1, 1.1)
161
```

Dans ce code nous liions un évènement à la molette de la souris. Si cet événement est égal à 4 on dezoom suivant l'échelle. Si l'évènement est égal à 5 on zoom.

3. Question (1_f).

```
164     def processMouseMotionEvent(self):
165
166         if pygame.mouse.get_pressed()[0]:
167             gl.glRotate(self.event.rel[0], 0, 1, 0)
168             gl.glRotate(self.event.rel[1], 1, 0, 0)
169
170         elif pygame.mouse.get_pressed()[2]:
171             gl.glTranslatef(self.event.rel[0]/10, self.event.rel[1]/10, 0)
172
173
```

IV. - Création d'une section

1. Question (2-a).

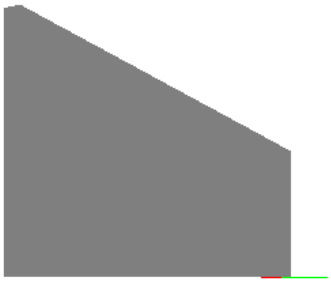
```
54      def generate(self):
55          self.vertices = [
56                  [0, 0, 0 ],
57                  [0, 0, self.parameters['height']],
58                  [self.parameters['width'], 0, self.parameters['height']],
59                  [self.parameters['width'], 0, 0],
60                 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

Dans la première partie du code nous définissons les coordonnées des différents sommets du parallélépipède. En fonction des paramètres d'entrées.

Puis nous définissons chaque face en fonction des points précédemment définit.

2. Question (2-b).

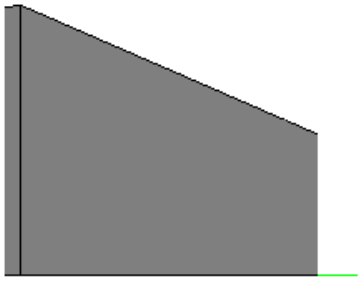
```
90      def draw(self):
91          for face in self.faces :
92
93              gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
94              gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
95              gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
96              gl.glVertex3fv([self.vertices[face[0]]])
97              gl.glVertex3fv([self.vertices[face[1]]])
98              gl.glVertex3fv([self.vertices[face[2]]])
99              gl.glVertex3fv([self.vertices[face[3]]])
100             gl.glEnd()
101
```



Ce code permet de lire les points des différentes faces puis assembles les coordonnées correspondantes de manière à construire les faces.

3. Question (2-c).

```
85      def drawEdges(self):
86
87          for face in self.faces :
88
89              gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)
90              gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
91              gl.glColor3fv([0, 0, 0]) # Couleur gris moyen
92              gl.glVertex3fv([self.vertices[face[0]]])
93              gl.glVertex3fv([self.vertices[face[1]]])
94              gl.glVertex3fv([self.vertices[face[2]]])
95              gl.glVertex3fv([self.vertices[face[3]]])
96              gl.glEnd()
97
98      # Draws the faces
99      def draw(self):
100         if self.parameters['edges']:
101             self.drawEdges()
102
```



Nous modifions le code précèdent pour y ajouter les bords.

V. Création des murs

1. Question (3a).

```
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
```

Ces lignes de code permettent de définir des paramètres par default ce ceux-ci ne sont pas renseigné par l'opérateur.
Excepté pour les paramètre Width et Height où lorsque ceux-ci ne sont pas renseigné un message d'erreur apparait.

```
38      self.objects = []
39
40      # Adds a Section for this object
41      self.parentSection = Section({'width': self.parameters['width'], \
42                                    'height': self.parameters['height'], \
43                                    'thickness': self.parameters['thickness'], \
44                                    'color': self.parameters['color'],
45                                    'position': self.parameters['position']})
46      self.objects.append(self.parentSection)
47
```

Nous créons une liste vide appelé objet appartenant à la classe Wall.
Ou dans celle-ci nous y ajoutons les paramètres de section.

```
70      def draw(self):
71
72          gl.glPushMatrix()
73          gl.glTranslatef( self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
74          gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
75          self.parentSection.draw()
76          gl.glPopMatrix()
77
```

Ce code permet de créé les mur en respectant leur position dans l'espace, grâce aux fonctions gl.glTranslatef, gl.glRotate encadré par la fonction gl.glPushMatrix et gl.glPopMatrix qui crée un nouveau repère qui appartient à la pièce crée pour ensuite à la fin du code le supprimer.

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

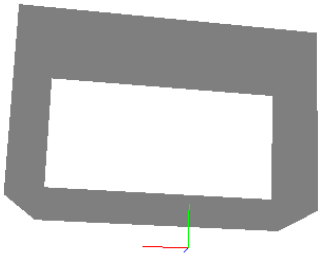
Nous créons un mur en fonction des paramètre Position, Width, Height et Orientation.

VI. Création d'une maison

1. Question (1).

```
42  def Q4a():
43      # Ecriture en utilisant des variables : A compléter
44      wall1 = Wall({'position': [0, 0, 0], 'width':7, 'height':3, 'orientation': 0  })
45      wall2 = Wall({'position': [0, 4.2, 0], 'width':7, 'height':3, 'orientation': 0  })
46      wall3 = Wall({'position': [0.2, 0.2, 0], 'width':4, 'height':3, 'orientation': 90  })
47      wall4 = Wall({'position': [7, 0.2, 0], 'width':4, 'height':3, '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 créons des murs en insèrent des dimensions puis de ces dimensions nous en déduisons leurs positions par rapport au repère de la maison.

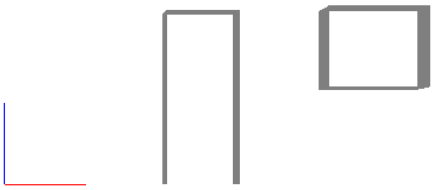
VII. Création d'ouvertures

1. Question (5-a).

```
    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'],0],
                [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
                ]

        self.faces = [

                [0, 4, 5, 1],
                [0, 4, 6, 3],
                [1, 7, 7, 5],
                [2, 7, 6, 3],
                ]

    # Draws the faces
    def draw(self):

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

        for face in self.faces :

            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([self.vertices[face[0]]])
            gl.glVertex3fv([self.vertices[face[1]]])
            gl.glVertex3fv([self.vertices[face[2]]])
            gl.glVertex3fv([self.vertices[face[3]]])
            gl.glEnd()

        gl.glPopMatrix()
```

Pour ces lignes de code nous nous sommes servi des codes que nous avons écrit dans la classe section puis nous avons enlevé deux face pour crée ces ouvertures.

Puis dans le code de la fonction draw nous ajoutons une fonction permettant de définir les positions des différentes faces par rapport au repère



Nous sommes désolé qu'une partie de la préparation n'apparaissent pas dans le rapport due l'arrêt de notre ordinateur alors que nous n'avions pas enregistré le début de notre rapport.