Unity - Outils de script

**Variables accessibles depuis n'importe quel script**

enabled

* Permet d'activer ou désactiver un script (enabled = true; par exemple). Peut-être utilisé sur quasiment tous les components.

gameObject

* Permet d'accéder au gameObject sur lequel se trouve l'instance de votre script (par extension, on peut dire que cela permet d'accéder aux gameObjects des différentes instances du script qui sont en train de s'exécuter dans la scène).

tag

* Permet d'accéder au tag du gameObject sur lequel se trouve votre script (c'est un raccourci, vous pouvez aussi y accéder via gameObject.tag)

transform

* Permet d'accéder au component « transform » du gameObject sur lequel se trouve votre script. Tout bonnement indispensable.

name

* Permet d'accéder au nom du game object sur lequel se trouve votre script (c'est un raccourci, vous pouvez aussi y accéder via gameObject.name)

**Qu’est-ce qu’une fonction et comment en créer une ?**

Une fonction est une suite d’instructions que l’on condense en une ligne.

Par exemple, on pourrait créer une fonction tout à fait fantasque :

Fonction faire un gâteau : //Voici la fonction écrite en une ligne, faire un gâteau…

-Casser les œufs.

-Mettre de la farine.

-Mélanger.

-Mettre au four.

-etc…

Pour créer cette fonction, on fera :

void FaireUnGateau()

{

-Casser les œufs.

-Mettre de la farine.

-Mélanger.

-Mettre au four.

-etc…

}

Il faut absolument créer cette fonction au même endroit que vous déclarez les variables public !  
Les fonctions ne doivent pas être créer dans les void Update() ou void Start() !

Pour appeler une fonction, on fera :

FaireUnGateau() ;

Cela exécutera tout ce que vous aurez écrit dans la fonction que vous avez créée.

Un exemple concret :

AfficherDesMessages() //Je déclare ma fonction

{

Debug.Log(« Bonjour ») ;

Debug.Log(« tout le monde ») ;

}

Void Start()

{

AfficherDesMessages() ; //J’appelle ma fonction dans la fonction Start() d’Unity. Ce qui fait que lorsque je lance mon jeu, « AfficherDesMessages() » est exécuté (du coup, les 2 messages s’affiches dans la console).

}

**Fonctions utiles accessibles depuis n'importe quel script**

Destroy

* Sert à détruire un gameObject
* Utilisation:
* Destroy(gameObject); // détruit le game object sur lequel notre script se trouve
* Destroy(GameObject.Find("MainCamera") );

Instantiate

* Générer un gameObject à partir d'un modèle (prefab)
* Utilisation:
* // on suppose ici que l'on a une variable prefab\_to\_spawn qui contient un prefab valide.
* // on le fait apparaître en 0,0,0 avec une rotation neutre
* Instantiate(prefab\_to\_spawn, Vector3.zero, Quaternion.identity);
* Vector3.zero veut dire que le vecteur ne contient que des valeurs égales à 0 (C’est la même chose que Vector3(0,0,0) ) ; Dans l’exemple précédent, on crée un objet à la position X = 0, Y = 0 , Z = 0 ;
* Quaternion.identity veut dire que l’on met les rotations à 0 (C’est la même chose que Vector3(0,0,0) )

**Plus compliqué ici**

Invoke

* Permet de planifier l'exécution d'un morceau de programme dans le futur
* Utilisation:
* Invoke("Destruction", 2.5f); // appellera une fonction "Destruction" dans 2,5 secondes
* // la fonction Destruction en question doit avoir la même forme que Start ou Update
* // (Destruction n'est qu'un exemple, vous pouvez créer autant de ces fonctions que vous voulez

InvokeRepeating

* Même chose que Invoke mais la fonction est appelée en boucle
* Utilisation:
* InvokeRepeating("Destruction", 1, 2.5f); // le premier paramètre dit dans combien de temps se fait la 1ère exécution, le 2ème paramètre tous les combien de temps la fonction s'exécutera ensuite.

CancelInvoke

* Annule les Invoke en cours
* Utilisation:
* CancelInvoke();

**Informations utiles accessibles depuis une variable de type GameObject (la variable gameObject de votre script, une variable contenant un prefab, un gameObject que vous récupérez lors d'une collision)**

tag, name, transform

Trouver un gameObject dans la scène depuis le code (par exemple un gameObject qui s'appelerait "avatar"):

* GameObject objet = GameObject.Find("avatar");

Trouver un gameObject dans la scène depuis le code, en utilisant son tag (par exemple un game object qui aurait comme tag "Player"):

* GameObject objet = GameObject.FindWithTag("Player");

Récuppérer un component sur le gameObject sur lequel se trouve votre script (un SpriteRenderer par exemple) :

* SpriteRenderer sprite = gameObject.GetComponent("SpriteRenderer");
* // Ou autre syntaxe (mais résultat équivalent)
* // exemple avec un gameObject de la scène
* SpriteRenderer sprite = GameObject.Find("vaisseau").GetComponent<SpriteRenderer>();

Désactiver un gameObject sans le détruire (par exemple le gameObject "avatar" que l'on aurait récupéré précédemment et stocké dans une variable « avatar »)

* avatar.SetActive(false);
* // exemple pour désactiver le gameObject sur lequel on se trouve
* gameObject.SetActive(false);

**MOUVEMENTS/ROTATIONS**

Accéder à la position d'un transform

* transform.position

Modifier la position d'un transform (par exmple en x=1, y=-2, z=0). Modifier la position revient à une téléportation

* transform.position = new Vector3(1, -2, 0);

Modifier la position locale d'un transform (position par rapport au parent de l'objet, si l'on est dans une hiérarchie)

* transform.localPosition = new Vector3(1, -2, 0);

Modifier une seule composante d’un transform (par exemple uniquement le y, sans modifier la position en x ni celle en z)

* // Vous n’avez pas le droit de modifier directement une seule composante (vous ne pouvez pas écrire transform.position.y = -4; par exemple)
* // à la place, il faut écrire:
* transform.position  = new Vector3( transform.position.x, -4, transform.position.z );
* // ou
* Vector3 position = transform.position; // on stocke la position du transform (imaginons que sa position est de X = 0, Y = 22, Z = 0) dans la variable « position » de type Vector3.
* Une variable de type Vector3 possède 3 paramètre que l’on désigne par X,Y et Z.
* Par exemple, « position » a une valeur de : X = 0, Y = 22, Z =0 ;
* position.y = -4; // on a le droit de modifier cette copie. Ici on peut accéder à la valeur « X » du Vector3 position en ajoutant à la fin de la variable un « .X »
* transform.position = position; // on assigne la nouvelle position au transform

Faire référence aux directions du repère local d'un objet (repère qui tient compte de l'orientation de l'objet, c’est-à-dire que l’on regarde en haut de la tête de l’objet et non en haut vers l’axe Y)

* transform.up (axe Y), transform.right (axe X), transform.forward (axe Z)

Faire référence aux directions du repère global de la scène

* Vector3.up, Vector3.right, Vector3.forward

Faire bouger un objet (par exemple de 1,2 unité vers la droite et 0,5 unité vers le bas)

* transform.Translate(1.2f, -0.5f, 0, Space.World);
* ou
* transform.position += new Vector3(1.2f, -0.5f, 0); // remarquez que l'on utilise +=, à la différence du cas où l'on téléporte l'objet à une nouvelle position
* += veut dire que l’on ajoute une valeur et non qu’on la remplace
* Par exemple, je prends une variable « vitesse » qui vaut 5. On peut ajouter une valeur à cette variable en faisant : vitesse += 10. Vitesse vaudra alors 10 car 5+10 = 15.

Même chose mais avec un déplacement suivant les axes du repère local de l'objet

* transform.Translate(1.2f, -0.5f, 0);
* ou
* transform.position += transform.right \* 1.2f + transform.up \* -0.5f + transform.forward \* 0;

Accéder à la rotation x, y, z d'un gameObject

* transform.eulerAngles.x, transform.eulerAngles.y, transform.eulerAngles.z

Modifier la rotation d'un gameObject (par exemple 90, 45, 0)

* transform.eulerAngles = new Vector3(90, 45, 0)

Rotation locale d'un objet (par rapport au parent dans une hiérarchie)

* transform.localEulerAngles

Faire tourner un gameObject autour de son axe central (de 45 degrés sur son axe X par exemple)

* transform.Rotate(45,0,0);

Faire tourner un gameObject autour d'un point quelconque de l'espace

* transform.RotateAround( new Vector3(1, 10, 0) );

Faire tourner un gameObject autour d'un autre gameObject

* GameObject other = GameObject.Find("autre");
* transform.RotateAround(other.transform.position );

Accéder au scale d'un objet

* transform.localScale

/!\ Attention, le scale d'un object n'est pas forcément sa taille absolue. Par exemple, si mon scale est de 1 mais que je suis enfant d'un objet qui a un scale de 2 dans ce cas ma taille n'est pas 1.

De plus, si j'ai importé une image de 200 pixels x 200 pixels et que la valeur de pixels per unit est à 100 (valeur par défaut) alors mon sprite fera 2 unités de taille mais le gameObject qui utilisera mon sprite peut tout à faire avoir une taille de 1.

**SPRITES**

Accéder à la taille d'un sprite en unités dans le monde

* gameObject.GetComponent<SpriteRenderer>().sprite.bounds.size (On accède à la taille de l’image et non la taille du transform du gameObject !)

Accéder à la demi-taille d'un sprite

* gameObject.GetComponent<SpriteRenderer>().sprite.bounds.extents (On accède à la taille de l’image et non la taille du transform du gameObject !)

Modifier la couleur du sprite du gameObject qui se trouve sur votre script (la couleur s'exprimer en Red pour rouge, Green pour vert, Blue pour bleu, Alpha, entre 0 et 1 pour la transparance)

* gameObject.GetComponent<SpriteRenderer>().color = new Color(0.9f, 0, 0, 1);

Masquer le sprite d’un objet

* SpriteRenderer renderer = gameObject.GetComponent<SpriteRenderer>();

renderer.enabled = false;

**CAMERA (plus technique)**

Accéder à la caméra principale de la scène:

* Camera.main

Donc la position de la caméra est : Camera.main.transform.position

Récupérer la taille de la caméra :

* float hauteur\_camera = Camera.main.orthographicSize \* 2; // orthographicSize donne la demi-hauteur
* float largeur\_camera = hauteur\_camera \* Camera.main.aspect;

Transformer une position “World” (par exemple 0,0,0) dans le repère de la caméra (le 0,0 est en bas à droite de la caméra, le 1,1 en haut à droite)

* Vector3 position\_in\_camera\_space = Camera.main.WorldToViewportPoint(new Vector3(0,0,0) );
* C’est un moyen simple de vérifier si un objet est dans la caméra ou non

Transformer une position exprimée dans le référentiel de la camera en une position dans le monde (par exemple le point en haut à droite de la caméra)

* Vector3 position\_in\_world\_space = Camera.main.ViewportToWorldPoint(new Vector3(1,1,0) );

**DIVERS**

Récupérer un nombre aléatoire entre -11 et 15,7

* Float random\_number = Random.Range(-11, 15.7f);

Charger une scène (qui s'appellerait level\_2.unity par exemple)

* UnityEngine.SceneManagement.SceneManager.LoadScene("level\_2");
* // si cette scène fait partie de la build, elle peut aussi être chargée en utilisant son numéro au lieu de son nom

Quitter le jeu

* Application.Quit();

**Générer des assets de Sprite dans le menu d’Unity**

* Aller dans le dossier « Asset » dans Unity
* Clique droit
* Create
* Sprite
* Choisissez entre : square/circle/hexagon…

**Outils de debug**

Ecrire un message dans la console (log)

* Debug.Log(“mon message”);

Stopper l’exécution du jeu (mettre en pause uniquement utile en tant que développeur, cela ne quitte pas le jeu une fois finalisé)

* Debug.Break();

**Evènements de collision**

***Rappel: pour pouvoir recevoir les événements de collision dans le code, il faut que votre objet ait un collider ET un rigidbody (cela fonctionne avec un collider2D et un rigidbody2D).***

Pour détecter des collisions de type **trigger** dans un script :

// pour détecter que l’on **vient** d’entrer en collision avec un trigger

* void OnTriggerEnter2D(Collider2D other)
* {
* //Votre code
* }

// pour détecter que l’on **est** en collision avec un trigger (donc appelé à toutes les frames tant que l’on est en collision avec cet élément)

* void OnTriggerStay2D(Collider2D other)
* {
* //Votre code
* }

// pour détecter que l’on **vient** de quitter une collision avec un trigger

* void OnTriggerExit2D(Collider2D other)
* {
* //Votre code
* }

Pour détecter des collisions de type **tangible (le mode par défaut)** dans un script :

// pour détecter que l’on **vient** d’entrer en collision avec un objet tangible (non trigger)

* void OnCollisionEnter2D(**Collision2D** other)
* {
* //Votre code
* }

// pour détecter que l’on **est** en collision (donc appelé à toutes les frames tant que l’on est en collision avec cet élément)

* void OnCollisionStay2D(**Collision2D** other)
* {
* //Votre code
* }

// pour détecter que l’on **vient** de quitter une collision avec un objet tangible (non trigger)

* void OnCollisionExit2D(**Collision2D** other)
* {
* //Votre code
* }

Récupérer la taille d’un Collider2D

* Soit un script qui se trouve sur votre avatar et supposons que cet avatar ait un Collider2D, on pourra écrire:
* Vector2 taille\_collider = gameObject.GetComponent<Collider2D>().bounds.size;

**Maths**

Toutes les fonctions mathématiques se trouvent dans Mathf (<https://docs.unity3d.com/ScriptReference/Mathf.html>)

Quelques fonctions utiles hors des classiques :

Clamp

* Permet de restreindre une valeur à l’intérieur d’un intervalle.
* Ex: float x = Mathf.Clamp(10, 0, 5); // renvoie 5. La première valeur est le nombre à restreindre, les 2 suivantes sont l’intervalle. Ici la fonction analyse le « 10 » et cette fonction se rend compte que le 10 est plus grand que 5, alors Mathf.Clamp renvois 5, la valeur la plus haute de l’intercale.

MoveTowards

* Permet de faire bouger une valeur vers une autre. Pratique pour faire avancer une position vers une autre.
* float position\_x\_ennemi = 10; // une position quelconque
* float vitesse = 6;
* float x = Mathf.MoveTowards(transform.position.x, position\_x\_ennemi, vitesse \* Time.deltaTime);
* // cette ligne permet de faire avancer la position en x de notre objet vers une autre position, à une certaine vitesse (et on récupère le résultat dans une variable)

Les paramètres sont : valeur actuelle, valeur cible, quantité de déplacement

La particularité de cette fonction est que si le déplacement effectué venait à nous faire dépasser la valeur cible, MoveTowards nous positionne pile sur la cible.

Existe aussi pour les Vector3 ! (Vector3.MoveTowards)

**Time**

On trouve dans Time des informations liées au temps qui passe (pour les poètes)

Time.deltaTime correspond au temps qui s’est écoulé depuis la dernière frame

Time.time correspond au temps qui s’est écoulé depuis le lancement du jeu

Time.timeScale permet de modifier la vitesse d’écoulement du temps. Un timeScale de 1 est la valeur par défaut. S’il vaut 2, tout ce qui dépend du temps dans votre jeu ira 2 fois plus vite, s’il vaut 0.5, tout ira 2x moins vite, etc.