RAPPORT DE STAGE

Enzo Martins Lopes – Promo 2022/2023

Tuteur enseignant : Delperie Jérôme

Tuteur entreprise : Béchu J./ Vincent L.

Adresse enseignement professionnel : 1 AV CARNOT 91300 MASSY

Titre du projet : Outil de test d'une bibliothèque de guidage

Sommaire

REMERCIEMENTS :

Avant tout, je souhaite remercier l'ensemble de l'équipe AMC de m'avoir permis d'effectuer mon stage au sein du service IAS de THALES.

Merci tout particulièrement à mon tuteur de stage, de m'avoir accueilli chaleureusement dans l'équipe et guidé tout au long de ce stage.

Je tiens à remercier également tous les développeurs que j'ai pu côtoyer au cours de ce stage pour la bonne ambiance régnant dans le service et les discussions Merci à eux pour leur soutien et d'avoir répondu à mes questions parfois peu pertinentes.

J'ai bien sûr une pensée toute particulière pour l'ensemble des stagiaires qui ont également contribué à l'atmosphère détendue mais malgré tout productive qui a régné durant ce stage. Je salue d'autre part leurs participations aux conversations passionnantes et passionnées qui furent échangées sur les sujets les plus divers.

Introduction

J'ai eu l'opportunité de réaliser un stage au sein de l'entreprise Thales dans le cadre de ma formation en BUT Génie électrique et informatique industrielle pendant 9 semaines, du 2 janvier au 3 mars. Cette expérience a été très enrichissante, me permettant de découvrir le fonctionnement de l'entreprise, d'acquérir de nouvelles compétences et d'approfondir mes connaissances théoriques.

J'ai effectué ce stage sur le site de Massy, dont l'activité porte sur le développement de systèmes de défense aérienne. Plus précisément, ma mission s'est déroulée au sein de l'équipe Air Mission Control du projet ACCS. Le développement de l'ACCS a débuté en 1999 avec pour objectif de remplacer de nombreux systèmes de commandement et de contrôle aériens des pays de l'OTAN. Il est développé sous la responsabilité de Thales-Raytheon Systems. Il fournira un système de commandement et de contrôle aériens unifié, permettant aux pays de l'OTAN de gérer tous les types d'opérations aériennes (planification, attribution et exécution) au-dessus du territoire des pays européens de l'OTAN ou en opération extérieure.

L'Air Mission Control est un système de commandement et de contrôle aériens de l'OTAN.

Dans ce rapport, je vais vous présenter dans un premier temps l'entreprise, puis nous aborderons un chapitre concernant la montée en compétence pour ma mission en Python, et enfin nous finirons par la mission elle-même, qui consiste à améliorer un outil de visualisation.

Présentation de l’entreprise

Thales est une entreprise française de haute technologie spécialisée dans les domaines de la défense, de l'aérospatiale, de la sécurité et des transports. Elle a été fondée en 1893. Elle est actuellement basée à La Défense, en France. Thales est présent dans plus de 56 pays à travers le monde et emploie plus de 80 000 personnes.

Activités :

Thales opère dans cinq principaux domaines d'activités :

Défense et sécurité : Thales fournit des solutions de défense, de sécurité et de cyber sécurité aux gouvernements, aux forces armées et aux forces de l'ordre. Cela comprend des systèmes de communication, des systèmes de surveillance et des équipements de défense.

Aérospatiale : Thales fournit des solutions pour l'aéronautique, les hélicoptères, les avions commerciaux, les avions d'affaires et les satellites. Cela comprend des systèmes de navigation, des systèmes de communication et des systèmes de contrôle de vol.

Transport terrestre : Thales fournit des solutions pour les transports terrestres, tels que les trains et les métros. Cela comprend des systèmes de signalisation, des systèmes de communication et des équipements de contrôle des trains.

Transport maritime : Thales fournit des solutions pour les transports maritimes, tels que la navigation et les systèmes de communication pour les navires.

Identité et sécurité numérique : Thales fournit des solutions pour la sécurité numérique, telles que la protection des données, la biométrie et la gestion des identités.

Produits et services :

Thales propose une large gamme de produits et services dans les domaines de la défense, de l'aérospatiale, de la sécurité et des transports. Parmi les produits et services proposés par Thales, on peut citer :

1. Des systèmes de communication pour les forces armées et les forces de l'ordre
2. Des systèmes de contrôle de vol pour les avions commerciaux
3. Des équipements de contrôle pour les trains et les métros
4. Des solutions de cyber sécurité pour les gouvernements et les entreprises
5. Des systèmes de navigation pour les navires et les avions
6. Des solutions de protection des données pour les entreprises

Partenariats et collaborations :

Thales collabore avec un certain nombre d'entreprises et d'organisations pour fournir des solutions de haute technologie dans le monde entier.

Conclusion :

Thales est une entreprise de haute technologie qui fournit des solutions de défense, d'aérospatiale, de sécurité et de transports dans le monde entier. Avec plus de 80 000 employés dans plus de 56 pays, Thales est un leader dans son domaine et continue de développer

Projet de qualification d'un répertoire

1) Problématique :

J'ai donc eu comme problématique de devoir faire un algorithme en Python qui doit permettre à l'utilisateur d'obtenir une vue d'ensemble des fichiers présents dans un répertoire donné en fournissant des informations pertinentes et utiles sur ces fichiers. En utilisant des bibliothèques Python telles que os, ce script sera capable d'analyser chaque fichier dans le répertoire et d'extraire des informations telles que la taille du fichier, l'extension du fichier et le nombre de fichiers pour chaque extension.

Ce projet peut être très utile pour les utilisateurs qui ont besoin de gérer un grand nombre de fichiers dans un répertoire donné, en leur permettant de mieux comprendre la composition de ces fichiers et de prendre des décisions éclairées sur la façon de les gérer.

En utilisant Python, nous sommes en mesure de fournir une solution efficace et flexible qui peut être facilement adaptée pour répondre à différents besoins.

2)Etude et développement du sujet :

2.1) Algorithme Python :

Explication :

Comme dit précédemment, j'ai dû faire un algorithme en python qui a pour objectif de devoir récupérer les fichiers locaux et devoir donner le chemin.

Ce code Python permet de parcourir récursivement un répertoire de départ spécifié et d'afficher tous les fichiers et sous-répertoires contenus dans ce répertoire.

Pour chaque fichier, le code affiche son nom, sa taille en bytes et son extension.

Plus précisément, le code utilise la fonction **os.walk** de la bibliothèque **os** pour parcourir récursivement le répertoire de départ et récupérer tous les noms de fichiers et de sous-répertoires. Ensuite, pour chaque fichier, il utilise les fonctions **os.path.getsize** et **os.path.splitext** pour récupérer la taille et l'extension du fichier.

Enfin, il affiche toutes les informations récupérées de manière formatée à l'écran en utilisant des indentations pour représenter la structure hiérarchique des fichiers et sous-répertoires.

Ce code en Python liste tous les fichiers dans un répertoire de départ donné, en incluant tous les sous-répertoires et affiche les informations suivantes pour chaque fichier : nom, taille en octets et extension.

Le module os est importé pour permettre l'utilisation des fonctions de système d'exploitation, en particulier la fonction os.walk () qui permet de parcourir tous les fichiers et répertoires à partir d'un chemin de départ donné.

Le code utilise une boucle for pour parcourir chaque répertoire et fichier dans le chemin de départ en utilisant la fonction os.walk (). Pour chaque répertoire trouvé, il affiche le nom du répertoire avec un niveau d'indentation approprié. Pour chaque fichier trouvé, il affiche le nom du fichier avec un niveau d'indentation approprié et affiche également la taille du fichier en octets et l'extension du fichier.

Difficulté :

Les difficultés auxquelles j'ai pu faire face sont tout d'abord le manque de connaissance que j'avais au tout début. J'ai donc dû d'abord me renseigner sur internet pour pouvoir trouver des solutions. Mais je ne pouvais pas prendre n'importe quoi en effet Thales dispose d'un environnement très contraint, lié à la sécurité, ce qui m'a imposé d'être vigilant sur la récupération de code et d'exemples d'internet. Je devais donc faire attention et tester un par un mes différentes fonctions de mon côté lors de mon stage.

J'ai donc dû faire attention aux différents codes qui possédaient différentes licences j'ai dû apprendre et essayer de m'inspirer de mon côté.

Détail du code :

Voici mon premier code qui a pour but de donner uniquement le chemin, l'extension et la taille.

```
import os #Allows to give access to the file of the computer (module os)

def list_files(startpath):
    for root, dirs, files in os.walk(startpath):
        level = root.replace(startpath, '').count(os.sep)
        indent = ' ' * 4 * (level)
        print(f'{indent}[{os.path.basename(root)}]')
        subindent = ' ' * 4 * (level + 1)
        for f in files:
            print(f'{subindent}{f}')
            file_path = os.path.join(root, f)
            file_size = os.path.getsize(file_path)
            print(f'{subindent} - Size: {file_size} bytes')
            file_extension = os.path.splitext(f)[1]
            print(f'{subindent} - Extension: {file_extension}')

list_files('/path/to/directory')
```

Ici nous pouvons voir que j'utilise « import os » qui va m'accompagner dans mon projet tout du long. En effet le module ***os*** en Python permet d'interagir avec le système d'exploitation, ainsi de manipuler les répertoires c'est-à-dire il fournit des fonctions permettant de créer et de supprimer un fichier, de récupérer son contenu, de modifier et d'identifier le répertoire courant. On a donc la possibilité de beaucoup manipuler grâce à la fonction import os.

Ensuite, la fonction "list_files" est définie avec l'argument "startpath" :

```
def list_files(startpath):
```

Cette fonction prend un chemin de départ "startpath", à partir duquel elle commencera à parcourir les fichiers et les répertoires. Le but de cette fonction est d'afficher tous les fichiers et les répertoires dans l'arborescence à partir de "startpath" et d'afficher quelques détails sur chaque fichier, tels que la taille et l'extension.

La fonction utilise la méthode "os.walk" pour parcourir tous les sous-répertoires et les fichiers à partir du "startpath". La ligne suivante utilise "os.walk" pour obtenir les informations de chaque répertoire, y compris le nom du répertoire, les noms des sous-répertoires et les noms des fichiers :

```
for root, dirs, files in os.walk(starpath):
```

Cela crée une boucle qui parcourt chaque sous-répertoire, les sous-répertoires et les fichiers contenus dans chaque sous-répertoire.

Ensuite, pour chaque sous-répertoire, la fonction affiche le nom du répertoire entre crochets, pour indiquer que c'est un sous-répertoire. La ligne suivante crée une indentation pour afficher le nom du répertoire :

```
level = root.replace(startpath, '').count(os.sep)
indent= ' ' * 4 * (level)
print(f'{indent}[{os.path.basename(root)}]')
```

La variable "level" est utilisée pour déterminer la profondeur du sous-répertoire par rapport au "startpath", en remplaçant "startpath" par une chaîne vide et en comptant le nombre de séparateurs de chemin d'accès (os.sep) restants. La variable "indent" est utilisée pour créer une indentation pour chaque niveau de profondeur du sous-répertoire. La méthode "os.path.basename" est utilisée pour obtenir le nom du répertoire à partir du chemin d'accès complet, et ce nom est ensuite affiché entre crochets.

Ensuite, la fonction utilise une indentation supplémentaire pour afficher les noms de fichiers et leurs détails tels que la taille et l'extension. La ligne suivante crée une indentation pour chaque fichier :

```
subindent= ' ' * 4 * (level + 1)
```

La variable "subindent" est utilisée pour créer une indentation supplémentaire pour chaque niveau de profondeur du sous-répertoire, afin d'afficher les noms de fichiers à l'intérieur de chaque sous-répertoire.

Ensuite, la fonction utilise une boucle pour parcourir chaque fichier dans chaque sous-répertoire et afficher des détails sur chaque fichier. La ligne suivante crée une boucle pour parcourir chaque fichier :

```
for f in files :
```

Pour chaque fichier, la fonction affiche le nom du fichier, sa taille et son extension. La ligne suivante affiche le nom du fichier :

```
print(f'{subindent}{f}')
```

La variable "subindent" est utilisée pour créer une indentation pour chaque niveau de profondeur du sous-répertoire, afin que le nom du fichier soit affiché sous le nom du sous-répertoire..

Ensuite, la fonction utilise la méthode "os.path.join" est utilisée pour joindre plusieurs chemins de fichiers ou de répertoires en un seul chemin de fichier. Elle prend un ou plusieurs arguments qui représentent les parties du chemin, et renvoie un chemin unique qui est valide pour le système d'exploitation sur lequel vous exécutez votre programme pour obtenir le chemin d'accès complet de chaque fichier :

```
file_path = os.path.join(root, f)
```
La méthode "os.path.join" combine le chemin d'accès du sous-répertoire et le nom du fichier pour créer le chemin d'accès complet du fichier.

Ensuite, la fonction utilise la méthode "os.path.getsize" pour obtenir la taille de chaque fichier :

```
file_size = os.path.getsize(file_path)
```
La méthode "os.path.getsize" prend le chemin d'accès complet du fichier en argument et retourne la taille du fichier en octets.

Par la suite, la fonction affiche la taille de chaque fichier avec une indentation supplémentaire :

```
print(f'{subindent} - Size: {file_size} bytes')
```
Enfin, la fonction utilise la méthode "os.path.splitext" pour obtenir l'extension de chaque fichier :

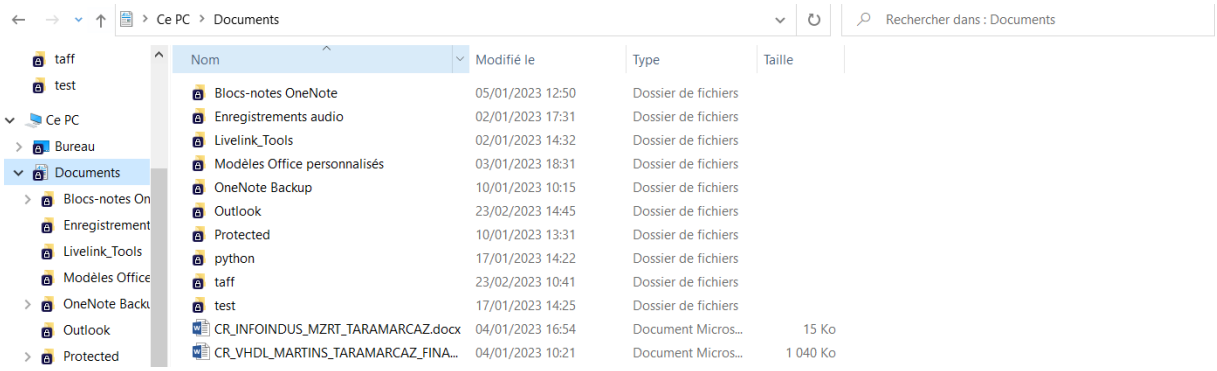
```
file_extension = os.path.splitext(f)[1]
```
La méthode "os.path.splitext" prend le nom du fichier en argument et retourne une liste contenant le nom du fichier et son extension. La deuxième valeur de la liste est l'extension, qui est stockée dans la variable "file_extension".

Enfin, la fonction affiche l'extension de chaque fichier avec une indentation supplémentaire :

```
print(f'{subindent} - Extension: {file_extension}')
```
En résumé, la fonction "list_files" parcourt tous les sous-répertoires et les fichiers à partir du "startpath" et affiche le nom de chaque sous-répertoire, le nom de chaque fichier, sa taille et son extension. La fonction utilise la bibliothèque "os" pour obtenir des informations sur les fichiers et les répertoires. La fonction utilise également des indentations pour afficher les fichiers à l'intérieur de chaque sous-répertoire et pour afficher les détails de chaque fichier avec une indentation supplémentaire.

Voici donc le résultat :

```
>>> def list_files(startpath): #We give the name to starpath
...     for root, dirs, files in os.walk(startpath): #
...         level = root.replace(startpath, '').count(os.sep)
...         indent = ' ' * 4 * (level)
...         print(f'{indent}[{os.path.basename(root)}]')
...         subindent = ' ' * 4 * (level + 1)
...         for f in files: # exemple de for
...             print(f'{subindent}{f}')
...             file_path = os.path.join(root, f)
...             file_size = os.path.getsize(file_path)
...             print(f'{subindent} - Size: {file_size} bytes')
...             file_extension = os.path.splitext(f)[1]
...             print(f'{subindent} - Extension: {file_extension}')
...
>>> list_files('Documents')
[Documents]
    CR_INFOINDUS_MZRT_TARAMARCAZ.docx
     - Size: 14384 bytes
     - Extension: .docx
    CR_VHDL_MARTINS_TARAMARCAZ_FINAL.docx
     - Size: 1064524 bytes
     - Extension: .docx
    desktop.ini
     - Size: 402 bytes
     - Extension: .ini
    prodact.dat
     - Size: 118 bytes
     - Extension: .dat
    stage.txt
     - Size: 948 bytes
```



Ici nous pouvons voir que le résultat est en effet correct avec ce que nous avons trouvé dans la console.

2.2) Amélioration de l’approche en catégorisant les types de fichiers :

Par la suite j’ai eu une nouvelle problématique. On m’a demandé de rajouter dans mon code la fonction qui parcourt la liste de fichiers récupérée et extrait l'extension de chaque fichier et compte le nombre de fichiers ayant la même extension. Le résultat est stocké dans un dictionaire appelé extensionCount.

J’ai donc garde le même principe pour mon code mais j’ai uniquement apporté une fonction qui stocke les détails de chaque fichier dans un dictionnaire. La clé de chaque élément du dictionnaire est le chemin d'accès complet du fichier, et la valeur est un autre dictionnaire contenant des détails sur le fichier, tels que la taille et l'extension qui nous sera très utiles.

En effet j’ai donc fait ceci pour mon premier parti :

```
import os

def listFiles(startpath):
    results = {}
    for root, dirs, files in os.walk(startpath):
        for fileName in files:
            filePath = os.path.join(root, fileName)
            fileSize = os.path.getsize(filePath)
            fileExtension = os.path.splitext(fileName)[1]
            results[filePath] = {
                'size': fileSize,
                'extension': fileExtension,
                'name' : fileName,
            }
    return results
```

Par la suite j'ai donc fait le deuxième parti de mon code qui est :

```
def countExtensionFiles(startpath) :
    filesDict = listFiles(startpath)
    extensionCount = {}
    for filePath in filesDict:
        fileExtension = filesDict[filePath]['extension']
        if fileExtension in extensionCount:
            extensionCount[fileExtension] += 1
        else:
            extensionCount[fileExtension] = 1
    for extension in extensionCount:
        print(f"{extension} : {extensionCount[extension]} files")
    return extensionCount

if __name__ == '__main__':
    filesDict = countExtensionFiles('C:\\Users\T0281448\Documents')
    print(filesDict)
```

Donc ici nous pouvons voir que la fonction utilise une autre fonction appelée "listFiles" pour trouver tous les fichiers présents dans le dossier et stocke les informations sur chaque fichier dans un dictionnaire appelé "filesDict".

La fonction "countExtensionFiles" parcourt ensuite chaque fichier dans "filesDict" et récupère l'extension de chaque fichier. Elle utilise ensuite un autre dictionnaire appelé "extensionCount" pour compter le nombre de fichiers pour chaque extension.

Enfin, la fonction "countExtensionFiles" affiche le nombre de fichiers pour chaque extension et renvoie le dictionnaire "extensionCount".

La dernière partie du code utilise la fonction "countExtensionFiles" pour analyser le dossier spécifié et stocke le résultat dans une variable appelée "filesDict". Le programme imprime ensuite ce dictionnaire pour afficher le nombre de fichiers pour chaque extension de fichier présente dans le dossier analysé.

Voici le code complet :

```python
import os

def listFiles(startpath):
    results = {}
    for root, dirs, files in os.walk(startpath):
        for fileName in files:
            filePath = os.path.join(root, fileName)
            fileSize = os.path.getsize(filePath)
            fileExtension = os.path.splitext(fileName)[1]
            results[filePath] = {
                'size': fileSize,
                'extension': fileExtension,
                'name' : fileName,
            }
    return results

def countExtensionFiles(startpath) :
    filesDict = listFiles(startpath)
    extensionCount = {}
    for filePath in filesDict:
        fileExtension = filesDict[filePath]['extension']
        if fileExtension in extensionCount:
            extensionCount[fileExtension] += 1
        else:
            extensionCount[fileExtension] = 1
    for extension in extensionCount:
        print(f"{extension} : {extensionCount[extension]} files")
    return extensionCount

if __name__ == '__main__':
    filesDict = countExtensionFiles('C:\\Users\T0281448\Documents')
    print(filesDict)
```

Et voici le résultat :

```
>>> def countExtensionFiles(startpath) :
...     filesDict = listFiles(startpath)
...     extensionCount = {}
...     for filePath in filesDict:
...         fileExtension = filesDict[filePath]['extension']
...         if fileExtension in extensionCount:
...             extensionCount[fileExtension] += 1
...         else:
...             extensionCount[fileExtension] = 1
...     for extension in extensionCount:
...         print(f"{extension} : {extensionCount[extension]} files")
...     return extensionCount
...
>>> if __name__ == '__main__':
...     filesDict = countExtensionFiles('C:\\Users\T0281448\Documents')
...     print(filesDict)
...
.docx : 7 files
.ini : 1 files
.dat : 1 files
.txt : 2 files
.py : 5 files
.sav : 4 files
.one : 2 files
.onetoc2 : 1 files
.xml : 1 files
.ost : 1 files
{'.docx': 7, '.ini': 1, '.dat': 1, '.txt': 2, '.py': 5, '.sav': 4, '.one': 2, '.onetoc2': 1, '.xml': 1, '.ost': 1}
>>>
```

Ici nous pouvons donc voir qu'il compte belle et bien le nombre de fichier.

Présentation du projet d'amélioration de l'interface graphique de test

Pour ce projet Thales avait besoin de visualiser leurs données lors de l'exécution de test. Comme par exemple, visualiser la trajectoire d'un avion de chasse vers sa cible.

Mon objectif sera donc de de pouvoir faire visualiser en 3D ces différent test et de devoir améliorer les demandes.

Dans le cadre de l'ACCS et de l'équipe AMC de nombreux tests sont réalisés afin d'éprouver nos logiciels.

Lors de ces tests, un besoin est apparu de visualiser le résultat de certains algorithmes. Plus précisément, dans le cadre de test d'une bibliothèque de guidage l'équipe a besoin de visualiser la trajectoire en 3D.

Le framework de test disposait à mon arrivé des technologies suivantes pour l'interface graphique : Javascipt pour la partie Web et Python pour la partie server.

J'aurais donc un cahier des charges avec toutes les demandent de Thales pour pouvoir faire différentes formes ou qui répondent aux besoins utilisateur.

Cahier des charges :

1. L'objectif est de développer une bibliothèque d'objet en JavaScript qui permettra à l'utilisateur de créer des formes géométriques personnalisées

- Une croix positionnable avec des options de personnalisation (taille, couleur, épaisseur)
- Faire une forme géométrique à partir de la partie utilisateur.
- Ainsi qu'un cylindreOffsett qui pourra être décalé en hauteur et bas.
- Puis d'un Billboard une image qui est une phrase en 2D qui s'affichera.

2. Fonctionnalités La bibliothèque doit permettre à l'utilisateur de :

- Créer des formes géométriques personnalisées horizontales via la méthode "shapeGeometry".
- Ajouter une croix personnalisable (taille, couleur, épaisseur) positionnable à l'aide de la méthode "crossShape".
- Créer un cylindre avec décalage en hauteur personnalisable (Bottomheigh, TopHeight) via la méthode "cylinderOffset".
- Pouvoir ajouter une phrase via l'utilisation de Billboard.

3. Contraintes techniques que la bibliothèque devra respecter les contraintes techniques suivantes :

- Être développée en JavaScript.
- Être compatible avec les navigateurs web courants (Chrome, Firefox, Safari, Edge).
- Respecter les normes de codage et de documentation pour faciliter la maintenance et l'évolution future.
- Fournir une documentation claire et complète.

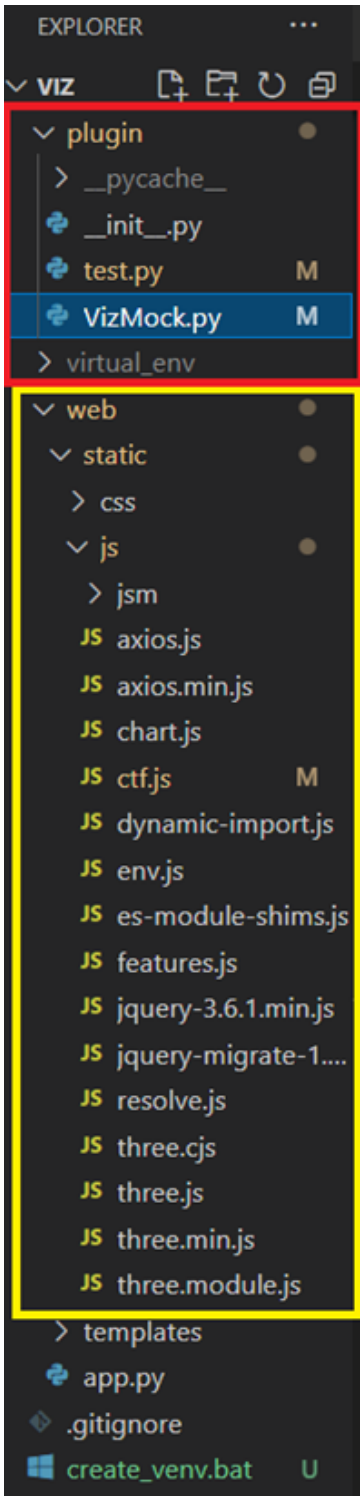
4. Délais

- Le développement de la bibliothèque devra être effectué dans un délai de 1 mois à partir de la validation du cahier des charges

5. Maintenance et évolution La bibliothèque devra être maintenue et évolutive.

- Une formation pour l'utilisation et la mise à jour régulière de la bibliothèque seront prévues. Des mises à jour régulières devront être effectuées pour assurer la compatibilité avec les nouvelles versions de navigateurs web et les nouvelles technologies.

Interface graphique pour le Framework de test :



Environnement de travail :

Je travaille donc sur Visual studio code. Lors de cette mission je vais devoir travailler en JavaScript qui est un langage de programmation de haut niveau principalement utilisé pour créer des applications web interactives.

Mais je vais aussi utiliser du Python qui va servir davantage à envoyer les commandes demandées.

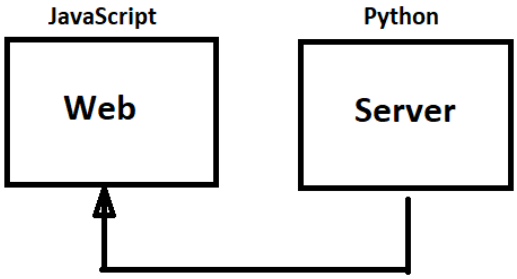
JavaScript est un langage de script orienté objet et utilise des événements pour interagir avec les utilisateurs. Il peut être utilisé pour créer des animations, des effets visuels, des formulaires interactifs, des jeux, des applications de chat et bien plus encore.

Ici nous pouvons voir les différents partis en Python et en JavaScript.

La partie en rouge sera ma partie Python et ma partie en jaune sera celle en JavaScript.

Désormais je vais vous parlez du lien entre l'aspect JavaScript et le code Python

Lien entre Python et JavaScript :



Ici grâce à ce schéma nous pouvons voir que la partie python fait des requêtes c'est-à-dire que python utilise une bibliothèque requests, qui permet de faire des requêtes http donc il envoie l'information qu'il veut afficher sur le site qui utilise JavaScript.

Personnellement je vais travailler uniquement sur la partie Test.py et VizMock.py pour la partie Python et pour la partie JavaScript ça sera CTF.js

Voici donc la partie Python qui se nomme test.py:

```python
import VizMock
#plugin.setPosition(cube, (350, 10, 10))
#
#plugin.setRotation(cube, (0.5, 0.3, 0.2))
#
#plugin.drawCone((-100, -100, 100), (0, 0.3, 0), 5, 30, '#CCCCFF')
#
#plugin.drawSphere((-120, -100, 100), 5, '#F00000')
#
#plugin.drawCylinder((-10, 0, 10), (0, 0, 0), 1, 21, '#00FF00')
#
#plugin.drawArrow((0, 0, 0), (2, 1, 0), 1, '#FF0000')
#
#plugin.drawPath(
#   (( -10, 0, 10 ),
#   ( -5, 5, 5 ),
#   ( 0, 0, 0 ),
#   ( 5, -5, 5 ),
#   ( 10, 0, 10 )
#    ),
#    color='#0000FF'
#)
#
if __name__ == '__main__':
    plugin = VizMock.VizMock()
    plugin.removeAll()
    plugin.drawPlane(width= 1 , height =1 ,color='#FF0000')
```

Ce sera donc ici que je vais demander d'afficher, par exemple, un avion. La demande de la partie client sera donc faite ici. L'utilisateur n'aura qu'à choisir l'un des plugins pour pouvoir afficher une sphère ou un cône, par exemple.Il pourra aussi choisir l'emplacement par

exemple avec la position x, y et Z. Il pourra aussi choisir la couleur comme par exemple : blue =0000FF ;(utilisation hexadecimal)

Et il aura d'autre options a choisir en fonction de l'objet choisit.

Voici desormais la parti Viz qui est aussi en python:

```python
def drawSphere(self, position, radius, color):
    return self.send({
        'action': 'sphere',
        'position': position,
        'radius': radius,
        'color': color
    })
```

Ici, la méthode utilisée utilise un protocole de communication spécifique pour envoyer une demande à un serveur, qui est représenté ici par une structure de données contenant une action (dans ce cas, drawsphere) et les paramètres nécessaires pour effectuer cette action. Le serveur (test.py) devrait être configuré pour comprendre et répondre à ce type de demande en affichant le texte spécifié sur le panneau d'affichage aux coordonnées spécifiées. C'est donc ici que je vais devoir choisir les différentes options de mon objet que je vais afficher sur mon interface.

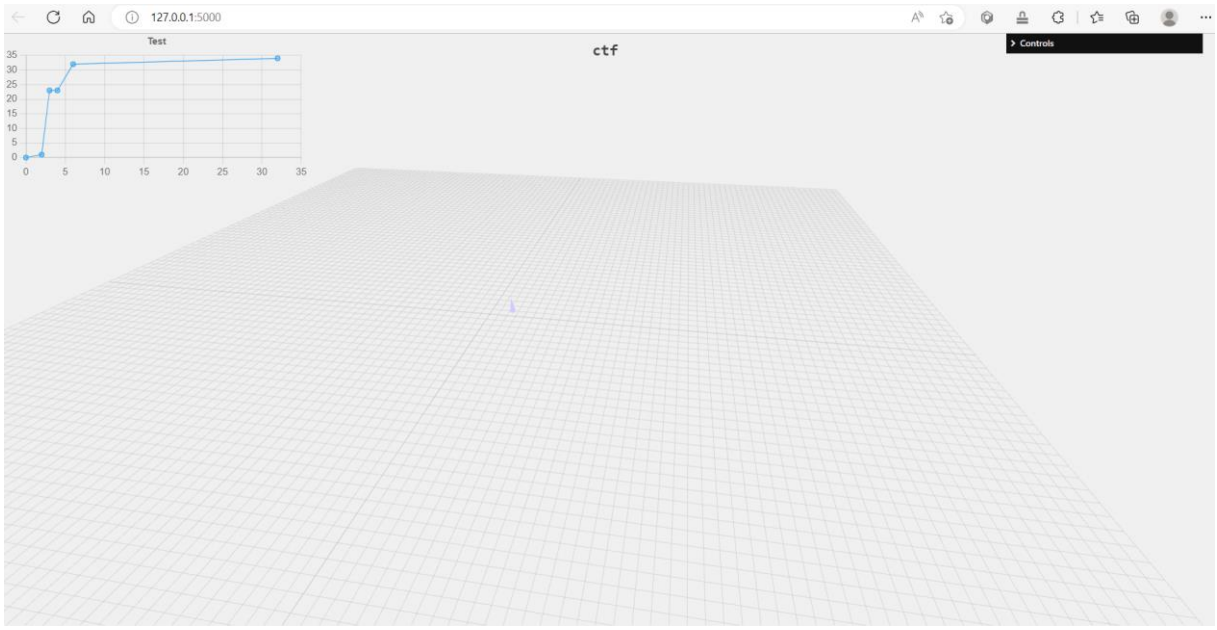
Voici la partie JavaScript:

```javascript
function addSphere(position, radius, color) {
    const geometry = new THREE.SphereGeometry( radius, 32, 16 );
    const material = new THREE.MeshBasicMaterial({ color: color });
    const object = new THREE.Mesh(geometry, material);
    object.position.x = position[0];
    object.position.y = position[1];
    object.position.z = position[2];
    return object;
}
```

En général, les codes suivent le paradigme de programmation orientée objet (POO) en utilisant les concepts de la géométrie, des matériaux et des maillages. Cela permet de construire des objets 3D complexes en combinant différentes géométries et matériaux pour créer des maillages de plus en plus complexes.

J'utilise donc la bibliothèque THREE.JS pour créer et d'afficher des graphismes 3D interactifs dans un navigateur web. Ici j'utilise donc THREE.SphereGeometry pour pouvoir afficher une sphère. On rentrera dans les détails du code par la suite.

Voici donc l'interface Web :



Ici nous avons donc l'interface de la bibliothèque de guidage. C'est donc ici que on n'affichera nos différents Object et qu'on pourra faire nos tests. Il recevra donc les demandes du server (Test.py) et les affichera donc ici.

ShapeGeometry :

Explication :

Nous allons donc devoir faire une fonction qui a pour but de faire une n'importe quelle figure choisie par l'utilisateur dans la partie test.py en python. Cette fonction va devoir prendre un tableau de points pour créer une forme géométrique en deux dimensions. Les points choisis se placeront uniquement en [x] [y]. De plus nous pourront choisir une couleur.

Difficulté :

Ici la difficulté va être de devoir faire un programme ou le client choisi la forme a créé. En effet il existe une fonction dans la bibliothèque THREE.Js qui se nomme ShapeGeometry mais malheureusement celle-là ne fera uniquement la figure dans la partie Web client et non dans server test.py.

Pour ça je vais vous montrez un exemple pour que nous comprenions mieux comment fonctionne la fonction shapeGeometry :

```
function addShapeGeometry(points, color) {
    console.log('coucou')
    const x = 0, y = 0;

    const heartShape = new THREE.Shape();

    heartShape.moveTo( x + 5, y + 5 );
    heartShape.bezierCurveTo( x + 5, y + 5, x + 4, y, x, y );
    heartShape.bezierCurveTo( x - 6, y, x - 6, y + 7,x - 6, y + 7 );
    heartShape.bezierCurveTo( x - 6, y + 11, x - 3, y + 15.4, x + 5, y + 19 );
    heartShape.bezierCurveTo( x + 12, y + 15.4, x + 16, y + 11, x + 16, y + 7 );
    heartShape.bezierCurveTo( x + 16, y + 7, x + 16, y, x + 10, y );
    heartShape.bezierCurveTo( x + 7, y, x + 5, y + 5, x + 5, y + 5 );

    const geometry = new THREE.ShapeGeometry( heartShape );
    const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
    const mesh = new THREE.Mesh( geometry, material ) ;
    scene.add( mesh );
}
```

Pour mon premier test j'ai donc essayé de faire un cœur et en mettant le programme dans la partie web pour voir si tout fonctionne.

Explication :

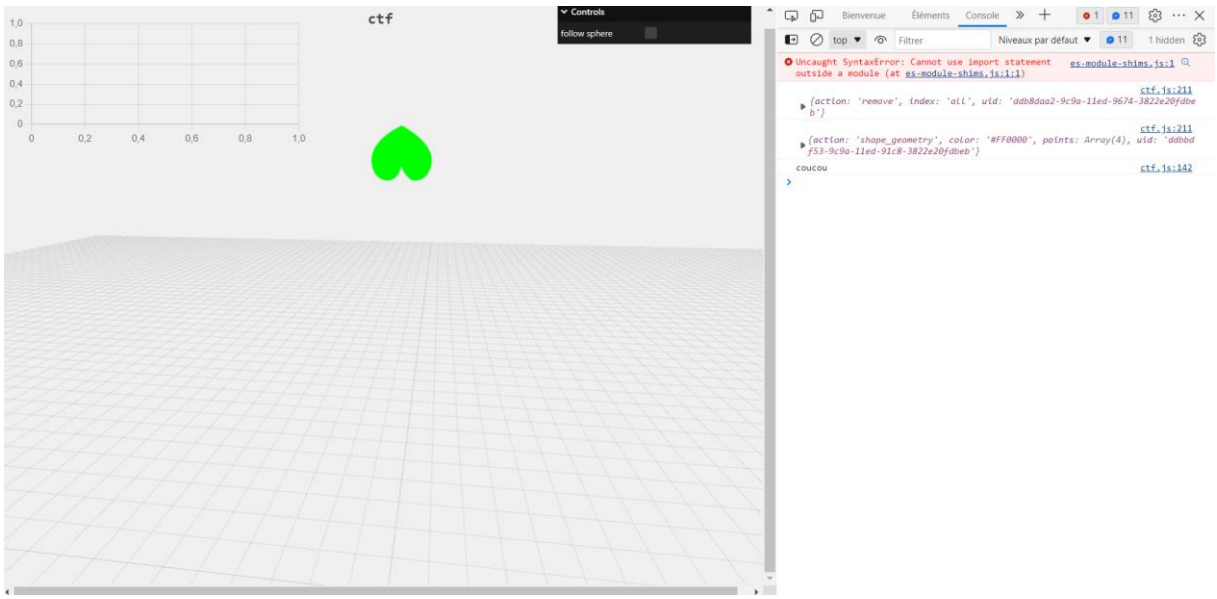
Les trois premières lignes créent des constantes x et y à 0 pour servir de coordonnées de base pour la forme du cœur. Ensuite, une nouvelle forme de Three.js est créée avec new THREE.Shape().

La forme du cœur est dessinée en utilisant des courbes de Bézier à l'aide de la méthode bezierCurveTo() de la forme. Les points de contrôle de la courbe sont définis en ajoutant ou en soustrayant des valeurs aux coordonnées de base x et y.

Une fois que la forme du cœur est définie, elle est utilisée pour créer une géométrie Three.js avec new THREE.ShapeGeometry(heartShape). Cette géométrie est ensuite utilisée pour créer un matériau avec new THREE.MeshBasicMaterial({color: 0x00ff00}), qui est défini avec une couleur verte.

Enfin, un objet mesh est créé en utilisant la géométrie et le matériau, puis ajouté à la scène avec scene.add(mesh). Cela permet à la forme du cœur d'être rendue dans la scène 3D.

Résultat :



Nous pouvons donc voir que notre cœur est correctement apparu mais il y a plusieurs soucis, nous pouvons déjà voir que notre figure n'est pas droite en effet dans le cahier des charges il est précisé que nous devons avoir une forme horizontale. Ensuite il va falloir faire en sorte que l'utilisateur puisse choisir sa forme et non qu'elle soit déjà prédéfinie dans la partie JavaScript.

Par la suite j'ai donc tenté un algorithme avec un « for » et des « if » pour pouvoir faire une liaison avec la partie de l'utilisateur :

```
function addShapeGeometry(points, color) {
    const shape = new THREE.Shape();
    for (i = 0; i < points; i++)
        if (i == 0);
        {
            shape.moveTo(points[i][x], points[i][x]);
        } else {
            shape.lineTo(points[i][x], points[i][x]);
        }
    const geometry = new THREE.ShapeGeometry( shape );
    const material = new THREE.MeshBasicMaterial( { color: color } );
    const mcuh = new THREE.Mesh( geometry, material );
    object.position.x = position[0];
    object.position.y = position[1];
    object.position.z = position[2];
    object.rotation.x = rotation[0];
    object.rotation.y = rotation[1];
    object.rotation.z = rotation[2];
    return object;
}
```

Explication :

Le for a pour objectif de bouclé sur chaque élément dans l'objet points en utilisant une boucle for et vérifie si l'élément actuel est le premier élément en utilisant un if:

Le bloc de code suivant le if utilise la méthode moveTo() de l'objet shape pour définir le premier point de la forme géométrique. La méthode moveTo() prend deux arguments, qui sont les coordonnées x et y du point:

Pour tous les autres points de la forme géométrique, le code utilise la méthode lineTo() de l'objet shape pour définir une ligne reliant le point précédent au point actuel:

Une fois que tous les points de la forme géométrique ont été ajoutés à l'objet shape, la fonction crée un objet ShapeGeometry.

Donc pour résumer nous venons de faire un algorithme qui permet l'utilisateur de faire un Object. Mais ce n'est pas tout désormais nous allons devoir le lier à la partie python (Viz et test.py) pout que l'utilisateur puisse manipuler.

```python
def drawShapeGeometry(self, points, color):
    return self.send({
        'action': 'shape_geometry',
        'points': points,
        'color': color
    })
```

J’ai choisi points et color dans ma partie Viz comme action à faire.

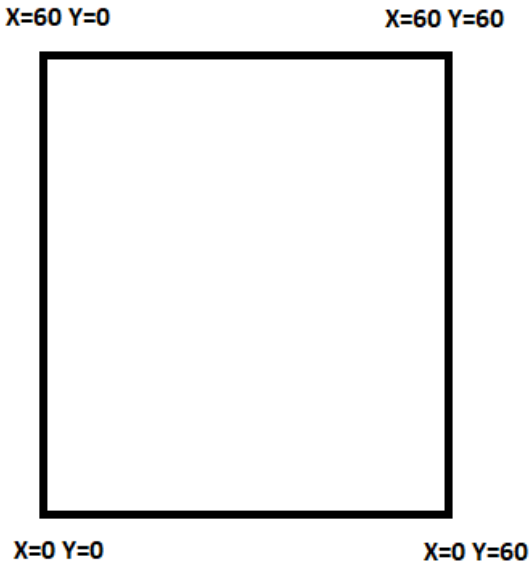
```python
if __name__ == '__main__':
    plugin = VizMock.VizMock()
    plugin.removeAll()
    plugin.drawShapeGeometry(points=[[0, 0], [0, 60], [60, 60], [60, 0 ], [0, 0]], color='#FF0000')
    #plugin.setOffset(100, 100, 100)
    #plugin.scale(0.5, 0.19, 5)
    #plugin.setSceneFactor(0.1)
```

Ici nous avons donc la partie test .py ces donc ici que l’utilisateur va devoir choisir les coordonnées de ça figure.

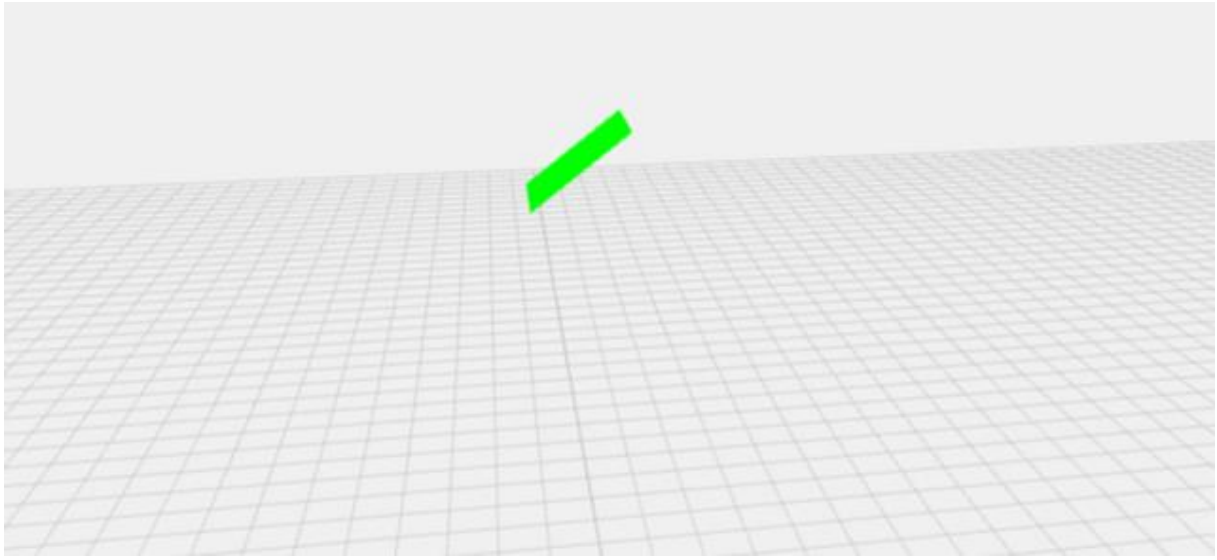
Mais il y a un sens précis pour utiliser ShapeGeometry voici comment fonctionne ShapeGeometry

Donc ici nous voyons que les premiers coordonnés sont de x=0 et y=0 donc rien ne bouge. Ensuite x=0 et y=60 donc un trait se fait à droite puis x=60 et y=60 nous restons au même endroit mais montons avec un trait de 60. Par la suite nous faisons x=60 et y=60 nous gardons donc la même hauteur mais nous allons vers la gauche. Pour finir nous avons x=0 et y=0 pour retourner au tout début.

Cela devrait faire ceci :



Résultat :



Ici nous pouvons voir qu'il y a un problème de rotation avec la figure est que ce n'est pas ce que nous voulons car nous voulons que la figure soit obligatoirement horizontale.

Pour ce faire je vais donc améliorer mon code en utilisant autre chose que mon for et rajouter une rotation correcte à ma figure.

Code :

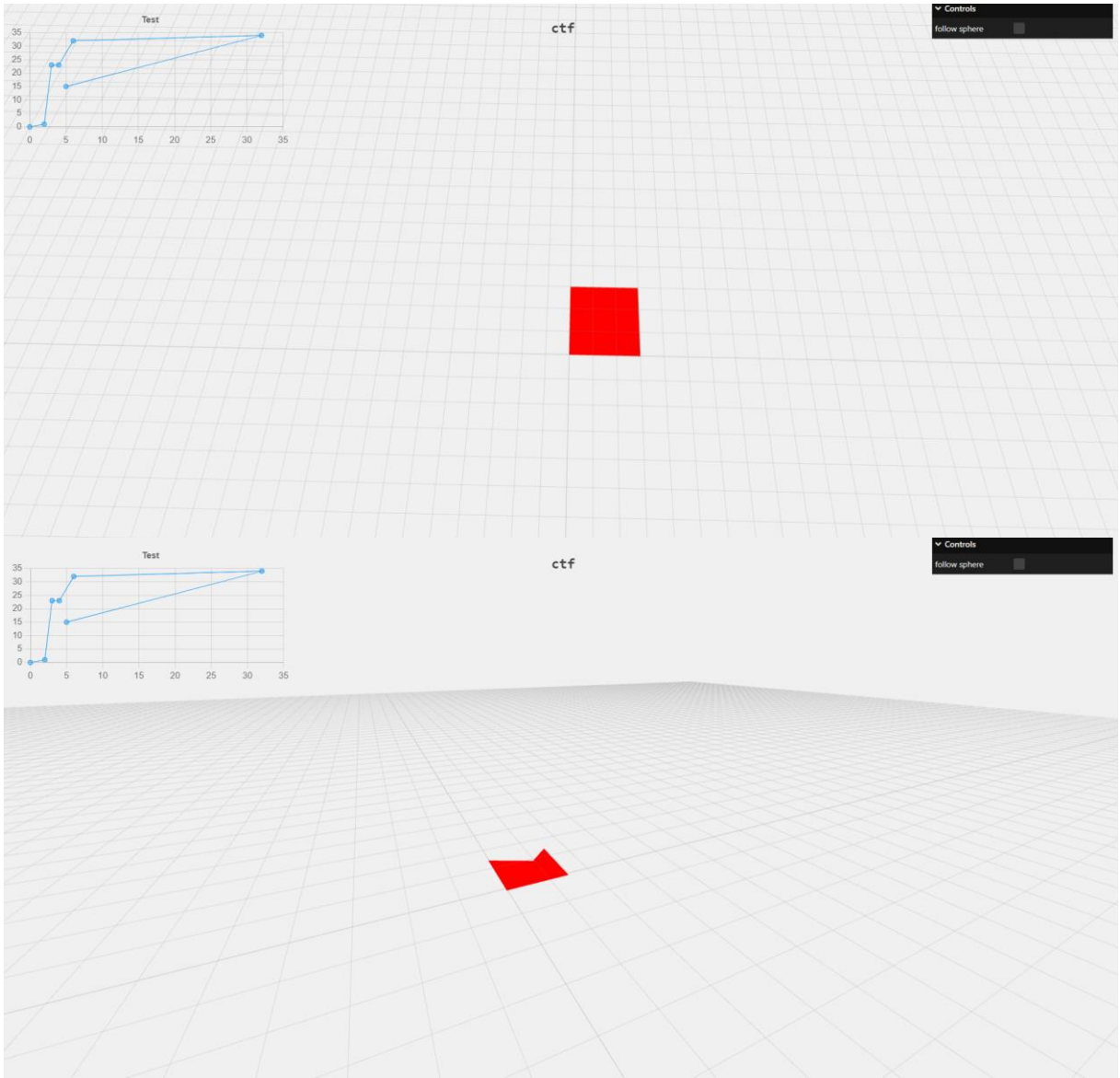
```
function addShapeGeometry(points, color) {
    const shape = new THREE.Shape();

    points.forEach(point =>
        shape.lineTo( point[0], point[1] )
        );

    const geometry = new THREE.ShapeGeometry( shape );
    const material = new THREE.MeshBasicMaterial( { color: color } );
    const mesh = new THREE.Mesh( geometry, material ) ;
    mesh.rotation.set(-Math.PI/2, 0, 0);
    mesh.position.set(0, -200, 0);
    return mesh;
}
```

Explication :

Ici nous pouvons voire que mon code n'a plus de « for » ni de « if » en effet j'ai pu le remplacer par une fonction de la bibliothèque THREE.JS qui a le même principe forEach.

Sur le tableau "points" en utilisant la méthode "lineTo" de l'objet Shape pour définir un chemin (path) à travers les points fournis. La forme ainsi créée est stockée dans une variable "geometry" qui est utilisée pour créer un nouveau maillage (mesh) en utilisant la classe THREE.Mesh. La couleur pour ce maillage est spécifiée en utilisant le paramètre "color" pour que l'utilisateur coté python puisse choisir la couleur. Puis pour finir j'utilise mesh.rotation.set(-math.PI/2,0,0) pour faire une rotation de 90° et régler le problème pour que la figure soit toujours horizontale. Puis j'utilise la même chose mais avec la position et je la mets en -200 sur l'axe Y pour pouvoir faire en sorte qu'il soit bien visualisé vers le sol.

Résultat :



Cela marche nous pouvons désormais faire différente figure a la bonne position et la bonne rotation.

AddCross :

Je vais donc désormais devoir afficher une croix. Mais ça ne va pas être aussi simple. En effet malheureusement Three.js intègre pas de croix précise donc je vais devoir en faire une.

Je pense donc à faire deux box qui se croise dans une rotation et une position précise mais pour ça je vais devoir utiliser une fonction qui se nomme group.Add.

Elle va me servir à pourvoir faire un groupe de plusieurs figures avec lesquelles vais pouvoir les faires croiser en précisant la rotation et la position.

Code :

Avant d'explique le code Voici sur Viz ce les actions que je mets :

```
def drawCross(self, position, rotation, size, color, thickness=5):
    return self.send({
        'action': 'cross',
        'position': position,
        'rotation': rotation,
        'size': size,
        'thickness': thickness,
        'color': color
})
```

En effet dans le cahier des charges il est précisé que je vais devoir faire une rotation et de mettre un paramétré de taille et de couleur. Mais nous pouvons voir que j'ai décidé de mettre la thickness a 5 car il lui faut donner d'abord une épaisseur pour pouvoir la modifier par la suite.

Désormais je vais vous montrer la partie test.py :

```
if __name__ == '__main__':
    plugin = VizMock.VizMock()
    plugin.removeAll()
    plugin.drawCross((20, 0, 0), (0, 0, 0), 30,'#003300')
    #plugin.setOffset(100, 100, 100)
    #plugin.scale(0.5, 0.19, 5)
    #plugin.setSceneFactor(0.1)
```

Difficulté :

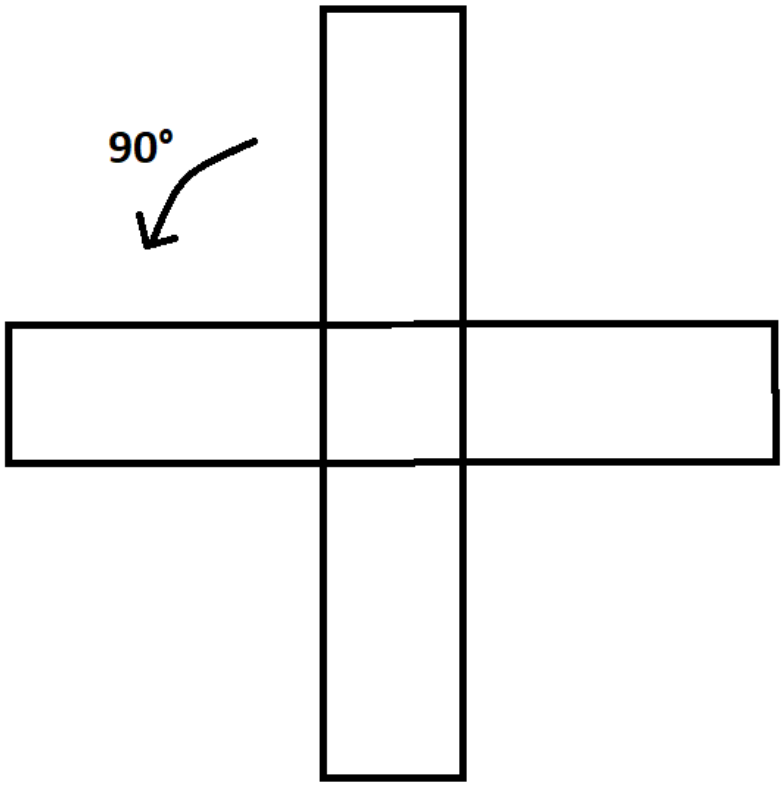
Les difficultés ici ont été de devoir faire en sorte de pouvoir avoir une croix sans même utiliser de fonction déjà faite en effet comme dit précédemment je vais devoir trouver une solution pour pouvoir faire une croix tout en pouvant modifier toutes les caractéristiques voulu. J'ai pu réfléchir à différente solution comme faire deux flèches qui se croise, mais ici la difficulté a eu été de devoir gérer pour les deux flèches les caractéristiques en même temps. Mais j'ai pu trouver une solution qui va permettre de pouvoir gérer deux figures en même temps, en effet tout d'abord j'ai décidé d'utiliser des box pour pouvoir faire une croix en les mettant dans une positions et rotation précisent et d'utiliser la fonction group.add pour pouvoir faire un groupe de figure et de faire en sorte de considérer deux figure comme si ça n'aurait été qu'une et de pouvoir les traiter en même temps.

Code :

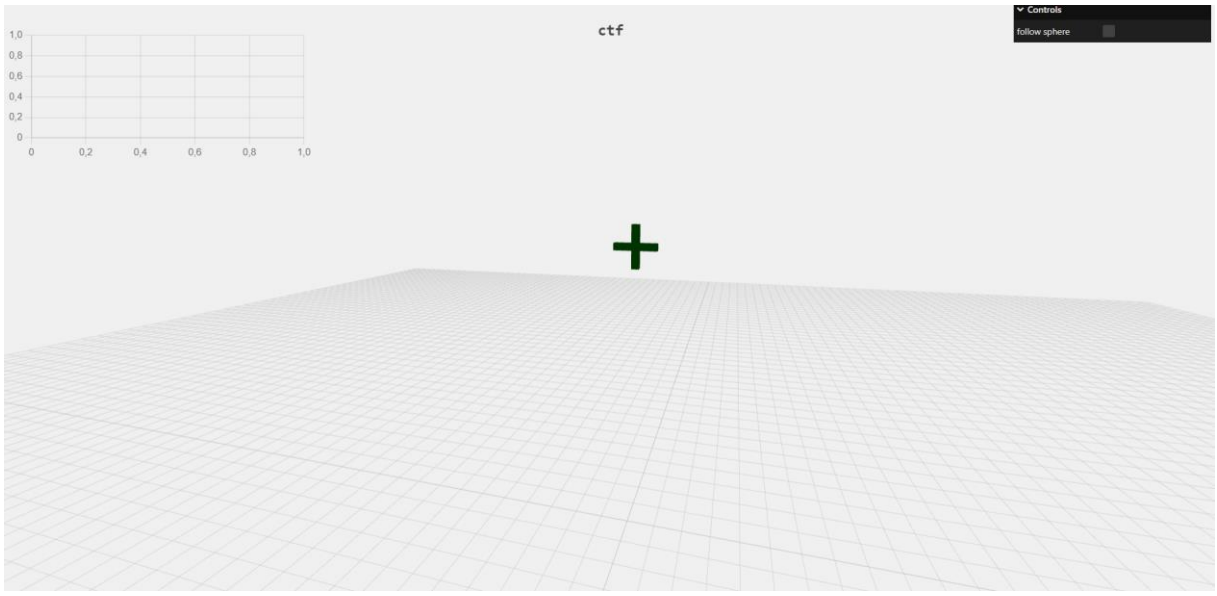
```
function addCross(position, rotation, size, color, thickness){
    const geometry = new THREE.BoxGeometry( 5, thickness, size);
    const material = new THREE.MeshBasicMaterial( {color: color } );

    const cubeA = new THREE.Mesh( geometry, material );
    cubeA.position.set( 0, 0, 0 );
    cubeA.rotation.set(-Math.PI/2, 0, 0);

    const cubeB = new THREE.Mesh( geometry, material );
    cubeB.position.set( 0, 0, 0 );

    const group = new THREE.Group();
    group.add( cubeA );
    group.add( cubeB );
    group.position.x = position[0];
    group.position.y = position[1];
    group.position.z = position[2];
    group.rotation.x = rotation[0];
    group.rotation.y = rotation[1];
    group.rotation.z = rotation[2];

    return group; //cubeA,cubeB
}
```

Ici nous pouvons voir que je crée deux box grâce à THREE.BoxGeometry. Dont l'un s'appellera CubeA et aura donc une position en x=0 y=0 mais avec une rotation de 180/2 = 90°. Et un autre CubeB aura uniquement une position comme cube A, en effet on veut qu'il soit les deux au même endroits pour qu'il puisse se croiser et former une croix. Puis on ne choisit pas de rotation pour cubeB elle sera déjà mis de base.

Voici le résultat souhaité :



Résultat :



Nous pouvons voir que nous avons exactement les résultats souhaiter avec les caractéristiques demandées.

CylindreOffset :

Explication :

Dans la bibliothèque THREE.JS il y existe différentes figures déjà faite dont une qui est un cylindre mais comme écrit dans le cahier des charges on ne me demande pas un simple cylindre on me demande de faire un cylindre qui a pour but de pouvoir être modifier en « haut » et en « bas ». En effet le cylindre donner par THREE.JS a la possibilité d'être modifier en taille mais celui-ci est modifier en fonction de la taille écrite je m'explique. Lorsque nous donnons une taille à notre cylindre il divise cette somme et la met de chaque coter de l'endroit où le cylindre est placer donc la position initiale du cylindre. Mais dans mon cahier des charges on me demande de faire un cylindre avec une position mais je dois pouvoir changer le coter bas du cylindre et celui du haut sans devoir toucher la position.

Maintenant que nous avons compris notre problématique nous allons maintenant devoir trouver une solution pour pouvoir faire ce que l'on nous demande.

Je suis donc au début parti sur ça :

```
function addCylinder3d(position, topHeight, bottomHeight,  color) {
    const height = topHeight + bottomHeight;
    const y_offset_position = height + topHeight;
   const geometry = new THREE.CylinderGeometry(topHeight, bottomHeight, height, 32);
    const material = new THREE.MeshBasicMaterial({ color: color });
```

```
    const object = new THREE.Mesh(geometry, material);

    object.position.x = position[0];
    object.position.y = y_offset_position[1];
    object.position.z = position[2];
    object.rotation.x = rotation[0];
    object.rotation.y = rotation[1];
    object.rotation.z = rotation[2];

    return object;
}
```

Nous pouvons voir que je fais apparaitre un simple cylindre. Mais je décide de créer deux actions qui se nomme TopHeight et BottomHeight. Je décide de les faires s'additionner pour récupérer la valeur et mettre dans la size du Cylindre pour qu'il est les valeurs demander par l'utilisateur. Mais par la suite je me rends compte que les résultats demander n'est pas celui que j'ai obtenu car on me demande d'avoir un cylindre qui est modifiable en haut et en bas.

Donc voici ce que j'avais mis dans mon appel d'action dans ma partie python :

```
def drawCylinder3d(self, postion, TopHeight, BottomHeight, color):
        return self.send({
            'action': 'cylander3d',
            'postion': postion,
            'TopHeight ': TopHeight,
            'BottomHeight':BottomHeight,
            'color': color
     })
```

Donc ici la difficulté été de devoir faire en sorte de récupérer les valeurs de l'utilisateur et de les mettre dans mon cylindre et de faire en sorte qu'il est possible de modifier la taille manuellement.

J'ai donc pensé à ceci :

```
function addCylinder3d(position, topHeight, bottomHeight,  color) {
    const height = topHeight + bottomHeight;
    const geometry = new THREE.CylinderGeometry(topHeight, bottomHeight, height, 32);
    const material = new THREE.MeshBasicMaterial({ color: color });
    const object = new THREE.Mesh(geometry, material);

    object.position.x = position[0];
    object.position.y = position[1] - topHeight;
    object.position.z = position[2];
```

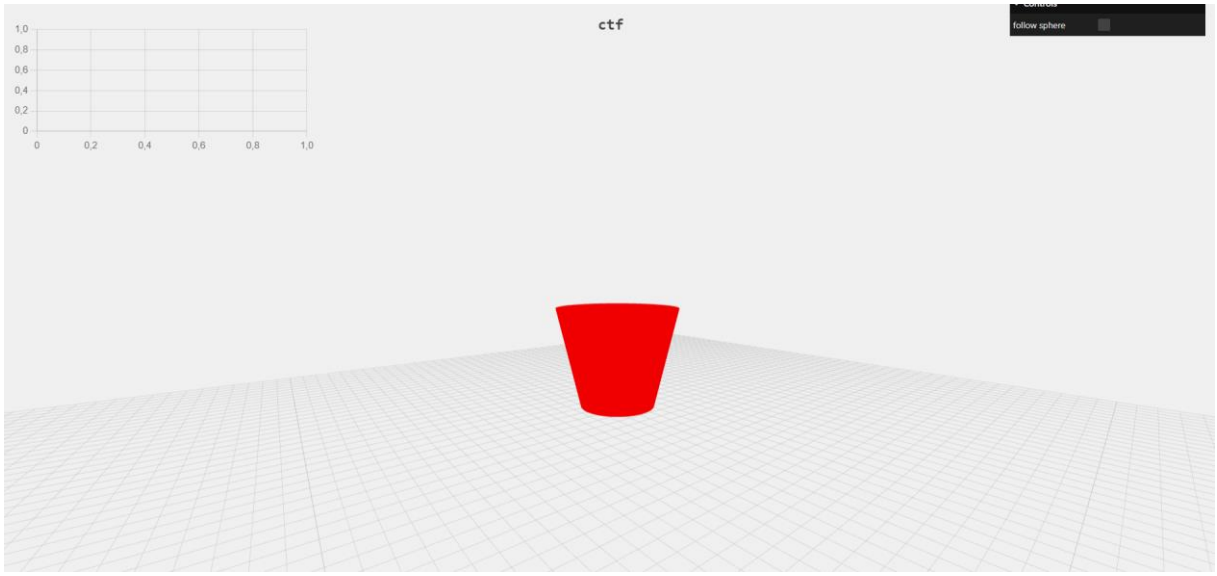
```
    return object;
}
```

Ici noud pouvons voir que je récupère les valeurs TopHeight et BottomHeight de l'utilisateur et que je les additionnes pour pouvoir modifier la tailles de mon cylindre de base en effet car la fonction cylindres marchent ainsi :

```
CylinderGeometry(radiusTop : Float, radiusBottom : Float, height : Float,
```

Donc apres avoir placer mes valeurs dans mon cylindre je décide de récuperer la valeur du TopHeight et de la soustraire a ma posistion Y, Ainsi nous pourront avoir notre cylindre correctement placer dans l'espace 3D avec la possibilé de modifier ca valeur en haut en bas.

Voici le resultat :



Nous avons donc notre cylindre qui peut être modifier en haut et en bas par l'utilisateur dans la partie python.

Par la suite je décide de rajouter l'action qui est l'opacité donc je décide de mettre mon action dans Viz :

```
def drawCylinder3d(self, position, TopHeight, BottomHeight, color, opacity=0):
    return self.send({
        'action': 'cylinder3d',
        'position': position,
        'TopHeight': TopHeight,
        'BottomHeight':BottomHeight,
        'color': color,
        'opacity': opacity
})
```

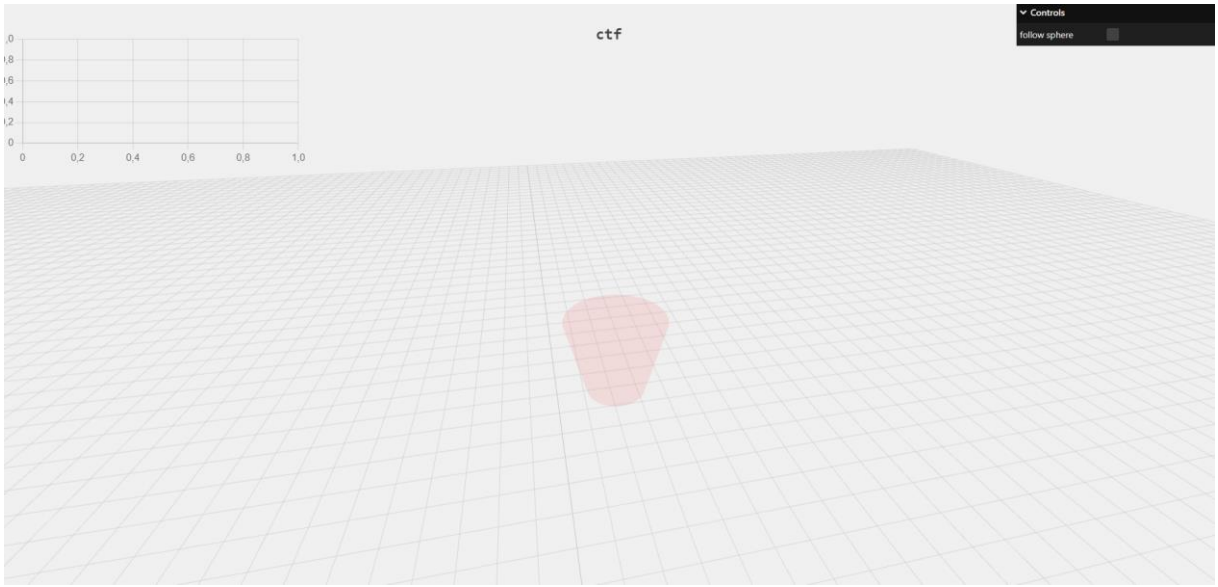
Je rajoute donc l'opaciter dans mon code :

```
function addCylinder3d(position, TopHeight, BottomHeight,  color) {
    const height = TopHeight + BottomHeight;
    const geometry = new THREE.CylinderGeometry(TopHeight, BottomHeight, height,
32);
    const material = new THREE.MeshPhongMaterial({ color: color, opacity: 0.2,
transparent: true });
    const object = new THREE.Mesh(geometry, material);


    object.position.x = position[0];
    object.position.y = position[1] - TopHeight;
    object.position.z = position[2];
    return object;
}
```

Ici nous pouvons voir qu'il suffit de rajouter l'opaciter dans la fonction MeshPhongMaterial qui a pour but de modifier la texture de la figure.

Donc plus l'opaciter est proche de 1 plus notre cylindre sera opaque, inversement si plus l'opaciter se raproche de 0 plus il sera transparent.

Voici le resultat avec l'action de l'opaciter :



Billboard:

Explication :

Pour finir j'ai eu comme objectif de faire un billboard. On ma donné comme contrainte de devoir faire apparaitre un affichage en 2D dans un espace 3D. J'ai donc reflechis et chercher dans la bibliothéques JavaScript et j'ai donc trouver la possibilité de faire afficher une phrase en 2D dans un espace 3D.

```
function billboard(text) {
    const canvas = document.createElement("canvas");
    const context = canvas.getContext("2d");
    context.font = "bold 48px Verdana";
    const textWidth = context.measureText(text).width;
    canvas.width = textWidth;
    canvas.height = 64;
    context.font = "bold 48px Verdana";
    context.fillStyle = "white";
    context.fillText(text, 0, 48);

    const texture = new THREE.CanvasTexture(canvas);
    const material = new THREE.SpriteMaterial({map: texture});
    const billboard = new THREE.Sprite(material);



    billboard.scale.set(50, 50, 1); // la taille de mon billboard

    return billboard;
}
```

Ce code définit une fonction nommée "billboard" qui prend un argument "text". La fonction crée ensuite un élément canvas dans le document HTML et récupère son contexte de dessin en 2D.

Ensuite, le code détermine la largeur du texte en utilisant la méthode "measureText" du contexte de dessin, puis définit la largeur du canvas en conséquence et fixe également sa hauteur à 64 pixels.

La fonction définit ensuite une police de caractères en utilisant "context.font" et dessine le texte sur le canvas en utilisant "context.fillText". La couleur de remplissage est fixée sur blanc à l'aide de "context.fillStyle".

La fonction crée ensuite une texture THREE.CanvasTexture à partir du canvas et l'utilise pour créer un matériau THREE.SpriteMaterial. Un objet THREE.Sprite est créé à partir de ce matériau, puis sa taille est fixée à 50x50 et sa profondeur est fixée à 1.

Enfin, la fonction renvoie l'objet THREE.Sprite résultant.

Dans l'ensemble, cette fonction crée un sprite qui affiche un texte spécifié sous forme de texture dans une scène 3D.

```
if __name__ == '__main__':
    plugin = VizMock.VizMock()
    plugin.removeAll()
    plugin.billboard("hello")
    #plugin.drawCross((20, 0, 0), (0, 0, 0), 30,'#003300')
    #plugin.setOffset(100, 100, 100)
    #plugin.scale(0.5, 0.19, 5)
    #plugin.setSceneFactor(0.1)
    # print('test')
    # print(plugin)
    #cube = plugin.drawCube((300, 100, 10), (0.3, 0, 0), 50, '#003300')
```

Ici nous pouvons voir je vais faire afficher une fonction qui se nomme « hello »

Voici le code que j'ai pu implementer dans Viz :

```
def billboard(self, text):
    return self.send({
        'action':'billboard',
        'text': text
})
```

Nous allons donc désormais passer à la conclusion.

Conclusion :

Au cours des neuf semaines passées dans l'entreprise de Thales, j'ai acquis une expérience précieuse et des connaissances pratiques dans le domaine de la programmation. J'ai eu la chance de travailler sur plusieurs projets intéressants, en utilisant des technologies variées, ce qui m'a permis de renforcer mes compétences en programmation et de découvrir de nouvelles méthodes de travail.

Au fil des semaines, j'ai appris à travailler en autonomie et à collaborer avec des collègues ayant des expériences et des compétences différentes des miennes. J'ai également été initié à l'utilisation d'outils tels que Git, ce qui a grandement facilité la gestion de projet.

En outre, j'ai eu la chance de travailler avec des clients et de développer mes compétences en communication et en gestion de projet. Cela m'a permis de mieux comprendre les besoins des clients et de leur offrir des solutions adaptées.

En somme, ce stage a été une expérience très enrichissante, tant sur le plan professionnel que personnel. J'ai acquis de nouvelles compétences, renforcé mes connaissances en programmation et en entreprise dans un environnement professionnel stimulant. Je suis reconnaissant envers Thales et les personnes avec lesquelles j'ai travaillé pour cette opportunité de stage enrichissante.