Révision de notions de programmation

Un script = une classe

En C# chaque script créé devient une nouvelle classe. Le nom que vous donnez au script devient le nom de la classe.

Voici un exemple : disons que vous créez un script C# et que vous le nommez *DeplacementObjet*. En ouvrant ce script, vous retrouverez ceci :

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

public class DeplacementObjet : MonoBehaviour {

   // Utiliser pour initialisation des éléments
   void Start () {

   }

   // Mise à jour des éléments à chaque cadence (frame)
   void Update () {

   }
}
```

En C#, chaque script (ou chaque classe!) doit préciser ce dont il a besoin pour fonctionner. Il faut donc "importer" dans le script les classes nécessaires à l'aide de la commande "using". Dans cet exemple, on importe des collections de classes nécessaires pour pouvoir programmer en C# dans Unity. Les deux premières lignes importent des bibliothèques de classes .Net et la troisième ligne les collections de classes relatives à Unity.

Pour plus d'informations :

https://msdn.microsoft.com/fr-fr/library/system.collections(v=vs.110).aspx
https://msdn.microsoft.com/fr-fr/library/system.collections.generic(v=vs.110).aspx

```
public class DeplacementObjet : MonoBehaviour
{

}
```

En C#, il faut obligatoirement déclarer la classe. Le nom de cette dernière doit être le même que le nom du script dans l'interface de Unity sinon il y aura une erreur. Notez la présence des accolades ({ }). L'ensemble du code (variables, fonctions) devra être placé entre ces accolades. **MonoBehaviour** indique de quelle classe est dérivé le script que vous créez. MonoBehavior est la librairie de Unity qui contient les classe GameObject , Transform, Update(), Start() etc.

Les variables (ou propriétés)

En C#, les variables se déclarent sous cette forme: ***type nomVariable ; ou type nomVariable = valeur; ou var nomVariable = valeur;*** ***(la troisième forme fonctionne seulement pour les variables locales)***

À noter : En C# les variables sont **privées par défaut**. Pour avoir une variable publique (accessible dans l'inspecteur de Unity et par les autres scripts), il faut le préciser avec le mot clé "**public**" juste avant le type de variable.

À noter : les nombres à virgules (float) en C# doivent être suivis de la lettre **"f"**. Exemple **5.45f**

Exemples:

```
public int compteBallon;
public float degreRotation = -2.0f;
public bool finJeu = false;
string  monNom = “DarkToro” ;
public Vector3 vitesse;
public GameObject objetARamasser;          //initialiser dans l’Inspecteur en général
private Rigidbody rigidBodyCube;
public AudioClip sonExplosion;
public GameObject[] objetsInventaire;   //tableau de GameObject pour l’inventaire, initialisé dans l’Inspecteur
```

Exemples d’utilisations

```
public class ScriptObjet: MonoBehaviour
{
  public Vector3 vitesse;       // les 3 valeurs sont 0, on peut les initialiser dans l’inspecteur si la
                                // variable est "public".

//ou    initialiser ici
  public Vector3 vitesse = new Vector3(20,45,10);  // x = 20, y = 45 et z = 10

  void Start ()
   {
      vitesse.y = 45;         // l’initialiser dans Start remplace la valeur qui se trouve dans l’Inspecteur
                              //  et on peut  modifier une seule valeur à la fois si nécessaire
   }
}
```

Les Fonctions (ou Méthodes)

En C#, les fonctions se déclarent sous cette forme:

typeDeValeurRetournée NomDeFonction()

typeDeValeurRetournée NomDeFonction(typeParamètre1 nomParametre1, etc.)

À noter : si la fonction ne retourne pas de valeur, il faut utiliser le mot clé "void" avant le nom de la fonction.

Exemples en C#

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

public class DeplacementObjet : MonoBehaviour
{
    void Start ()
    {
        Salutation("BAREV");
        Salutation("SALAM");                          //appel d'une fonction qui ne retourne pas de valeur
        floattotal = MonAddition (5f, 10f);           //appel d'une fonction qui retourne une valeur

        print (total);
    }
    void Salutation(string mot)
    {
        print(mot);
    }

    float MonAddition(float chiffre1,float chiffre2)
    {
        float resultat = chiffre1 + chiffre2;
        return resultat;
    }
}
```

Le composant transform d'un objet 3D

transform.position -> la position en x,y,z de l'objet qui possède le script
transform.rotation -> la rotation en x,y,z de l'objet qui possède le script
transform.localScale -> l'agrandissement de l'objet qui possède le script

Remarque : position et **scale** sont de type **Vector3**
rotation est de type **Quaternion** (représente les angles mais pas en degré, plus optimal pour 3D)

REMARQUE: En C#, il n'est pas possible de modifier directement les valeurs .x , .y et .z d'une propriété d'une classe interne de Unity de type Vector3 (ex: position, rotation, velocity, etc.). Il faut utiliser une variable temporaire ou une nouvelle Vector3 et modifier les 3 valeurs en même temps, comme dans l'exemple ci-bas. Autrement, une erreur de compilation se produit.

Exemple

```
print(transform.position);
transform.position = new Vector3(10,20,10);  // pour changer les 3 valeurs, en C# il faut créer un nouveau Vector3 à
```

```
                                                    // l'aide de la commande new
transform.position = transform.position + new Vector3(0,10,0); // pour augmenter seulement le y de 10

// Pour multiplier par 2 la dimension Z d'un objet
 transform.localScale = new Vector3(transform.localScale.x, transform.localScale.y, transform.localScale.z * 2);

//ou en utilisant une variable temporaire
Vector3 tailleActuelle = transform.localScale;
 tailleActuelle.z *= 2;
transform.localScale = tailleActuelle;

//Modifier la rotation----------------------------------------------------------

transform.rotation = Quaternion.Euler(45, 0, 0);   // pour fixer la rotation, on peut utiliser  Quaternion.Euler et exprimer
                                                   // la rotation désirée en degré selon les axes x, y, z.
//ou

transform.localEulerAngles =  new Vector3(45,0,0)

// pour modifier seulement une des valeurs
transform.localEulerAngles =  new Vector3(transform.localEulerAngles.x , transform.localEulerAngles.y , 90);
transform.rotation = Quaternion.Euler(transform.localEulerAngles.x , transform.localEulerAngles.y , 90);
```

Tourner un objet : transform.Rotate

transform.Rotate(float xAngle, float yAngle, float zAngle, Space relativeTo = Space.Self);

transform.Rotate(Vector3 eulerAngles, Space relativeTo = Space.Self);

Description

Applique une rotation de xAngle degrés autour de l'axe des X, yAngle degrés autour de l'axe Y, et zAngle degrés autour de l'axe des Z (dans cet ordre).

On peut aussi fournir les 3 valeurs de x, y et z dans une variable de type Vector3.

Si relativeTo est laissé de côté ou fixé à ***Space***. Avec ***Self*** la rotation est appliquée par rapport aux axes locaux de l'objet (les axes x, y et z de l'objet dans la scène selon son point de pivot).

Si relativeTo est ***Space.World*** la rotation est appliquée par rapport au système de coordonnées du monde.

Exemple

```
public float degreRotation = -2f;   //angle de rotation à appliquer
public Vector3 anglesRotationV3 = new Vector3(0,0,5);     //vecteur de rotation à appliquer


void Update()
{
   transform.Rotate(1, 0, 0);     // Tourne l'objet autour de son axe X de 1 degré/frame.

   // Tourne l'objet autour de son axe Y selon la valeur de la variable degreRotation
   transform.Rotate(0, degreRotation , 0);
```

```
    // Tourne l'objet autour de l'axe Y du monde de 1 degré/seconde par rapport à l'axe du monde
    transform.Rotate(0, Time.deltaTime, 0, Space.World);

    //tourne l'objet selon les valeurs d'un Vecotr3
    transform.Rotate(anglesRotationV3);
  }
```

Accès aux autres composants de Unity : GetComponent<*typeDeComposant*>()

```
  GetComponent<AudioSource>();      // pour accéder au composant AudioSource d'un gameObject
  GetComponent<Rigidbody2D>();    // pour accéder au composant Rigidbody2D d'un gameObject
  GetComponent<Rigidbody>();       // pour accéder au composant Rigidbody2 d'un gameObject
  GetComponent<LeScriptDeplacement>(); // pour accéder au script LeScriptDeplacement d'un gameObject
```

Pour simplifier l'écriture du code, il est recommandé de déclarer des variables au début du script et les initialiser par les bons composants dans la fonction Start(); et par la suite utiliser les variables pour référer aux composants.

Exemples

```
AudioSource audioCube;       // variable privée de type AudioSource
Rigidbody objetCube;         // variable privée  de type Rigidbody
                             // les variables privées ne sont pas visibles dans l'inspecteur
```

```
//fonction d'initialisation des variables de référence pour les composants
void Start()
{
   audioCube = GetComponent<AudioSource>();   //on assigne à la variable le composant AudioSource de l'objet
   objetCube = GetComponent<Rigidbody>();    //on assigne à la variable le composant Rigidbody d'un objet 3D
   audioCube.Play();
   rigidbodyCube.useGravity = true;
 }
```

Détection des touches du clavier

Input.GetKey(touche:String)

Description

Retourne **continuellement** « vrai » si la touche indiquée est appuyée.

```
void Update()
  {
    if (Input.GetKey ("a"))
    {
      print ("la lettre « a » est appuyée");
      transform.Rotate(0,10, 0);
    }

    if (Input.GetKey ("space"))
    {
      print ("la barre d'espace est appuyée");
      transform.Translate(0, 0, 1);
    }
  }
```

La touche peut aussi être identifiée par la propriété KeyCode.

Exemple : KeyCode.A, KeyCode.Escape, KeyCode.Space , KeyCode.UpArrow etc.

```
if (Input.GetKey (KeyCode.Space)...   //pour détecter la touche "espace"

if (Input.GetKey (KeyCode.Return)...        // pour détecter la touche "Return" (Enter)
```

Input.GetKeyDown(touche:String) et Input.GetKeyUp(touche:String)

Description

Retourne « vrai » si la touche indiquée vient d'être appuyée (GetKeyDown) ou soulevée (GetKeyUp) **une seule fois**.

Remarque : Ces méthodes doivent être placées dans **Update**() et **jamais** dans **FixedUpdate**(). FixedUpdate() peut manquer la touche parce que son FPS est plus lent que Update(), pour optimiser les calculs de l'engin physique.

Input.GetAxis(axe:String)

Description

Retourne les valeurs virtuelles de l'axe de mouvement identifié par *axe ("Horizontal" ou "Vertical")*

La valeur est comprise entre -1 et 1 des entrées clavier et joystick. Les valeurs changent graduellement et permettent un mouvement plus fluide de l'objet.

Input.GetAxis ("Vertical") : détecte les flèches « haut,bas » et les touches « w, s »

Input.GetAxis ("Horizontal") : détecte les flèches «gauche, droite » et les touches « a, d »

Exemples

```
float vitesse = 10f;
float vitesseRotation = 100f;

void Update()
{
   var vitesseVetricale = Input.GetAxis ("Vertical") * vitesse;
   var rot = Input.GetAxis ("Horizontal") * vitesseRotation;

   transform.position = transform.position +  new Vector3(0,vitesseVetricale,0);  //monte l'objet vers le haut
   transform.Rotate (0, rot, 0);
}
```

Communication entre 2 objets

Accès aux composants d'un autre objet à partir d'un script :

Il faut dans le script:
- déclarer une variable public de type *GameObject,*
- assigner l'objet à cette variable dans l'inspecteur
- utiliser GetComponent pour accéder aux propriétés ou aux variables du script de cet objet.

Exemple

```
//Script du personnage
public GameObject unDiamant; // variable publique qui contient la référence à un autre gameObject. À définir
                             // dans l'inspecteur.

void Start()  //ou n'importe quelle autre fonction
{
   unDiamant.transform.position = new Vector3(10,20,10);   // change la position de l'objet en référence dans la
                                                           // variable
   unDiamant.GetComponent<AudioSource>().Play();          // lance la lecture d'un son associé à l'objet en
                                                           // référence
   unDiamant.GetComponent<ScriptDeDiamat>().vitesse.x = 10;  //modifie la valeur de la variable vitesse de
                                                             // l'objet en référence,
   Remarque:  la variable vitesse doit être public sinon erreur de compilation "variable is inaccecible..."
   unDiamant.GetComponent<ScriptDeDiamat>().Exploser();     //appel la fonction Exploser de l'objet en
                                                             // référence.
}
```

À noter : Pour accéder aux variables et fonctions d'un autre script, ces dernières doivent obligatoirement être publiques.

```
// Script de l'objet diamant
public class ScriptDeDiamat: MonoBehaviour
```

```
{
        public Vector3 vitesse = new Vector3(5,0,0);

      // fonction publique. Si "public" n'est pas mentionné, la fonction sera privée par défaut.
       public void Exploser()
       { .....
       }
}
```

Accéder directement au script d'un objet (façon alternative)

Si nous avons besoin d'accéder seulement au script d'un objet alors on peut mémoriser la référence du script.
Il faut:
- déclarer une variable public de type *NomScriptObjet*, (le nom du script de l'objet i.e. sa classe)
- assigner l'objet à cette variable dans l'inspecteur
- accéder aux propriétés ou aux variables du script de cet objet.

Exemple

```
public ScriptDeDiamant scriptObjet ; // variable publique qui contient la référence au script d'un autre gameObject.
                                     //À définir dans l'inspecteur.
void Start()
{
  scriptObjet.vitesse.x = 10;  //modifie la valeur de la variable vitesse de l'objet en référence,
  scriptObjet.Exploser();      //appel la fonction Exploser de l'objet en référence,
}
```