# Gestion de caméras

## Création d'une caméra

Pour créer une nouvelle caméra il faut choisir ***Create/camera*** à partir du menu de la fenêtre *Hierarchy* ou passer par le menu *GameObject/camera*.

Ctrl+Shift+f : permet d'aligner la caméra pour visionner la même vue que la fenêtre *Scene*. La caméra doit être sélectionnée dans la fenêtre *Hierarchy.*

Il est possible de créer plusieurs caméras, **mais il est impératif d'en garder seulement une active à la fois**.

**Remarque:** Chaque caméra possède un Audiolistener. S'il y a plus d'une caméra active alors le message d'erreur "there is 2 or more AusioListener ..." apparaît dans la fenêtre "console". Il est possible d'avoir plus qu'une caméra, mais il faut le dimensionner (exemple: minimap) et désactiver son AusioListener dans l'Inspecteur.

---

## Activer et désactiver un Objet : gameObject.SetActive(true) ou gameObject.SetActive(false)

## Identifier la caméra active : Camera.main.gameObject

Pour pouvoir changer de caméra de façon dynamique, la meilleure façon est d'identifier chaque caméra par l'étiquette (Tag) ***MainCamera*** dans la fenêtre ***Inspector***.


Lorsque les caméras possèdent ce ***Tag***, on peut trouver le **gameObject** de la caméra active dans le jeu en utilisant la commande :
**Camera.main.gameObject**

Ensuite, on peut la désactiver par sa méthode ***SetActive()***.

**Camera.main.gameObject.SetActive(false);**

**Exemple**

```
// change la caméra en désactivant la caméra actuelle et en activant la caméra choisie
public GameObject cameraChoisie;
Camera.main.gameObject.SetActive(false);  //désactive la caméra actuelle
cameraChoisie.SetActive(true);                //active la nouvelle caméra
```

---

## Remarque sur les « Tags »

On peut utiliser la propriété **« Tag »** d'un gameObject pour créer des regroupements d'objets.
Exemples de tags : ObjetARamasser, Ennemis, Argent, Vie .

---

## Création d'un contrôleur de caméras

Lorsque plusieurs caméras sont utilisées à l'intérieur d'une scène, il est souhaitable de se créer un contrôleur de caméras qui permet de faire la gestion de l'activation et de la désactivation des caméras.

Le gestionnaire de caméras peut être créé en ajoutant un gameObject vide à votre scène (*menu GameObject/Create empty)*. Nommez cet objet *GestionnaireCameras* . Le script de détections des touches et du changement de caméras doit être placé sur cet objet.

**Exemple**

```
// Script de gestion des caméras à l'aide des touches

public GameObject camFPS ;                    // il faut initialiser dans l'Inspecteur

void Update ()
{
  if(Input.GetKeyDown("1"))
  {
    ActiverCamera(camFPS);  // Appel de fonction. Il faut avoir une fonction ActiverCamera() dans le script...
  }
}
// Désactive la caméra actuelle et en active la caméra choisie
void  ActiverCamera(GameObject  cameraChoisie)
{
   ...
}
```

---

## Pointer un objet vers un autre objet : LookAt(objet.transform)

Permet de faire pivoter un objet pour qu'il pointe vers un autre objet « cible » (toujours sur l'axe Z). Peut être utilisé pour aligner une caméra ou un autre type d'objet (exemple : un ennemi, une fusée, etc.)

La cible peut être un objet (« gameobject.transform ») ou une position dans le monde( Vector3(0,10,10) ).

**Exemple**

```
// Aligne l'objet pour qu'il regarde la cible

public GameObject cible;                    // la cible est déterminée dans la fenêtre inspector
void Update ()
{
   transform.LookAt(cible.transform);       // regarde la cible, il faut utiliser le transform de l'objet cible ou sa position
}
```

# Les types de mouvement de caméras

Il existe autant de techniques de contrôle de caméras qu'il y a des styles (gamepLay) de jeux vidéo. Elles peuvent avoir quelques lignes de code ou des centaines de lignes de code. Voici quelques caméras simples.

## Caméra première personne (FPS)

Il suffit de positionner et parenter la caméra au personnage. S'il n'y a pas de personnage 3D alors on peut utiliser une capsule créée dans Unity pour simuler l'espace physique du personnage.

## Caméra de surveillance (CameraFixe)

- Positionnez la caméra dans le monde.
- Placez un script avec la fonction **LookAt( )** pour regarder le personnage (ou une autre cible) . (voir l'exemple plus haut)

## Caméra de suivie à distance fixe (CameraDistanceFixe)

Dans le script de cette caméra, **à chaque frame** :

- positionnez la caméra à une distance fixe par rapport à la cible par programmation. La position d'un objet est obtenue par sa transform.position, alors:
  transform.position = (position de la cible + la distance voulue en **Vector3**)
- orientez la caméra vers la cible à l'aide de la fonction **LookAt ( )**

## Caméra troisième personne avec mouvement de suivi fluide, méthode Lerp( ) (CameraSuivieFluide)

Nous avons utilisé cette méthode dans Unity 2D pour suivre le personnage.

**Mathf.Lerp(valeur de départ, valeur de fin , facteurAmortissement); (en anglais : linear progression)**

Cette fonction permet de passer d'une valeur à une autre selon un facteur d'amortissement qui affecte le temps total de la transition. Le facteur est compris entre 0 et 1.

Plus le facteur est grand plus la transition est rapide. Si la valeur est de 0, alors il n'y aura pas de changement de valeur.

**Exemple** : si on veut augmenter la valeur d'une vitesse graduellement, à chaque frame Lerp fournit la prochaine valeur selon le facteur d'amortissement. Si la valeur est 0.1 alors la vitesse va s'approcher de 10% à chaque frame à la valeur maximale (0.2 = 20% etc.).

**Exemple (utilisé en 2D pour faire suivre la caméra)**

```
float vitesseMax=100.0f ;
float maVitesse=0.0f;
void Update ()
{//fournit la prochaine valeur en entre la valeur de départ et la valeur de fin selon un facteur
      maVitesse = Mathf.Lerp(maVitesse,vitesseMax, 0.1f);
      print (maVitesse);    // après la 1e fois, maVitesse sera 10, après la 2e fois 19, ensuit 27 .... à la fin 100
 }
```

### Vector3.Lerp(valeur de départ en vector3, valeur de fin  en vector3 , amortissement);

Cette fonction permet de passer d'un vecteur (x,y,z)  à une autre vecteur (x,y,z) selon le facteur d'amortissement spécifié.

**Exemple**

```
// Déplace l'objet du script (caméra, ennemi, balle etc.) à la position du joueur

public GameObject cibleJoueur ;                          // la cible est déterminée dans la fenêtre inspector
void Update ()
{
    //fournit la prochaine position entre la position de la caméra et la position du joueur selon une fraction de la distance
   //ici la caméra  va s'approcher de 20%
    transform.position = Vector3.Lerp(transform.position, cibleJoueur.transform.position, 0.2f);
}
```

### transform.TransformPoint(Vector3)

La position de la caméra 3e personne doit être à une certaine distance **derrière** le joueur (derrière l'axe Z locale et un peu haut selon l'axe Y). Pour déterminer cette position, on peut utiliser une des méthodes du transform de l'objet .

**Exemple**

```
// Définie une position 5 mètres en hauteur et 10 mètres en arrière de la cible (selon les axes locaux de la cible)
 Vector3 positionFinale = cible.transform.TransformPoint(Vector3(0, 5, -10));
```

Le script final est placé dans **FixedUpdate()** pour mieux synchroniser le mouvement d'un personnage rigidbody et celui de la caméra.

**Exemple**

```
// Script de la caméra 3e personne qui suit le joueur graduellement

public GameObject  cibleJoueur;                    // la cible est déterminée dans la fenêtre inspector

void FixedUpdate ()
{
    // Définie une position 5 mètres en hauteur et 10 mètres en arrière de la cible (selon les axes locaux de la cible)
    Vector3 positionFinale = cibleJoueur.transform.TransformPoint(0, 5, -10);

    // prochaine position entre la position de départ de la caméra et la position finale désirée selon un facteur 0.5
    transform.position = Vector3.Lerp(transform.position, positionFinale, 0.5f);

    // regarde toujours la cible
    transform.LookAt(cibleJoueur.transform);

    }
```

## Random.Range

static function **Range (min :** float**, max :** float**) :** float
static function **Range (min :** int**, max :** int**) :** int

**Description**

Retourne un nombre (float ou integer) à l'intérieur du min et du max (en incluant la valeur min et max).
Notez cependant que si des nombres entiers sont utilisés, la valeur max est exclue des possibilités.

**Exemple**

```
int aleatoire = Random.Range(5,10); // Nombre entier aléatoire entre 5 et 9

float  aleatoire2 = Random.Range(5f,10f); // Nombre à virgule aléatoire entre 5 et 10

//variable qui contiendra un vector3 avec un X et un Z aléatoires entre -10 et 10.

Vector3 position = new Vector3(Random.Range(-10f, 10f), 0, Random.Range(-10f, 10f));
```