ATI L3 2021-22

Support de cours Unity

Cours 1 : Transform et Scripting

10/10/2019

Présentation générale

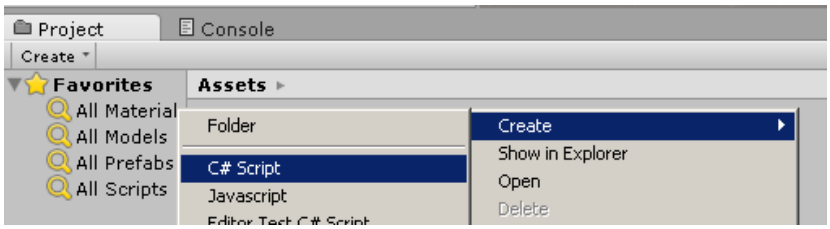
Dans ce cours nous allons voir les bases du scripting dans Unity, essentiellement présentées sous forme de mouvements basiques.

Nous verrons également les variables et opérateurs basiques que vous verrez également en C++ et Python. Vous remarquerez beaucoup de point communs entre le C# et le C++.

Points Essentiels

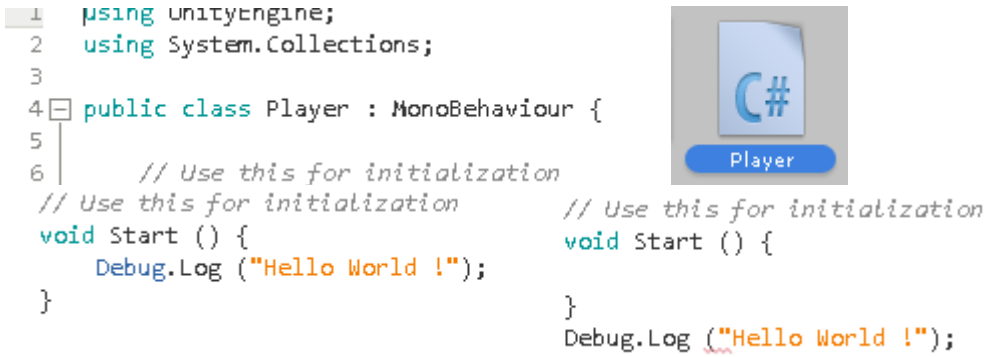
Ecriture d'un Script C#

Pour créer un script C#, faites un clic droit dans la fenêtre *Project*, puis Create > C# Script.



Lorsque vous créez un script, vous créez plus précisément une « classe ». Vous verrez plus tard dans l'année, dans mes cours comme dans les cours de Python ou C++, de quoi il s'agit précisément. La classe est visible dans les premières lignes du script.

Donnez-lui un nom simple, sans espaces, sans accents ou autres caractères spéciaux. A cause de cette histoire de classe, si vous avez appuyé sur Entrée trop tôt, ou si vous voulez renommer votre script, il faudra aussi changer son nom dans le script.

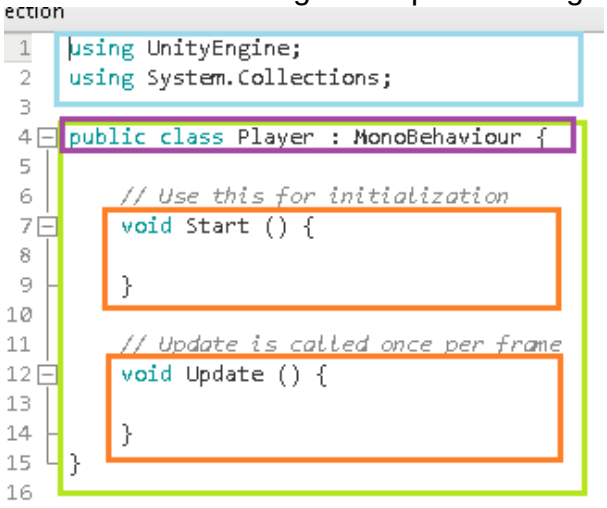


Tout comportement de votre script doit être écrit entre les accolades **{ }** de Start ou Update.

Lecture d'un script C#

Un script C# pour Unity est composé, par défaut, de la manière suivante :

A. La section « using ». La première ligne vous permettra d'employer des commandes propres à Unity (comme le Transform), la seconde des commandes basiques de C#.



B. La **première ligne** de cette partie risque de vous échapper pour le moment, mais c'est pas grave. Vous n'aurez à vous en soucier que pour renommer votre script, à l'heure actuelle. Il s'agit en réalité de la déclaration de la classe de votre script. La partie « MonoBehaviour » permet à votre script d'être accroché à un GameObject.

C'est dans ce bloc (entre les accolades) que vous pouvez déclarer vos variables. Vous trouverez également deux **blocs** Start() et Update(), chacun avec des accolades.

- Le bloc Start() sert à exécuter des lignes de code une seule fois au lancement de la scène. Ca peut être utile pour faire certains calculs avant de se lancer dans la boucle temps réel.
- Le bloc Update() sert à ce qui doit se produire en temps réel, et sera lancé intégralement à chaque frame de calcul. Ce qui est contenu dedans sera répété jusqu'à la fin du programme, ou un changement de scène, ou la destruction de l'objet contenant le script.

Conventions et syntaxe

Comme mentionné plus tôt, le nom de script doit coïncider avec le nom de la classe, et toute commande doit être écrite dans une fonction, notamment Start() ou Update().

Il y a plusieurs termes qui peuvent soit désigner une variable, soit un objet, selon si le mot commence par une majuscule ou non. Ainsi :

- Je peux écrire `transform.Translate (Vector3.up);` car « transform » désigne le component Transform de l'objet contenant le script.
- Je ne peux pas écrire  car « Transform » est un type de variable, au même titre que « float » ou « int ». On s'en sert pour stocker le component Transform d'un autre objet que celui contenant le script.

Et il en est de même pour « gameObject » qui désigne le GameObject auquel est attaché le script, et « GameObject » qui sert à stocker un autre objet que celui qui a le script.

En restant sur l'exemple de la commande Translate, pensez également à :

- Mettre un point entre « transform » et « Translate ».
- Mettre un point-virgule à la fin de la ligne.
- Mettre des parenthèses après « Translate ».

- Mettre le Vector3 entre ces parenthèses.

Variables

Une variable sert à stocker une valeur qui va nous être utile tout le long du script. Vous en avez déjà utilisé dans vos cours de c++ ou python.

Il en existe différents types, les principaux que nous allons voir sont int, float, bool, et Vector3.

```
// un int sert à stocker des entiers
// 0, 7, 21, -13, 9586, ...
int mon_int = 3;

// un float sert à stocker des valeurs décimales
// 0.0, 1.5, 3.45, -9.6, 78569.256, ...
// on peut lui donner une valeur d'entier aussi, comme 2.
// lorsqu'une valeur décimale est employée,
// elle doit impérativement être suivie d'un f
float mon_float = 0.5f;

// un bool stocke une information binaire
// true / false
bool mon_bool = true;

// un Vector3 stocke trois valeurs de type float
// il faut l'initialiser avec la commande new
Vector3 mon_vector = new Vector3 (21, 0.5f, 15);
```

La déclaration d'une variable et l'assignation d'une variable sont deux choses différentes : on déclare une variable une seule fois (habituellement avant la fonction Start()), mais on peut lui assigner une valeur quand on veut. On ne peut pas assigner de valeur à une variable non déclarée, et on ne peut pas déclarer plusieurs fois une même variable.

Une variable est déclarée selon la syntaxe suivante :

```
public int nombreDeBananes = 1;

private float tailleDeMaBanane;
```

- **accessibilité** : si votre variable est publique ou privée. Une variable publique peut être modifiée dans l'Inspector. Ne rien mettre revient à avoir une variable privée.
- **type** : le type de la variable, qui dépendra de ce que vous voulez en faire. Voir des exemples de types plus haut.
- **nom de la variable** : le nom de votre variable. Vous ne pouvez pas avoir deux variables avec le même nom. Il ne doit pas contenir d'espaces ou de caractères spéciaux.
- **valeur (facultatif) :** vous pouvez assigner une valeur à une variable au moment où elle est déclarée. Il est cependant plus difficile d'assigner la valeur d'un Transform ou un GameObject, je vous conseille donc d'en faire une variable publique afin de pouvoir la déclarer à la main dans Unity.

Il est possible de déclarer une variable dans le bloc Start() ou Update(), mais elle ne sera

accessible que dans ce bloc. Elle ne doit pas contenir de mot-clé d'accessibilité (public ou private).

Attention ! Si vous déclarez une variable en public, comme sa valeur sera changeable depuis l'interface Unity, il ne sera plus possible de changer sa valeur lors de sa déclaration dans le script. Il reste cependant possible de modifier sa valeur une fois dans le bloc Start() ou Update() .

Vector3

Le Vector3 est un type de variable propre à Unity. Il stocke trois valeurs de type float. Comme mentionné plus tôt, lui assigner une valeur s'effectue à l'aide du mot-clé « new ».

```
// un Vector3 stocke trois valeurs de type float
// il faut l'initialiser avec la commande new
Vector3 mon_vector = new Vector3 (21, 0.5f, 15);
```

Mais il est aussi possible d'assigner des valeurs classiques et prédéfinies par Unity.

```
Vector3 vectorA = Vector3.up; // lui assigne la valeur (0,1,0)
Vector3 vectorB = Vector3.down; // lui assigne la valeur (0,-1,0)
Vector3 vectorC = Vector3.right; // lui assigne la valeur (1,0,0)
Vector3 vectorD = Vector3.left; // lui assigne la valeur (-1,0,0)
Vector3 vectorE = Vector3.forward; // lui assigne la valeur (0,0,1)
Vector3 vectorF = Vector3.back; // lui assigne la valeur (0,0,-1)

Vector3 vectorG = Vector3.one; // lui assigne la valeur (1,1,1)
Vector3 vectorH = Vector3.zero; // lui assigne la valeur (0,0,0)
```

Opérateurs basiques

Les opérateurs permettent de gérer les variables pour leur assigner des valeurs ou gérer des calculs mathématiques simples.

Nous avons vu le « = » qui permet d'assigner une valeur

- On peut écrire « public float speed = 5f ; », ce qui nous déclare une variable publique d'une valeur de 5.
- Mais on peut aussi dans le bloc Start ou Update écrire « speed = 0f ; » afin de changer la valeur de cette variable.

Il en existe d'autres pour réaliser des opérations simples :

- + : permet de faire une addition.
- - : permet de faire une soustraction.
- * : permet de faire une multiplication.
- / : permet de faire une division, mais à éviter autant que possible car très gourmand pour la machine. Écrivez « * 0.1f » plutôt que « / 10 ».

Il est possible d'assigner des valeurs à des variables en faisant des calculs :

```
mon_int = 3 * 7;

mon_float = 2 + 0.1f;

vectorA = Vector3.up * 3;

vectorB = Vector3.left + Vector3.forward;
```

Il est même possible d'utiliser des variables dans ces calculs.

Effectuer un « += » signifie ajouter la valeur (ici 4) à celle de la variable. Les opérateurs « -= », « *= » et « /= » existent également. Comme dans d'autres langages, « ++ » signifie « += 1 ».

```
mon_float = mon_int + 4;
mon_int += 4;
mon_int++;
```

Mais attention ! Il n'est pas possible de faire de calculs pendant la déclaration !

Commandes du Transform

Le Transform gère la position, la rotation, et l'échelle de l'objet. C'est à travers lui que nous pourrons agir sur ces paramètres. Il s'utilise avec des Vector3.

- **transform.position** permet de définir ou récupérer la position de l'objet auquel est assigné le script.
- **transform.rotation** permet de définir ou récupérer la rotation. Mais attention ! L'angle est ici stocké dans un Quaternion. Pour employer un Vector3, vous devrez utiliser les **transform.eulerAngles**.
- **transform.localScale** permet de définir ou récupérer la taille de l'objet.
- **transform.Translate(<Vector3>) ;** est une commande qui permet d'effectuer une translation, c'est à dire un déplacement dans le temps, selon le vecteur saisi.
- **transform.Rotate (<Vector3>) ;** est une commande qui permet de faire tourner un objet sur lui-même autour de l'axe défini par le vecteur saisi.
- **transform.LookAt(<Vector3>) ;** ou bien transform.LookAt(<Transform>) ; permet de demander à l'objet de se tourner vers la position dans l'espace définie par le Vector3, ou bien la position dans l'espace du Transform.
- **transform.RotateAround (<Vector3>, <Vector3>, <float>) ;** est une commande plus complexe qui fait tourner un objet autour d'un point dans l'espace (le premier Vector3), en respectant l'axe indiqué (le second Vector3), et une vitesse précisée (le float).
    - RotateAround n'est pas la meilleure méthode pour qu'un objet tourne autour d'un autre : il vaut mieux parenter l'objet à un autre Transform, qui lui tournera sur lui-même à l'aide d'un Rotate.

Time

Votre objet va vite lorsque vous réalisez une translation ? C'est normal ! En fait, si vous faites un Translate dans le Update, Unity va réaliser ce mouvement en fonction du nombre de frames qu'il est capable de calculer par seconde. Pour éviter ce problème, il est courant de multiplier les vecteurs de mouvement par : « **Time.deltaTime** ».

Le deltaTime renvoie la durée que met la machine à calculer chaque frame. Ainsi, votre objet se déplacera à la même vitesse peu importe la puissance de la machine sur laquelle vous lancerez le script.

Vous obtiendrez le résultat suivant :

```
transform.Translate(Vector3.forward * Time.deltaTime);
```