Chapitre 11 - Atelier : Raycast

Les raycast ouvrent la porte à beaucoup de mécanique de jeu et valent la peine à ce qu'on leur consacre un cours entier pour les maitriser. Ils auront une grande place dans l'examen final et dans votre TP3. Vous avez l'occasion aujourd'hui de vous pratiquer.

11.1 Ray

Un "Ray" est définit par un point d'origine et une direction. Pour créé un rayon, nous avons vu comment le faire avec un point d'origine dans la camera avec :

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
```

Pour dessiner notre rayons, Unity nous fourni une méthode dans Debug :

```
Debug.DrawRay(origin, direction * k, color, t);
```

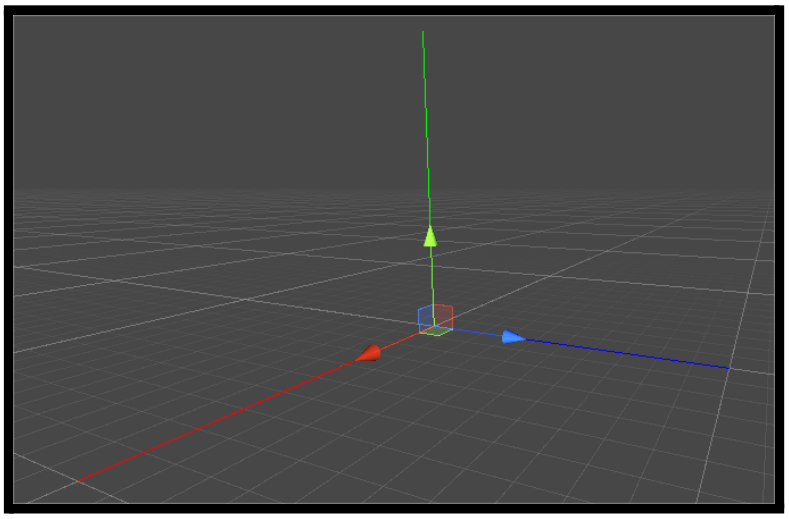
Origin : l'origine du rayon;
Direction : la direction du rayon;
K : une magnitude (pour l'affichage, nous n'utilisons pas l'infini);
Color : la couleur du rayon a l'écran;
T : un délai d'affichage.

Cela peut être extrêmement utile de pouvoir voir où sont vos rayons lors du développement d'un jeu. N'hésitez pas à utiliser cette méthode pour vous donner des chances.

Nous pouvons aussi créer un rayon sans l'aide de la caméra en utilisant « new » et en lui fournissant les informations nécessaires : un point d'origine et une direction.

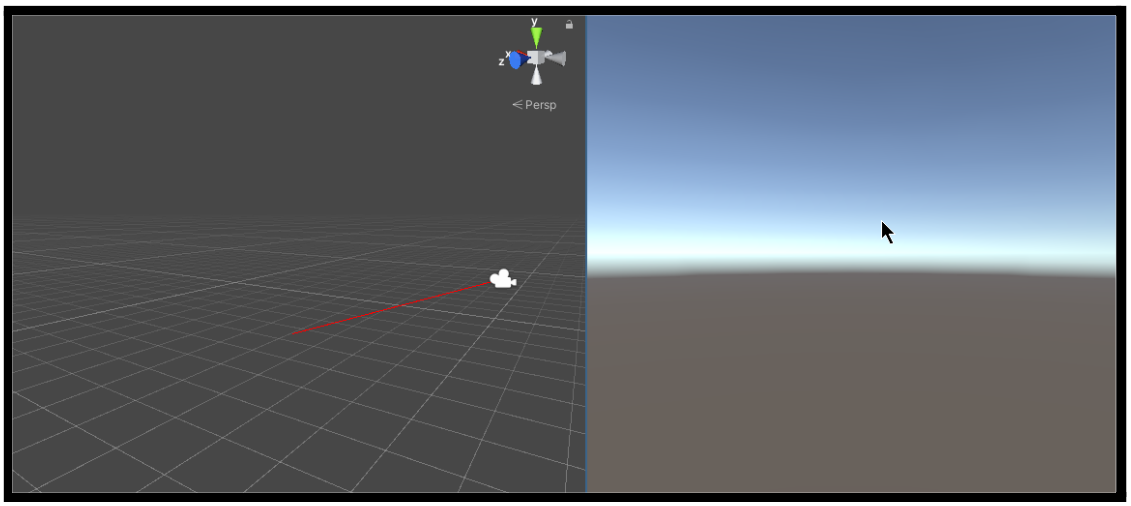
11.2 World Cross

Notre première pratique consiste à dessiner, à l'aide de rayon, les axes du monde dans une scène.



11.3 Camera Ray

Dessinez un rayon qui commence à la position du curseur de la sourie sans avoir à cliquer.



11.4 Raycast

Avec de tels rayons, l'engin de physique de Unity est capable de nous dire s'il y a une intersection entre le « ray » et un collider dans la scène.

```
bool Raycast(Vector3, Vector3, float, int, QueryTriggerInteraction);
```

1. L'origine du rayon;
2. La direction du rayon;
3. La distance maximale;
4. Un masque de layer pour spécifier quel layer on veut utiliser.
   Exemple: `LayerMask.GetMask("TestLayer");`
5. Le dernier paramètre est pour spécifier si l'on désire détecter les triggers ou pas.

À noter que les paramètres qui ont des valeurs par défaut n'ont pas à être fournis si la valeur par défaut vous convient. Avec cette version, nous n'avons qu'une valeur de retour vrai ou faux qui nous dit : oui, le rayon entre en collision ou pas.

```
bool Raycast(Vector3, Vector3, out RaycastHit, float, int, QueryTriggerInteraction);
```

Cette version est la même chose, excepter qu'une variable de type RaycastHit sera remplie d'information sur le point de collision. Vous pouvez avoir la position de la collision et le transformé et donc le game object qui est entrée en collision avec le rayon.

```
bool Raycast(Ray, float, int, QueryTriggerInteraction);
```

Cette version vous permet de passer directement un objet « Ray » et optionnellement une distance maximale, un masque de layer et si vous voulez détecter les triggers. Une fois encore, cette version ne retourne qu'un booléen comme information sur l'intersection éventuelle.

```
bool Raycast(Ray ray, out RaycastHit, float, int, QueryTriggerInteraction);
```

Cette dernière version nous permet d'avoir en plus, un objet RaycastHit comme information sur la collision. Voici un exemple simple de Raycast et comment aller chercher les informations du l'objet qui est entré en collision.

```
public class CastSimpleRay : MonoBehaviour
{
    private void Update()
    {
        if(Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            RaycastHit hitInfo;
            if(Physics.Raycast(ray, out hitInfo, 1000.0f))
            {
                string objectName = hitInfo.transform.name;
                string objectTag = hitInfo.transform.tag;
                Debug.Log($"name: {objectName}, tag: {objectTag}");
            }

            Debug.DrawRay(ray.origin, ray.direction * 100.0f, Color.red, 3.0f);
        }
    }
}
```

11.5 RaycastAll

Jusqu'à maintenant, la méthode de Raycast nous retourne l'information seulement sur le premier objet que nous détectons. La méthode RaycastAll nous retourne elle, une liste de tous les objets qui sont dans le chemin du rayon.

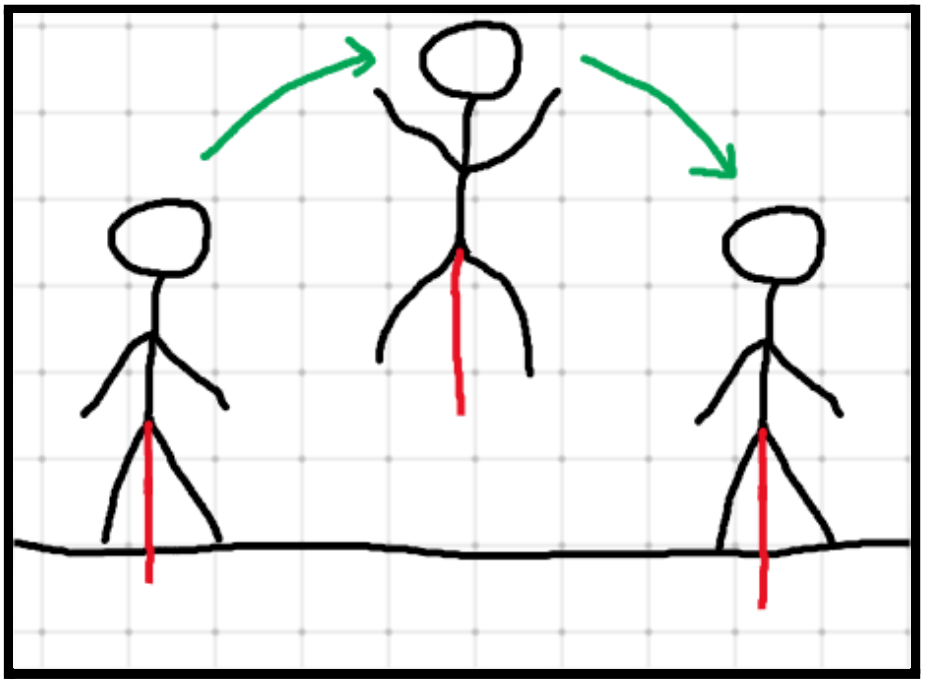
```
public class CastAllRay : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            RaycastHit[] hitInfos = Physics.RaycastAll(ray, 100.0f);

            if(hitInfos.Length > 0)
            {
                foreach (RaycastHit hit in hitInfos)
                {
                    string objectName = hit.transform.name;
                    string objectTag = hit.transform.tag;
                    Debug.Log($"name: {objectName}, tag: {objectTag}");
                }
            }

            Debug.DrawRay(ray.origin, ray.direction * 100.0f, Color.red, 3.0f);
        }
    }
}
```

11.6 Application des Raycasts

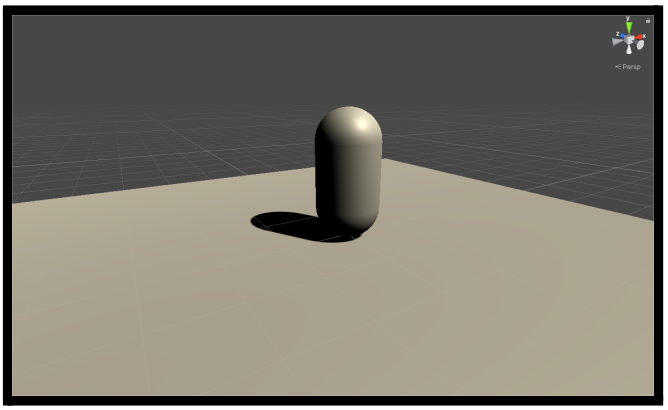
Maintenant, vous savez comment utiliser les Raycasts, le moment d'étudier un peu leur application est venu. Dans cette section et pour le reste du cours, vous allez avoir des labos à faire. Ces labos vous préparent à l'examen final et au dernier travail pratique. C'est le moment de vous pratiquer et de poser des questions. Allez-y à votre rythme et chacun votre tour je répondrai à vos interrogations.

11.6.1 Jump

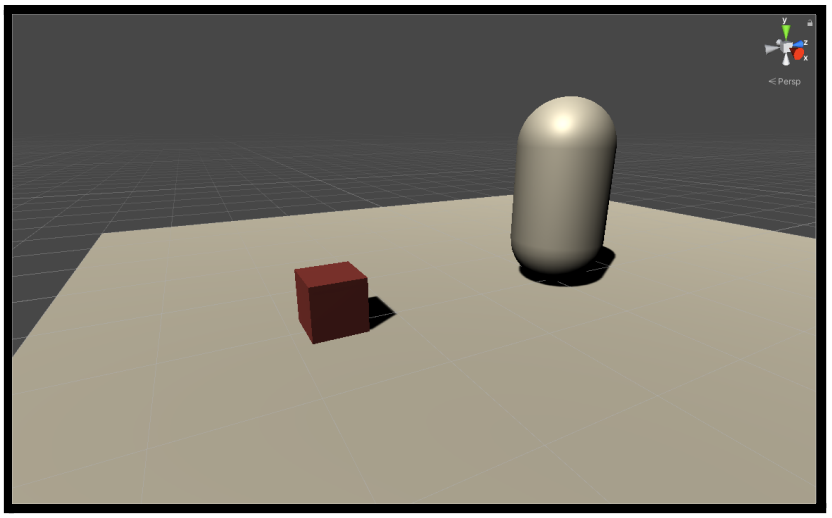
L'utilisation des raycasts est la méthode standard pour détecter le « ground » pour empêcher le joueur de sauter s'il n'y touche pas.



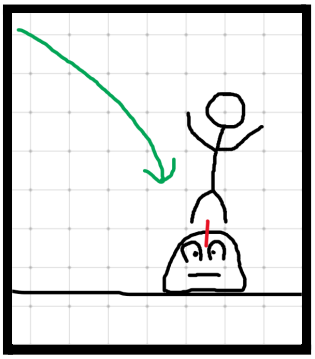
Montez la scène suivante et empêchez les sauts si le joueur (*la capsule*) ne touche pas au plancher (*le plane*).



11.6.2 Jump sur un objet



Ajoutez un cube et si votre personnage entre en collision avec le cube, il est détruit. Mais s'il saute sur le dessus du cube, c'est le cube qui est détruit.



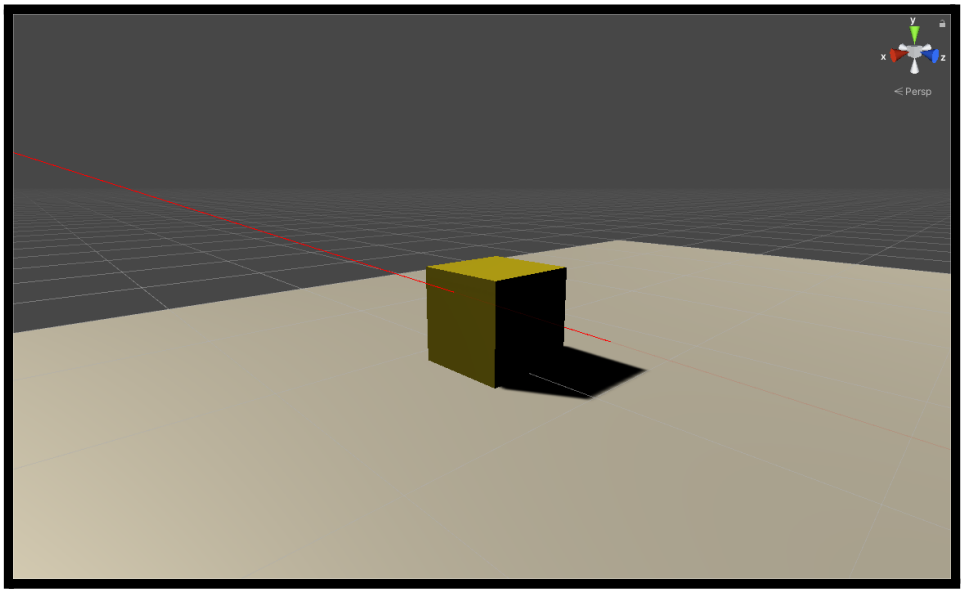
NOTE : je vous rappelle que la méthode pour détruire un objet est « Destroy ».

11.6.3 Interaction

Dans une scène, préparer une scène avec un cube. Lorsque vous cliquez dessus, le cube doit changer de couleur.

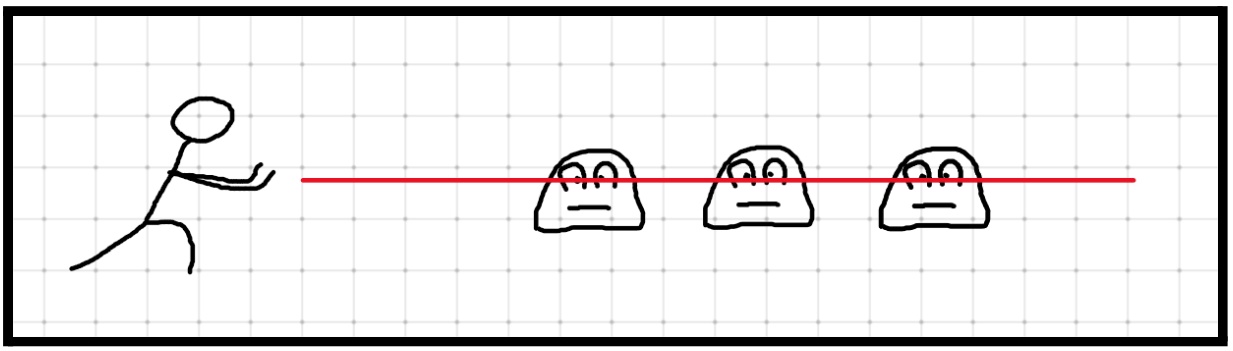
Indice : utilisez ce code pour changer la couleur du matériel sur le cube.

```
hitInfo.transform.GetComponent<MeshRenderer>().material.color = Color.yellow;
```



11.6.4 Tir

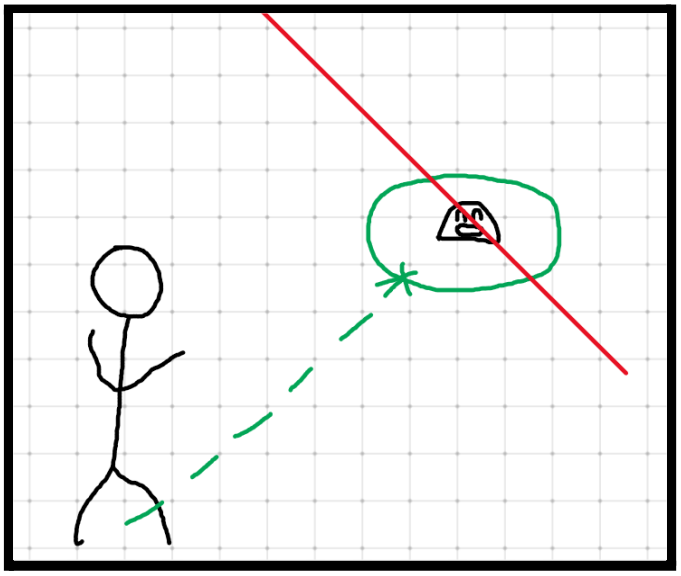
Lancer un Raycast et afficher, dans la console, combien de cible vous avez touché.



11.6.5 Attaques, P1

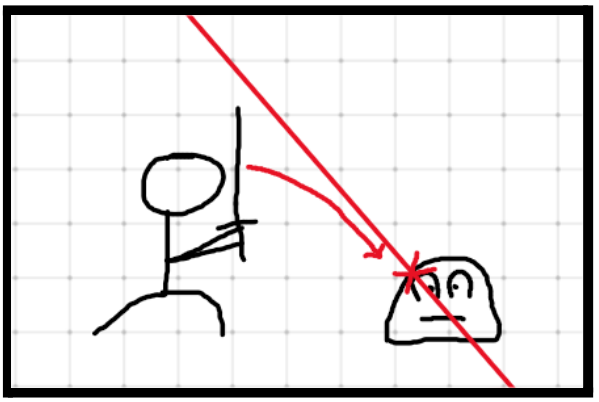
Monter une scène avec un NavMesh. Déplacer votre personnage où vous avez cliqué. Votre joueur doit avoir un « attack range » et arrêter lorsqu'il est assez proche pour attaquer.

Indice : pour arrêter un NavMeshAgent, utilise la propriété isStopped.



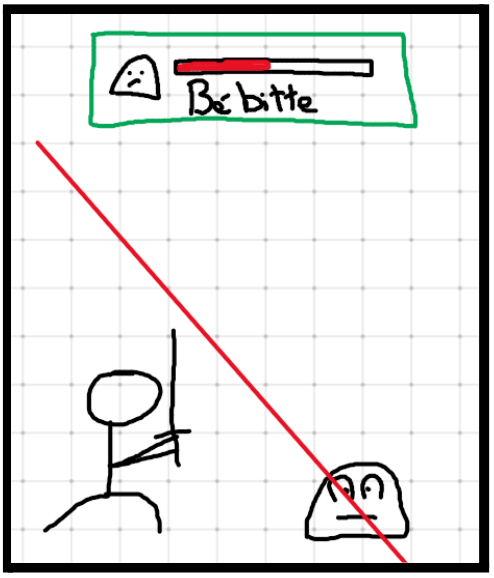
11.6.6 Attaques, P2

Une fois dans le range d'attaque, votre personnage doit aller chercher le bon composant de l'ennemie et lui donner du dégât. Ajoutez une fonction TakeDamage dans le script de votre ennemie et faites un GetComponent sur le hit du raycast.



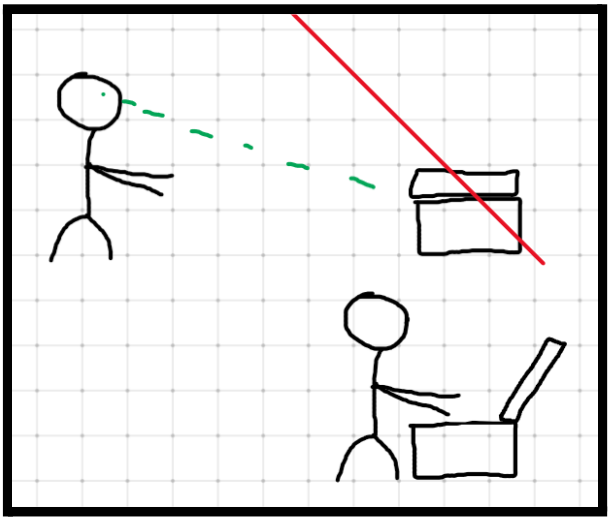
11.6.7 Informations

Lorsque le curseur de la souris passe au-dessus d'un ennemi, afficher des informations sur celui-ci dans la console.



11.6.8 Activation d'animations

Lorsque vous cliquez sur un objet (porte, coffre, etc.), lancez une animation de celui-ci, si votre personnage est à porter. Sinon, déplacer le personnage puis activer l'objet.



11.6.9 Selection

Placez plusieurs objet, un à la suite de l'autre. Ajouter un script "Enemy" a l'un d'eux et lancer un rayon à travers tous les objets. Détruire seulement selui qui a le script.