Gestion de la souris

Les interactions avec les objets 3D ayant des colliders

Lorsqu'on objet 3D possède un Collider , il est possible d'utiliser les fonctions reliées aux événements de la souris dans son script.

void OnMouseEnter ()	**void OnMouseOver ()**	**void OnMouseExit ()**
void OnMouseDown ()	**void OnMouseUp ()**	

Exemple

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

// Modifie la couleur du mesh de l'objet en fonction de la souris. (effet de RolleOver sur un objet 3D)
public class EvenementsSouris : MonoBehaviour {

   // Variable qui contiendra la référence au Renderer de l'objet
   Renderer rend;

   void Start () {
      rend = GetComponent<Renderer>();
   }

   // Le mesh devient rouge lorsque la souris entre sur l'objet.
   void OnMouseEnter () {
      rend.material.color = Color.red;
   }

   // ...Le rouge tend vers cyan lorsque la souris est sur l'objet..
   void OnMouseOver () {
      rend.material.color += new Color(-0.1f, 0.1f, 0.1f) * Time.deltaTime;
   }

   // Le mesh devient blanc lorsque la souris quitte l'objet.
   void OnMouseExit () {
      rend.material.color = Color.white;
   }

   // La nouvelle scène est lancée avec un clique gauche de la souris
   void OnMouseUp () {
      SceneManager.LoadScene("niveau1");
   }
}
```

Déplacement du personnage selon les axes du monde (sans tenir compte de son orientation)

Ce type de déplacement permet de déplacer le personnage selon les axes X et Z du monde en modifiant sa vélocité en X et en Z.

Remarque: On n'a plus besoin de le déplacer selon son transform.forward i.e. son axe Z local.

Exemple:

```
rigidbodyPersonnage.velocity = new Vector3(depHorizontal,0, depVertical);
```

Pour orienter le personnage, on le tourne vers la position de la souris.

```
transform.LookAt(positionSouris);     //la position de la souris est obtenu par la technique vue plus haut
```

Vector3( ).normalized

Permet de normaliser la valeurs du vecteur et les ramener entre 0 et 1.

Par exemple, lorsque le personnage se déplace en diagonal (en x et en z du monde) alors il va plus vite. La solution est de normaliser le vecteur de déplacement à l'aide de Vector3( ).normalized avant de l'affecter à la vélocité du personnage.

Exemple:

```
Print(new Vector3( 4,0,4).normalized);     affiche Vector3(0.7,0, 0.7)

Print(new Vector3( 1.5,0,1.5).normalized);     affiche Vector3(0.7,0, 0.7)

bebeRigid.velocity = new Vector3(dHorizontal, 0, dVertical).normalized * vitesseAvant ;
```

Exercice 0 (cette partie est déjà faite :)

- Téléchargez le package *"SurvieShooterDebut"* qui se trouve sur le site du cours. Ouvrez un nouveau projet et importez le package. Ouvrez la scène *"SceneDebut"*.
- Vérifiez dans l'inspecteur si le *Rig* du personnage est bien en mode *Generic* et vérifiez ses animations.
- Vérifiez la présence des composants *Rigidbody* et *CapsuleCollider* sur le personnage.
- Vérifiez que les contraintes du *Rigidbody* du personnage : Freez rotation en X, Y, Z
- Le contrôleur d'animation pour le personnage est est déjà créé, *PersonnageAnimControleur* associez-le au personnage dans l'inspecteur.
- Dans la fenêtre *Animator,* créez les transitions entre l'animation de repos et de déplacement, il y a une variable booléenne ("marche") pour les transitions.
- Complétez le script du personnage qui permet de:
    - Contrôler le déplacement en modifiant la vélocité de rigidbody du personnage. Attention en C#, il est interdit de changer les propriétés x, y ou z individuellement. Le personnage se déplace selon les axes X et Z du monde.
        - le personnage avance et recule à l'aide des touches verticales (w,s et flèches haut et bas). La vélocité en z du rigidbody est affectée par cette valeur.
        - le personnage se déplace vers la gauche et vers la droite à l'aide des touches horizontales (a,d et flèches gauche et droite).
          La vélocité en x du rigidbody est affectée par cette valeur.
        - le personnage ne doit pas aller plus vite, s'il se déplace diagonalement. (voir Vector3().normalized)
    - Gérer la transition entre les animations de repos et de déplacement.
      Si le déplacement est différent de 0, on active l'animation de déplacement (paramètre "marche" devient true, sinon false. (Vous pouvez utiliser *bebeRigid.velocity.magnitude* pour savoir si le personnage est en mouvement).
    - La caméra peut suivre le personnage en gardant une distance fixe. La caméra ne doit pas tourner.

Détection des clics de la souris (dans la fonction *Update()* d'un objet)

Input.GetKey(KeyCode.Mouse0) -- détecte le bouton gauche de la souris
(KeyCode.Mouse1: bouton milieu, KeyCode.Mouse2: bouton de droite)

ou

Input.GetMouseButton(int numBouton) ,

Permet de détecter quel bouton de la souris est appuyé (0 : bouton gauche, 1 : bouton de centre et 2 : bouton de droite).

Exemple

```
void Update ()
  {      //si le bouton gauche est appuyé et qu'on peut tirer,  alors on tire continuellement
     if(Input.GetKey(KeyCode.Mouse0)  && peutTirer == true)
     {
        TirerBalle();
     }
     if(Input.GetMouseButtonDown(2) && peutLancerBanane == true)
     {
        LancerBanane();
     }
  }
```

Remarque: Dans Unity il existe des boutons virtuels qu'il est possible de définir dans: **Edit → Project Settings → Input**

Exemple:
Input.GetButton("Fire1") // détecte le bouton gauche de la souris ou la touche CTRL du clavier ou le bouton 0 d'une manette de jeu.

Manette : Pour détecter les boutons d'une manette de jeu

Input.GetKey(KeyCode.Joystick1Button0) etc.

Création d'objets en temps réel

Tous les *gameObjects* peuvent être copiés dynamiquement. Il est également possible de créer une copie (ou instance) d'un élément préfabriqué (« Prefab »). Lorsque l'on instancie un *Prefab*, la position et l'orientation d'origine seront appliquées si aucune autre valeur de position et de rotation n'est fournie.

Instantiate (Object objetADupliquer**,Vector3** position**, Quaternion** rotation**)**
Instantiate(PrefabParticule, Vector3 position**, Vector3** rotation**);**
Instantiate(objetADupliquer);

var clone = Instantiate(objetADupliquer); //on mémorise l'objet cloné pour pouvoir y accéder

D'autres variations sont possibles. Plus de détails ici :
https://docs.unity3d.com/ScriptReference/Object.Instantiate.html

Exemple → CECI EST UN EXEMPLE...

```
//Cloner un objet Prefab et le placer au même endroit que l'objet du script
//Clone est détruit après un délai (sinon les objets clonés vont se cumuler dans le hierarchy)
    public GameObject PrefabParticule; // À définir dans l'inspecteur. Prendre un Prefab de l'onglet project.

   // Crée un clone de l'objet à la même position et rotation que l'objet du script.
    gameObject cloneParticule = Instantiate(PrefabParticule,transform.position, transform.rotation);
     //ou
     gameObject cloneParticule = Instantiate(PrefabParticule);
     cloneParticule.transform.position = transform.position;
     cloneParticule.transform.rotation = transform.rotation;
     Destroy(cloneParticule, 0.8f);
```

Exercice partie 1 (À COMPLÉTER ET À REMETTRE sur Remise Ou montrer au prof)

- Dans le projet le Prefab de particules de tir (*ParticlesTire*) qui se trouve dans Project/Prefabs a été ajouté sur la tête de fusil *(objet GunBarrelEnd).*
- Une lumière de type *pointLight* de couleur jaune a été placée sur la particule de tir.
- Testez et désactivez la particule. Lorsque la particule sera activée, la lumière s'activera aussi. Il ne faut pas désactiver la lumière puisque son parent (la particule) est désactivé.
    - **Remarque: Assurez-vous que la propriété PlayOnAwake est cochée.**
- Dans le script du personnage :
    - Déclarez deux variables de types *GameObject*, une qui contiendra la référence à la particule de tir *ParticulesTire* de l'onglet *hierarchy* et l'autre pour la particule de contact de tir *ParticulesHit.* Initialiser ces variables dans l'inspecteur.
    - Lorsque le bouton gauche de la souris est appuyé : Input.GetKey(KeyCode.Mouse0)
        - La particule de tir s'active (la durée de la particule: 0.1 seconde). Elle est ensuite désactivée. Pour arriver à faire cela, vous aurez besoin d'une coroutine (et d'une fonction de type *IEnumerator)* ou encore de la commande *Invoke() ;*
        - Pour empêcher de tirer trop rapidement , utilisez une variable booléenne (ex. *peutTirer*) qui se met à *false* lorsque la balle est tirée et qui redevient *true* lorsque le délai pour jouer la particule de tir est terminé.

- Le son *PlayerGunShot* doit se faire entendre, placez-le sur l'objet ... ?;
  (ce son se trouve dans le dossier Project/Audio)

- Lancement d'une balle:
  - Créez une clone de la balle en le dupliquant avec Instantiate(), ayant la position et la rotation actuelle de la balle,
  - Activez le clone et lancé le vers l'avant dans son axe Z, en lui donnant une vélocité. (.....volecity = clone.transform.forward * vitesseBalle)

- Créez un script pour la balle pour gérer la colision des balles:
  - Déclarez une variable qui permet de mémoriser la particule de contact *ParticulesHit*.
  - Lorsque la Balle fait une collision avec un objet alors un clone de la *ParticulesHit* est créé à la position de contact,
  - le clone est activé,
  - Le clone est détruite à la fin de sa durée de vie,
  - La balle est détruite.

  REMARQUE: La balle reste toujours désactivée pour ne pas être détruite. Ces sont les Balles clonées qui ont le même script que la balle qui vont exécuter les actions du script.

FIN Partie 1

Remise pour les étudiants n'ayant pas Unity:

- donner tous les composants que le personnage doit avoir,
- Le hierarchie des éléments que le personnage possède, ainsi que leur états (active ou désactive) au départ du jeu.
- donner tous les composants que la balle doit avoir,
- donner les particules utilisés dans le jeu,
- tous les scripts du jeu avec des commentaires très détaillés.

Défi OPTIONNELLE :

- **Au lieu de cloner la balle pour la lancer:**

  - Il faut utiliser un *Physics.Raycast* pour détecter quel objet est devant le fusil.

  - l'origine et la direction du *Raycast* peuvent être celles de la tête de fusil,

Truc de débogage: imprimer le nom de l'objet touché avec un print() et dessiner le rayon *Raycast* à l'aide de Debug.DrawRay(), avant d'aller plus loin.

→ Debug.DrawRay(Vector 3 *PositionOrigine*, Vector 3 *DirectionEtLongueur*, Color *Couleur*);

- Créez dynamiquement une copie de la particule *HitParticle* à la position de contact du *Raycast* avec l'objet touché → propriété *point* de la variable de type *RaycastHit* que vous utiliser.
- Détruisez le clone après la durée de la particule. Indice: Destroy(objet, délais)

- **autre Défi :** ajoutez une ligne de tir pour voir la trajectoire du tir.
    - Vous pouvez utiliser un objet ayant son point de pivot à sa base, et qui est placé au même endroit que la tête de fusil. Sa longueur est ajustée selon la propriété *distance* de la variable de type *RaycastHit* que vous utiliser.
    - **Encore mieux,** ajoutez un composant LineRenderer (ADDComponent dans l'inspecteur) sur la tête de fusil ou du personnage. Pour configurer adéquatement ce composant, vous aurez besoin :
        - De créer un nouveau matériel et de changer la couleur de l'émissive (propriété *Emission*).
        - Ce matériel devra être ajouté à la liste de matériels du composant *LineRenderer* (spécifiez un *size* de 1 dans la propriété *materials* et glissez votre nouveau matériel.
        - Le nombre de positions (*Positions*) doit rester à 2 (valeur par défaut)
        - Pour ajuster l'épaisseur du rayon, laissez la propriété *with* à 1 et ajuster la courbe dans le tableau. Cette courbe permet de contrôler l'épaisseur du rayon, du début à la fin. Pour un rayon d'épaisseur uniforme, utilisez une droite plutôt qu'une courbe.
        - Désactiver le composant *lineRenderer* dans l'inspecteur.
    - Lorsque le personnage tire :
        - Activez le composant *lineRenderer*; (GetComponent<LineRenderer>().enabled = true)

Optionnelle, Rotation personnage (Pour mieux viser):

Position de la souris dans le monde 3D et la rotation vers la souris

La position de la souris est en coordonnées 2D (x,y) par rapport à l'écran (onglet *game*). Pour obtenir sa position dans le monde 3D, il faut utiliser un rayon (*Raycast*) à partir de la caméra vers le monde 3D.

La fonction Camera.main.ScreenPointToRay(Input.mousePosition)

Retourne un rayon (Type *Ray*) créé à partir d'une position de l'écran (ici, la position de la souris) dirigé vers l'avant de la caméra. Ce rayon identifie la position et la direction du rayon. La variable de type Ray possède les propriétés origin: (x, y, z) et direction (x, y, z) qui peuvent être utilisées pour créer un Raycast.

Exemple

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

//Contrôle de l'orientation d'un gameObject en fonction du déplacement de la souris.
public class ExempleScreenToPoint : MonoBehaviour {
  void Update ()
  {
    //crée un rayon à partir de la caméra vers l'avant à la position de la souris
    Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);

    // variable locale : contiendra les infos retournées par le Raycast sur l'objet touché
    RaycastHit infoCollision;

    // lance un rayon de 5000 unités à partir du rayon crée, vérifie seulement la collision avec le plancher
    // Le plancher doit avoir un layerMask (exemple:"Plancher") assigné dans l'inspecteur
    if( Physics.Raycast(camRay.origin, camRay.direction, out infoCollision , 5000, LayerMask.GetMask("Plancher")))
    {
      transform.LookAt(infoCollision.point);  // L'objet regarde vers le point de contact
      transform.localEulerAngles = new Vector3(0, transform.localEulerAngles.y, 0);  //élimine les rotations en X et Z

       print(infoCollision.collider.gameObject);    //l'objet touché par le rayon
     }

    Debug.DrawRay(camRay.origin, camRay.direction * 100, Color.yellow);  //outils pour visualiser le rayon dans
                                                                          // l'onglet scene
  }
}
```

Rappel sur le Raycast:

Physics.Raycast(position , direction, out infoCollision, longueur, filtre de LayerMask)

Cette fonction trace un rayon invisible à partir d'une position donnée et dans une direction donnée. Si un objet est touché par le rayon, la fonction retourne la valeur *vraie (true)* et elle retourne les infos sur la collision dans une variable de type **RaycastHit** (ici, *infoCollision*).

Quatre paramètres sont nécessaires (et un optionnel) :

1- La position d'origine du rayon (un Vector3, qui peut être le *transform.position* de l'objet du script)

2- La direction du rayon (un Vector3, ex: Vector3(0,1,0) vers le haut, Vector3(0,-1,0) vers le bas, Vector3(0,0,1) vers l'avant de l'objet.

3- La variable infoCollision contiendra les informations suivantes si le rayon touche un objet :

- infoCollision.collider.name : le nom de l'objet touché (l'objet doit avoir un Collider)
- infoCollision.collider.gameObject : l'objet touché (l'objet doit avoir un Collider)
- infoCollision.distance : la distance à laquelle un objet a été touché (en mètre)
- infoCollision.point : le point de contact (position) entre le rayon et l'objet touché, (en Vector3)
- infoCollision.point.y : la position y du point de contact entre le rayon et l'objet touché,

4- La longueur du rayon en mètre.

5-Le filtre de LayerMask permet d'identifier sur quel objet le Raycast doit détecter les collisions. L'objet doit avoir un LayerMask assigné dans l'inspecteur.

LayerMask.GetMask("*NomDuLayer*")) permet d'obtenir le Mask désiré.

Remarque: Pour que le Raycast ignore un objet, il faut mettre cet objet sur le layer IgnoreRaycast.
Exemple: si on ne veut pas que l'hélico soit détecté