## ATI L3 2021-2022

## Support de cours Unity

## Cours 4 : Character Controller, Components

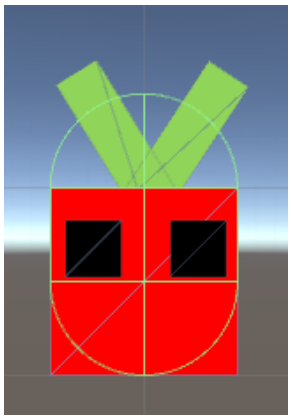
## Présentation

Dans ce cours, je vais vous parler du Character Contoller, un Component de Unity qui facilite largement la création de personnages qui se déplacent. Nous aborderons ensuite les Components avec un peu plus de détails, notamment par les commandes GetComponent et RequireComponent. J'expliquerai également comment accéder aux components d'une instance. Je terminerai avec la création de fonctions et d'un chronomètre.

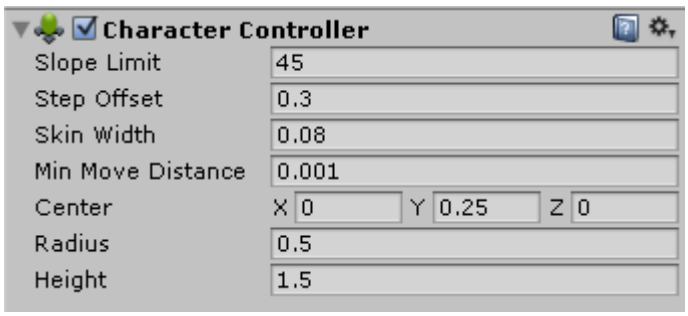
## Character Controller

### Définition

Le Character Controller est un des outils les plus pratiques dans la création d'un jeu vidéo. C'est un script qui gère le mouvement et la collision d'un personnage. Il est conseillé de retirer les components de collisions (genre BoxCollider) de l'objet sur lequel on le place, car il le remplacera de toute manière. Une fois placé sur un GameObject, la collision prend la forme d'une capsule.



Le Component se trouve dans Add Component > Physics > Character Controller.




Pour les quatre premiers paramètres, je vous invite à jeter un œil au manuel. Vous aurez rarement à les modifier. Ils permettent de gérer le comportement des contacts au sol, notamment pour les pentes.

Les trois derniers nous intéressent davantage. Center, Radius, et Height permettent d'établir la forme et la taille de la capsule. Elle doit recouvrir le personnage, afin de donner l'impression au joueur que c'est bien le personnage qui rentre en collision, et non une capsule.

## Usage

Pour déplacer le CharacterController, nous procédons d'une façon similaire au Translate : en employant un Vector3.

```
public Vector3 mouvement;
private CharacterController controller;

    void Awake()
    {
        controller = gameObject.GetComponent<CharacterController>();
    }
   void Update()
    {
        mouvement.x = Input.GetAxisRaw("Horizontal");
        controlle.Move(mouvement * Time.deltaTime);

    }
```

Remarquons l'aspect pratique d'avoir un Vector3 mouvement. Elle permet de modifier ses axes directement avec un GetAxis. Pour rappel, celui-ci récupère la touche gauche comme étant -1 et droite comme +1. Si aucune touche n'est appuyée, GetAxis renvoie 0.

Il existe deux commandes de déplacement : Move et SimpleMove. La seconde gère automatiquement la gravité, mais ne va pas nous servir puisque nous voulons justement jouer sur ce paramètre.

Il existe également une condition spéciale, le isGrounded, qui vérifie si le personnage est au sol.

```
mouvement.x = Input.GetAxisRaw ("Horizontal");
mouvement.y -= gravity * Time.deltaTime;

if (controller.isGrounded) {
    mouvement.y = 0;
}

controller.Move (mouvement * Time.deltaTime);
```

# Components

## Définition

Les Components sont les éléments placés sur l'objet qui déterminent son comportement. Vous pouvez tous les voir et els modifier dans l'Inspector lorsque vous sélectionnez un objet. Mais il est également possible de les modifier dans le script.

## GetComponent

Nous allons ici prendre l'exemple du CharacterController, mais le GetComponent peut aussi s'effectuer sur tous les autres éléments de l'objet.

Le CharacterController doit être stocké dans une variable avec un GetComponent si on veut l'utiliser dans le script (notamment avec un Move).

La formule simple donne ceci :

```
private CharacterController controller;

void Awake() {
    controller = GetComponent<CharacterController>();
}
```

Remarquez la déclaration à l'origine, puis l'assignation dans le Awake. A partir de ce moment là, nous avons accès à toutes les commandes qu'offre le CharacterController depuis notre script.

Il est important de bien respecter la syntaxe du GetComponent : avec le nom exact du component entre des singletons ( < > ), et suivi de parenthèses ouvertes/fermées.

Vous pouvez également utiliser un GetComponent sur un autre Script que vous avez créé, en mettant le nom du script entre les singletons. Mais si vous voulez utiliser des variables ou fonctions de ce script, elles doivent être publiques.

## RequireComponent

Lorsque vous créez un script, vous pouvez avoir besoin qu'un autre script ou component soit impérativement également présent sur l'objet. Par exemple, si vous avez un script qui emploie des commandes de Rigidbody ou de CharacterController, vous pouvez lui imposer d'avoir ce component.

La commande se place avant la déclaration de classe.

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(CharacterController))]
public class Player : MonoBehaviour {
```

Ici, j'indique que mon script a absolument besoin d'un CharacterController pour fonctionner. Le jeu ne se lancera pas s'il n'y en a pas.

Également, lorsque vous placerez ce script sur un objet qui ne contient pas de CharacterController, il y en aura un qui se placera automatiquement. Mais si vous rajoutez la ligne après avoir ajouté le script à l'objet, il n'y aura pas de CharacterController placé automatiquement.

## Accès au Component d'une instance

Lorsque vous créez une instance, il peut être utile d'accéder à un script de cette instance. Dans le cas de MeatBoy, nous réalisons une instance d'une goutte et nous transformons sa vélocité. Pour cela, il faut déclarer une variable GameObject temporaire qui contient l'instance. Nous accédons ensuite au script qui nous intéresse par ce GameObject.

```
cptGoutte -= Time.deltaTime;

if (cptGoutte <= 0f)
{
    GameObject goutte = Instantiate (gouttePrefab, transform.position + goutteOffset, Quaternion.identity) as GameObject;
    goutte.GetComponent <ScriptGoutte>().velocity = new Vector3 (-mouvement.x, speed, 0f);
}
```

Plusieurs points importants :

- On déclare un GameObject de n'importe quel nom tant qu'on le retrouve.
- On assigne à ce GameObject notre instance.
- On n'oublie pas le « as GameObject » à la fin. Instantiate créer un Object et non un GameObject. Il faut réaliser une conversion car la commande GetComponent n'existe qu'avec un GameObject.
- Nous pouvons ensuite utiliser le GetComponent en cherchant le nom de notre script. Il faut bien être sûr que l'objet contienne ce script.

## Fonctions

### Fonctions de Unity

Nous avons vu plusieurs fonctions propres à Unity. Ces fonctions ont un nom particulier que Unity peut reconnaître et utiliser de manière particulière. Nous avons vu :

- Start, qui permet de lancer une commande au démarrage.
- Update, qui permet de lancer une commande à chaque frame.
- OnCollision/TriggerEnter/Exit/Stay, qui permettent de détecter une collision, ou bien un objet qui passe à travers un autre.
- OnBecameInvisible/Visible, qui permettent de détecter quand un objet devient invisible/visible.

Mais il en existe encore d'autres. Notamment, j'ai envie de vous parler du :

- Awake est très similaire au Start et se lance au démarrage. Mais le Awake se lance avant le Start. D'autre part, le Awake se lance au démarrage de la scène que le script soit actif ou non, alors que le Start se lance la première fois que le script est actif.

### Fonctions personnelles

Si vous créez une fonction avec un nom que Unity ne reconnaît pas, il ne fera rien de cette fonction par défaut. Créer une fonction s'écrit de la sorte :

```
void MaFonction () {
    // faire des trucs
}
```

Le nom de la fonction doit être précédé de void, afin d'indiquer qu'il s'agit d'une fonction.

Les intérêts d'une fonction sont multiples :

- Ça structure votre code, et le rend plus lisible, aussi bien pour vous que vos camarades ou votre prof.
- Elles permettent d'avoir une série de commandes qui peuvent être appelées n'importe quand et plusieurs fois.
- Si une fonction est publique, elle peut même être appelée de l'extérieur. Ainsi, chaque script peut se gérer à sa manière et il n'y a pas besoin d'un script de « chef d'orchestre ».

### Fonctions publiques

Pour qu'une fonction puisse être appelée depuis un autre script (avec un GetComponent, par exemple), il faut qu'elle soit publique.

```
public void MaFonctionPublique () {
    // faire des trucs
}
```

## Chronomètre et Score

### Créer un compteur

Un compteur n'est rien d'autre qu'une variable flottante qu'on incrémente ou décrémente par le Time.deltatime. Tout l'intérêt d'un compteur est de réaliser des opérations rythmées.

L'exemple ci dessous montre un compteur basique qui diminue et réalise quelque chose lorsqu'il atteint zéro.

```
public float delay;
private float cpt;

void Awake() {
    cpt = delay;
}

void Update() {
    cpt -= Time.deltaTime;

    if(cpt <= 0f)
    {
        // réaliser mon comportement ici
    }
}
```

Notez la présence de deux variables : un délai en public qui peut être modifié par l'utilisateur (vous), et un compteur privé qui sera décrémenté en temps réel. Avoir ces variables séparées permet de réinitialiser le compteur à la valeur de delay pour avoir une action régulière.

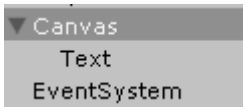
Notez également qu'à cause du caractère imprécis des float, il vaut mieux vérifier pour une valeur inférieure ou supérieure, plutôt que strictement égale.

### Mettre à jour un GUI (afficher un score)

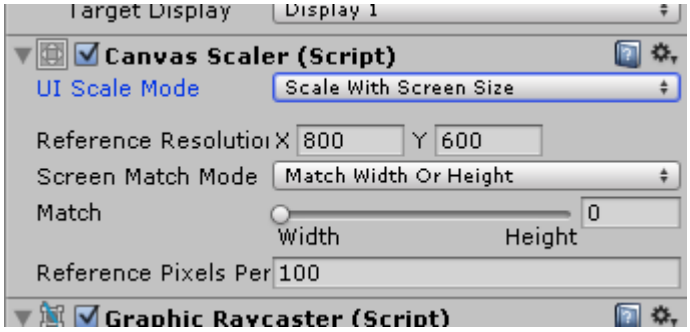
Je vous ferai un cours complet sur les GUI une autre fois. Je vais juste vous décrire les étapes à suivre pour mettre en place un texte modifiable.

### *Créer un GUI Text*

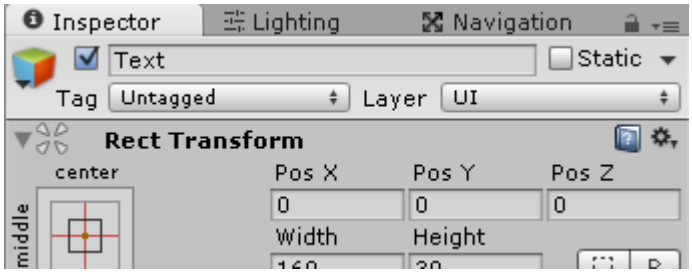
Faites un Create > UI > Text. Unity va vous créer 3 objets : un Canvas avec un Text parenté, et un EventSystem.



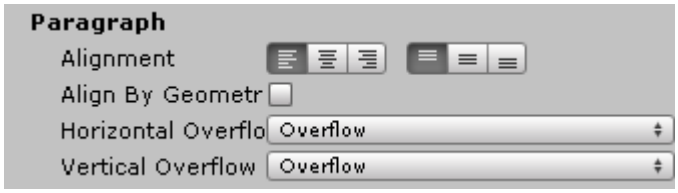

Selectionnez votre Canvas et mettez son Canvas Scaler en Scale With Screen Size.




Selectionnez votre Text et mettez le en (0,0,0) afin de le centrer sur l'écran. Replacez-le où vous voulez.




Toujours sur le texte, changez son Overflow horizontal et vertical pour Overflow. Ça permet au texte de ne pas être limité par sa « boite » et de déborder.




### *Mettre à jour un GUI Text par le script*

Tout d'abord, il va falloir rajouter quelque chose dans la section « using » du script.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class ScoreManager : MonoBehaviour {
```

Ensuite, déclarez un public Text dans vos variables. Assignez-lui, dans l'Inspector, l'objet dont vous voulez modifier le texte.

```
public Text monTexteDeScore;
```

Il vous faudra ensuite assigner une valeur (de type string, une chaîne de caractères) à la

variable « text » de cet objet.

```
public Text monTexteDeScore;
private int score;

void SetScoreToText () {
    monTexteDeScore.text = "Score : " + score.ToString();
}
```

## Enregistrer un score

Les PlayerPrefs sont un moyen pour Unity d'enregistrer des valeurs sur la machine de l'utilisateur afin de pouvoir les retrouver lors d'un autre lancement du jeu. Les PlayerPrefs peuvent être utilisés pour enregistrer la progression du joueur, en enregistrant sa position, son inventaire, etc.... Ou bien ils peuvent être employés de manière plus simple, pour enregistrer un meilleur score.

Les PlayerPrefs peuvent enregistrer trois types de variables : int, float, et string.

Pour enregistrer une valeur dans les PlayerPrefs, il faut employer un PlayerPrefs.SetInt (ou SetFloat ou SetString).

```
private int score;

void SetBestScore () {
    PlayerPrefs.SetInt ("BestScore", score);
}
```

Le string saisi en premier paramètre est le nom de la variable que vous voulez enregistrer. Assurez-vous de bien la réécrire de la même manière à chaque fois que vous voulez l'utiliser, ou vous enregistrerez deux variables différentes !

Pour utiliser une de ces valeurs, il vous faudra faire un **GetInt/Float/String** à la place. Mais si la variable n'a pas encore été crée, notamment lors du premier lancement du jeu, il pourrait y avoir un souci ! Il va falloir utiliser une vérification **HasKey**, de la sorte :

```
private int bestScore;

void Start () {
    // si la variable existe sur le PC et a déja une valeur
    if (PlayerPrefs.HasKey("BestScore") == true)
    {
        // je récupère sa valeur
        bestScore = PlayerPrefs.GetInt("BestScore");
    }
    else // sinon, donc si la variable n'existe pas sur le PC
    {
        PlayerPrefs.SetInt ("BestScore", 0f); // je la crée
        bestScore = 0f;
    }
}
```