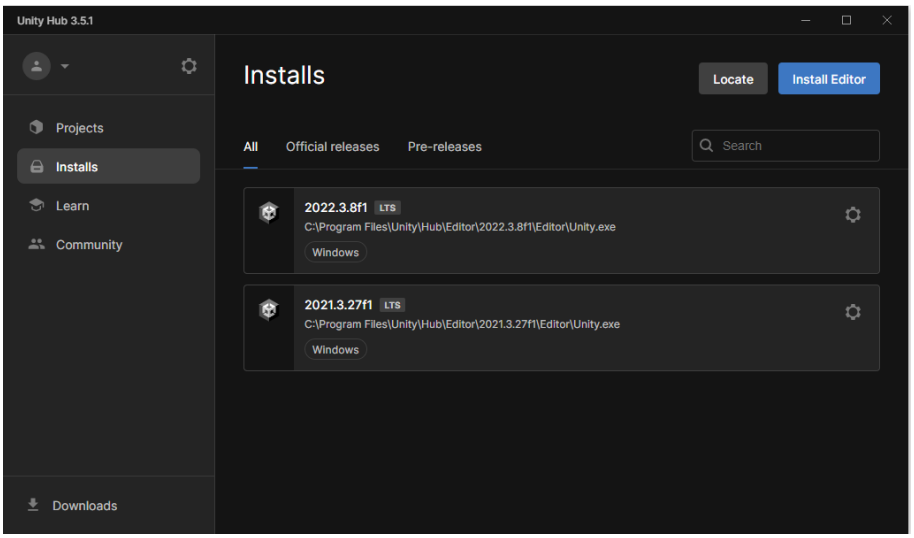
Date : 17/09/2024

Séance 3 – Classes sur Unity

Avant de commencer :

- Téléchargez « Unity Hub » : https://unity.com/download
- Utilisez le bouton « Install Editor » pour télécharger la dernière version LTS d'Unity



Objectifs :

- Apprendre à utiliser de classes prédéfinies des bibliothèques Unity
- Création de nouvelles classes sur Unity

Concepts :

Classes sur Unity : le moteur de jeu Unity propose un ensemble de classes pour faciliter la modélisation des jeux vidéo. Ces classes représentent des composants d'un jeu vidéo comme des objets 2D, des objets 3D, des effets spéciaux, de l'audio, *etc*.

- **Classe « GameObject »** : c'est la classe utilisée pour modéliser des éléments de la scène d'un jeu vidéo. Une bonne partie des éléments que nous utiliserons pour modéliser nos jeux sur Unity sont des classes de type « GameObject ». Vous trouverez sur le lien ci-dessous une description détaillée des attributs et méthodes fournis par cette classe. Lien : https://docs.unity3d.com/ScriptReference/GameObject.html
- **Classe « Transform »** : cette classe permet de stocker et manipuler des informations relatives à la position et au déplacement des objets dans la scène, comme les attributs de « position », « rotation » et « scale », en français : position, rotation et échelle.
- **Classe « MeshRender »** : cette classe offre des attributs et méthodes pour contrôler le rendu des éléments de votre jeu vidéo.

Script comportemental : code en langage C# utilisé pour ajouter des comportements (ou des fonctionnalités) à une instance d'un *GameObject*. En pratique, lorsqu'on définit un script sur Unity, on est en train de définir une nouvelle classe. Nous sommes libres pour ajouter les méthodes et les attributs que nous voulons sur ces classes.

Néanmoins, le moteur de jeu Unity propose un modèle de script (ou classe) qui nous permet de profiter des comportements prédéfinis pour les éléments du jeu.

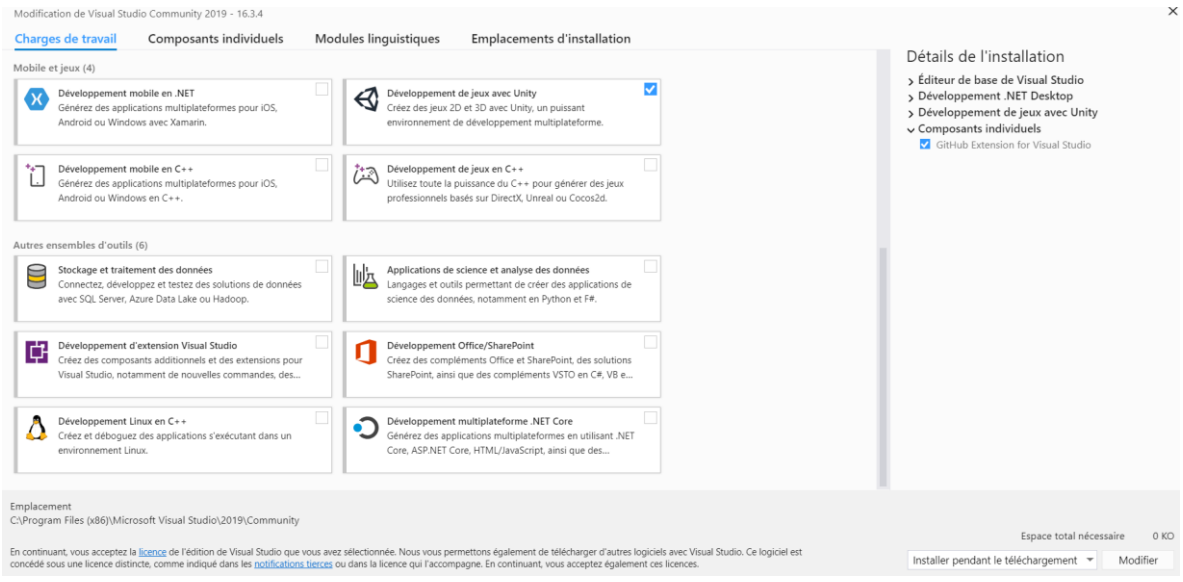
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class MonScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {    }
    // Update is called once per frame
    void Update()
    {    }
}
```

Par exemple, lorsqu'on définit les méthodes « Start » et « Update » avec les signatures proposées par le modèle, on a les comportements suivants :

- La méthode « Start » est appelée automatiquement par Unity avant l'exécution de la première image (« frame ») du jeu vidéo. Elle peut être utilisée pour activer certaines fonctionnalités des éléments d'un jeu juste avant le démarrage du jeu.
- La méthode « Update » est appelée à chaque image de votre jeu vidéo. Cette méthode est donc utile pour toutes les mises à jour ordinaires de votre jeu, comme :
    - ✓ Déplacement des objets non physiques.
    - ✓ Minuteur/chronomètre simples
    - ✓ Traitement des interactions avec l'utilisateur.

Visual Studio pour Unity

Avant de commencer les exercices, vérifiez que vous avez installé le module « Développement de jeux avec Unity » sur Visual Studio Community. Si ce n'est pas le cas, vous devez lancez « Visual Studio Installer » pour installer ce module.



Démonstration

- Création d'un projet
- Ajout d'un objet de type « GameObject » dans la scène
- Modification des composants d'un objet (« Transform », « MeshRenderer »).
- Création d'un script
- Visualisation des attributs d'un script sur « Inspector »
- Fonction « Debug.Log »

Comment ajouter un script sur un objet ?

Plusieurs options sont possibles :

a) Clic droit sur la zone de l'onglet « Asset » sous « projet ». Choisissez l'option « Create » et ensuite la sous-option « C# script ». Ensuite, sélectionnez le fichier créé et déplacez-le sur l'objet.
b) Sélectionnez l'objet à intégrer dans le script. Cliquez sur le bouton « Add Component » dans l'onglet « Inspector » et choisissez l'option « New script ».
c) Cliquez sur l'objet à intégrer dans le script et ensuite sur le menu « Component ». Sélectionnez soit l'option « script » pour créer un nouveau script ou le nom d'un script existant sur la liste.

Exercices :

1. **Utilisation des « assets »** : Maintenant, nous utiliserons des ressources préexistantes pour construire le jeu, des « assets ». Téléchargez les matériels du cours « Create with code » de Unity Learn.

Lien : https://connect-prd-cdn.unity.com/20191004/11d568c2-0b8a-4eca-9219-52d05a07eba1/Prototype%201%20-%20Direct%20Download.zip?_ga=2.261656845.1451003127.1571749928-1045493195.1570353603

- Décompressez l'archive téléchargée. Ensuite, allez sur le menu « Asset » d'Unity, et choisissez l'option « Import Package ». Chargez le fichier « Prototype-1_Starter-Files.unitypackage »
- Placez les assets « Prototype 1 » ou « Ground road » et « Veh_Ute_Red_Z » sur la scène. Ils se trouvent dans la fenêtre « Assets ». Ensuite, renommez ces assets avec des noms plus pertinents. Par exemple, voiture et environnement.
- Lancez le jeu pour valider l'exercice.

2. Cet exercice a pour objectif d'ajouter du mouvement à l'objet voiture. Le contrôle du mouvement des objets sur Unity est normalement fait à partir du composant « Transform ». Mais, comment y accéder ?

- **Instance « gameObject »** : dans tous les scripts Unity, un raccourci vers l'objet « GameObject » sur lequel le script C# est rattaché est automatiquement créé. Ce raccourci, nommé « gameObject » (attention à la première lettre en minuscule), donnes un accès direct aux attributs et méthodes de l'instance du GameObject en question, ainsi qu'aux composants de l'objet.

La construction suivante permet de récupérer l'attribut « position » de l'objet :

```
gameObject.transform.position
```

- La position des objets dans un jeu 3D sur Unity est représentée par un vecteur à trois dimensions : x, y, z. Le code suivant incrémente de « 1 » la position de l'objet sur l'axe y.

```
Vector3 vecteur = new Vector3(1,0,0);
gameObject.transform.position+= vecteur ;
```

- Ajoutez un script, nommé « déplacement », sur l'objet « voiture » et intégrez le code ci-dessus à la méthode que vous jugerez la plus pertinente pour déplacer la voiture. Testez le fonctionnement du code !
- Maintenant, essayez de modifier le code ci-dessus pour déplacer la voiture, automatiquement, dans l'axe de la route.

3. **Time.deltaTime** : vous avez peut-être remarqué que le déplacement de la voiture s'est fait trop vite ou trop lentement, selon la vitesse de votre ordinateur. Multipliez la mise à jour du vecteur « position » par « Time.deltaTime ». Cet attribut de la classe Time permet de contrôler la mise à jour de paramètres du jeu selon la vitesse de l'ordinateur. Est-ce que le déplacement vous semble plus naturel maintenant ?
4. **Réglage de la vitesse** : le mouvement de la voiture est actuellement constant. Ajoutez un attribut à votre script pour contrôler la vitesse de la voiture. Cet attribut doit être intégré au calcul de la mise à jour de la vitesse de la voiture et il doit être modifiable via « Inspector ». Testez différentes valeurs pour l'attribut « vitesse » pendant l'exécution de notre jeu pour valider votre code.
5. **Utilisation de classes** : créez un nouveau script et nommez-le « GestionVoiture ». Ce script doit contenir la classe décrite ci-dessous qui sert à calculer la consommation d'essence de la voiture et à signaler son arrêt dès que le réservoir est vide.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class GestionVoiture
{
    private double essence;

    public GestionVoiture() { //Constructeur
        essence = 10;
        Debug.Log("Essence :" + essence);
    }
    public double getEssence() {
        return essence;
    }
    public void setEssence(double valeur) {
        essence = valeur;
    }
    public bool roule(double consommation) { //à compléter }
}
```

- Complétez la méthode « roule ». Elle doit décompter la valeur du paramètre « consommation » de l'attribut « essence » et évaluer l'état du réservoir d'essence. Si

la valeur de l'attribut « essence » est supérieure à zéro, la méthode renvoie « true » et « false » dans le cas contraire (réservoir vide).

- Ajoutez une instance de la classe « GestionVoiture » au script responsable du déplacement de la voiture. Ce script doit arrêter le déplacement de la voiture dès que la méthode « roule » signale que le réservoir d'essence est vide.

Perfectionnement

6. Pour les plus expérimentés, ajoutez à votre voiture les fonctionnalités suivantes liées à la conduite de la voiture :
   - Accélérer ou décélérer la voiture en utilisant les touches Z et S.
   - Tourner à droite et à gauche en utilisant les touches Q et D.

Vous pouvez utiliser le code source ci-dessous pour capturer l'interaction du joueur avec les flèches du clavier.

```
if (Input.GetKeyUp(KeyCode.LeftArrow))    {…}
else if (Input.GetKeyUp (KeyCode.RightArrow)){…}
else if (Input.GetKeyUp (KeyCode.UpArrow))    {…}
else if (Input.GetKeyUp (KeyCode.DownArrow)) {…}
```

7. Rajoutez des véhicules contrôlés par le jeu à votre projet. Ces véhicules doivent parcourir la route en sens inverse.
8. Rajoutez des obstacles sur la route, telles que des boîtes. Utilisez le composant de type « RigidBody » aux obstacles afin de rajouter des fonctionnalités liées à la physique de collisions à votre jeu.