Visualisation de données

Créer un nouveau projet (ce référer aux précédents TP)

Le nommer “VisuGlobe”

Depuis le cadre”Hierarchy” créer une sphere, renommer en “Globe” et s’assurer que sa position est (0,0,0)

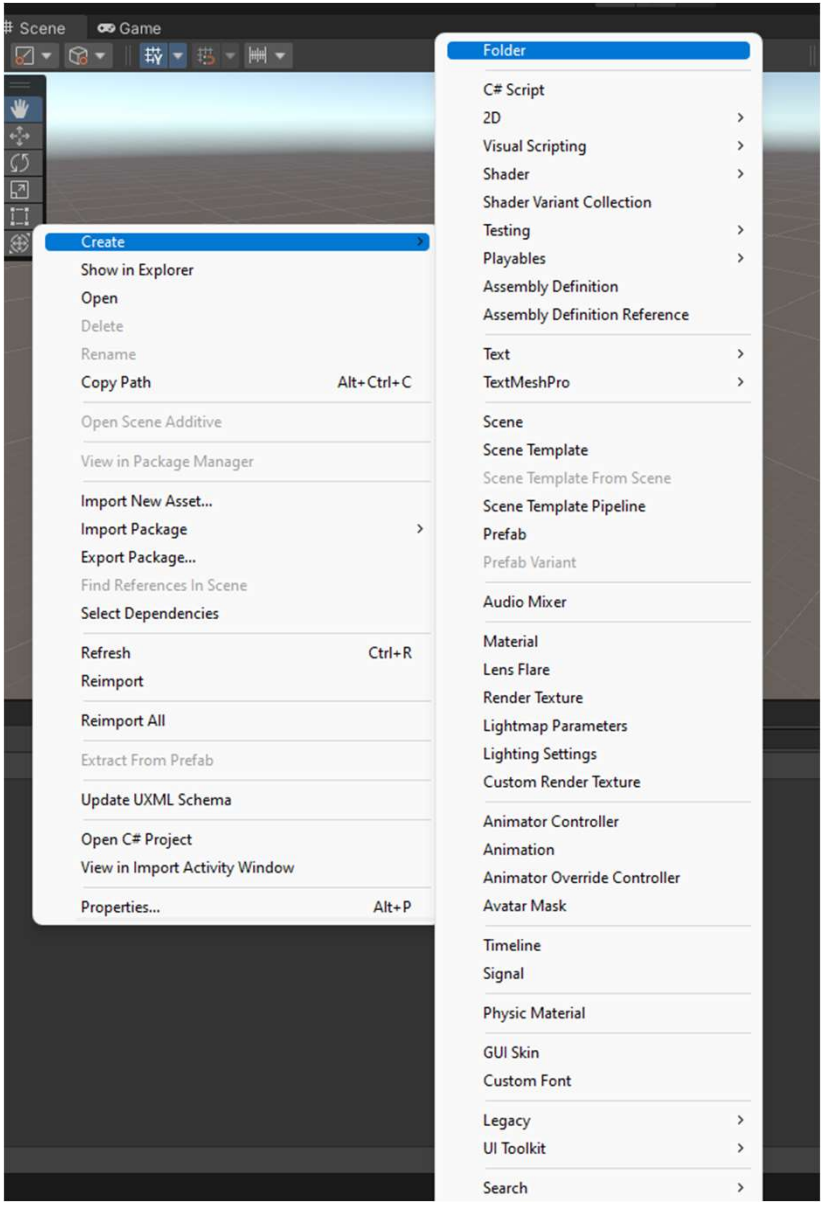
Visualisation de données

Depuis le cadre Asset, créer un dossier “Materials” (click droit Create > Folder)

Créer aussi un dossier “Textures”

Entrer dans “Materials” et créer un nouveau Material (click droit Create > Material)

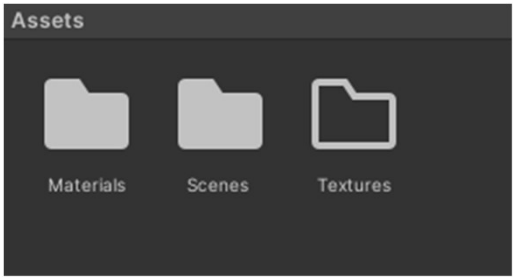
Le renommer “MaterialTerre”

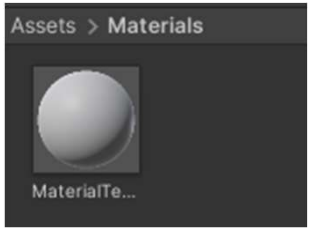


Visualisation de données

Le contenu de “Assets” devrait ressembler à çà

Et le contenu de “Assets/Materials” à çà





Visualisation de données

Télécharger les textures utilisées dans ce TP sur le lien suivant :

https://www.solarsystemscope.com/textures/

Choisir 8k ou 2k en function de la connexion internet et de la puissance de la machine

Télécharger

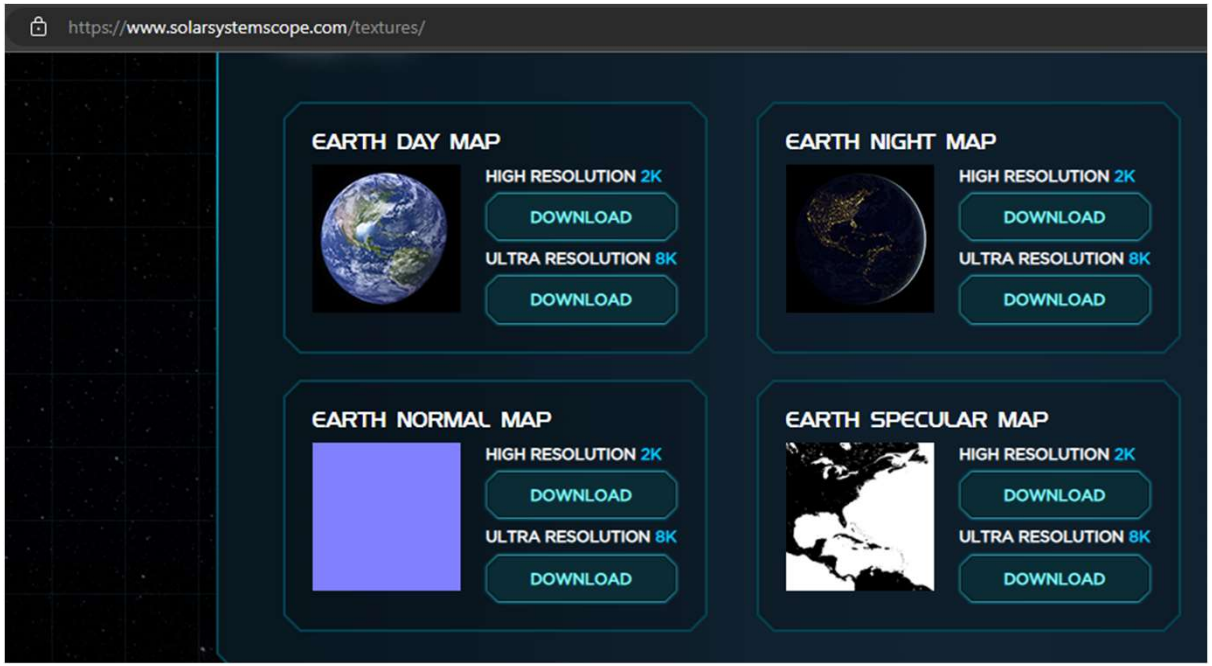
“Earth Day map”,

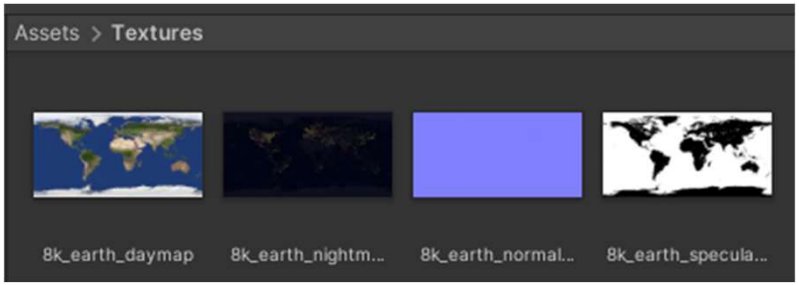
“Earth Night map”,

“Earth Normal map”,

“Earth Specular map”

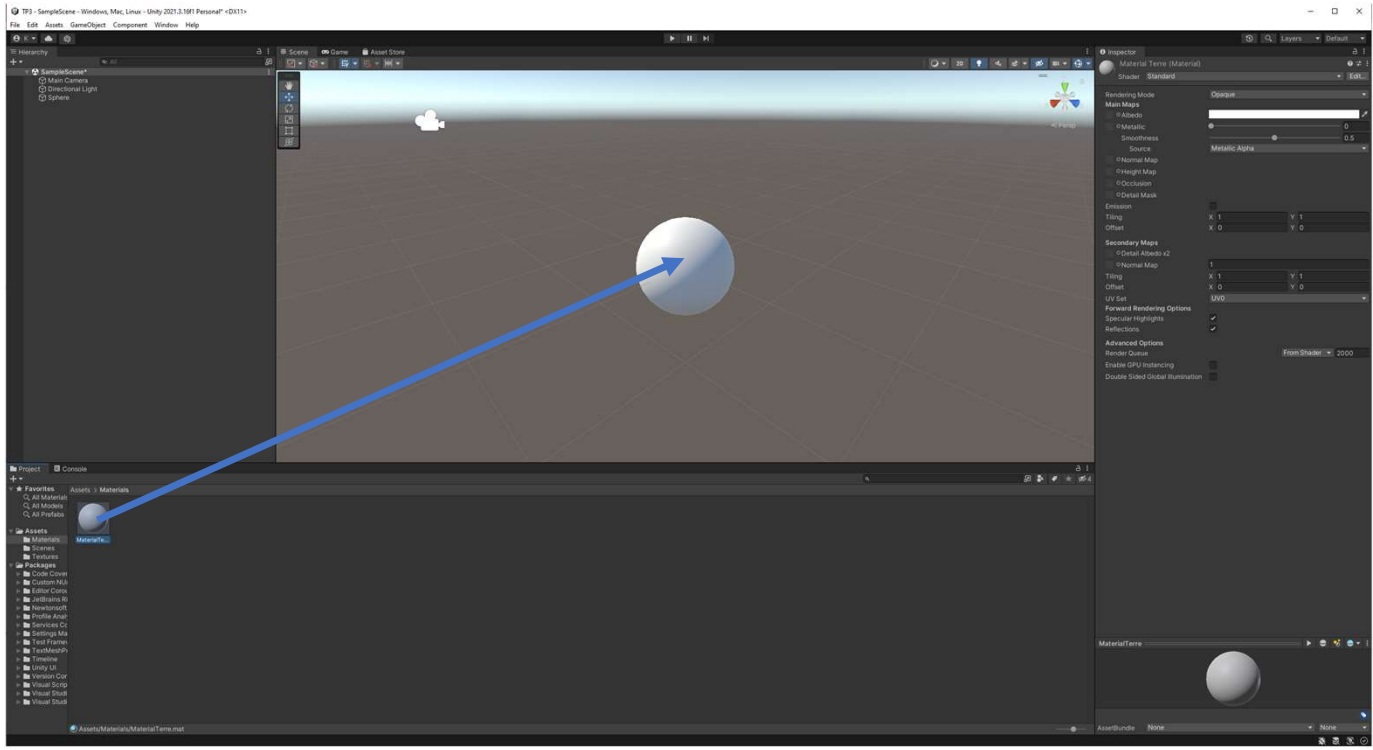
Et les mettre dans le dossier “Textures” du projet





Visualisation de données

Assigner “MaterialTerre” à la sphere en glissant le materiel sur l’objet dans la scène



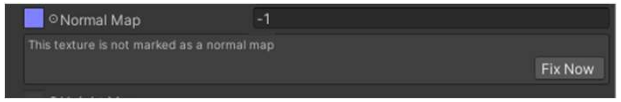
Visualisation de données

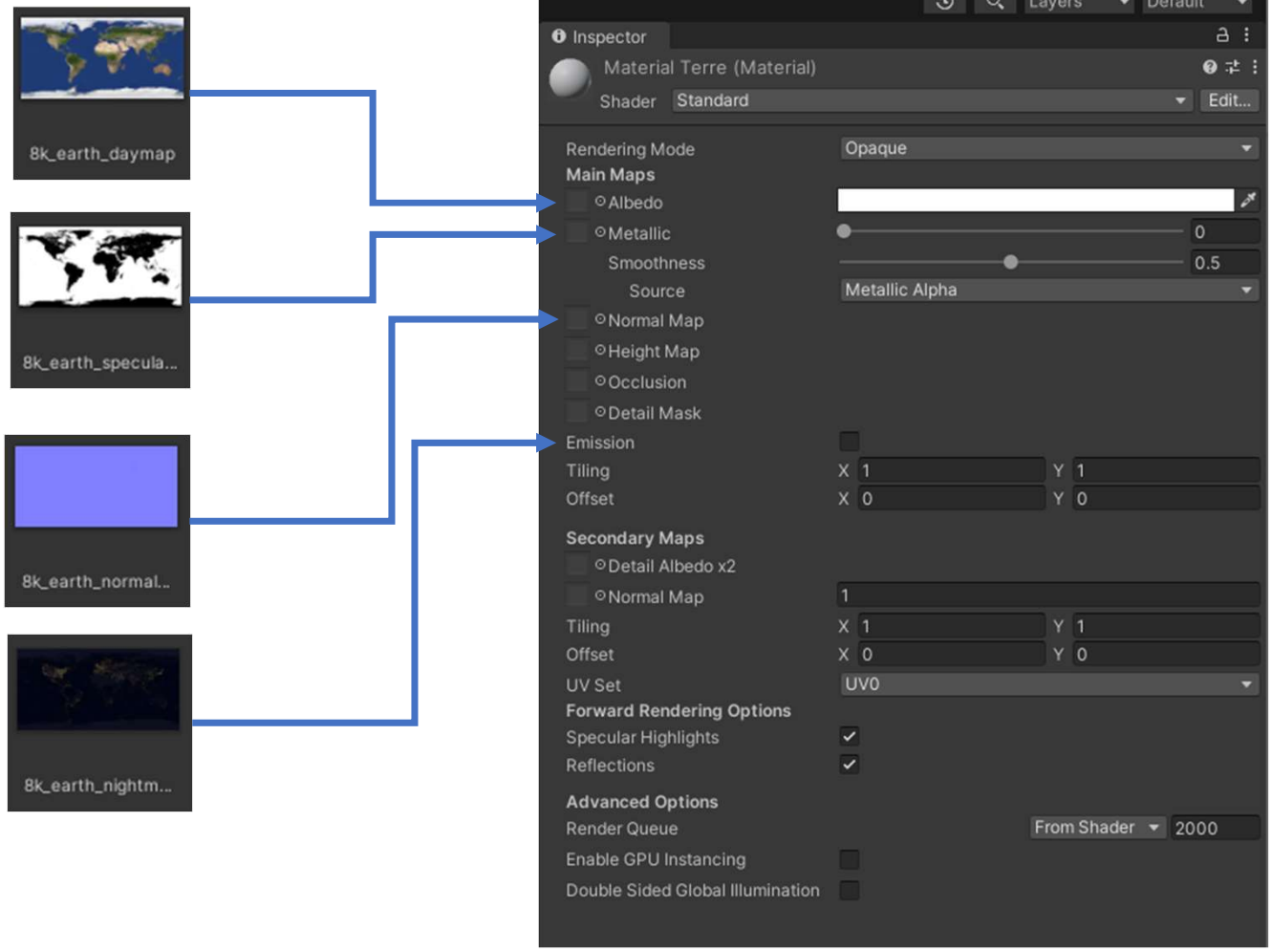
Cliquer sur “MaterielTerre” et assigner les textures à leur canaux respectifs

Faire bien attention :

Avant de pouvoir utilizer le canal “Emission” il faut l’active en cochant la case à côté.

La première fois que la texture des normales est appliquée un message d’avertissement est susceptible d’apparaître. Cliquer sur “Fix Now” le cas écheant. Il faut aussi remplacer 1 par -1 (les valeur des normales sont inverse)





Visualisation de données

Dans “Assets” créer un repertoire “Scripts”

Dans “Hierarchy” sélectionner la camera et ajouter un nouveau composant “add new component > script”

Renommer le script “CameraOrbite”

Déplacer le script dans le repertoire “Scripts”

Déclarer les champs public suivants :

“cible” l’objet que la camera va viser

“anlgesParSeconde” la vitesse à laquelle la camera va tourner

```
public GameObject cible;


public float anglesParSeconde = 45;
```

Visualisation de données

Modifier Start pour initialiser la position et la rotation de la camera au démarrage

Modifier Update pour que la camera tourne autour de la cible (le globe) à la Vitesse anglesParSeconde.

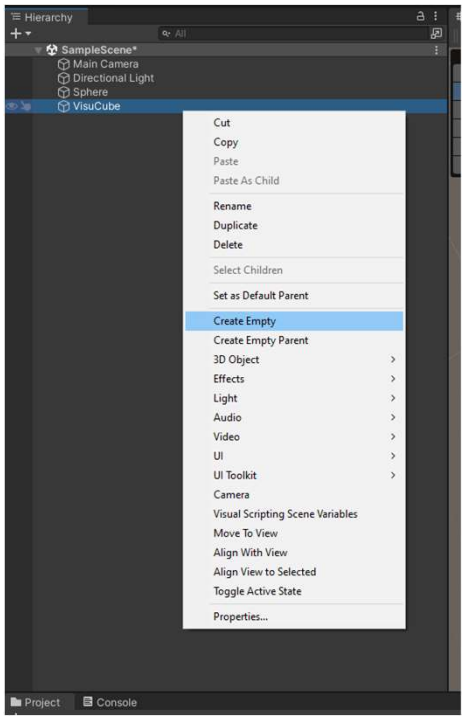
Après avoir assigné “Globe” au champs “Cible” du script “CameraOrbite”, en lançant le jeu, la fenêtre “Game” devrait afficher un globe tournant.

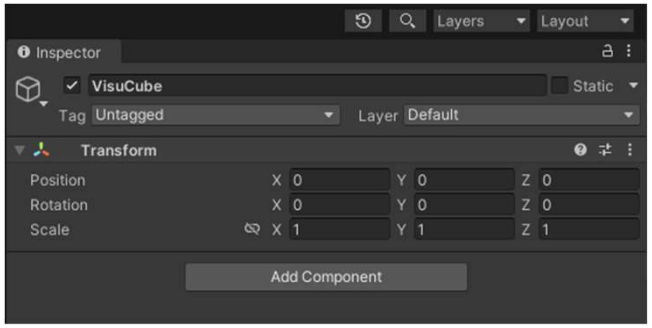
```
// Start is called before the first frame update
    void Start()
    {
        this.transform.transform.position = new Vector3(2, 0, 0);
        this.transform.transform.rotation = Quaternion.Euler(0,-90f,0);
    }

    // Update is called once per frame
    void Update()
    {
        transform.RotateAround(cible.transform.position, Vector3.up,
anglesParSeconde * Time.deltaTime);
    }
```

Visualisation de données

Depuis hierarchy créer un Empty et le nommer “VisuCube”

Vérifier dans l’Inspector que le transform est bien tel qu’illustré





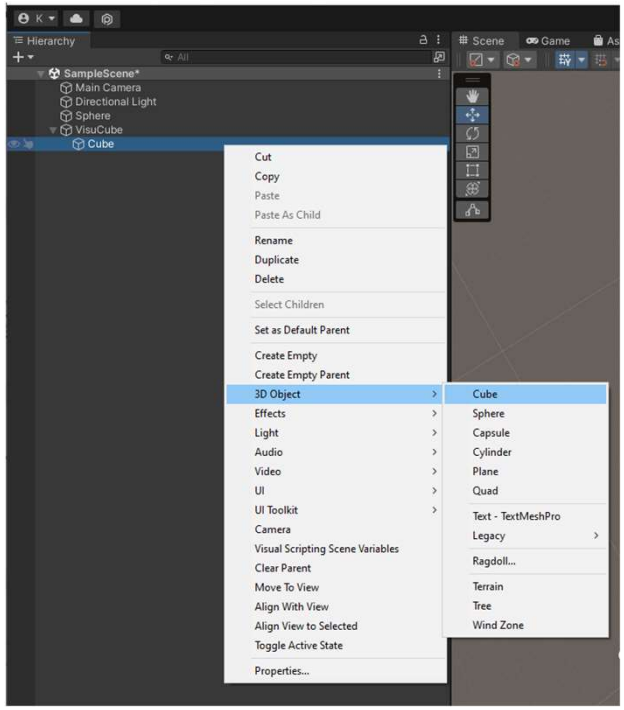
Visualisation de données

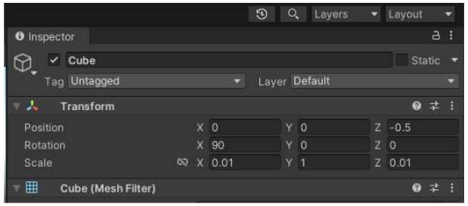
Depuis hierarchy créer un cube parenté à “VisuCube”

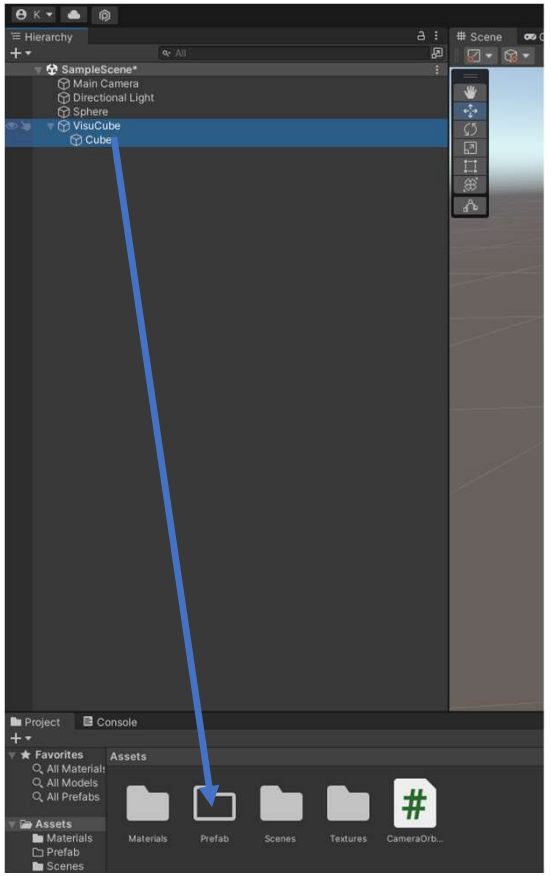
Changer les valeurs de “transform”

Créer un dossier “Prefab” dans “Assets” et déplacer “VisuCube” dedans.

“Visucube” devient un prefab qui peut être instancié sur demande

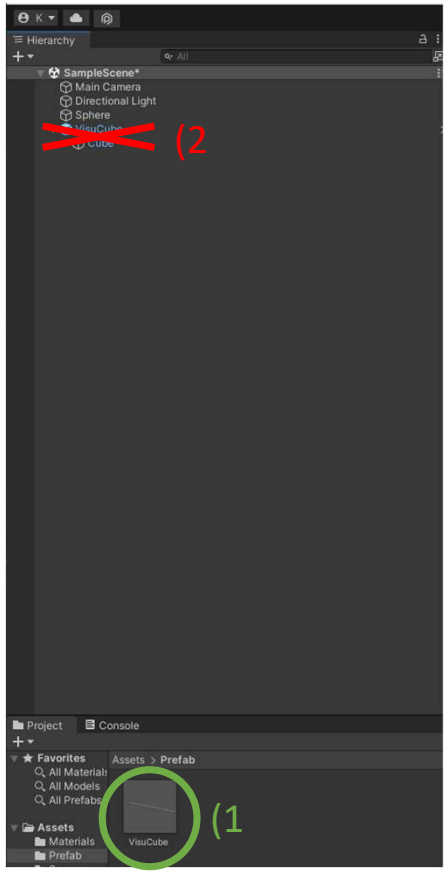






Visualisation de données

Une fois le prefab fabriqué (1), il est possible d’effacer VisuCube (2) de la scene



Visualisation de données

Dans le repertoire “Scripts” créer un script et l’appeler “VisuData”.

Effacer “Start” et “Update”, ils ne sont pas utile ici

Déclarer trois champs :

“Cible” sert à orienter le cube de visualization

“VisuCube” pointe vers le prefab “VisuCube” précédemment construit

“multiplier” sert à ajuster la taille de “VisuCube” en function de la taille de l’information affichée

```
public Transform Cible;
public GameObject VisuCube;
public float multiplier;
```

Visualisation de données

La méthode latloncart sert à convertir des coordonnées de latitude longitude en coordonnées cartésiennes x,y,z

```
Vector3 latloncart(float lat, float lon)
    {
        Vector3 pos;
        float x = 0.5f * Mathf.Cos(lon) * Mathf.Cos(lat);

        float y = 0.5f * Mathf.Cos(lon) * Mathf.Sin(lat);

        float z = 0.5f * Mathf.Sin(lon);

        pos.x = 0.5f * Mathf.Cos((lon) * Mathf.Deg2Rad) * Mathf.Cos(lat * Mathf.Deg2Rad);
        pos.y = 0.5f * Mathf.Sin(lat * Mathf.Deg2Rad);
        pos.z = 0.5f * Mathf.Sin((lon) * Mathf.Deg2Rad) * Mathf.Cos(lat * Mathf.Deg2Rad);

        return pos;
    }
```

Visualisation de données

La méthode VisualCube instancie un prefab “VisualCube” pour une position donnée en latitude longitude (conversion via latloncart) et change l’echelle du cube en fonction d’une valeur “val” donnée.

```
public void VisualCube (float lat, float lon, float val, float multiplier)
    {
        GameObject cube = GameObject.Instantiate(VisuCube);
        Vector3 pos;

        pos = latloncart(lat, lon);
        cube.transform.position = new Vector3(pos.x, pos.y, pos.z);
        cube.transform.LookAt(Cible, Vector3.back);

        Vector3 echelle = cube.transform.localScale;
        echelle.z = val*multiplier;
        cube.transform.localScale = echelle;
    }
```

Visualisation de données

En préparation de la lecture du fichier worldcities.json, nous créons deux classes sérialisables :

“City” et “Cities”.

Dans le répertoire script, créer City.cs et Cities.cs.

Effacer leur contenu par défaut auto-généré par Unity et remplacer par ces lignes.

Cities.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class Cities
{
    public City[] cities;
}
```

City.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class City
{
    public string city;
    public float lat;
    public float lng;
    public int population;
}
```

Visualisation de données

Créer la classe LectureData qui va utiliser les classes sérialisées Cities et City ainsi que la classe VisuData pour lire un fichier json

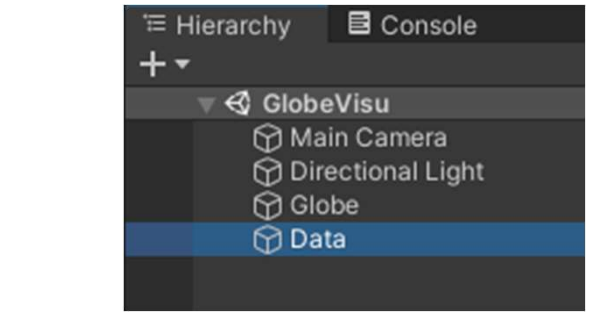
```
using System.IO;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;


public class LectureData : MonoBehaviour
{
    public TextAsset jsonFile;
    public VisuData Visualize;

    void Start()
    {
        Cities employeesInJson = JsonUtility.FromJson<Cities>(jsonFile.text);

        foreach (City cities in employeesInJson.cities)
        {
            Visualize.VisualCube(
                cities.lat,
                cities.lng,
                cities.population,
                Visualize.multiplier
                );
        }
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

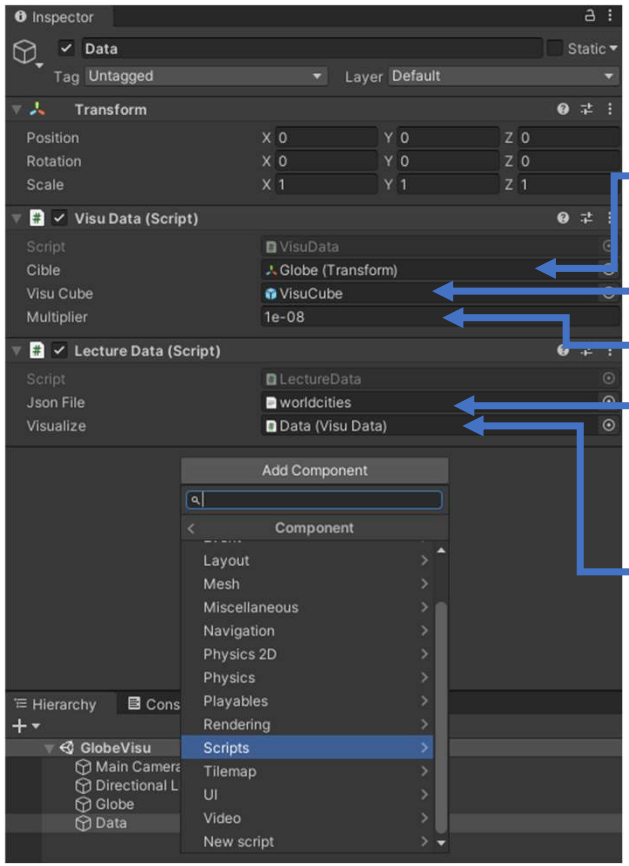
Visualisation de données

Dans “Hierarchy” créer un “Empty” nommé “Data”

Cliquer sur “Add Component” et ajouter les scripts “LectureData” et “VisuData”

Récuperer le fichier json sur le moodle du cours dans le repertoire “Textures et fichiers json”





Le globe qui sert de reference pour orienter les instances de VisuCube

Le prefab VisuCube qui sera instancié

Le multiplier réglé pour accomoder des villes dont la population dépasse la centaine de million

Le fichier json contenant les data représentées

Se réfère au composant VisuData de l’objet Data