TP 5 : Développer une application 3D sous Unity

1. Introduction

L'objectif de ce TP est de développer une application en utilisant le logiciel Unity tout en maintenant une structure de code et une conception de projet claire.

L'application à développer est une application permettant diverses visualisations du système solaire (visualisation globales, statiques de dynamiques, différents points de vue, etc.). Les données proposées sont calculées à partir de positions approximatives calculées via les formules képlériennes[1]. Les fonctions permettant d'effectuer ces calculs sont déjà implémentées dans la classe statique « PlanetData ».

2. Quelques éléments de conception

Les singletons

Une façon de structurer l'application avec Unity est de séparer les fonctionnalités dans des GameObjects spécifiques, souvent appelés « Managers ». Ces objets (souvent composés d'un simple GameObject avec un seul script) gèreront l'ensemble des données ou le comportement d'un ensemble d'éléments. Dans notre cas, nous allons implémenter un « PlanetManager » qui permettra de gérer les positions des astres, les trajectoires, et les paramètres temporels. Ceci pourrait être fait dans la classe statique « PlanetData », mais l'utilisation d'un Manager permet de rendre visible dans l'inspecteur les paramètres de l'application.

Voici un exemple pour la structure de cette classe :

[1] https://ssd.jpl.nasa.gov/planets/approx_pos.html

```
using UnityEngine;

public class PlanetManager : MonoBehaviour
{
    public static PlanetManager current;

    private void Awake()
    {
        if (current == null)
        {
            current = this;
        }
        else
        {
            Destroy(obj: this);
        }
    }
}
```

Notez la structure permettant de 1) s'assurer que l'objet est unique (Singleton), 2) rendre l'objet accessible partout via l'attribut publique.

Concernant l'attribut de date, vous pourrez utiliser la classe `UDateTime` fournie qui encapsule `DateTime` mais permet de le rendre visible via l'inspecteur.

Une première représentation

Vous allez maintenant créer les premières représentations de planètes.

Commencez par créer toutes les sphères correspondantes aux planètes, vous pouvez utiliser par exemple des textures de planètes en ligne, ou de simple sphères de couleur. Puis, créez un nouveau script, qu'on pourra nommer `SolarSystemController`, qui permettra de gérer le positionnement de ces objets.

Enfin, on créera une fonction permettant de placer toutes les planètes : `UpdatePosition(DateTime t)`

Testez le fonctionnement (par exemple en appelant cette fonction dans l'Update).

La gestion des évènements

Nous allons maintenant modifier le PlanetManager créé précédemment pour lui permettre de générer un évènement.

Par exemple, nous allons créer un évènement `OnTimeChange` qui sera appelé à chaque fois que la date considérée sera modifiée.

Voici la syntaxe à utiliser :

```
    public event Action<DateTime> OnTimeChange;
    public void TimeChanged(DateTime newTime)
    {
        OnTimeChange?.Invoke(newTime);

    }
```

Notez le mot-clé `event` et la structure permettant de passer un argument de type `DateTime`.

Ensuite, il faut appeler cet évènement à chaque fois que la date est modifiée. Une façon de faire cela est d'encapsuler le champ `date` précédemment créé dans une propriété :

```
[SerializeField]
private UDateTime date;
public UDateTime Date
{
    get => date;
    set
    {
        date = value;
        TimeChanged(value.dateTime); //Fire the event
    }

}
```

Enfin, il ne reste plus qu'à associer les fonctions qui doivent être exécutées à l'évènement. Par exemple, pour mettre à jour la position des planètes à chaque fois que la date est modifiée (dans le `SolarSystemController`) :

```
void Start()
{
    PlanetManager.current.OnTimeChange += UpdatePosition;

}
```

3. Concevoir et implémenter le reste de l'application

Vous avez maintenant tous les éléments pour implémenter une application de façon structurée (en suivant le formalisme PAC de préférence). Vous allez maintenant compléter votre application de façon à enrichir ses fonctionnalités.

Gestion des interactions

Concernant la gestion des interactions, il faudra utiliser le package Input System (comme dans les TPs Unity précédents).

Vous pourrez créer un nouvel objet « Input Action » dans votre projet, où vous définirez les actions à générer. Vous pourrez ensuite utiliser ce fichier pour toutes vos interactions (via les attributs `InputAction`, cf TP 1 de TERV).

Fonctionnalités à implémenter

Pour compléter l'application, vous ajouterez une interface utilisateur (via les GameObject « UI » disponibles dans Unity), et implémenter les fonctionnalités suivantes :

- Vue héliocentrique
    - Déplacer le point de vue de la caméra (rotation & zoom)

    - Afficher / cacher les trajectoires des planètes via l'UI.
    - Changer l'échelle d'affichage des planètes (réaliste vs adaptés pour voir les objets) via l'UI.
    - Définir la date via l'UI + commandes clavier.
    - Créer un mode dynamique permettant de voir le mouvement des planètes (en définissant la vitesse de défilement du temps via l'UI).
- Vue planète
    - Placer la caméra avec une planète au centre de la fenêtre lorsqu'on clique sur l'astre.
    - Afficher les informations de la planète
- Bonus
    - Gérer la rotation propre des planètes
    - Ajouter la voute céleste et créer une vue à la surface de la planète (pour afficher la position des constellations et des planètes vues de la surface).
    - Ajouter les satellites de planètes et leurs mouvements.

4. Rendu

Pour ce TP, il est attendu un compte rendu présentant :

- Des captures d'écran de votre programme
- Une courte présentation des fonctionnalités implémentées
- Le diagramme UML final de votre application