UNITY 102: Intro au Scripting

I L'Interface de scripting:

a) Introduction

D'abord, pour créer un script, rien de + simple: Allez dans l'inspecteur pour un objet, "Add Component", "New script", et Visual Studio s'ouvrira avec un début de classe pré-faite :

```
public class MyScript : MonoBehaviour {

    // Use this for initialization
    0 references
    void Start () {

    }

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

    }
}
```

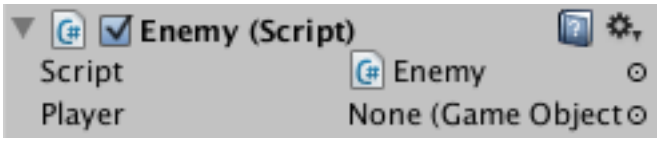
Cette classe contient déjà deux fonctions : une appelée à l'initialisation de la classe, et une appelée à chaque frame.
A cette classe, on peut rajouter ses propres fonctions, ainsi que des variables.

b) Les variables dans l'Inspecteur

Quand vous créez une variable, vous pouvez la définir comme "privée" ou "publique".
Une variable publique peut être initialisée depuis l'Inspecteur, comme paramètre de votre script .
Cette variable peut contenir n'importe quoi: un int, une str, ou un GameObject complet : Si on déclare "*public GameObject player;*", alors ceci apparaitra dans l'inspecteur : Plus qu'a y glisser son GameObject, et la variable sera initialisée dans le script !

c) Interaction avec les GameObjects

On peut récupérer un Composant de son GameObject affilié ainsi :
ClasseComposant nom_de_variable = GetComponent<ClasseComposant>().



D’autres méthodes existent pour récupérer un GameObject complet, dans lequel ensuite récupérer ses composants. Un exemple :

player = GameObject.FindWithTag(“Player”);
player.GetComponent<RigidBody>.AddForce(0, 10 0);

(***Attention :*** GetComponent<> peut renvoyer null… Et paf Segfault!)

On vient ici de récupérer un objet avec le Tag “Player”, puis appliqué la fonction “AddForce” a son rigidbody.

II: Les Évènements (de collision)

Quand 2 Colliders entrent en collision, une fonction particulière est appelée dans le script relié à l’objet : “OnCollisionEnter()”. Pour chaque frame ou le contact est maintenu, “OnCollisionStay()”, puis, quand la collision s’arrête, “OnCollisionExit()”. Ces fonctions reçoivent toutes le collider “other” en paramètre, afin de pouvoir l’affecter en conséquence.

La même chose existe pour les “Triggers”.

Si vous définissez le booléen “IsTrigger” d’un collide à True, alors celui-ci deviendra un “Trigger”: il ne sera plus affecté par la physique (pas de gravité, passe à travers les objets) mais vous permettra de lancer des fonctions “OnTrigger”.

III: Les Vecteurs et leurs usages

Quelques petites astuces pour manipuler des objets avec des vecteurs :

Les fonctions de “AddForce” / “AddTorque”:

Ces fonctions donnent de l’élan à un Rigidbody avec la force voulue.

Elles permettent de déplacer un objet beaucoup mieux qu’en manipulant son transform.

Il y a même différents types de Forces à disposition: Force continue / instantanée, et vélocité continue / instantanée . (Voir la doc)

Les Quaternions et la rotation:

Les quaternions sont un élément mathématique très utiles dans Unity, afin d’effectuer des rotations.

Par exemple, pour qu’un objet en regarde toujours un autre, on peut utiliser ceci:

```
Vector3 relativePos = target.position - transform.position;
transform.rotation = Quaternion.LookRotation(relativePos);
```

Appliquez ça sur l’update de votre objet, et voilà !

On peut aussi les utiliser pour faire rotationner un objet facilement sur lui même :